Repositorio Institucional de la Universidad Autónoma de Madrid

https://repositorio.uam.es

Esta es la **versión de autor** del artículo publicado en:
This is an **author produced version** of a paper published in:

Simulation 73.1 (1999): 5 – 12

DOI: http://dx.doi.org/10.1177/003754979907300102

Copyright: © 1999 The Society for Modeling and Simulation International

El acceso a la versión del editor puede requerir la suscripción del recurso
Access to the published version may require subscription

Semiautomatic generation of web courses by means of an Object Oriented Simulation Language

Manuel Alfonseca, Juan de Lara, Estrella Pulido
Universidad Autonoma de Madrid, Dept. Ingenieria Informatica
{Manuel.Alfonseca, Juan.Lara, Estrella.Pulido}@ii.uam.es

Abstract

This paper describes the procedure we have used to semiautomatically generate three different courses for the web. The simulations used in these courses have been written in our special-purpose object-oriented continuous simulation language (OOCSMP). A compiler we have written for this language automatically generates Java code and html pages, which must be completed manually with the text and images associated to each.

Keywords

Simulation in the Web, Education, Object Oriented Languages, Java code generation, Continuous Simulation.

Introduction

The World-Wide-Web (WWW) is becoming a very important tool in the educational field. Every day there are more and more courses based on its use [1-2], which go from a simple transposition of classroom notes, to the inclusion of more sophisticated elements, such as simulations, animated graphics, and so forth. In particular, the Java language has made the courses more interactive, faster to execute and easily transportable to multiple platforms.

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 [3-4]. In the words of the "Multimedia Educational Software Task Force", dependent on the European Commission Directorate-General XIII [5]: "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)".

Educational multimedia systems [6] should be based on an interaction between the knowledge to be learned and the student who will learn it. The main objective should not be a mere acquisition of information: the student should attain the ability required to effectively execute the procedures needed for problem resolution in the field of his studies.

Simulation can interact with the web in several interesting ways [7]:

- Building and executing distributed models.
- Providing web-based tools, such as simulation interpreters.
- Integrating previously generated simulation models in educational courses (this is our approach).

Models available in the web may be written in Java, or in any other language accessible through CGI. There are two main ways of reducing the effort needed to build the models:

- Providing a library that contains pre-defined classes which 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 C++ [8-9] or Java [10-13].
- 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, and our language is called OOCSMP.

This paper describes the OOCSMP language and the COOL compiler which translates OOCSMP code into Java and C++. It also presents the procedure to follow in order to integrate complex continuous simulations in interactive WWW pages by means of the Java programs automatically generated by COOL. Next, the problems to be solved to generate code are described. Finally, the procedure has been applied to the construction of three different courses on the web: a course on Newton's gravitation law, a practical course on Ecology and an introductory course on Electronic circuits.

Semiautomatic generation of educational courses on the WWW

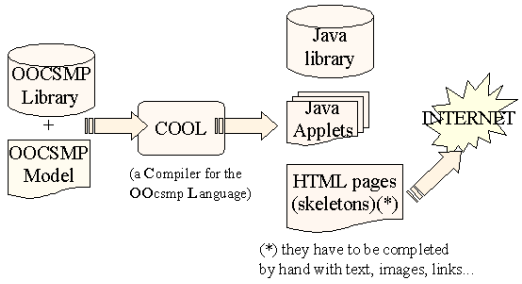

Figure 1: Procedure for course generation

The procedure proposed to generate courses for the web consists of the following steps (see figure 1):

1. Designing on paper an interactive course based on continuous simulation models. Depending on the course, a single model may be used in one or more (sometimes all) pages.
2. Building the models in our own continuous simulation language, OOCSMP. This language is an object-oriented [14] extension of the old CSMP simulation language [15].
3. Designing the different simulation runs for each page in the course. The same basic model may be tested in different situations and provides interactive facilities that make it possible for the student to experiment.
4. Translating the models into C++ and testing them. The compiler provides a fast, easy-to-use stand alone environment that simplifies testing and allows the course-writer to experiment many different situations.
5. Translating the models and run situations into Java applets. The same compiler is used for both this and the previous step, as the object language is a parameter of the compiler.
6. Automatic generation of html skeletons for each page in the course. The Java applets embodying the models are automatically embedded in the html pages.
7. Manual addition of text, images, and internal/external references to the skeletons. This adjustment is needed to fill the html skeletons with explanations, images and cross references to other pages. The resulting set of pages is ready to be made available to the students through the Internet. Some of the courses we have generated will be discussed in the remainder of the paper.

We will explain all these steps in detail with examples in subsequent sections.

All these capabilities are incorporated into a single tool that we call COOL (a Compiler for the OOcsmp Language), a compiler that translates OOCSMP source code and, depending on the invocation options, generates Java and/or html skeletons, or C++ for DOS, WINDOWS 95 or Unix environments. An additional set of options makes it possible to select many different presentation possibilities in the Java applets.

The OOCSMP continuous simulation language

The OOCSMP language is a true extension of the old CSMP simulation language, in the sense that CSMP programs can still be compiled and executed by OOCSMP compilers. The extensions added to the language make it possible to build extremely compact object-oriented models when the system to be simulated consists of many similar interacting parts, as in the examples we present here. Additionally, the OOCSMP language also includes capabilities to solve partial differential equations and implements an algebra for matrices and vectors. All this adds greatly to the power of the language and extends its domain of application.

The object-oriented extensions added to OOCSMP can be found in [16]:

A few additional extensions increment the capabilities of the OOCSMP language:

- Solution of partial differential equations by the finite elements [17] and/or the finite differences methods [18].
- Several graphical outputs: two-dimensional and three-dimensional plots, two types of iconic representations, maps of isosurfaces, and a plot for the partial differential equations grids.
- The PARAMETER sentence makes it easy to modify the values of some constants during program invocation.
- The run separator sentence, consisting only of the symbol \, may be used to define different runs of the same model. All the instructions after this sentence, to the end of the program or the next \ sentence, will be considered as a different run. The model itself cannot change, but all the declarative instructions: TITLE, DATA, TIMER, object constructions, object arrays and the PRINT/ PLOT/ICONICPLOT/CONNECTIONPLOT directives may be modified. Every \ sentence restores the original state of the model. The changes specified afterwards modify that

state.

The generated Java code

COOL can translate OOCSMP source code into Java and C++. A class (C++ or Java) is generated for each OOCSMP class, and a main file is generated to handle the user interface, and to perform the simulation loop. The structure of this loop varies depending on the selected integration method.

The actions of the main simulation loop (OOCSMP `DYNAMIC` section outside classes) are accomplished in a function called `<NAME>_s2()`, both in the C++ and Java cases. This function is called several times in the simulation loop (depending on the integration method).

Apart from these similarities, the task of generating code [19] from the OOCSMP models is quite different depending on which code (C++ or Java) has to be generated.

The main difference is the graphical user interface generated. Some of the Java graphical objects are not possible in the C++ case. In Java, two different threads are launched : one for the graphics, another for the simulation loop.

Some of the graphical representations create their own thread too.

In the Java case, a file is generated for each OOCSMP class. In the C++ case, a header file is generated too. In this last case, the generation of the constructors is simplified due to the possibility of using parameters with default values. In the Java case, a constructor must be generated for every parameter with a default value.

Another difference is the absence of pointers in Java. The pointers are 'simulated' by means of simple variables, which are updated with the correct values at some points of the simulation execution.

If the option of generating a standalone Java program is selected, a file (called `<NAME>.java`) is generated to make possible starting the application as an Applet too. If this option is not present, only the main file (called `frm_<NAME>.java`) is generated (plus all the class files).

Listing 1 shows a brief scheme of a typical main Java file generated with COOL:

```
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
{ ... }
public void stop()// Stops the thread
{ ... }
void <NAME>_s2()  // implements the calculus to be done in the simulation loop
{ ... }
void initAllArrays()// initializes the arrays of  simulated pointers...
{ ... }
public void frm_<NAME>()
// constructor,adds graphical objects and initializes data
{ ... }
public void <NAME>_sim( ... ) // The simulation Loop
{ // Initialize the selected graphical representations
...
for (;;){
       <NAME>_s2();
       // Print the selected variables
```

```
        ...
        // Plot the selected variables
        ...
        // 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 1 : A scheme of Java generated code

A course on Newton's gravitation law

This course may be found at the following web address:

http://www.ii.uam.es/~epulido/newton/grav.htm

(Step 1) The course consist of six which show:

- A simple description of Newton's Mechanics with different solutions to the two-body problem (a free fall following different orbits: a circle, an ellipse, a parabola, a hyperbole and a straight line).
- A model of the solar system, as a practical example of the n-body problem. For convenience, the solar system is shown in two separate parts: the inner system (from Mercury to Jupiter) and the outer system (from Jupiter to Pluto), with different time scales.
- A model of the Sun-Earth-Moon system. The time and plot scales are adjusted to make the two orbits distinguishable.
- The discovery of Neptune by John Couch Adams and Urbane Jean-Joseph Le Verrier, indicating how this discovery transformed an apparent failure of Newton's Mechanics into an outstanding success. The discovery is illustrated by a double simulation of Uranus's orbit, in the presence and in the absence of Neptune.
- A geo-stationary satellite which keeps constant its distance to the Earth has been simulated using the same model, changing only the values of the constants and the instanced objects (members of the class Planet). The effect of the Moon on the satellite's orbit is illustrated by performing a double simulation in the presence and in the absence of the Moon. To test the second case, we only have to change the mass of the Moon to zero.
- Finally, the last page leaves the student freedom to experiment with the simulated solar system, providing the ability to change the planet parameters and the universal constants (the mass of the Sun or the gravitational constant), to play at answering what-if questions.

(Step 2) This course is based on a single continuous simulation model of the n-body problem, which is applied in the different pages to several situations which provide an overview of the possible uses of Newton's gravitation law. The model defines a single class (Planet) which may be used indistinctly to represent the planets in the solar system, natural or artificial satellites.

Listing 2 shows the OOCSMP definition of the Planet class, which is used in all the pages described above. A more detailed description of this model appears in reference [20]. Figure 2 shows one of the pages in the course.

```
*********************************
* Definition of Planet class    *
*********************************
CLASS Planet {
 NAME name
 DATA M, X0, Y0, XP0, YP0, FI
 INITIAL
 FIR:=FI*PI/180
 CFI:=COS(FIR)
 SFI:=SIN(FIR)
*********************************
* Calculations for a planet     *
*********************************
 DYNAMIC
* Distance to the Sun
  R2   := X*X+Y*Y
  R    := SQRT(R2)
  Y1   := Y*CFI
  Z    := Y*SFI
* Mutual influences
```

```
* The Sun on this planet
  APS := G*MS/R2/R
* This planet on the Sun
  ASP := G*M/R2/R
  XPP := -(ASP+APS)*X
  YPP := -(ASP+APS)*Y
  XP  := INTGRL(XP0,XPP)
  YP  := INTGRL(YP0,YPP)
  X   := INTGRL(X0,XP)
  Y   := INTGRL(Y0,YP)
*********************************
* Mutual actions of two planets *
*********************************
 ACTION Planet P
* Distance to another planet
  DPP2 := (P.X-X)*(P.X-X)+
   (P.Y-Y)*(P.Y-Y)+(P.Z-Z)*(P.Z-Z)
  DPP  := SQRT(DPP2)
* Influences
* The other planet on the Sun
  ASP1 := G*P.M/P.R2/P.R
* The other planet on this planet
  APP1 := G*P.M/DPP2/DPP
* Coordinate conversion
  Y2 := P.Y*COS(P.FIR-FIR)
* Actual action of the planet
  XPP += APP1*(P.X-X) - ASP1*P.X
  YPP += APP1*(Y2-Y) - ASP1*Y2
*********************************
* Other data                    *
*********************************
 PRINT R
 PLOT Y,X
 FINISH R=.0001
}
```

Listing 2: Declaration of the Planet class in the course on Newton's gravitation law

(Step 3) Although the same model is used in the whole course, slight modifications need to be done for each HTML page. As an example, for the two body system in the first page, the ACTION term is not required, as this is only required for the case of three or more bodies. For the remaining pages, the original model may be used as-is. The only modifications needed may be to inhibit some objects in the plot or to modify the plot scales.

The rest of the course generation process has to do with code and html skeleton generation and is done automatically by the system. As mentioned when describing the process, these skeletons need to be completed manually by the course designer with additional text, images and hyperlinks.

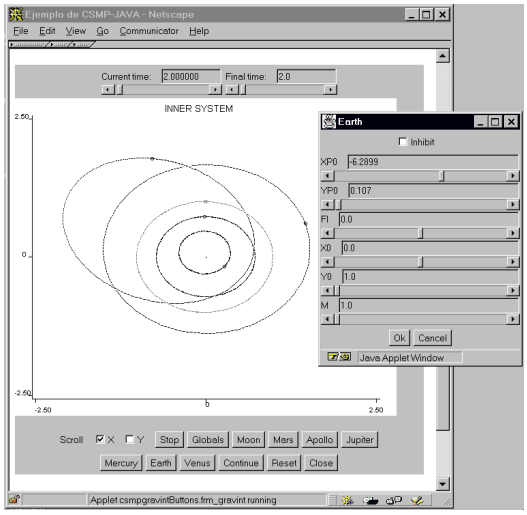

Figure 2: A course on Newton's gravitation law

A practical course on Ecology

The course can be found at the following address:

http://www.ii.uam.es/~epulido/ecology/simul.htm

The course consists of seven pages presenting the following ecosystems:

- An isolated three-species system (one primary producer, one prey and one predator), with the appropriate parameters to bring the ecosystem out of equilibrium. The student can observe the periodicity of this kind of systems.
- A three-species system in equilibrium.
- A three-species system, originally in equilibrium, which is invaded after some time, first by a new prey, then by a new predator. The first invasion takes the system out of equilibrium. After the second invasion, the whole five-species system reaches a new periodical stability.
- The same system, invaded after some time, first by a new predator, then by a new prey. The periodical stability attained after some time is completely different to the preceding case.
- A five species system in equilibrium.
- A five-species system with an user interface that lets the student modify the different parameters to perform experiments. Some of these experiments are suggested by the text of the page, but the student may perform many more.
- A simulation of an ecosystem with fifteen different species that interact to build complicated trophic chains and ecological niches.

As mentioned above, the main components of the html pages are the simulations of ecosystems presenting different features. The outputs of these simulations are presented graphically in two different ways:

- A plot of the evolution of the populations of the different species along time.
- An iconic representation of the same populations. The number of icons of a given species shown is proportional to its population at any moment. As populations wax and wane, icons appear or disappear.

One of the pages makes it possible to dynamically add objects (species) during the execution of the model.

Our model is based upon the Volterra equations [21], with a few modifications. We consider three different types of species:

- Primary producers, usually plants, which are preyed upon, but do not prey.
- Superpredators, which prey but are not preyed upon.
- Intermediate consumers, which prey and are preyed upon.

Each species can have up to two vectors of coefficients which represent its "appetence" for other species and vice versa. These coefficients are not constant, but a function of the population proportions. Listing 2 depicts the resulting model, written in OOCSMP. This model is described in more detail in reference [22], which also includes a version of the same program written in the Modelica [23] language.

We have used the model to simulate a part of the African savanna ecosystem. The data used for the parameters and coefficients are more or less real, as found in the literature [24]. Our model contains fifteen different species, four of which are carnivore, six herbivore, two omnivore, and three primary producers (plants). There are several trophic chains, the longest of which consists of five species: lion-jackal-rat-locust-grass.

Figure 3 shows on of the pages in the course on Ecology, which uses icons to represent the populations of the different species.

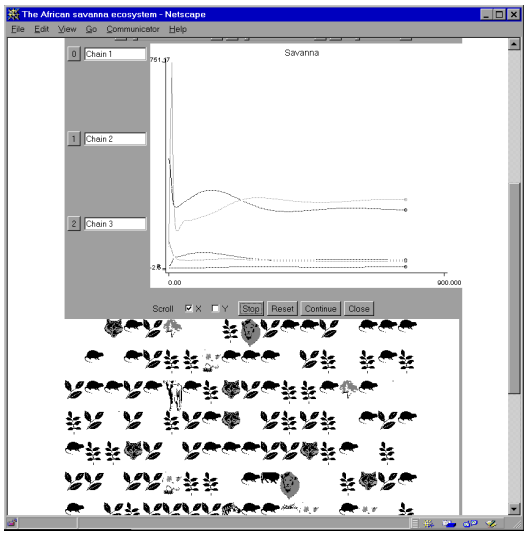

Figure 3: A practical course on Ecology

An introductory course on electronic circuits

This course may be found at the following web address:

http://www.ii.uam.es/~epulido/circ/modules.htm

The course shows how to build more complex systems incrementally by using simpler circuits.

The simpler circuits are encapsulated as OOCSMP classes, which are reused in the complex circuits. It would also be possible to connect the electronic circuits with other components, but this is not shown in the course.

The models use the OOCSMP directive *CONNECTIONPLOT* to produce a graphical representation appropriate for electronics circuits. Each block in the model appears as a graphical image with its inputs and outputs correctly connected, and special widgets for the global input/output of the model. The user can change the inputs to the circuit by clicking on the widgets, watch the outputs and display the values of the intermediate blocks by clicking on them. It is also possible to visualise the output as a seven segment led.

The course is divided into two main sections: combinatorial circuits and sequential circuits. The first includes adders (1 and 4 bits), multiplexers (2x1 and 4x1) and decoders (2x4, 4x16 tree-decoders, coincident-decoders and a decoder whose output is a seven segment led).

In the second section (sequential circuits) we show a D-type flip-flop.

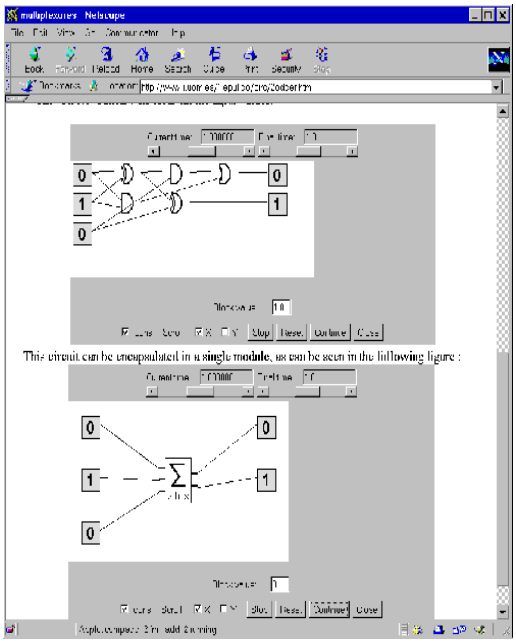

Figure 4: A 1-bit adder

The previous figure 4 shows the page of the course that describes the 1-bit adder.

This course is described with more detail in Spanish in reference [25].

Solution of partial differential equations

We have generated some pages with examples of the solution of PDEs [26] in one and two dimensions. There are also examples of the grid generation capabilities of OOCSMP. They can be found in the following address :

http://www.ii.uam.es/~jlara/oocsmp/pdes.html

As an example, figure 5 shows the solution of the diffusive transport equation in 1D.

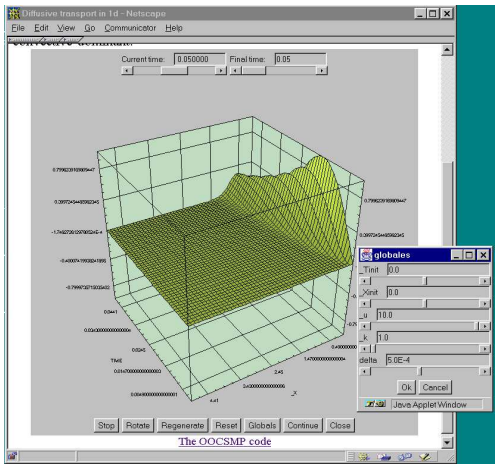

Fig 5: Solving the non-diffusive transport equation

Conclusion

The procedure we have used to build courses on the web based on simulation Java applets is very general, as proved by the three different courses we have designed up to now. The set of tools we have developed make it very easy to build the courses by combining pages that use the same models or quite different ones. The OOCSMP language is much simpler to use than Java in this area of application. Object orientation further simplify the models, which can be built in a hierarchical sequence, the simplest being used as new building blocks by the more complex.

Manual construction of the pages is reduced to a minimum: only descriptive texts, static images and cross-references have to be added by hand. The Java-generated models provide automatically a flexible user interface and a complete set of output capabilities, including two and three-dimensional plots, iconic plots and electronic diagrams. The course writer may select those most appropriate for the subject matter of the course, or combine several in any way.

In the future, we intend to extend the language and the compiler in such a way that it would be possible to include the text for the html pages in the model, in the form of comments. We also intend to include some code and access to a database, so as to be able to control the performance of each student with the courses we generate.

We are also beginning to work with distributed objects. Given an OOCSMP model, COOL will optionally generate stand-alone code, or distributed code. This will speed-up considerably some simulations.

Acknowledgment

This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project numbers TIC-96-0723-C02-01 and TEL97-0306.

References

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

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

[3] Aviation Industry CBT Committee Computer Managed Instruction. 1977. *Computer Managed Instruction Guidelines and Recommendations*, AGR 006, Version 1.1, AICC. http://www.aicc.org/agr006.htm.

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

[5] Directorate-General XIII. 1996. *Educational Multimedia: first elements of reflection. Task force on Multimedia Educational software.*

[6] Allen,J.T., Chance,R. 1998. *Maximizing the Learning of Information Systems via World Wide Web*. In Proceedings of Web Net 98. Vol. 2. pp.1245.

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

[8] Jones,J., Roberts,S. 1996. *Design of Object-Oriented Simulations in C++* . In Proceedings of the 1996 Winter Simulation Conference, Coronado.

[9] Schwetman.H. 1995. *Object-oriented Simulation Modeling with C++/CSIM17*. In Proceedings of the 1995 Winter Simulation Conference.

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

[11] Page,E.H., Moose Jr.,R.L., Griffin,S.P. 1997. *Web-based Simulation in Simjava using Remote Method Invocation.* In Proceedings of the 1997 Winter Simulation Conference, Atlanta, pp. 468-474.

[12] Howell,F., McNab,R. 1998. *A Discrete Event Simulation Library for Java*. In Fishwick P., Hill D., Smith R., Ed: Proceedings of the 1st International Conference on Web-based Modeling and Simulation, SCS San Diego.

[13] Page,E.H., Griffin,S.P. 1998. *Transparent Distributed Web-Based Simulation using Simjava*. In Fishwick P., Hill D., Smith R., Ed: Proceedings of the 1st International Conference on Web-based Modeling and Simulation, SCS, San Diego.

[14] Prapidapis,K.S, Colley,M., Chernett,P, *An object oriented automated framework for simulation of complex physical systems*. ESS'97. Passau, pp 31-35.

[15] 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.

[16] Alfonseca, M., Pulido, E., Orosco, R., de Lara, J. *OOCSMP : an object-oriented simulation language*. ESS'97, Passau, pp. 44-48.

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

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

[19] Acebes,L.F., de Prada, C.*SIMPD : An intelligent modelling tool for dynamic processes*. ESS97. Passau, pp. 177-181.

[20] Alfonseca, M., de Lara, J. And Pulido, E. November 1998. *An object-oriented continuous simulation language and its use for training purposes*. 5th International Workshop on Simulation for European Space Programs, Noordwijk, The Netherlands.

[21] Volterra, V. 1931. *Leçons sur la Théorie Mathématique de la Lutte pour la Vie*, Gauthier-Villars, Paris.

[22] Alfonseca, M., de Lara, J. and Pulido, E. October 1998. *Educational simulation of complex ecosystems in the World-Wide Web*, Proceedings ESS'98, Nottingham, pp. 248-252.

[23] Elmqvist, H. and Mattson, S.E. 1997. *An Introduction to the Physical Modeling Language Modelica*, Proceedings 9th European Simulation Sympossium ESS97, SCS Int., Erlangen, pp. 110-114. See also http://www.Dynasim.se/Modelica/index.html.

[24] Rodríguez de la Fuente, F. 1970. *Enciclopedia Salvat de la Fauna*, Salvat.

[25] Alfonseca, M., de Lara, J. and Pulido, E. 1998. *Generación semiautomática de cursos de Electrónica para Internet mediante un lenguaje*

de simulación continua orientado a objetos. III Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica (TAEE'98), Madrid, pp. 125-130.

[26] de Lara, J., Alfonseca, M. *Simulating Partial Differential Equations in the World-Wide-Web*. EUROMEDIA'99. Munich. In press.