Automatic Generation of Simulation-Based Web Courses and Model Documentation

Manuel Alfonseca and Juan de Lara

Dept. Ingeniería Informática, Universidad Autónoma de Madrid, Ctra. De Colmenar, km. 15, 28049 Madrid, Spain; E-Mail: {Manuel.Alfonseca}{Juan.Lara}@ii.uam.es

This paper presents the procedures and tools that we are using to generate fully automatic multimedia Web courses based on simulations. These courses are composed of HTML pages with interactive simulations that help the students understand the subject of the course. This is accomplished by means of an object-oriented simulation language (OOCSMP) that allows us to include information about the appearance of the HTML page where the simulation model is going to be included. The language incorporates constructions to synchronize multimedia elements with the simulation execution, and to produce distributed simulations. The compiler for this language (C-OOL) generates Java applets for the simulation problem, and can generate automatically the documentation for the models, in the form of HTML pages, using information in the symbol table and special comments included in the model. One example of the construction of a multimedia simulation page with the simulation of an ecosystem is presented, and its extension to a distributed simulation environment is explained.

Keywords: Continuous simulation, course generation, Web-based simulation, distributed simulation, multimedia-enriched simulation, automatic documentation, education, OOCSMP, HTML, Java, C-OOL

1. Introduction

The currently most successful hypermedia system is the World Wide Web (WWW), which has many advantages over traditional hypertext applications. This has brought about the current proliferation of educational courses on the WWW [1, 2], which run from a simple transposition of lecture notes, to pages including more sophisticated elements, such as animated graphics, simulations and so forth. Students are also more and more familiar with browsing the Web and playing computer games. Such Web users, in order to use simulation, desire tools that allow quick and easy experimentation [3].

The clear interest towards this field has created a need for adequate tools to help in the elaboration of the courses, which should make it possible to express all the possibilities offered by WWW teaching [4, 5]. In the words of the Directorate-General XIII [6]: "*From now on, there must be stress on helping to develop authoring tools, easy to use by the teachers who wish to include in their teaching methods multimedia elements (both local and on the Web).*"

Integration of simulation and Web services can be done in several ways [7]:

- Thick server approach. In this case, the simulation programs execute at the server, programmed in any language accessible through the Common Gateway Interface (CGI). This approach centralizes the execution of the models, but increases the traffic of data in the Web, which may generate performance problems.
- Thick client approach [8]. The simulation tools (interpreters, plug-ins, etc.) have to be preloaded or downloaded to every client, and then execute there. This approach decreases the data traffic at the cost of stringent client requirements. If executable programs are downloaded, there is a danger of client incompatibility and virus transmission.
- Pure navigator approach. The models are integrated with the HTML pages making up the educational courses and executed as Java "applets." Java has many interesting properties, such as "*write-once, run-everywhere,*" that provide client independence. This approach means the initial loading of the pages will be slower (the applets have to be downloaded), but the model execution may be fast.
- Distributed execution. The models execute simultaneously in several machines that cooperate with each other. If Java is used as the programming language, such programs may communicate by means of Remote Method Invocation (RMI) [9], the Java analogue to the traditional and well-known Remote Procedure Call (RPC) for distributed computing. In the Java distributed object model, a remote object is one whose methods can be invoked from another Java Virtual Machine, potentially on another host.

Received: Month Year; Revised: Month Year; Accepted: July 2000

TRANSACTIONS of The Society for Computer Simulation International
ISSN 0740-6797/00

Volume 17, Number 3, pp. ##-##

Acknowlegement

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

- Distributed modeling [10]. Models can be distributed across the Web. They reside where they can best be cataloged, indexed, maintained, etc. The model designer can browse different components and combine them to build a more complex model. This approach simplifies collaborative model definition and model reuse.

There are two main ways of reducing the effort needed to build the models:

- Providing a library that contains pre-defined classes that may be used by the user to build the model. In this case, the model builder usually programs in a general purpose language, such as Java [11–15].
- Using a special-purpose simulation language and a compiler that translates the models into other languages, such as Java and C++. This is our approach.

Parallel and distributed simulation (PADS) [16] has flourished in the latter 1990s [17], having been included in the High Level Architecture (HLA) [18], and in air traffic and transportation systems. Transparency as to machine assignment in heterogeneous environments is one of the essential properties of distributed simulation applications. A proper synchronization of components is also very important. The Java language is especially applicable to solving this problem.

We have been working for some time on the development of advanced simulation tools that simplify the generation of educational courses on the WWW. The courses contain interactive simulation programs that allow the users to explore the proposed problem, to make changes, etc., promoting a more active role of the student in the learning process and "*learning by discovery*." This follows the tendency of higher education to move from a teacher-centered paradigm to a student-centered paradigm [19]. Our work is motivated by the lack of tools directed to integrating simulation models in courses for the Web. The languages, tools and procedures that will be presented in this paper are a first step towards an authoring tool for the construction of simulation-based Web courses.

The language we are using is an extension of the old CSMP (Continuous System Modelling Program) language, sponsored by IBM [20]. We call the new language OOCSMP [21], for its main difference from CSMP is the addition of object-oriented constructs which make much easier the simulation of complex systems based on the mutual interaction of many similar agents (which can be modeled as collections of objects). The language can also solve systems of PDEs using finite difference and finite element methods, and has constructions to produce distributed simulations.

This language has been used to build a course on Newton's gravitation and the solar system [22], a course on ecosystems [23], and a course on partial differential equations [24]. Capabilities to handle multimedia elements and discrete events have also been added to the language.

In this paper, we explain the methodology we have developed to:

- Generate the courses (from the design of the simulation model, to the publishing on the Web).
- Integrate the multimedia elements in the simulation.
- Automatically generate the model documentation, in the form of HTML pages.
- Produce distributed simulations.

The third and the fourth point, as well as the instructions to describe the appearance of the HTML pages, are our new contributions described in this paper.

The paper is organized as follows: Section 2 presents the procedure that we follow to generate courses based on simulation for the Web. An example of a page of a course on ecology is shown. In Section 3, the compiler that generates the courses is presented. Section 4 discusses the extensions added to the language to produce distributed simulations, and describes an ecosystem model that can be executed in a parallel manner. Finally, in Section 5, we present the conclusions and our future work.

2. Automatic Generation of Courses

We have developed a procedure to generate generic Web courses based on simulation. The procedure can be applied to any course dealing with a scientific or technical subject. The course will consist of HTML pages with simulation applets in them. The procedures and tools allow us to program (in a high-level language) and obtain both the HTML pages and the applets.

In a previous publication [25], this procedure was described as partially automatic. It has been improved to achieve full automation, and to permit including multimedia elements in the simulations. This procedure is shown in Figure 1.

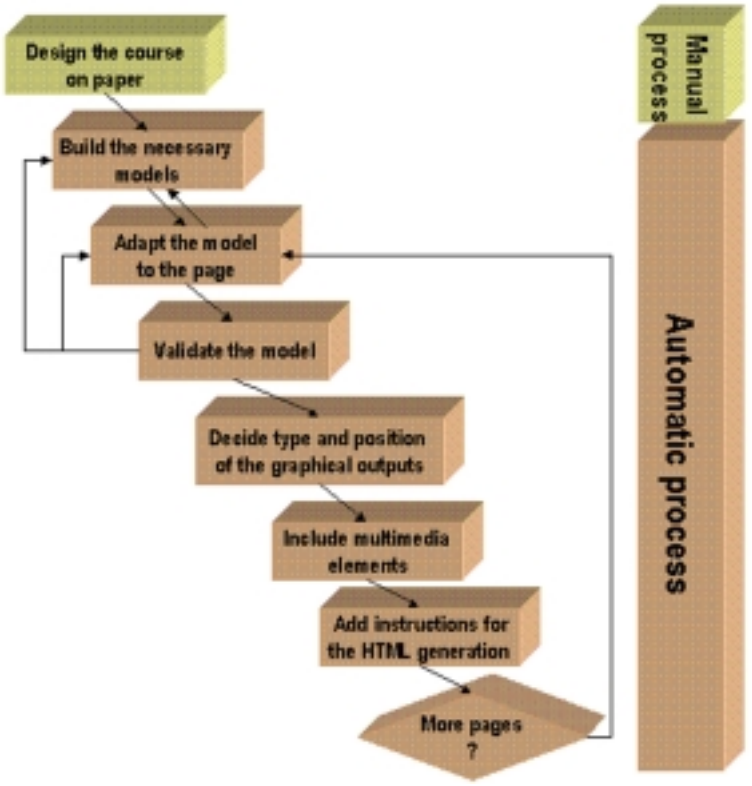

Figure 1. Procedure to generate Web courses

The steps of the procedure will be explained in the following sections. At the same time, the design of a course on ecology and the construction of a course page will be shown in detail.

2.1 Step 1: Designing the Course on Paper

The first step is to design the course on paper. Depending on the course, a single model may be used in one or more (sometimes all) pages. The steps of the procedure can be applied to any course that contains simulations. In our case, we want to design a course on ecology; in another application, the models would be different, but the steps to be followed remain the same.

The models will be based on the Volterra equations [26] generalized to make them applicable to multi-level multi-species ecosystems. Other approaches to ecological simulations are based on multi-agent simulations [27], multimodels [28], formal grammars [29], cellular automata [30], etc. The course will consist of seven pages:

- An introductory page, with a three-species ecosystem (one primary producer, one prey and one predator), with the appropriate parameters to bring the ecosystem out of equilibrium. This page will demonstrate the periodicity of this kind of system.
- A three-species system in total equilibrium.
- A three-species system, originally in total equilibrium, which is invaded after some time by a new predator. The invasion takes the system out of total equilibrium, but it soon reaches an oscillatory stability.
- The same system, this time invaded by a new prey. The periodical stability attained after some time is different from the preceding case.

Listing 1. Species class (file Species.csm)

```
INCLUDE "macros.csm"
* TITLE class Species
* BACKGROUND "ck040bg.gif"
* ABSTRACT This file contains the \ITALIC{Species} class, used to encapsulate
** all the behaviour of a \ITALIC{Species} in one ecosystem.
* AUTHOR   Juan de Lara (\LINK{"www.ii.uam.es/~jlara" www.ii.uam.es/~jlara })
* EMAIL Juan.Lara@ii.uam.es
* DATE 2/11/99
* BAR [C,75]
CLASS Species
{
 * name : The Name of the Species.
 NAME name
 * icname : The icon that will be used in some graphical representations.
 ICON icname
* X0 : Initial population.
* M, N1, N2 (\N2) : Parameters for the Volterra equations.
* start : (\start) time when the species enters in the ecosystem.
* max : (\max) maximum population of the species.
* K : (\K) proportionality constant.
 DATA X0, M, N1, N2:=0, start:=0, max:=1000, K:=1
* Ill (\Ill), When (\When), Int (\Int) : Variables to cause epidemics.
 DATA Ill:=10, When:=1000, Int:=1000
 DYNAMIC                                        * Begin main simulation loop
    X:=STEP(start)*LIMIT(0,max,XT)              * Limit the population at reasonable size
    XT:=INTGRL(X0,XP)                           * Calculate actual population
    XP:=K*X*(M-IMPULS(When,Int)*Ill)            * Produce an epidemic, if appropriate
    Xpdel1 :=0
    Xpdel2 :=0
    Teats  :=0
    Teats0 :=0
 ACTION Species S, Percent, Last
*  Interaction between species of the same trophic chain
   Xpdel1 +=INSW(Percent, Percent*S.X*S.Teats0/S.Teats, 0)
   Xpdel2 +=INSW(Percent, 0, Percent*S.X*S.X/S.X0)
   Teats  +=INSW(Percent, 0, 1)*S.X
   Teats0 +=INSW(Percent, 0, 1)*S.X0
   XP +=INSW(Percent, Last*K*N2*X*Xpdel1*X/X0, Last*K*N1*X*Xpdel2*Teats0/Teats)
}
```

- A five-species system in equilibrium.
- A five-species system with a user interface that lets the student modify the different parameters to perform experiments. Some of these experiments are suggested by the text in the page, but the student may perform many more.
- A simulation of an ecosystem with 15 different species that interact to build complicated trophic chains and ecological niches. This ecosystem is a simplification of the savanna ecosystem.

Each page will provide explanations in the appropriate moment of the simulation. This will be done by including multimedia elements synchronized with the simulation execution.

2.2 Step 2: Building the Necessary Models

In the second step, we build the models in our continuous simulation language, OOCSMP. In our case, all the models will be based on a single OOCSMP class, called *Species* that will encapsulate the behavior of a species in one ecosystem. Then, we will declare one object for each species in the ecosystem. The OOCSMP code for the *Species* class is shown in Listing 1.

In the listing, we have added some special comments (the lines beginning with an asterisk) to help the compiler generate the model documentation. To achieve this task, the compiler also takes advantage of the information held in the symbol table (the classes declared, the methods, parameters and types they have; the objects and their types; the procedures declared, etc.). The documentation consists of an HTML file for the main model, plus an extra page for every class used in the model. Parameter and object classes are linked to the corresponding class documentation page. We have added new entries in the compiler symbol table to hold the name of the HTML file and the location inside the file where each class documentation is located.

Several new special comments have also been included to provide the compiler with extra information about the model construction. They indicate the model author, the e-mail address (a "*mailto*:" HTML tag is created), an abstract of the model behavior, the date when the model was programmed, its title, etc. All the comments can be included in the OOCSMP main model or inside the definition of a class. Additional comments control the visual aspect of the documentation (bars, links, tables, images with associated explanations, etc.) HTML native code can also be included inside the OOCSMP model, by means of the *HTML* instruction.

We can tell the compiler how to translate some sentences to HTML (or any other tagged language such as Tex), by means of macros. For example, the following macro defines how to change the background appearance:

```
TRANSLATE "BACKGROUND bckg",
  "<BODY BACKGROUND=bckg>"
```

The file "*macros.csm*" includes this and other macros to make sections with associated targets, to include preformatted text, etc.

It is also possible to format the text by means of special tags. Those tags make it possible to do things such us:

- Change the text style, the size, the font; insert list items, targets, links, etc.
- Give access to the name of the author, the e-mail address and the date (if specified before) by means of *\AUTHOR*, *\EMAIL* and *\DATE*.
- Access the initial values of the simulation variables (in Listing 1 this has been done for variables *N2*, *start*, *max*, *K*, *Ill*, *When* and *Int*). If the variable is a vector or a matrix, an *HTML* table is created to show its values.
- Compute simple expressions (not involving *OOCSMP* blocks, just diadic and monadic operators). This is done by inserting the expression between two "*$*" symbols.
- The compiler counts the number of tables, images and items in a list, and those counters can be accessed by means of the tags *\TCOUNT*, *\ICOUNT* and *\ITEMCOUNT*, which can take part in any expression and are useful to make ordered lists, and to reference images and tables automatically.
- Access other variables, such as the current date, the current time, the name of the file, etc.

Compound formats can be created, by means of the *STYLE* instruction. For instance:

```
STYLE "\LLINK{ ", "\ITEM{ \LINK{ "
```

creates a compound style called "*\LLINK*" that can be used to create a list of links. The styles in a compound style can also be compound.

2.3 Step 3: Adapting the Model to the Page

In this stage we have to design the simulation runs for every page in the course. In our example, class *Species* will be used, but each ecosystem will contain a different number of species with different *M*, *N1* and *N2* coefficients, to simulate equilibrium and oscillating equilibrium situations. In the last page (the one with the simulation of the simplified savanna ecosystem) we will use realistic data [31] for the coefficients and the trophic chains.

As an example for this stage, the construction of page three (invasion of a predator) will be shown. In this page, we have initially an ecosystem with three species in equilibrium (*Lion*, *Gnu* and *LGrass*), a second predator (*Cheetah*) will invade the ecosystem at time = 50. We will create four *Species* objects, and the values of the coefficients are tailored so that the system is initially in total equilibrium (the derivatives of all the populations must be equal to zero). For the *Cheetah* object, the default value of variable *start* will be overridden with a value of 50. The resulting model is shown in Listing 2.

2.4 Step 4: Validating the Model

At this point, we have to test the models. The compiler provides a fast, easy-to-use standalone environment that simplifies testing and allows the course-writer to experiment with many different situations. Depending on compiler options, we can choose between generating C++ or Java code for the problem. For testing

Listing 2. Model for the predator invasion page (file is *africa1.csm*)

```
TITLE Three species , invasion of predator
INCLUDE "Species.csm"
* Actual species
Species Cheetah("Cheet", "icons/wcat002.gif",   4,-.028,.0014, .0   , 50)
Species Lion   ("Lion",  "icons/lion002.gif",   2,-.02, .001        )
Species Gnu    ("Gnu",   "icons/bovin008.gif",  20,-.02, .0001, .016666666 )
Species LGrass ("LGrass","icons/leafs015.gif", 400, .01, 0,     .0005)
DYNAMIC                                        * Begin main simulation loop
   Species.STEP()                              * Invoke main section on all Species
   Cheetah.ACTION(Gnu,       1, 1)             * Begin interactions ...
   Lion.ACTION   (Gnu,       1, 1)             * ...between species
   Gnu.ACTION    (Lion,   - .6, 0)             * the methods are invoked...
   Gnu.ACTION    (Cheetah,- .4, 1)             * following the trophic chain.
   Gnu.ACTION    (LGrass,    1, 1)
   LGrass.ACTION (Gnu,     - 1, 1)
TIMER delta:=0.01,FINTIM:=900,PRdelta:=.5,PLdelta:=5   * Declare control variables
METHOD ADAMS                                           * Select the integration method
```

purposes, we usually choose a simple print or a two-dimensional plot of "interesting" model variables.

C++ is more suitable if the calculation cost of the simulation is great. On the other hand, we use Java if the model is simple but requires a complicated output visualization, since we provide more output forms when generating Java code. The Java and the C++ code generated are different in some details: the Java code is multithreaded and has to simulate the pointers; the C++ code can take advantage of default parameters in constructors and functions, but some optimizations related to memory management in expressions involving vectors and matrices have to be done by the compiler. An example of the generated Java code is shown in Listing 3.

2.5 Step 5: Deciding Type and Position of the Graphical Outputs

Several output forms can be included in a single simulation problem. If we choose to generate Java applets, the main panel of the simulation will be embedded inside the HTML page of the course. We can assign up to nine graphical outputs to this panel (in a 3×3 grid), but more outputs can be added as separate windows.

Several graphical outputs can be chosen by the course designer, such as: animated two-dimensional plots, three-dimensional plots, iconic plots, graphics to show the equations graphically, maps of isosurfaces, graphics to show the nodes of the grid used to solve a partial differential equation, etc.

In our example, we selected two graphical outputs:

- An animated two-dimensional plot that will be used to show the populations of all the species in the ecosystem.
- An iconic plot which shows a number of icons representing each species in proportion to its population.

To complete this stage, we have to add to Listing 2 the instructions shown in Listing 4 (see next page).

The first parameter in each instruction defines the position where the output is going to be placed (at the **C**enter and at the **S**outh of the main panel). The second parameter (*Species.X*) means the *X* attribute of every object belonging to the class *Species*. In our case, this would be equivalent to specifying "*Lion.X, Cheetah.X, Gnu.X, LGrass.X*," but obviously more general.

2.6 Step 6: Including Multimedia Elements

If we decide to include multimedia elements in the simulation, in the next step we have to synchronize them with the simulation execution. The general procedure designed to include multimedia elements in a simulation is shown in Figure 2.

The conditions to change between a multimedia element and another element are OOCSMP logic expressions. When one of these conditions becomes true, the corresponding element is launched. In this way, it is easy understand what is happening in the simulation.

We will explain each step in the procedure shown in Figure2 with our example.

- In the first step we identify the appropriate multimedia elements, in our case, a text panel, explaining what happens in the model (there will be three explanations: before the invasion, when the predator breaks the equilibrium, and when the oscillatory equilibrium is reached), and an image showing the trophic chain at every moment (two different trophic chains: before and after the predator invasion).
- In the second step we identify the conditions to change the texts and the images. Three different intervals are needed:
  - From the beginning of the simulation till the invasion of the predator. This condition can be expressed in OOCSMP as: `START ( (TIME>=0)&&(TIME<50) )`
- From the invasion of the predator to the setting of stability. This can be set in OOCSMP as:

```
START ((TIME>=50)&&(Lion.XP<0)&&
   Cheetah.XP<0)&&(Gnu.XP<0))
```

(continued on page 112)

Listing 3. A scheme of the Java-generated code

```
package <NAME>
import java.awt.*;
...
//import objects from our Java library
import csmp.plot.PlotData;
...
public class frm_<NAME> extends (Frame|Applet) implements Runnable [  ,...]
// other interfaces, depending on the graphical outputs selected...
{
// Declare arrays of simulated pointers to the variables beeing integrated,
// plot and printed ...
...
// Declare the model Data
...
// Declare the graphical objects
...
public void run() // launchs a thread for the calculus (not done in C++)
{  ... }
public void stop()// Stops the thread (not done in C++)
{  ... }
void <NAME>_s2()  // Calculus to be done in the simulation loop, the same in C++
{  ... }
void initAllArrays()
{  ... } // initializes the arrays of  simulated pointers...(not necessary in C++)
public void frm_<NAME>()
// constructor,adds graphical objects and initializes data, (not necessary in C++)
{  ... }
public void <NAME>_sim( ... ) // The simulation Loop, is almost the same for C++
{  // Initialize the selected graphical representations
...
for (;;){
        <NAME>_s2();
        // Print the selected variables
        ...
        // Plot some variables (varies depending on the chosen graphical output)
        ...
        // Perform integration, depending on the selected integration method...
        ...
}
}
public boolean handleEvent(Event e) // Handles user actions
{...}
private void updateArrays()
// Updates the array of pointers to the variables beeing integrated
{...}
private void updatePlots() // Updates the array of variables to be plotted
{...}
private void updatePrints() // Updates the array of variables to be printed
{...}
// Some other functions depending on the graphical outputs selected
}
```

Listing 4. Adding the graphical outputs to the file *africa1.csm*

```
...
PLOT                    [C], Species.X, TIME
ICONICPLOT              [S], Species.X
```

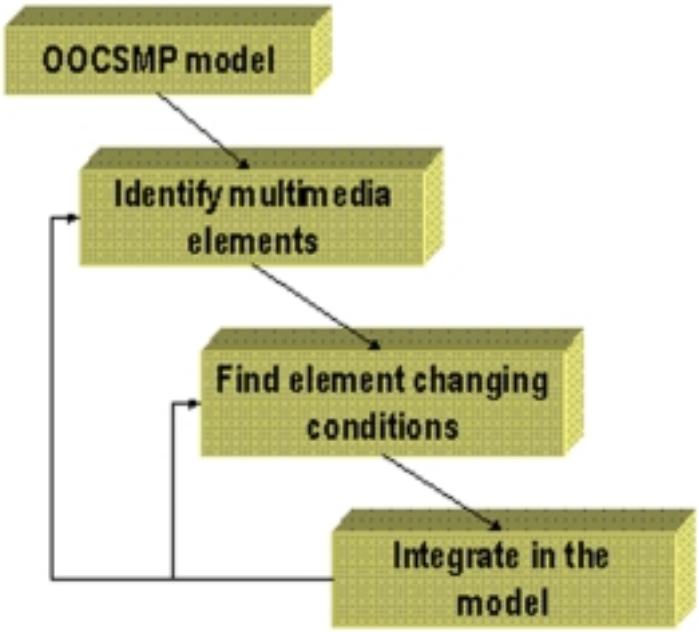

Figure 2. Procedure to integrate the multimedia elements with the simulation

Where *XP* is the derivative of the population of the corresponding species. At the beginning of this interval, we have to change the text and the image.

- After a state of oscillatory equilibrium is reached. This can be expressed by means of the *DEFAULT* clause, which launches the corresponding multimedia elements when no other expression is true. At the beginning of this interval, we have to change the text explanation.

The two multimedia elements will be located in the main panel, to the right of the previous graphics (*E* and *SE*). Listing 5 shows the instructions that must be added to Listing 2 to include the multimedia elements.

2.7 Step 7: Adding Instructions for the HTML Generation

In the earlier versions of the system [25], links, images, etc., had to be manually added to the generated HTML pages. This process can be now avoided, due to the fact that the simulation language has been extended with instructions to control the appearance of the HTML page where the applet will be placed. We have already described some for the automatic generation of documentation, which can also be used in this step. Thus there are two kinds of "HTML appearance" instructions: those for the documentation (with the asterisk), and those for the course page. In addition to those, there are other instructions to add descriptive text, or to include previously compiled models, which is useful when several simulations have to be placed in the same page. In these instructions, the compiler translates appropriately special symbols, such as accented vowels, and others.

Advantage can also be taken of the OOCSMP *INCLUDE* instruction, which provides reusing of HTML sections common to several pages, such as indexes to other course pages, footnotes, headings, etc.

Listing 6 shows the OOCSMP code necessary to obtain the

Listing 5. Adding the multimedia elements to the file *africa1.csm*

```
IMAGEPANEL [E],          START ((TIME>0)&&TIME<50)), "inicio.gif",
                         DEFAULT, "inv.gif"
TEXTPANEL [SE],          START ((TIME>0)&&TIME<50)), "inicio.txt",
                         START ((TIME>=50)&&(Lion.XP<0)&&(Cheetah.XP<0)&&(Gnu.XP<0)),
                         "inv.txt",
                         DEFAULT, "equilib.txt"
```

Listing 6. Instructions to generate the HTML page

```
* Third page of the course on ecology
* AUTHOR Juan de Lara
* EMAIL Juan.Lara@ii.uam.es
* DATE   21/12/99
TITLE The introduction of a predator breaks the equilibrium
DESCRIPTION Let's see how the introduction of a new predator (at TIME = 50)
DESCRIPTION affects our system in equilibrium.
IMAGE [C], "../images/africa1.JPG", "Example's trophic chain"
DESCRIPTION As you can see, the introduction of a new predator causes a
DESCRIPTION decrease on the population of the herbivores.
DESCRIPTION Because of this decrease, an increase on the population of the
DESCRIPTION plants takes place.\n
MODEL [670;630], [C], "africa1.csm", "/thesis/courses"
INCLUDE "ecoindex.csm"
INCLUDE "footnote.csm"
```

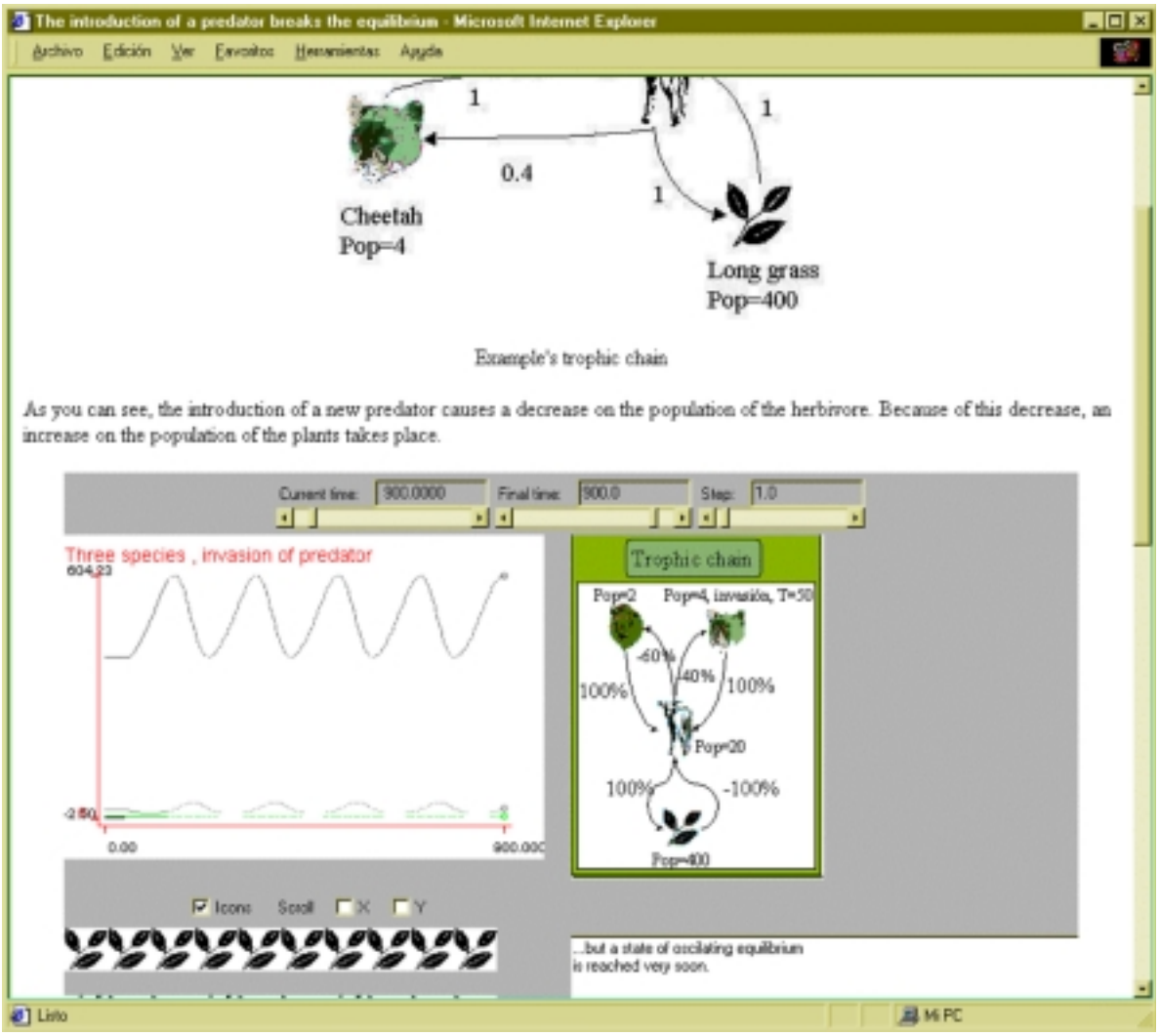

Figure 3. The generated course page

third page of the course. This page is shown in Figure 3.

The file `ecoindex.csm` contains an index, common to all the pages in the ecology course. The `footnote.csm` file contains a footnote common to all the courses we have generated.

This course, with the others we have generated with our language, can be found on the Internet at:

http://www.ii.uam.es/~jlara/investigacion

3. The Compiler

The compiler we are using to generate the courses is called C-OOL (a **C**ompiler for the **OO**CSMP **L**anguage), and its working scheme is shown in Figure 4.

The compiler can generate code for three different object languages: Java (applets or programs), plain C++, or C++ that uses the Amulet [32] library. In every case, it is possible to generate documentation for the models, in the form of HTML pages.

Other systems, such as AME [33] or OOPM [10] also generate C or C++ code from the simulation models; but in general, there is a lack of systems that can integrate executable models with educational courses for the Web.

MGEN (**M**esh **GEN**erator) is a graphical tool programmed in Java that can be used to generate OOCSMP code for the domains, meshes and conditions that will be used to solve a system of partial differential equations. The user can also make use of the standard components in the OOCSMP library, which includes electronic components, mechanical components, and components to solve typical partial differential equations.

A user interface is generated automatically, and can be configured by means of compiler options. This is useful if we want to restrict the possibilities of interaction of the user with the simulation. The interface allows the student to experiment with the problem, and to answer "*what if...?*" questions, in a Visual Interactive Simulation paradigm [27]. The interface is much more complete, and the OOCSMP programmer has more output forms available when generating Java code.

4. From Standalone Applications to Distributed Applications

Traditionally, the term parallel simulation has been applied to discrete event simulation (PDES) [34]. When it was applied to

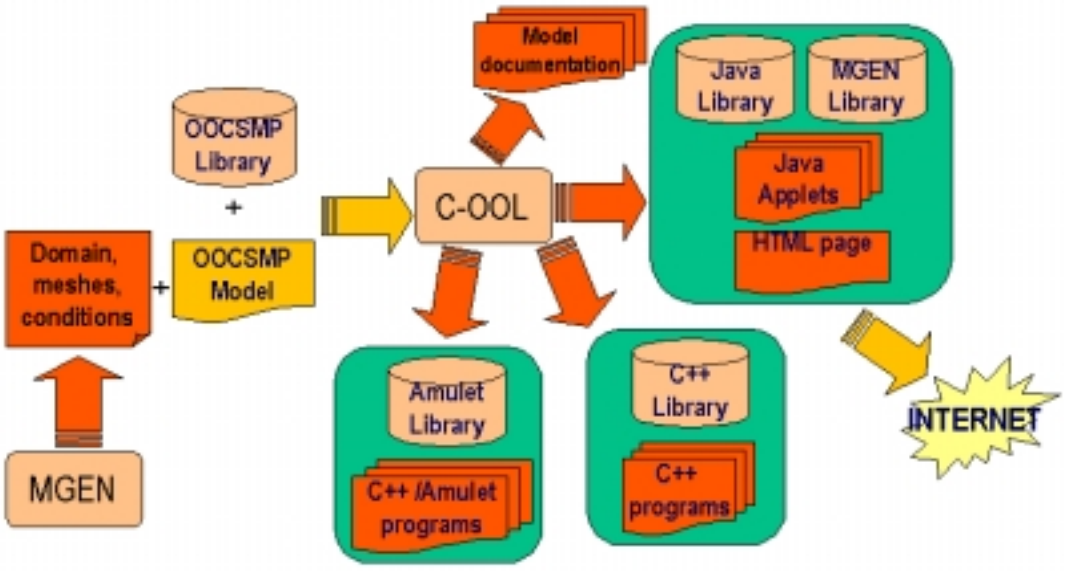

Figure 4. Our C-OOL compiler.

continuous simulation, the parallelization efforts have been centered on the algorithms for the solution of systems of equations, as well as matrix operations [35]. In our environment (the Web), this approach is not useful, because it usually implies a fine grain parallelism (for example, in the resolution of elliptic problems with iterative methods, each processor is associated with one or a few matrix elements). The Web environment can have a very high latency; therefore, we have to increase the granularity, to minimize communications and maximize computation. On the other hand, the Internet offers us some advantages, such as:

- It is an existing infrastructure.
- It has common and well established communication protocols and mechanisms.
- It is highly scalable.

These advantages are converting the Internet into an interesting framework to solve more interesting and complex problems [36, 37].

Distribution has been incorporated into OOCSMP using the *rmi* Java packages [9] at object level. It is possible to specify in which machine each object must be created, or if the object must be replicated in every computer. Each machine taking part in the simulation has a label, used to specify it independently of its actual address. In this way, it is easy to change the distribution scheme by changing the machines assigned to each label. A simple OOCSMP model is compiled for every machine taking part in the simulation, with the appropriate compiler options.

Different output visualization forms can be used in each machine, which makes it possible to visualize (and change) local or remote objects. The distribution scheme allows us to use different integration methods in each machine.

Synchronization points (semaphores) are added automatically by the compiler, assuring the serialization of the distributed code. Another synchronization point is added at the end of each simulation step. This guarantees the same simulation time in all the machines. There is also the possibility of not generating semaphores.

This scheme of distribution is appropriate when we can identify clusters of objects that interact only between themselves. In the ecological example, distribution would not be a good solution, because all the objects interact. Suppose, however, that we want to simulate several separate ecosystems at the same time. The species in these ecosystems do not interact, but migration is possible. The proper solution to this problem is distribution, as we can assign each ecosystem to a different machine.

Two types of migration have been implemented: seasonal and due to overpopulation. The migration target is selected between all the ecosystems in the simulation (the target ecosystem can be in another machine); the migrating species will choose the ecosystem with fewer individuals of the same species, to minimize competition.

The information associated with a generic species (food preference coefficients, type of migration it performs...), was encapsulated on a class called `MSpecies`. We have developed another class (`Population`) to represent the behavior of a particular species on a particular ecosystem (containing the initial and actual number of individuals, a reference to the generic species it belongs to,...). Each ecosystem has been modeled as a collection of species. Except for migration, each population interacts only with other species in the same ecosystem.

Using these classes, we have simulated a set of three ecosystems. One of them is in equilibrium, and has three species (Lions, Gnus and Grass). The others are not in equilibrium and have four species (Lions, Gnus, Zebras and Grass). We can observe an invasion of the Zebras into the first ecosystem, breaking the equilibrium.

All the species in the same ecosystem are created in the same machine, and the classes with information about a generic species have been replicated in all the machines. The distributed model is not much faster than the single processor model, due to the fact that network time dominates computation time in this particular example, but as we increase the number of species in each ecosystem and the number of ecosystems, the performance of the parallel version increases.

Listing 7 shows the *OOCSMP* code for this simulation.

The file *machines.csm* contains the *ip* address of machines with labels *m1*, *m2* and *m3*. For testing purposes, different labels can point to equal addresses.

The species in each ecosystem invoke method *migrate*, which selects the appropriate migration scheme for the current species. If the migration condition holds, migration takes place. This is taken as a discrete event (handled by the *FCNSW* and *INSW* instructions), and the integrators are reset, as if we had an "initial" condition, because an abrupt modification in variable *XP* takes place, and this variable is being integrated. *FCNSW* and

(continued on page 118)

Listing 7. OOCSMP code for the distributed ecosystems simulation

```
INCLUDE "machines.csm"
* Type of migration constants
DATA EXCESS:=0, SEASON:=1

CLASS MSpecies                          * Keeps generic information about a species
{
 NAME name                              * Name of the Species
 ICON spIcon                            * Icon associated for the output form
 DATA CANMIGRATE, TYPE                  * Indicate if it can miggrate, and the migration type
 DATA Percent[ 4] , Last[ 4]            * Vectors that indicate the preferences(trophic chains)
 DATA M, N1, N2:=0                      * Coefficients of the Volterra equations
}

CLASS Population                        * Simulates the behaviour of a population of species
{
   NAME popName                         * Name of the population
   DATA X0, orden                       * Initial population, order in the trophic chain
   MSpecie Sp                           * Pointer to the class that it belongs to

   DYNAMIC                              * Beginning of the main simulation loop for this class

    XT:=INTGRL(X0,XP)                   * Calculate actual population
    XP:=X*Sp.M                          * Calculate population increment, with M coefficient.
    X :=INSW(XT,0,XT)                   * Prevent a population to be negative.
    XPdel1 :=0                          * Initializes endogen variables, they will be used...
    XPdel2 :=0                          * ...in the ACTION method, that implements species...
    TEats  :=0                          * ... interaction.
    TEats0 :=0


   DECX num                             * DECX: method that is called in migrations
    X-=num                              * Reduce the population in "num"
    XP-=num*Sp.M                        * Calculate the new increment to apply.

   MIGRATE1 Population Ecos[]                       * Migration due to overpopulation
    EXCESO := Sp.CANMIGRATE*(2*X0-X)                * Calculate the excess of individuals
    EMIGRAN:= INSW(EXCESO,EXCESO,0)                 * Flag to indicate migration
    POSMIN := POSITION ( MIN ( Ecos.X ), Ecos.X )   * Position of the ecosystem with less
                                                    * individuals of the same type.
    X+=EMIGRAN                                      * Reduce population (EMIGRAN <= 0)
    XP+=EMIGRAN*Sp.M                                * Reduce population increment
    INSW (EMIGRAN,Ecos[ POSMIN] .DECX(EMIGRAN),)    * Increment the other population.

   MIGRATE2 Population Ecosys[]                     * Seasonal Migration
    PMIN := POSITION ( MIN ( Ecosys.X ), Ecosys.X ) * Position of the ecosystem with
                                                    * less individuals of the same type
    DSEASON := TIME%50                              * Calculate if it is migration time
    FCNSW (DSEASON,,INSW(Sp.CANMIGRATE,,Ecosys[ PMIN] .DECX(-X)),)   * If so, discrete...
                                                    * event, if the species can migrate
```

(Listing 7 continued on next page)

Listing 7. OOCSMP code for the distributed ecosystems simulation (continued from previous page)

```
                                                    * increase target population.
    X-=X*FCNSW(DSEASON,0,Sp.CANMIGRATE,0)           * Reduce population to zero.
    XP-=XP*FCNSW(DSEASON,0,Sp.CANMIGRATE,0)         * Update increment

   MIGRATE Population Others[], Population Places[]          * Migration
    FCNSW ( Sp.TYPE, , MIGRATE1(Others), MIGRATE2(Places) )  * Invoke the appropriate
                                                             * method, depending on the
                                                             * migration type.

ACTION Population S                                      * Interaction between species
 XPdel1 +=FCNSW(Sp.Percent[ S.orden]*S.TEats,            * See [23] for details.
 Sp.Percent[ S.orden]*S.X*S.TEats0/S.TEats, 0, 0)
 XPdel2 +=FCNSW(Sp.Percent[ S.orden]*S.X0, 0, 0, Sp.Percent[ S.orden]*S.X*S.X/S.X0)
 TEats  +=FCNSW(Sp.Percent[ S.orden], 0, 0, 1)*S.X
 TEats0 +=FCNSW(Sp.Percent[ S.orden], 0, 0, 1)*S.X0
 XP +=FCNSW(Sp.Percent[ S.orden]*X0, Sp.Last[ S.orden]*Sp.N2*X*XPdel1*X/X0, 0,
Sp.Last[ S.orden]*Sp.N1*X*XPdel2*TEats0/TEats)
}

* Implement the preferences of each species (trophic chain)

DATA PercLi[ 4], PercLi[ ] := 0 0.7 0.3 0       * Lions eat gnus(70%) and Zebras(30%)
DATA LastLi[ 4], LastLi[ ] := 0   0   1 0       * Flags vector, with the last position

DATA PercGn[ 4], PercGn[ ] := -1 0 0 1          * Gnus eaten by Lions(100%), eat Grass(100%)
DATA LastGn[ 4], LastGn[ ] :=  1 0 0 1          * Flags vector, with the last position

DATA PercZb[ 4], PercZb[ ] := -1 0 0 1          * Zebras eaten by Lions(100%),eat Grass(100%)
DATA LastZb[ 4], LastZb[ ] :=  1 0 0 1          * Flags vector, with the last position

DATA PercLG[ 4], PercLG[ ] :=  0 -0.5 -0.5 0    * Grass eaten by Gnus(50%)and Zebras (50%)
DATA LastLG[ 4], LastLG[ ] :=  0     0   1 0    * Flags vector, with the last position

* Declare the species of the ecosystems

MSpecies Lion("Lion","lion002.gif",1,EXCESS, PercLi, LastLi,-.0195, .0013982857142857)
MSpecies Gnu ("Gnu",  "bovin008.gif", 1,SEASON, PercGn, LastGn, -.02, .0002, .03 )
MSpecies Zebra("Zebra","zebra003.gif", 1,EXCESS, PercZb, LastZb, -.01, .0000625, .0075)
MSpecies LGrass("LGrass","leafs015.gif", 0,EXCESS, PercLG, LastLG, .01, .0      , 0.001)

* Declare each population of each ecosystem.
* The first ecosystem will be placed in machine m1, The population of Zebras has 0
* individuals initially

Population Li1 ( "Lion1",  2,  0, Lion)       MACHINE m1
Population Gn1 ( "Gnu1",   20, 1, Gnu )       MACHINE m1
Population Zb1 ( "Zebra1", 0,  2, Zebra )     MACHINE m1
Population LG1 ( "LGrass1",400,3, LGrass )    MACHINE m1

* The Second ecosystem will be placed in machine m2.

Population Li2 ( "Lion2",  2,  0,  Lion)      MACHINE m2
Population Gn2 ( "Gnu2",   22, 1, Gnu )       MACHINE m2
Population Zb2 ( "Zebra2", 20, 2, Zebra )     MACHINE m2
Population LG2 ( "LGrass2",300,3, LGrass )    MACHINE m2
```

(Listing 7 continued on next page)

Listing 7. Continued from previous page

```
* The Second ecosystem will be placed in machine m3.

Population Li3 ( "Lion3",  5,  0,  Lion)       MACHINE m3
Population Gn3 ( "Gnu3",   3,  1, Gnu )        MACHINE m3
Population Zb3 ( "Zebra3", 34, 2, Zebra )      MACHINE m3
Population LG3 ( "LGrass3",720,3, LGrass )     MACHINE m3

Population Ecosystem1 := Li1, Gn1, Zb1, LG1          * Declare the collection of objects
Population Ecosystem2 := Li2, Gn2, Zb2, LG2          * of each ecosystem.
Population Ecosystem3 := Li3, Gn3, Zb3, LG3

Population Lions := Li1, Li2, Li3                    * 3 aditional collections with all
Population Gnus  := Gn1, Gn2, Gn3                    * the species of the same type.
Population Zebras:= Zb1, Zb2, Zb3

DYNAMIC                                 * Main simulation loop
  NOSORT                                * Don't allow the compiler to reorder equations
  Lions.MIGRATE(Lions,Lions)           * Perform migration of Lions...
  Gnus.MIGRATE(Gnus,Gnus)               * Perform migration of Gnus...
  Zebras.MIGRATE(Zebras,Zebras)         * Perform migration of Zebras
  Ecosystem1.STEP()                     * Main section of Ecosystem1's populations
  Ecosystem1.ACTION(Ecosystem1)         * Interaction of Ecosystem1's populations
  Ecosystem2.STEP()                     * Main section of Ecosystem2's populations
  Ecosystem2.ACTION(Ecosystem2)         * Interaction of Ecosystem2's populations
  Ecosystem3.STEP()                     * Main section of Ecosystem3's populations
  Ecosystem3.ACTION(Ecosystem3)         * Main section of Ecosystem3's populations

* Choose output forms, the population of each species will be plotted in the machine
* where the objects have been placed.

PLOT  [C], [MACHINE=m1], Ecosystem1.X, TIME    * Ecosystem1 is plotted in m1
PLOT  [S], [MACHINE=m2], Ecosystem2.X, TIME    * Ecosystem2 is plotted in m2
PLOT  [N], [MACHINE=m3], Ecosystem3.X, TIME    * Ecosystem3 is plotted in m3

* Choose the control variables and the integration method

TIMER delta:=0.01,FINTIM:=320,PRdelta:=.5,PLdelta:=5
METHOD ADAMS
```

`INSW` can be used as mathematical blocks (returning a value), or as "event handlers" (when they appear as an instruction), similar to the constructions `IF...THEN` and `IF...THEN...ELSE` of traditional programming languages. In OOCSMP, "abrupt changes" of variables being integrated are treated in the same way when those variables appear in discontinuous blocks, such as `STEP`, `PULSE`, `IMPULS`, etc.

Basically, methods `migrat1` (migration due to overpopulation) and `migrat2` (seasonal migration) test for the migration condition to take place. If so (discrete event), the population of the actual species is decreased (in the first case, by the excess of individuals; in the second case, all the individuals migrate) and the number of individuals in the species of the target ecosystem is increased. Methods `STEP` and `ACTION` are very similar to those in the previous Species class described above.

Objects of the `MSpecies` class are replicated (the compiler replicates the objects in all the machines if we don't specify a machine label). `Population` objects belonging to the same ecosystem have been assigned to the same collection of objects, and to the same machine.

Finally, a two-dimensional plot of the local objects will show the graphical output.

To execute this model, it has to be compiled three times, once for each machine (`m1`, `m2` and `m3`). The Java classes `MSpecies` and `Population` generated by the compiler remain the same for each machine, but the main file changes.

5. Conclusions and Future Work

We have presented several procedures and tools that simplify the generation of Web courses based on simulation. These tools automatically document the simulation models. The `OOCSMP` instructions to design the `HTML` pages of the course simplify the task of constructing the course. Portions of the page can be reused in different pages, or even in different courses.

Synchronization of the simulation execution with the presentation of multimedia elements can also be accomplished. The multimedia elements give the student a better understanding of what is happening at every moment. The information is given at the precise instant and in a more rich way than in static `HTML` explanations.

We have also detailed some techniques to generate distributed simulations. Our simulation scheme has the advantage of permitting the passing of models from single machine models to distributed models in a natural way, with minimum changes in the model. The simulation designer does not have to worry about the low-level implementation of distribution by the system, or about the synchronization points, as happens when programming with libraries in general purpose languages.

With our distribution scheme, a single model is needed. The distribution scheme can be easily changed. Each machine can use a different integration method. Manipulation and change of remote objects can also be achieved.

As future work, we have several working lines:

- With respect to the language:
  - A possible improvement of the instructions to generate HTML code would be the possibility of defining `frames`. Frames would be useful to present course indexes, headings and footnotes.
  - More multimedia elements can be added to the language, such as animations, virtual reality, etc.
  - The discrete event handling of our language can be improved by creating an event queue, event types, event handlers, etc. These extensions are also directed to enable agent-oriented simulation [38, 39, 40] in OOCSMP.
- With respect to the distribution scheme: we pretend to migrate from *rmi* to *Corba* [9] as the supporting package for distribution. In this way, we would be able to mix Java and C++ objects generated by `C-OOL` in the same problem.
- With respect to the compiler: we are considering the possibility of generating `Modelica` [41] code.
- With respect to the environment: we are also thinking of building a graphical environment to construct the courses. This tool would cover all the stages of the procedures presented in this paper and would turn OOCSMP and C-OOL into an authoring tool for simulation courses. Ideally the tool would also provide facilities for collaborative programming across the Internet, covering in this way another aspect of the term "Web-based simulation" [10, 28].

6. References

[1] Thomson Publishing. *Internet Distance Education with Visual C++*. *http://www.thomson.com/microsoft/visual-c/teacher.h*, 1997.

[2] The Globewide Network Academy. *http://gnacademy.org*, 1997.

[3] Page, E.H., Buss, A., Fishwick, P.A., Healy, K., Nance, R.E. and Paul, R.J. "Web-Based Simulation: Revolution or Evolution?" to appear in *ACM Transactions on Modeling and Computer Simulation*, 1999.

[4] Aviation Industry CBT Committee on Computer Managed Instruction. *Computer Managed Instruction Guidelines and Recommendations*, AGR 006, Version 1.1, AICC. *http://www.aicc.org/agr006.htm*, 1997.

[5] Schutte. "Virtual Teaching in Higher Education: The New Intellectual Superhighway or Just Another Traffic Jam?" *http://www.csum.edu/sociology/virexp.htm*, 1997.

[6] Directorate-General XIII. *Educational Multimedia: First Elements of Reflection*. Task Force on Multimedia Educational Software, 1996.

[7] Fishwick, P.A. "Web-based Simulation: Some Personal Observations." *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, pp 772-779, 1996.

[8] Schmid, C. "A Remote Laboratory Using Virtual Reality on the Web." Special issue of *SIMULATION*, Web-Based Simulation, Vol. 73, No. 1, July 1999, pp 13-21, 1999.

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

[10] Cubert, R.M., Fishwick, P.A. "OOPM: An Object-Oriented Multimodeling and Simulation Application Framework." *SIMULATION*, Vol. 70, No. 6, pp 379-395, June 1998.

[11] Healy, K.J., Kilgore, R.A. "Silk: A Java-Based Process Simulation Language." *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, pp 475-482, 1997.

[12] Page, E.H. and Moose, Jr., R.L., Griffin, S.P. "Web-based Simulation in SimJava using Remote Method Invocation." In *Pro-*

ceedings of the 1997 Winter Simulation Conference, Atlanta, pp.468-474, 1997.

[13] Howell, F. and McNab, R. "A Discrete Event Simulation Library for Java." *Proceedings of the First International Conference on Web-based Modeling and Simulation*, P. Fishwick, D. Hill and R. Smith (Eds), SCS, San Diego, 1998.

[14] Page, E.H. and Griffin, S.P. "Transparent Distributed Web-Based Simulation using Simjava." *Proceedings of the First International Conference on Web-based Modeling and Simulation*, P. Fishwick, D. Hill and R. Smith (Eds), SCS, San Diego, 1998.

[15] Perumalla, K.S. and Fujimoto, R.M., "Interactive Parallel Simulations with the Jane Framework." To appear in *Special Issue of Future Generation Computer Systems*, Elsevier Science, 2000.

[16] Fujimoto, R.M. *Parallel and Distributed Simulation Systems*, Wiley Interscience, 1999.

[17] Page, E.H., Nicol, D.M., Balci, O., Fujimoto, R.M., Fishwick, P.A., L'Ecuyer, P. and Smith, R. "Panel: Strategic Directions in Simulation Research." *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[18] Defense Modeling and Simulation Office, HLA Home page, *http://hla.dmso.mil/*.

[19] Maly, K., Overstreet, C.M., González, A., Denbar, M., Cutaran, R., Karunaratne, N., Srinivas, C.J. "Use of Web Technology for Interactive Remote Instruction." *Proceedings of the Web'97 Conference*. On Internet at *http://www7.scu.edu.au/programme/posters/1855/com1855.htm*, 1998.

[20] IBM72. **Dr. Alfonseca: please provide this reference citation**.

[21] Alfonseca, M., Pulido, E., Orosco, R., de Lara, J. "OOCSMP: An Object-Oriented Simulation Language." *ESS'97*, Passau, pp.44-48, 1997.

[22] Alfonseca, M., de Lara, J. and Pulido, E. "Semiautomatic Generation of Educational Courses in the Internet by Means of an Object-Oriented Continous Simulation Language." In *Proceedings of ESM'98*, SCS, pp 547-551, 1998.

[23] Alfonseca, M., de Lara, J. and Pulido, E. "Educational Simulation of Complex Ecosystems in the World-Wide Web." *Proceedings of ESS'98*, SCS, pp. 248-252, 1998.

[24] de Lara, J., Alfonseca, M. "Simulating Partial Differential equations in the World-Wide Web." *Proceedings of EUROMEDIA '99*, pp-45-52, Munich, 1999.

[25] Alfonseca, M., de Lara, J., Pulido, E. "Semiautomatic Generation of Web Courses by Means of an Object-Oriented Simulation Language." Special issue of *SIMULATION*, Web-Based Simulation, Vol 73, No. 1, pp. 5-12, July 1999.

[26] Volterra, V. *Leçons sur la Thèorie Mathèmatique de la Lutte pour la Vie*. Gauthier-Villards, Paris, 1931.

[27] Campos, A.M.C., Hill, D.R.C. "An Agent-Based Framework for Visual-Interactive Ecosystem Simulations." *TRANSACTIONS of the SCS*, Vol. 15, No. 4, pp 139-152, 1998.

[28] Fishwick, P.A. "A Multimodeling Basis for Across-Trophic-Level Ecosystem Modeling: The Florida Everglades Example." *TRANSACTIONS of the SCS*, Vol. 15. No. 2, pp 76-89, 1998.

[29] Csuhaj-Varjú, E., Kelemen, J., Kelemová, A., Paun, G. "Eco(grammars) Systems–A Grammatical Framework for Life-like Interactions." *Artificial Life*, Vol. 24, pp 1-28, 1997.

[30] Alfonseca, M., Ortega, A. "Representation of Some Cellular Automata by Means of Equivalent L Systems." Submitted to *Complexity International*, 2000.

[31] Rodríguez de la Fuente, F., et al. *Enciclopedia Salvat de la Fauna*, Salvat, 1970.

[32] Myers, B., et al. *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, 1997.

[33] AME Home page, *http://helios.bto.ed.ac.uk/ierm/ame/index.html*.

[34] Zeigler, B., and Ahang, G. "Mapping Hierarchical Discrete Event Models to Multiprocessor Systems: Concepts, Algorithm, and Simulation." *Journal of Parallel and Distributed Computing*, Vol. 9, pp 271-281, 1990.

[35] Kascic, M.J. "Vector Processing on The Cyber 200." Infotech State of the Art Report, "Supercomputers", Infotech International Ltd, Maidenhead, U.K., pp. 1-38, 1979.

[36] Interim Java Grande Forum Report. "Java Grande Forum." Technical Report JGF-TR-4, *http://www.javagrande.org/report.htm*.

[37] Serbedzija, N.B. "The Web Supercomputing Environment." *Proceedings of the WWW'97, http://www7.scu.edu.au/programme/posters/1838/com1838.htm*, 1997.

[38] Wooldridge, M., Müller, J.P., Tambe, M. *Intelligent Agents II. Agent Theories, Architectures and Languages*, Springer, 1995.

[39] Jennings, N.R., Sycara, K., Wooldridge, M. "A Roadmap of Agent Research and Development." *Autonomous Agents and Multi-Agent Systems*, Vol. 1, pp 7-38, Kluwer Academic Publishers, 1998.

[40] Swarm Development Group Home page: *http://www.swarm.org*.

[41] Elmqvist, H., Mattson, S.E. "An Introduction to the Physical Modeling Language Modelica." *Proceedings of the 9th European Simulation Symposium, ESS'97*, SCS, pp 110-114. See also The Modelica Home page, *http://www.Modelica.org*, 1997.

Manuel Alfonseca is a Doctor in Electronics Engineering and Computer Science, with both degrees from the Universidad Politecnica of Madrid. He teaches and does research at the Department of Computer Science of the Universidad Autonoma of Madrid, where he is the Subdirector of Research. Previously he worked at the IBM Madrid Scientific Center, where he reached the level of Senior Technical Staff Member. He is a Member of SCS, the New York Academy of Sciences, IEEE, ACM, the British APL Association, and the Spanish Association of Scientific Journalism. He has published more than 150 technical papers and several books on computer language translation, simulation, complex systems, graphics, databases, artificial intelligence, object-oriented technology, and theoretical computer science. He also writes science for the layman (six books and 70 papers in a major Spanish daily journal) and children's literature (19 published books), and has received the 1988 Lazarillo Award, sponsored by the Ministry of Culture and the Spanish branch of the IBBY.

Juan de Lara is a Lecturer at the Universidad Autonoma of Madrid. He obtained his PhD in June 2000, with a doctoral thesis on Web-based simulation. In 1996 he became a Higher Engineer in Computer Science. He is also a Technical Engineer in Computer Science, graduating in 1994 with a Top of the Class Award. He worked for Cap-Gemini Spain from 1996 through 1997.