Constructing Simulation-Based Web Documents

Juan de Lara and Manuel Alfonseca
Universidad Autónoma of Madrid

Our system for constructing Web documents features visual interactive simulations and other hypermedia elements. It uses a continuous simulation language composed of abstraction layers that describe the simulation models' behavior, pages or slides, and courses, articles, or presentations. The methods and tools presented stress key points in the development of Web-based applications such as maintainability, reusability integration, and easy testing.

The Web is changing the way we work. We see educational courses, articles, and presentations on the Web. These publications can sometimes be directly accessed online and other times must be downloaded. Online documents range from simple lecture notes to pages including more sophisticated elements such as animated graphics and simulations.

The Web environment as a hyper document container offers some advantages, including a common user interface (the browser), richer interaction possibilities, and platform independence.[1] More interactive possibilities are offered by plug-ins. For example, one of the most popular is the Java Virtual Machine, which makes it possible to run Java programs (or applets) embedded in hypertext markup language (HTML) pages (see http://java.sun.com).

Interactive simulation lets students experiment with a model and answer "what if ... ?" questions. It also lets users play a more active role in the learning process. Thus, current education is moving from a teacher-centered paradigm to a student-centered view.[2] Simulation applets in Web courses[3] can be an effective complement to lab experiments.[4] In particular, multimedia elements[5] synchronized with simulations let teachers present in a richer format and at the appropriate moment.

The term "Web-based simulation"[6] can be understood in several ways, but here we use it to describe a hypermedia element embodied in Web documents.

For the purpose of distance learning, there's a need for tools and procedures to integrate interactive simulations with Web documents. Simulation models and documents may not be completely independent since sometimes the document body may require information contained in the simulation model. Simulation models should also be reusable in pages or documents. We need some way to describe simulation models, the pages where these simulations reside, and the grouping of these pages to form Web documents. Because of the dynamic nature of Web-based information, these documents must be easy to modify. We also need procedures and techniques for designing Web applications in a systematic way.[7]

Here we present a three-layer language as an answer to these requirements. The lower layer programs the simulation models, the intermediate layer describes page content, and the upper layer groups pages and defines common appearance properties. The system liberates the end user from low-level details such as programming in Java or HTML. It also enhances productivity and stresses quality practices.

System overview

Here we describe our three-level system.

OOCSMP

The object oriented continuous system modeling program (OOCSMP) continuous simulation language was conceived in 1997[8] as an object-oriented language. A compiler (C-OOL) was built for this language that creates C++ code or Java applets from simulation models. Simulation-based Web courses for gravitation, partial differential equations, ecology, and basic electronics have been generated using this language (see http://www.ii.uam.es/~jlara/investigacion).

The language and the compiler have the following features, which provide an educational focus:

- Web developers can generate several forms of output displays in the same simulation.
- Web developers can configure the user interface with compiler options. For example, if we present a simulation to a naive user, we can restrict the interface to prevent the user from changing the parameters. On the contrary, if the user is an expert, we can provide more possibilities for changing the parameters such as

1070-986X/01/$10.00 © 2001 IEEE

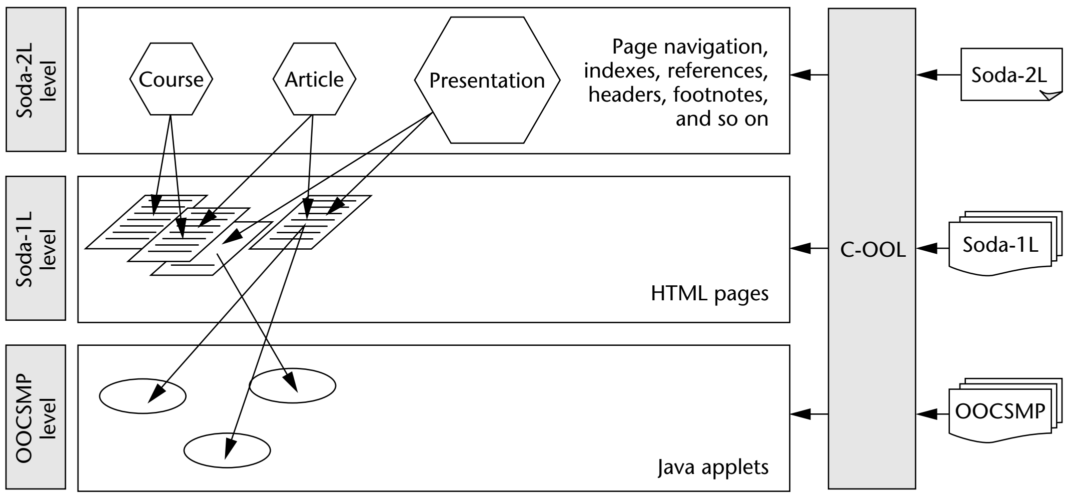


Figure 1. The three-level scheme used to generate simulation-based documents.

buttons to add or delete simulation objects.

- Users can modify, add, or delete parameters and object attributes during the simulation execution. This gives students more interaction possibilities.
- Teachers can design alternative simulations accessible from the main simulation making it possible to plan interesting situations that arise when a parameter is changed or an object is added.
- Teachers can include additional multimedia elements and synchronize them with the simulation. Thus, model behavior explanations may be presented at the appropriate moment.

The description of document pages is governed by a set of instructions termed simulation course description language 1st level (Soda-1L). This set of instructions describes Web documents containing hypermedia elements not available in HTML such as simulations, two- and three-dimensional graphics, and isosurface maps. Soda-1L forms a higher language abstraction layer than OOCSMP because the models defined in OOCSMP can be treated as hypermedia elements from the Soda-1L viewpoint.

An additional level, called Simulation Course Description Language 2nd Level (Soda-2L), can group several of the Soda-1L pages to form a course, a presentation or an article. Soda-2L has primitives that add navigation links to the pages, headers, footnotes, and that create and place indexes. They can be embedded in the resulting HTML pages or added as frames. At this level we usually add interface details common to all the pages to make the Soda-1L pages reusable.

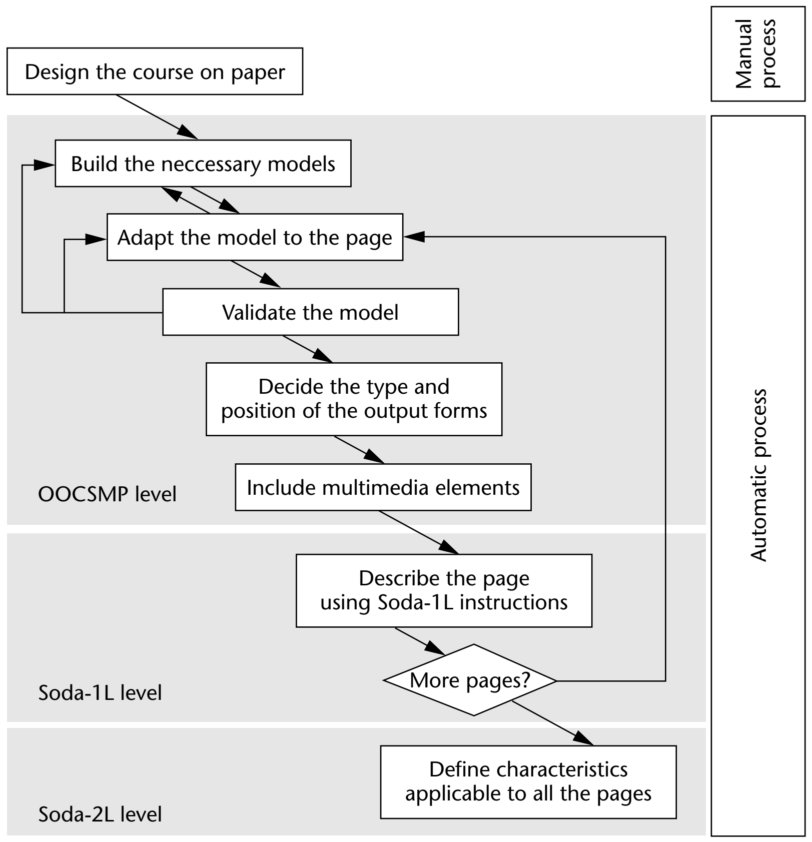
Figure 1 shows the three layers—OOCSMP, Soda-1L, and Soda-2L—and their interrelations.

Although the instructions of each level are usually included in different files, they can be mixed because they're part of a unique language and all three levels use the same compiler. Division in different files allows reuse, maintainability, and easy change of the documents.

Soda-1L

The Soda-1L level consists of a set of instructions to describe a document page. At this level we can use simulations defined in the OOCSMP level. The OOCSMP and Soda-1L aren't completely separated. From the Soda-1L level we can access the OOCSMP level to evaluate expressions and obtain values of variables. For example, if the result of an expression is a vector, in the Soda-1L level an HTML table shows the values of the elements in the vector. At this level the user can also manually insert HTML code.

We provide instructions to insert tables, images, links, sections, and references. We also offer table counters, image counters, and variables that define the author name, the author's email address, and the construction date. These variables can be accessed in footnotes and headers to make them more general and reusable. It's also possible to define writing styles and macros. Macros direct the compiler to translate some patterns to HTML. Also, the appearance of some of the Soda-1L built-in instructions can be modified using macros. This can be used to change the appearance with references, which are tracked by the compiler and sorted by name, year, and other criteria at the Soda-2L level.



Manual process
Design the course on paper
Build the neccessary models
Adapt the model to the page
Validate the model
Decide the type and position of the output forms
Include multimedia elements
OOCSMP level
Automatic process
Describe the page using Soda-1L instructions
More pages?
Soda-1L level
Define characteristics applicable to all the pages
Soda-2L level

Figure 2. Procedure to generate simulation-based Web documents.

Two- and three-dimensional graphics and isosurface maps can also be included. These elements are Java components reused from the OOCSMP output displays.

Soda-2L

The Soda-2L level lets us define features to apply to all the pages in a document, making it possible to change these pages simultaneously. Some of the available instructions in this level let us:

- Define the type of document (course, article, or presentation).
- Define common backgrounds, headers, and footnotes, and their placement as frames or embedded in the HTML page.
- Automatically generate an index for the document. The index elements could be the sections or the HTML pages. The location of this index (a separate page, a frame, and so on) can also be defined.
- Track references and order them by name, year, and other criteria. It's also possible to choose the location of the reference list (embedded in the last page, as a separate page, and so on). Macros can change the style of the references.
- Specify the navigation links between pages. We can do this for each page in the document or use a default. For example, if we're constructing a presentation, the default navigation is sequential and shows all the presentation slides.

Soda-2L scripts facilitate the modification of documents. Changing the navigation order of the pages of a document only requires a modification of the script; the pages remain unchanged. Thus, reuse of document pages is improved.

Three kinds of files can be compiled with C-OOL:

- single OOCSMP models that generate Java files,
- Soda-1L pages that generate the associated HTML file and the Java files (if the page contains a simulation model), and
- Soda-2L script where all the pages and the models contained in the script will be recompiled if needed.

Adding interactivity

With our system, users can generate simulation-based documents and interactive articles, presentations, and Web courses.

Generating simulation-based documents

Figure 2 depicts the following procedure to generate simulation-based documents.

1. Decide how many pages the document will have and which models will be contained in every page.
2. Build the models. When possible, encapsulate them in OOCSMP classes.
3. Adapt the model to the page. In most cases, creating a particular model requires declaring OOCSMP objects and connecting them by invoking methods.
4. Validate the models. The interface generated automatically by the compiler facilitates testing the models.

5. Decide which output forms to use and their position in the user interface. Users can place up to nine different output forms in the main panel and in separate windows.

6. Decide whether to include multimedia elements and identify the moments in the simulation where to place them.

7. Describe the document page using Soda-1L instructions. Users should only be concerned with the contents since most of the appearance details will be determined at the next step.

8. Define features applicable to all the pages such as background, navigation, and frames with indexes. Defining the user interface details at this level will improve the standardization of the appearance of all the document pages and facilitates their reuse.

Creating an interactive article for the Web

At an increasing frequency scientists submit papers to journals as HTML files and these articles are directly published on the journal's Web site. It would be beneficial to replace static images of the simulation results with the real simulations in papers dealing with computer simulations. Then readers would be able to experiment with the simulation models and draw their own conclusions. This can be accomplished with our system since the document viewer serves as a navigator.

For example, we submitted an article to an electronic journal (*Journal of Artificial Societies and Social Simulation*, http://jasss.soc.surrey.ac.uk/JASSS.html) using our system. The article contained embedded simulation applets instead of figures showing the simulation results (see http://www.ii.uam.es/~jlara/investigacion/ecomm/otros/JASSS.html). Figure 3 shows the Soda-2L script we used to compile the article.

Line 3 declares the document type and the background color. The next three lines define variables with data for the authors. These variables are reusable facilitating the definition of generic footnotes and headers. Line 7 sets options on the user interface. Line 8 causes the compiler to generate a section index that will be placed as a left frame, occupying 20 percent of the page width. Line 9 defines where the references must be located (in the last page), how they must be sorted (by shortname), and where their source is located (in file JASSrefs.csm). Lines 10 and 11 insert a header and a footnote in each page, the latter in the form of a frame. Line 12 declares the navigation links to be inserted in the pages in the form of a list. The last four lines declare the article pages (Soda-1L files) and the navigation between them. Each page will have a link to the next page and to the previous one. Declaring the navigation in this way provides reusability of the document pages.

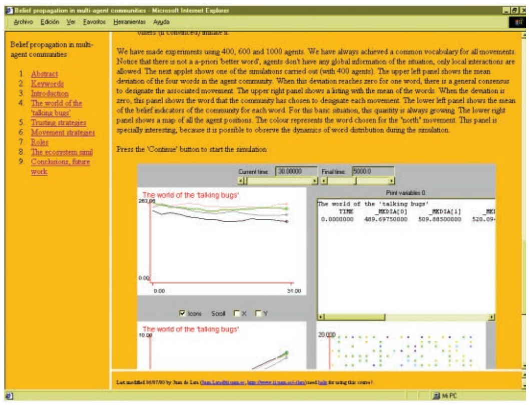
The cost of building the article was low because the models it includes were used previously in our research. In this way, the cost of including simulation models instead of pictures is negligible (in fact, it's less expensive, because we aren't required to capture a particular moment of the simulation execution). Figure 4 (next page) shows one of the pages of the article.

Creating an interactive presentation

Current applications, such as Microsoft PowerPoint, facilitate building visual presentations. It can launch external browsers, for example, to see Java applets. However, this interaction occurs outside PowerPoint control, and it's not

```
 [1] INCLUDE "macros.csm"
 [2] INCLUDE "styles.csm"

 [3] ARTICLE "Belief propagation in multi-agent
      communities"
     BACKGROUND="ORANGE"

 [4] AUTHOR     Juan de Lara, Manuel Alfonseca
 [5] EMAIL      Juan.Lara@ii.uam.es,
      Manuel.Alfonseca@ii.uam.es
 [6] WEBADDRESS http://www.ii.uam.es/~jlara,
      http://www.ii.uam.es/~alfonsec

 [7] SIMULATIONS-noFrame -noLeyenda -noScaleWindow
      -WIDTH=400 -HEIGHT=400
 [8] INDEX      [SECTIONS], [FRAME,W,20]
 [9] REFERENCES [LAST], [SORT SHORTNAME],
      "JASSrefs.csm"

[10] HEADER     "header1.csm"
[11] FOOTNOTE   [FRAME], "footnote1.csm"

[12] NAVIGATION [LIST]

[13] PAGE "JASSt1.csm" NAVIGATION [2]
[14] PAGE "JASSt2.csm" NAVIGATION [1,3]
[15] PAGE "JASSt3.csm" NAVIGATION [2,4]
[16] PAGE "JASSt4.csm" NAVIGATION [3]
```

Figure 3. Soda-2L script for compiling an interactive article.



Belief propagation in multi-agent communities - Microsoft Internet Explorer

Belief propagation in multi-agent communities

1. Abstract
2. Keywords
3. Introduction
4. The world of the 'talking bugs'
5. Trusting strategies
6. Movement strategies
7. Roles
8. The ecosystem simil
9. Conclusions, future work

We have made experiments using 400, 600 and 1000 agents. We have always achieved a common vocabulary for all movements. Notice that there is not a a-priori 'better word', agents don't have any global information of the situation, only local interactions are allowed. The next applet shows one of the simulations carried out (with 400 agents). The upper left panel shows the mean deviation of the four words in the agent community. When this deviation reaches zero for one word, there is a general consensus to designate the associated movement. The upper right panel shows a listing with the mean of the words. When the deviation is zero, this panel shows the word that the community has chosen to designate each movement. The lower left panel shows the mean of the belief indicators of the community for each word. For this basic situation, this quantity is always growing. The lower right panel shows a map of all the agent positions. The colour represents the word chosen for the "north" movement. This panel is specially interesting, because it is possible to observe the dynamics of word distribution during the simulation.

Press the 'Continue' button to start the simulation.

```
The world of the 'talking bugs'
      TIME        _MEDIA[0]       _MEDIA[1]
 0.0000000     489.69750000    509.88500000    520.09
```

Archivo Edición Ver Favoritos Herramientas Ayuda
Current time: 30.00000
Final time: 5000.0
The world of the 'talking bugs'
Print variables 0
Icons Scroll X Y

Figure 4. Screen shot of an interactive article page.

seamless. Furthermore, the results obtained when creating an online PowerPoint presentation aren't ideal because slides are converted to graphic files. One solution would be to make the presentation available in the form of a .ppt file to be downloaded, but this may not be accessible from all platforms, resulting in lower flexibility.

With our system, presentations can contain simulation applets with no need to launch external browsers. Furthermore, the presentation can be published on the Web and viewed online without any loss of interaction.

We prepared an interactive presentation for an educational simulation summer school. It contains simulation applets, and the students will be able to access the presentation online. Figure 5 shows a scheme of the Soda-2L script that we used to generate the presentation.

```
 [1] INCLUDE “macros.csm”
 [2] INCLUDE “styles.csm”
 [3] PRESENTATION “Web based Simulation”
      BACKGROUND=“BLUE”
 [4] FONT TITLE TYPE=“Tahoma”, SIZE=“+4”, COLOR=“BLUE”
 [5] FONT TYPE=“Arial,Helvetica”, SIZE=“+2”
 [6] AUTHOR     Juan de Lara, Manuel Alfonseca
 [7] EMAIL      Juan.Lara@ii.uam.es,
      Manuel.Alfonseca@ii.uam.es
 [8] WEBADDRESS http://www.ii.uam.es/~jlara,
      http://www.ii.uam.es/~alfonsec
 [9] NAVIGATION  [TABLE, 80, “GREEN”],[IMAGES
      “prev.gif”,”next.gif”]
[10] SIMULATIONS -noFrame -noScaleWindow -
      noLeyenda -WIDTH= 500 -HEIGHT= 350
[11] SLIDE “portada.csm”
[12] SLIDE “objetivos.csm”
[13]  ...
```

Figure 5. Soda-2L script for compiling an interactive presentation.

When presentations are compiled, the navigation is set by default to the next slide and the previous one. In this case, the hyper links used for navigation are placed as images (line 9) inside a green HTML table. Line 10 configures the user interface for the simulation applets. This can be changed in the Soda-1L level for special cases. We have also set the default font face and size for the titles and the texts. Line 11 inititates the slides that form the presentation (Soda-1L files). Slides and simulation models can be reused in other documents without any change. It's also easy to change the position of the slides in the presentation.

We created the slides, which with Soda-1L code can include simulations. The Soda-1L code for one slide is simple as Figure 6 shows.

```
[1] TITLE Gordon’s sine equation
[2] DESCRIPTION \CENTER {This is
     a quasi-linear equation that
     is given by:
[3] DESCRIPTION \ITALIC
     {U\SUB{xx}-
     U\SUB{tt}=sin(U)}.}
[4] MODEL “gordon.csm”
```

Figure 6. Soda-1L slide for the presentation.

Figure 6 declares the slide's title, incorporates descriptive text, and includes a simulation model (OOCSMP file gordon.csm). This model solves Gordon's sine equation using OOCSMP (see Figure 7). Details about the OOCSMP extensions to solve partial differential equations can be found in de Lara and Alfonseca.[9]

Figure 8 shows the resulting slide at a particular moment in the simulation. During the presentation, the speaker will be able to show the results of changing some of the parameters in the models.

The construction cost of this presentation was low because we reused all the models from several existing articles and courses. For example, we used the model in Figure 7 in the partial differential equations course described in the next section.

Creating a simulation-based Web course

Computer simulations prove useful to enhance laboratory learning, and they're commonly used in educational courses.[10] Thus, students explore the simulation models remotely in a hands-on paradigm.

We've generated several courses with our system, among them one on partial differential equations (PDEs). The course consists of 16 pages and contains an introduction to different numerical techniques to solve PDEs. Figure 9 shows one of the pages dealing with the finite element method. In this page, we placed several 3D graphics to show the 2D "shape functions" this method uses.

The second set of pages solves typical PDEs such as the heat equation (in one and two dimensions) and the transport equation (diffusive and nondiffusive). Also included are several pages showing examples of mesh generation.

The third set of pages contains more complex models, such as one solving the heat equation on moving pieces.

In this course, we spent most of our time validating the models. Our system facilitates assembling pages, incorporating indexes, and deciding the interface appearance.

Future work

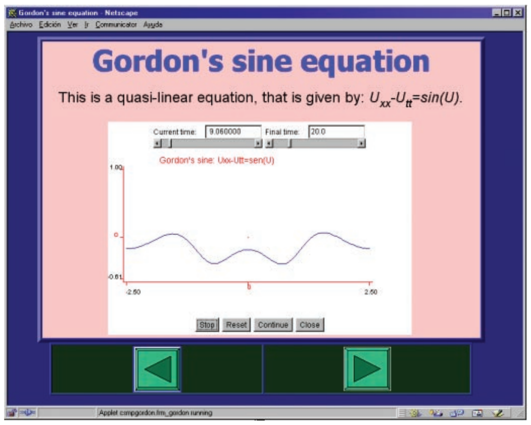
We presented several tools and techniques that simplify generating simulation-based documents. Our system stresses several key points in developing Web-based applications such as maintainability, reusability, easy testing, and integration.

The cost of constructing such Web documents is reduced considerably. There's no need to program in low-level languages such as Java or HTML. The user interface for the simulation applets is generated automatically, reducing the effort of building and testing the applets. Users will spend most of their time modeling the simulations, which can be reused between documents.

All the examples in this article can be accessed

```
[1] TITLE Gordon's sine: Uxx-
      Utt=sen(U)
[2] DOMAIN bar1d:=BAR(-
      5.0,5.0,INITIAL(EXP(-X*X)),
      INITIALDT(0),
      PERIODIC(EDGE(1,2)))
[3] MESH Res :=
      ISOPARAMETRIC(bar1d, LINE2,
      ELEMENTS(200) )
[4] PDE SG(-1, 1, 0, 1, 1, 0, 0,
      0, 0, 0, 0, 0, 0, 0, -
      SIN(SG), EXPLICIT)
[5] Res.setPDE(SG)
[6] DYNAMIC
[7] Res.STEP()
[8] TIMER FINTIM := 20.0, delta :=
      0.01, PLdelta:= 0.01
[9] PLOT2D [C], Res
```

Figure 7. OOCSMP model that solves Gordon's sine equation.

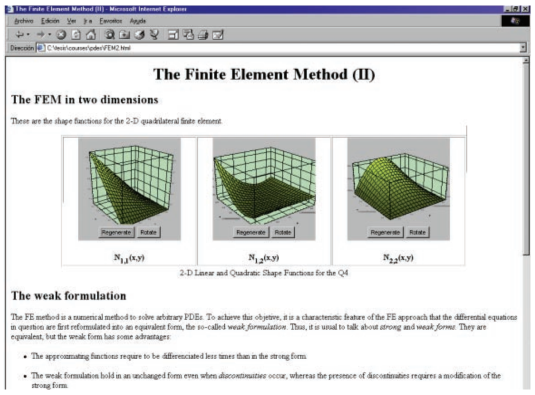


Gordon's sine equation - Netscape
Archivo Edición Ver Ir Communicator Ayuda
Current time: 9.060000
Final time: 20.0
Gordon's sine: Uxx-Utt=sen(U)
1.00
0
-0.61
-2.50
2.50
Stop
Reset
Continue
Close
Applet compgordon.fm_gordon running

Gordon's sine equation

This is a quasi-linear equation, that is given by: U_{xx}-U_{tt}=sin(U).

Figure 8. One of the slides from the presentation on Gordon's sine equation.



The Finite Element Method (II) - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Dirección C:\tesis\courses\pdes\FEM2.html
Regenerate
Rotate

The Finite Element Method (II)

The FEM in two dimensions

These are the shape functions for the 2-D quadrilateral finite element.

$N_{1,1}$(x,y) $N_{1,2}$(x,y) $N_{2,2}$(x,y)

2-D Linear and Quadratic Shape Functions for the Q4

The weak formulation

The FE method is a numerical method to solve arbitrary PDEs. To achieve this objetive, it is a characteristic feature of the FE approach that the differential equations in question are first reformulated into an equivalent form, the so-called *weak formulation*. Thus, it is usual to talk about *strong* and *weak forms*. They are equivalent, but the weak form has some advantages:

- The approximating functions require to be differenciated less times than in the strong form.
- The weak formulation hold in an unchanged form even when *discontinuities* occur, whereas the presence of discontinuties requires a modification of the strong form.

Figure 9. Page from the PDEs course.

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

Other methods have been proposed for constructing hypermedia applications such as relationship management methodology (RMM),[11] hypermedia design method (HDM),[12] and object oriented hypermedia design method (OOHDM).[13] Our approach proves beneficial with documents containing interactive simulations but can also be applied to other kinds of documents. In the future we plan to consider the possible interrelations of our work with some of these hypermedia development models.

We consider our system as a first step toward an authoring tool for simulation-based Web documents. The tool should include all the steps in our procedure (Figure 2) and should use graphical interfaces for building models and pages. We plan to address creating simulation models collaboratively, so that the model components could be distributed via the Internet.[14]

In the area of simulation-based presentations, we'd like to integrate our system with other efforts on synchronized multimedia on the Web, such as Synchronized Multimedia Integration Language (SMIL).

Finally, our system currently adapts the user interface of the simulation applet to a user's capabilities (a priori). However, we'd like to add adaptation capabilities to document contents, integrating our system with adaptive hypermedia systems, such as Tangow.[15] **MM**

Acknowledgment

We'd like to thank the Spanish Interdepartmental Commission of Science and Technology (CICYT) for sponsoring this article (project number TEL1999-0181).

References

1. A. Kristensen, "Developing HTML-Based Web Applications," *Proc.First Int'l Workshop on Web Engineering*, April 1998, http://www-uk.hpl.hp.com/people/ak/doc/webe98.html.
2. K. Maly et al., "Use of Web Technology for Interactive Remote Instruction," *Proc. Web 97 Conf.*, 1998, http://www7.scu.edu.au/programme/posters/1855/com1855.htm.
3. C. Schmid, "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.
4. R.C. Schank and C. Cleary, *Engines for Education*, Lawrence Erlbaum Associates, Hillsdale, N.J., 1995.
5. M. Alfonseca and J. de Lara, "Integration of Simulation and Multimedia in Automatically Generated Internet Courses," *Computers and Education in the 21st Century*, M. Ortega and J. Bravo, eds., Kluwer Academic Publishers, Dordrecht, 2000, pp. 47-54.
6. P.A. Fishwick, "Web-Based Simulation: Some Personal Observations," *Proc. 1996 Winter Simulation Conf.*, Soc. for Computer Simulation Int'l, San Diego, 1996, pp. 772-779.
7. S. Murugesan et al., "Web Engineering: A New Discipline for Development of Web-Based Systems," *Proc. First ICSE Workshop on Web Engineering, Int'l Conf. on Software Engineering*, Springer, London, May 1999, pp.1-9, http//fistserv.macarthur.uws.edu.au/san/icse99-webe/ICSE99-WebE-Proc/San.doc.
8. M. Alfonseca et al., "OOCSMP: An Object-Oriented Simulation Language," *Proc. European Simulation Symposium, 1997*, Soc. for Computer Simulation Europe, Ghent, Belgium, pp. 44-48.
9. J. de Lara and M. Alfonseca, "Simulating Partial Differential Equations in the World Wide Web," *Proc. Euromedia 1999*, Society for Computer Simulation Europe, Ghent, Belgium, pp. 45-52.
10. A.M.C Campos and D.R.C. Hill, "An Agent-Based Framework for Visual-Interactive Ecosystem Simulations," *Trans. Society for Computer Simulation*, vol. 15, no. 4, 1998, pp. 139-152.
11. A. Díaz et al., "RMC: A Tool to Design WWW Applications," *Proc. Fourth Int'l World Wide Web Conf.*, 1995, http://www.dsi.unive.it/~smm/docs/rmc/rmc.html.
12. F. Garzotto, D. Schwabe, and P. Paolini, "HDM—A Model Based Approach to Hypermedia Application Design," *ACM Trans. on Information Systems*, vol. 11, no. 1, Jan. 1993, pp. 1-26.
13. D. Schwabe, G. Rossi, and S.D.J. Barbosa, "Abstraction, Composition and Lay-Out Definition Mechanisms in OOHDM," *Electronic Proc. ACM Workshop on Effective Abstractions in Multimedia*, ACM Press, New York, 1995.
14. R.M. Cubert and P.A. Fishwick, "OOPM: An Object-Oriented Multimodeling and Simulation Application Framework," *Simulation*, vol. 70, no. 6, 1998, pp. 379-395.
15. R.M. Carro, E. Pulido, and P. Rodríguez, "Tangow: A System for Adaptive Distance Learning through Internet," *Computers and Education in the 21st Century*, M. Ortega and J. Bravo, eds., Kluwer Academic Publishers, Dordrecht, 2000, pp. 127-136.

Juan de Lara is an associate professor in the Department of Computer Science at the Universidad Autónoma of Madrid, where he teaches software engineering and programming. His research interests include Web-based simulation, distributed simulation, automatic code generation, distance learning and social agents. He received his PhD in June 2000, at the Universidad Autónoma of Madrid. 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.

Manuel Alfonseca is the Faculty Subdirector in the Department of Computer Science at the Universidad Autónoma of Madrid where he teaches and conducts research. His research interests include computer languages, simulation, complex systems, graphics, artificial intelligence, 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 Society for Computer Simulation, New York Academy of Sciences, IEEE Computer Society, ACM, British APL Association, and Spanish Association of Scientific Journalism.

Readers may contact de Lara and Alfonseca at Escuela Tecnica Superior de Informatica, Universidad Autónoma de Madrid, Campus de Cantoblanco, 28049, Madrid, Spain, email {juan.lara, manuel.alfonseca}@ii.uam.es.