Visual Interactive Simulation for Distance Education

Juan de Lara
Manuel Alfonseca
Department Ingeniería Informática
Universidad Autónoma de Madrid
Ctra. De Colmenar, km. 15, 28049 Madrid, Spain
Juan.Lara@ii.uam.es

The authors discuss the necessity of multiple flexible output forms in an educational Web simulation environment. Ideally, such an environment should allow the creation of visual, interactive simulations and their integration in the pages of the Web document being constructed. As a suitable candidate for this, the authors propose their system to generate documents for the Web that consists of three layers: continuous simulation language OOCSMP and its associated language layers SODA-1L and SODA-2L. OOCSMP and SODA are compiled into Java applets and HTML/VRML pages by a compiler called C-OOL. Different output forms can be combined to solve and show the results of a simulation problem. The authors propose a procedure to guide the construction of educational courses on technical or scientific subjects with their tools. This procedure is used to enhance an existing educational Web course with a new page showing the simulation of a robot arm Unimation PUMA260.

Keywords: Object-oriented continuous simulation, Web-based simulation, visual interactive simulation, virtual reality, tools for distance education, Web engineering

1. Introduction

The growing acceptance of the Internet in the present society makes it an ideal framework for distance education. The new technologies and techniques offered by this environment, as well as the possibility of reaching a huge audience, is forcing a large number of disciplines, such as computer simulation and educational sciences, to rethink their traditional philosophies and techniques [1]. One of the advantages of using the Internet as a means for distance education is that it offers common communication protocols and standards. This allows people access to information from heterogeneous platforms using a Web browser, which can have installed extensions to interact in a richer way with the HTML pages. These extensions are called plug-ins. Some of the plug-ins that we use to make available interactive simulations in the Internet are the Java Virtual Machine (http://java.sun.com) and VRML browsers (http://www.web3d.org/Specifications/VRML97).

Virtual reality (VR) immerses the users in a 3-D environment, where they can actively interact with virtual objects and explore the virtual world. Its advantages, such as new possibilities of interaction and more realistic and pleasant learning, are turning VR into a valuable tool for distance education [1, 2] (see also http://www.hitl.washington.edu/projects/knowledge_base/education.html). In the case of simulation, it reduces the semantic gap between the real system being simulated and the output obtained from the simulation, providing realistic visualization and richer interaction possibilities. Sometimes virtual reality simulations are embedded in games in which the students have to cooperate for the completion of complex tasks, such as in Scholz-Reiter et al. [4]. Without such realistic visualization and using only tables and graphics, it is sometimes difficult to relate the data obtained in a simulation with the behavior of the real system. For example, in the case of the simulation of a robot arm, it may not be easy to relate the angle values of the different joints to the real configuration of the robot. Visualizing the movement of the robot arm in a realistic way can improve understanding of the simulation model.

If the simulation environment makes it possible to see different output forms at a time, then one can combine the VR output—to visualize the behavior of the system—with other outputs such as tables or 2-D graphics for a more accurate visualization of the quantities of interest. Nowadays, the use of VR is not restricted to high-performance systems [5]; even personal computers are able to run browsers for a (limited) VR language called VRML. This language allows describing interactive 3-D objects and worlds and was designed to be used on the Internet. The VRML97 standard defines an external Application Programming Interface (API) that makes it possible to control VRML objects from external programs, thus facilitating the integration of VRML panels with other output forms written in Java.

SIMULATION, Vol. 79, Issue 1, January 2003 19-34

DOI: 10.1177/0037549703253455

Some educational environments facilitate learning by making the user interact with simulations dealing with the subject of study [6]. The use of simulations in education presents positive aspects [7] if the student cannot access the real system or if experiments with the real system are expensive (such as in the case of the robot arm), difficult, or just impossible (such as if we try to simulate the movements of the planets of the solar system; see Section 4). The simulation of the main characteristics of the system inside educational environments, especially if these environments are provided with multimedia [8] and VR elements, can be a good complement to laboratory experimentation.

There is a need for tools to help in the construction of educational interactive simulations that integrate different views of the simulation results with other explanations, possibly in the form of multimedia elements. Usually one would like to organize several of these simulations in a logical way to form educational materials such as courses. Thus, an authoring tool to assist in the construction of these materials should help in the tasks of building simulations, arranging and synchronizing the (multiple) output forms, embedding them in HTML pages, arranging the pages to form a course, and so forth. Moreover, very often the constructor of such documents is not an expert programmer in Java, VRML, HTML, and related Web technologies. Thus, it is desirable to hide these technical details from the course designer. In addition, the use of a high-level description for simulations, pages, and documents can reduce notably the time needed to build these materials, augment their maintainability, and reduce the time spent in testing [9].

Our aim is to provide such an environment to build Web documents containing interactive, multimedia simulations. Although there has been recent interest in the special needs of constructing applications for the Web (Web engineering), very little can be found in the current literature about the engineering of Web applications with a strong component of simulation. In this paper, we give a unifying view of our previous work with a Web-engineering perspective. We also present some extensions of our environment that allow the easy integration of (Java-based) simulation applets with (VRML-based) virtual reality worlds and multimedia elements, as well as the integration of all these elements in documents accessible from the Web. The description of the simulation and the output forms is made using a declarative high-level simulation language (designed by our group) called OOCSMP. Other language layers (SODA-1L and SODA-2L) allow including simulations in document pages and arranging the pages to form educational courses, interactive articles, and presentations. We also show a procedure that can be used when building Web documents with our tools, and we illustrate the use of this procedure and the proposed tools by enhancing an existing educational course.

This paper is organized as follows: Section 2 presents the problems and alternatives found in Web-based continuous simulation, together with the motivations of this work; Section 3 presents the basic architecture of our system; and Section 4 shows the extensions introduced to handle virtual reality. In Section 5, we provide a procedure to follow when building documents with our tools. Section 6 presents an example of the use of these extensions (the simulation of the PUMA270 robot arm) together with the inclusion of this model in an existing course, following the procedure presented in the previous section; Section 7 describes related research; and Section 8 gives the conclusions and future work.

2. Web-Based Simulation: Problems, Alternatives, and Motivations

There are several ways of accessing simulations through the Internet [10]. The first one is known as the *thick server approach*. In this case, the simulation programs execute at the server and are programmed in any language accessible through the Common Gateway Interface (CGI). This is not a good approach with interactive simulations, in which the user experiments during the simulation execution, trying to answer "what if" questions. In these experiments, the user is supposed to stop the simulation, change some parameters, and resume the simulation. In the thick server approach, these interactions may lead to long delays and inconsistencies due to Internet latencies: the user may try to interrupt the simulation long before the server receives the signal to stop. However, for noninteractive simulations, this can be a suitable approach (see, e.g., the GPSS Web simulator of the University of Magdeburg at http://www.cs.mcgill.ca/~hv/classes/MS/GPSSH/webgpssh.html).

In opposition to thick servers, the *thick client approach* servers do not run the simulations: they are downloaded and executed locally. This approach has the danger of client incompatibility or virus transmission and is not appropriate if the simulations are to be integrated with other services offered through the Web, such as cooperative learning support, learning guidance, and so forth. To solve these difficulties, in the *pure navigation approach*, the Web browser becomes the common access point to all the simulations. These are executed in the client machines as Java applets, possibly using other plug-ins. Java has many interesting properties (e.g., "write once, run everywhere") that provide client independence. This is our approach.

Other ways to integrate simulation and Web services [11] are distributed execution [12] and distributed modeling [13, 14]. In the distributed execution of continuous simulation models in the Internet, one cannot use traditional ways of fine-grain parallelization (such as solving a set of equations in parallel by assigning one or a few elements of the resulting equation matrix to each node in the network). A coarse-grain parallelism should be exploited due to Internet latencies. One approach is to express the model as a collection of objects that interact, group the objects that interact more frequently, and place them in the same machine. Objects in different machines will interchange messages through the network, possibly using

remote method invocation techniques [12]. This approach is similar to that taken by the High-Level Architecture (HLA) (see http://www.dmso.mil) for the parallel execution of discrete simulation models. On the other hand, for distributed modeling, when a group of designers collaborate in the modeling phase, current approaches take ideas from the Computer-Supported Cooperative Work (CSCW) community.

To reduce the effort required to build the simulation models in Java, one has two alternatives [15]. On one hand, a library can be provided containing predefined classes, which may be used by the programmer to build the model. In this case, the model builder programs directly in Java [16]. On the other hand, a special-purpose simulation language can be used, together with a compiler to translate the models into Java code. Some simulation languages are provided with compilers and environments able to translate models into Java—among them, GPSS, DEVS, and OOCSMP. This is our approach and has the advantage of simplifying the modeling process because simulation languages are higher level than regular programming languages, provide more powerful constructs, and are easy to use for nonprogrammers. These simulation languages are well known in the simulation community, and one may have access to a great number of models and libraries. A general-purpose language such as Java is more flexible but requires a higher level of expertise. Something similar happens with the increasingly popular system for graphical design, Flash (http://www.macromedia.com). This system is extremely powerful for graphics and animation, but one would have to code the simulation numerical solvers by hand.

As stated in the introduction, it is often desirable to integrate several simulations in a Web document. Again, one has two alternatives: doing it by hand (creating the HTML pages and embedding the applets inside the pages manually; for examples, see [17] and a virtual enginnering/science laboratory course at http://www.jhu.edu/virtlab/virtlab.html) or using an integrated environment for the creation of the simulations and the document pages. The second alternative is our approach. Creating the document pages by hand usually requires skill and expertise in HTML programming. Moreover, if the simulations make use of VRML plug-ins, arranging them in the HTML page may not be straightforward. In addition, it could be useful to show some simulation data (such as initial variable values) in the Web page outside the applets. Using the first approach means that if one changes the simulation, the Web pages must be changed too.

Three kinds of Web documents could be enhanced by interactive simulations [9]. The first kind, educational courses on technical or scientific topics, consists of a number of HTML pages containing simulations, possibly enriched with multimedia elements. It may be desirable to control the degree of interaction of the student with the simulations by adding interaction capabilities in a progressive way [18].

Scientific interactive articles also offer the possibility of taking advantage of visual, interactive simulations: when a simulation model is described, a real executable model may be added with which the reader can experiment. The simulation can incorporate further explanations of the results by the author of the article, synchronized with the simulation execution. It must be noted that nowadays, almost all scientific journals have Web servers through which electronic versions of the papers are made available.

Presentations describing simulation models and their execution are also good candidates for Web-based simulation techniques. In these documents, the slides should be HTML pages. When a simulation model is described, a Java applet may be added, so that the audience can see the execution. These presentations can also be published in the Web and seen by the readers using a Web browser. Typical presentation systems, such as PowerPoint, are not platform compatible and do not allow the execution of interactive simulations inside the slides.

Our approach is unique in the sense that it provides an integrated environment for the construction of the three types of documents. Simulations are specified in a high-level language called OOCSMP and may use different output formats, including VRML panels and multimedia elements (images, videos, or text) that can be synchronized with the simulation by means of language primitives. The pages in which these simulations are placed are specified using another language layer (SODA-1L), which can access information in the OOCSMP models, thus solving the problem of maintaining separate simulations and document pages. Controlling the way in which the pages are arranged to form articles, presentations, or courses is done in another, higher language layer called SODA-2L. Our system has been designed for reusability (not only of OOCSMP models but also of SODA pages), maintainability (in Section 6, we show how to extend an existing educational course), easy testing, standardization of the user interfaces, and so forth. The use of higher level languages increases notably the productivity of the document designer and avoids the need to know lower level languages such as Java, HTML, or VRML. To our knowledge, no other available tool can be used to build educational materials for the Web with a strong component of visual interactive simulations with these characteristics. On one hand, some of the existing tools are very good for graphics but lack powerful constructs for simulation (this is the case for Flash and Authorware). On the other hand, none of the existing simulation tools allows the seamless integration of simulations in educational materials for the Web.

3. The Integrated Simulation System: OOCSMP, SODA-1L, and SODA-2L

The lower layer of our integrated system is OOCSMP (http://www.ii.uam.es/~jlara/investigacion), an object-

oriented extension of the continuous simulation language CSMP [19], which we began to develop in 1997 [20]. This language is very suitable for models that can be expressed as similar interacting components and has extensions to solve partial differential equations (using finite elements or finite differences methods and structured or unstructured meshing techniques), manage discrete events, produce distributed simulations, and perform agent-based simulation [21]. A compiler called C-OOL (Compiler for the OOCSMP Language) has been developed for the language. C-OOL is able to generate Java programs or applets, plain C++, or C++ programs that use the Amulet library [22].

To make this approach useful for education, different output forms can be combined in the simulations. The C-OOL compiler provides a user interface that allows the student to answer "what if" questions and to interact with the simulation. The simulation results are presented as they are being calculated. The user interface can be configured by means of compiler options to adapt to the characteristics of the user by restricting or enhancing the possibilities of interaction, such as allowing the user to create new simulation objects at runtime [23] or preventing the modification of some parameters.

A typical Java user interface generated by our compiler consists of a main panel and several additional windows. The panel has a maximum capacity of nine output forms, in a 3×3 grid, but if not all locations are used, the grid automatically becomes as compact as possible. The panel also contains several scroll-like objects to adjust the simulation final time, the time step, and the current time, plus several buttons to view/modify the values of the parameters and state variables of the simulation objects and the global variables. Another nine windows can be displayed with different graphical representations. A scheme of the user interface is shown in Figure 1.

Each graphical output form can be placed in one or more of the nine panel positions or in a separate window. The types of graphical outputs available in OOCSMP include animated 2-D graphics (for 1-D functions and vectors); 3-D plots for matrices; iconic plots, which represent with icons the time-dependent values of variables (the number of visible icons in the same class is proportional to the value of the variable they represent); graphical representations of the equations in the model; outputs for agent-based simulation [21]; plots of the grid used to solve a partial differential equation; maps of isosurfaces; and a listing of variable values. Some of them are also input forms, such as the MGEN tool, which allows the user to interactively define a partial differential equations problem, including the domain, mesh, conditions, and the equations [18]. An example of all these outputs can be found at the OOCSMP homepage (http://www.ii.uam.es/∼jlara/investigacion).

The other two layers of the system address the integration of the simulation applets with the HTML pages of the Web documents [9]. They are called SODA-1L (Simulation Course Description Language–1st Level) and SODA-2L (Simulation Course Description Language–2nd Level). The SODA-1L level provides a set of instructions that describe document pages containing hypermedia elements not available in plain HTML, such as simulations, 2-D graphics for functions, 3-D graphics, and maps of isosurfaces. SODA-1L is at a higher language abstraction level than OOCSMP because the models defined in OOCSMP can be treated as hypermedia elements from the SODA-1L viewpoint. SODA-1L deals mainly with the contents of the pages. Details about their appearance (which are mostly common to all the document pages) are left to SODA-2L. At this level, one can group several of the SODA-1L pages to form an educational course, a presentation, or an article. SODA-2L provides primitives to add navigation links, headers, and footnotes; create and place indexes; and so on. They can be embedded in the resulting HTML pages or added as frames. At this level, one usually adds interface details common to all the pages, which makes the SODA-1L pages easy to reuse. Figure 2 shows the organization of all the language layers.

The system has been used for educational purposes: several courses containing simulations on ecology, gravitation, partial differential equations, and electronics [24] can be accessed at the OOCSMP homepage (http://www.ii.uam.es/∼jlara/investigacion). The system has greatly improved our productivity in creating and maintaining these courses, as we do not need to program the numerical methods, the simulation user interfaces, the graphical outputs, or the HTML of the documents: all these elements are provided or generated by the system and can be accessed using declarative, high-level constructs.

4. Adding VRML Panels to the Simulation System

As stated in the introduction, the use of VRML panels in simulations reduces the gap between the presentation of the results and the real behavior of the system, making the simulations much more realistic and pleasant to the student. In OOCSMP, it is possible to combine VRML panels with the other output forms or with multimedia elements. The VRML panels in our system also permit a richer interaction with the model during the simulation execution. For example, one can set hyperlinks in the VRML objects, which can be used to explain the role of the object in the simulation or to show additional data when the user clicks on them. We have used these possibilities in a simulation of the solar system in such a way that when the user clicks on one of the planets, its name, mass, radius, and other data are shown. VRML browsers also have controls to rotate the world, move away or nearer, change the viewpoint, and so forth. This permits the user to choose the most appropriate view of the simulation.

The OOCSMP VR extensions make it easy to assign a VRML node to each OOCSMP object. In the case of the simulation of the solar system, we can model each planet as an OOCSMP object of class *Planet* and assign to it

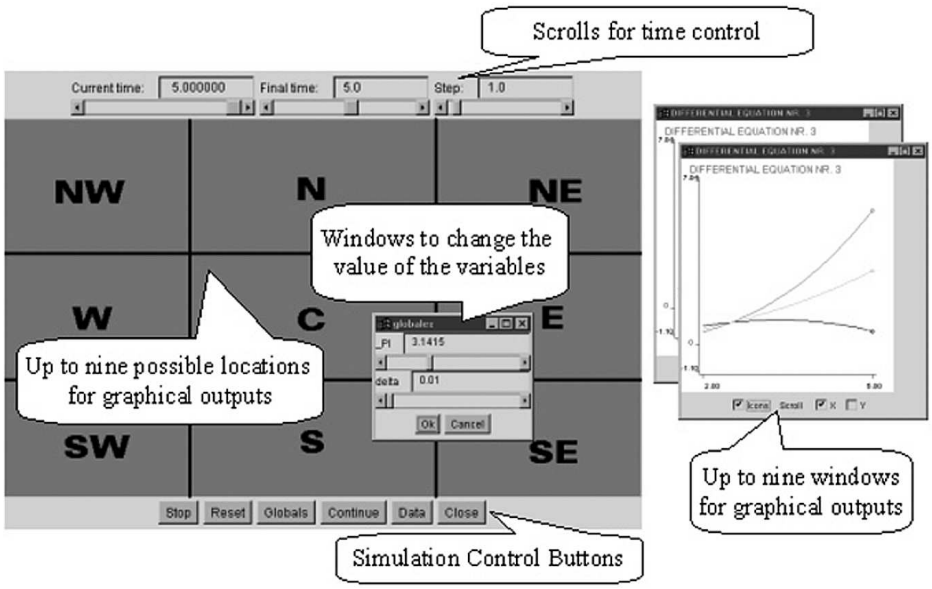

Figure 1. The scheme of a typical user interface for experimentation

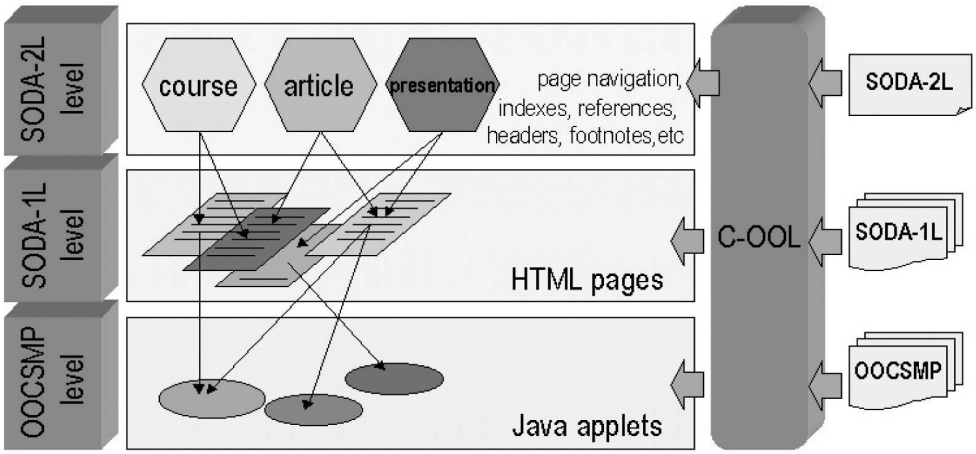

Figure 2. The three language layers used to generate simulation-based documents

a VRML node with its 3-D appearance. Another instruction (*VRMLworld*) assigns OOCSMP objects as dynamic components of a virtual world. Attributes of the OOCSMP object control properties of the VRML object (displacement, rotation, center of rotation, size, or color) in such a way that when the attribute changes, the visual appearance of the VRML world also changes. This instruction can be used in three ways, depending on where it is placed. At the end of a class or a model, the simulation engine will modify the property of the corresponding VRML object at every time step. For example, in the case of the simulation of the solar system, the *VRMLworld* instruction inside the OOCSMP *Planet* objects links the displacement of the VRML node with the position of the planet calculated by the simulation.

If the instruction is placed at the end of the main model, it can have a VRML file as a parameter. The contents of this file will be added to the virtual world generated by the simulation but will not be controlled by the simulation engine (they will be static elements of the virtual world).

In the case of the solar system, this feature can be used to make the Sun appear in the VRML world, assuming that the position of the Sun is not computed by the simulation and is placed by default at (0, 0). If the instruction is placed in a discrete event handler, when that event occurs, the corresponding property will be modified.

When compiling the model with C-OOL, one Java file and one VRML file are generated. The VRML file contains all the VRML objects declared in the model. The compiler assembles the VRML instructions associated with every OOCSMP object and also the static VRML elements declared outside OOCSMP classes. The Java applet contains the simulation logic, the simulation controls, and the other graphical output forms, if any has been selected. The Java applet allows the user to start and stop the simulation, change the model parameters, and so forth and communicates with the virtual world by means of the External Authoring Interface (EAI). Proprietary technologies such as Netscape's *LiveConnect* [25] are widely used [2, 26], but we think our solution is more appropriate because additional libraries are not needed: our solution is compatible with Internet Explorer, and communication between the Java applet and the virtual world is done directly from the program, rather than indirectly through JavaScript.

Figure 3 shows the working scheme of all these components, including a graphical Web browser, a VRML97-compliant plug-in such as CosmoPlayer, and a Java Virtual Machine plug-in that understands at least Java 1.1 code. The generated Java programs use a graphical and numerical Java library placed in the server (for developing purposes, it can be placed in the client). When the user accesses the simulation page, the VRML code, the Java applets, and the necessary Java classes are downloaded from the server and executed locally. The simulation engine, embedded in the Java applet executed in the client, keeps the Java-based outputs and the VRML outputs synchronized. For that purpose, it has to obtain a handler to the VRML plug-in through the EAI and explicitly perform the updating of the VRML nodes, depending on how the simulation variables influence each VRML node (through displacement, rotation, etc.). Figure 3 also shows a simulation of the inner solar system, using a 2-D (Java-based) output form and a VR panel (based on VRML). This simulation can be found on the Internet at http://www.ii.uam.es/~jlara/investigacion/ecomm/solar3.html.

An alternative to using VRML would be to implement the VR panels directly in Java, using technologies such as Java3D (http://java.sun.com/products/java-media/3D) or Shout 3D (http://www.shout3d.com). The advantage of using VRML is that it is an Internet standard faster than Java, and it is easier to find libraries of VRML objects to use in the simulations (see the Web3D Repository at http://www.web3d.org/vrml/vrml.htm). The advantage of using Java for 3-D modeling is that only the Java Virtual Machine is needed to run the models (without any VRML plug-in), and it can be better fitted with the simulation engine (Java applets) generated by our compiler. We deem the advantages of using VRML greater.

5. Procedure to Generate Web Documents

To carry out a certain activity—such as creating documents for the Web based on visual interactive simulations—one not only has to provide the tools to perform it but also has to propose a procedure showing the best way to do it (i.e., a sequence of activities, their relationships, and their inputs and outputs). Figure 4 shows the procedure we follow when we construct a Web document with our system. This procedure is an improvement to the one presented in de Lara and Alfonseca [9].

1. The first step is a manual activity, in which we plan the organization of our documents, deciding how many pages the document will have and which models are going to be placed in each page. The output of this step is a plan or a scheme for the Web document, containing descriptions of the models to be included in the pages. The next steps are performed using our tools.

2. In the second step, we make high-level representations of the systems under study, using notations such as Forrester system dynamics, statecharts, block diagrams, UML class diagrams, and so forth. The notation and the degree of formality in this phase depend on the nature and complexity of the system to be simulated. This step is also necessary in software engineering for the same reason as in modeling and simulation: one needs high-level representations of the systems to better understand them and to tackle their complexity. If the model is simple, it will be written on a sheet of paper and translated by hand into OOCSMP code in the next step. In harder cases, an automatic tool such as AToM^3 [27] can be used. This is a meta-modeling tool that accepts a description (meta-model) of the formalism one is going to use and automatically generates a tool to process models in the described formalism. It is also possible to define model manipulations, such as code generation, for other tools. In this way, we have defined transformations of some modeling formalisms into OOCSMP, such as causal block diagrams [28] and statecharts. The input to this activity is the plan of the Web document and, in particular, the description of the models in each page. The outputs of this activity are the models, expressed in an appropriate high-level modeling formalism.

3. In the third step, we code the models in our object-oriented continuous simulation language, OOCSMP. The models should be coded as OOCSMP classes in such a way that they can be reused throughout the Web document. Sometimes, the OOCSMP

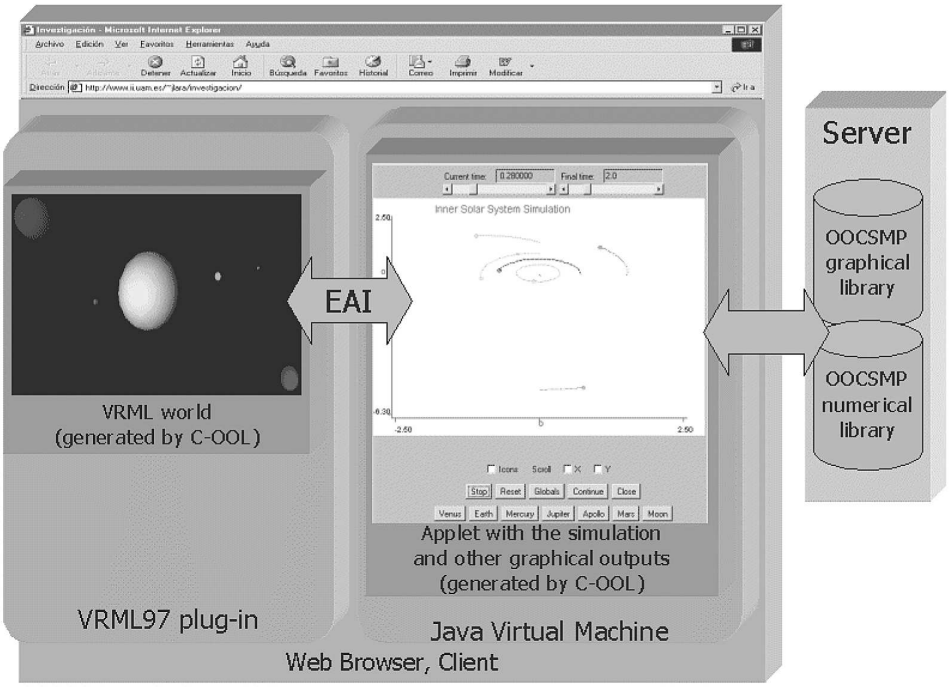

Figure 3. Using a virtual reality (VR) panel from a simulation: Inner solar system example

code can be generated automatically from higher level models by using AToM^3 or other customizable modeling tools (most tools supporting UML classes allow one to define templates for code generation). If this is the case, the generated code may have to be completed or modified by hand. The inputs to this activity are the models, described in a high-level formalism. The outputs of this activity are the OOCSMP files.

4. In the fourth step, and for each page in the document, we build the models to be included in the page. Usually, this means instantiating some of the classes, creating objects, and connecting them. The input to this activity is one of the OOCSMP files generated in the preceding step, and its output is a modified OOCSMP model adapted to the particularities of the page.

5. In the fifth step, the model is validated. The interface generated automatically by our compiler makes this testing process easier. For this step, we usually choose a simple output form, such as a listing of variable values, or 2-D graphics of some variables against time. If the model is not correct, we go back to the preceding steps (Step 2 or 3) to correct the model. The input to this activity is the OOCSMP model, and its output is the validated OOCSMP model.

6. Once the model has been validated, we select the output forms to be displayed. As we saw in previous sections, several output forms can be combined in a single model and placed in different positions in the user interface. The input to this step is an OOCSMP model, and the output is the model extended with appropriate output forms.

7. The next step is an optional activity to include and synchronize multimedia elements (if any) in the model. The inclusion of multimedia elements in the simulations makes the learning process more pleasant to the students. Explanations may be made richer than those that can be provided with simple static HTML text. The learning process is further improved if the explanations are synchronized with what is happening in the simulation.

   OOCSMP provides the possibility to include multimedia elements in the simulation, combined with other output forms. The multimedia elements available are image panels, dynamic text panels, video panels, and audio sequences. These multimedia elements can be synchronized with the simulation execution. In this way, appropriate explanations are

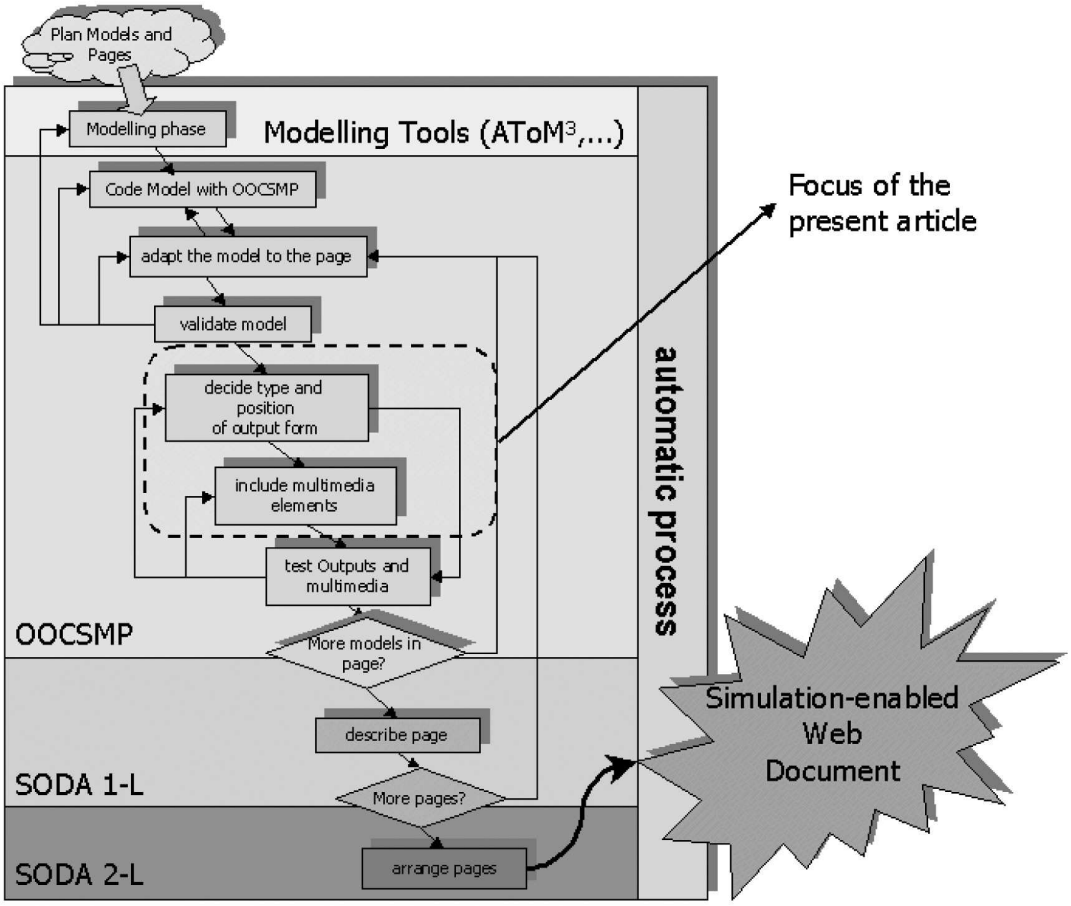

Figure 4. Our process to construct simulation-based Web documents

given to the student in the appropriate moment. Synchronization is done by specifying conditions in the simulations that trigger the execution of a given multimedia element. These conditions can be any valid OOCSMP expression. The multimedia element stops its execution when the associated condition is no longer true. If several conditions are true for the same panel, the last specified element is presented in the panel.

The input to this step is an OOCSMP model, and its output is the model extended with multimedia elements.

8. The next step tests the suitability of the output forms and the correct synchronization of the multimedia elements. The input to this step is an OOCSMP model provided with graphical outputs and/or multimedia elements, and its output is a validated OOCSMP model. If there are more models to be included in the current document page, the process goes back to Step 3 for each of the other models. Otherwise, we go to Step 9.

9. Step 9 uses the SODA-1L layer to describe the contents of the page and call the OOCSMP simulations produced in the preceding steps. SODA-1L makes it possible to include in the pages some elements not directly available in plain HTML, such as 2-D or 3-D graphics, maps of isosurfaces, and so forth. The inputs to this step are the models to be included in the page and the scheme of the page created in Step 1. The output of this step is one of the pages in the document. If there are more pages in the document, the process goes back to Step 3; otherwise, we continue with the final step.

10. This step is performed using the SODA-2L language. Here we have to create a script indicating the type of document (educational course, presentation,

or article) and define common appearance elements, such as backgrounds, font types, headers, footnotes, frames, and so forth. We can also indicate if we want to generate an index and how the references (if any) should be tracked. One important issue is that we can also specify the navigation throughout the document, so that, if we later decide to change the order of the pages, the SODA-1L pages remain unchanged and only the SODA-2L script has to be modified.

6. An Example: Enhancing an Existing Course with the Simulation of a Robot Arm Using VR

As an example of using the procedure and tools presented earlier, in this section we enhance an existing educational course with new pages. This will also show that our tools allow for the easy maintenance of the documents: when a new page is included, all the navigation links, indexes, and so on in the course are regenerated automatically. We will enhance the course with a page containing a simulation of the robot arm Unimation PUMA260 [29]. Robotic simulations can be interesting in the sense that industrial robots are so expensive that sometimes a university department cannot afford to buy one, and experiments with VR simulation may be a valid alternative. If the simulation can be accessed from the Internet, the student can also experiment at home. To build this page, we will follow the procedure in Figure 4, assuming that Step 1 (planning the course) has been performed before.

6.1 Step 2: Modeling Phase

In this step, we can make (several) models of the system using an appropriate formalism. The objective is to understand the dynamics of the system and, in our case, to plan the implementation. In this case, we use some of the UML diagrams described in Booch, Rumbaugh, and Jacobson [30]. To model physical systems, the UML diagrams can be complemented by simulation formalisms, such as partial differential equations (PDEs), ordinary differential equations (ODEs), and so forth. In robotic and mechatronic simulations, it is useful to use UML notations and map the models into an object-oriented simulation language such as OOCSMP. Figure 5 shows two of the UML diagrams that we have used for our problem (a class and a collaboration diagram).

The diagram to the left is a structural diagram used to describe the system architecture. Basically, we have a class named *PUMA260* (which represents the robot) containing four joints, the last one with a manipulator able to rotate through two different axes. Some of the robot attributes are its position (PX, PY, and PZ), the rotation degrees performed by each $Joint(G1 \ldots G4)$, and three rotation matrices (*ROTX*, *ROTY*, and *ROTZ*). In the methods section, the public interface allows rotating one of the joints a number of degrees (methods *ROTATE1* . . . *ROTATE4*). The first argument of these methods gives the order in which each rotation has to be performed. The other public methods are *GETANGLE1* . . . *GETANGLE4*, which return the angle value of each joint.

The *Joint* object is assigned a *VRMLObject* (a VRML file with the *Joint* graphical appearance) whose initial position is given by ($X0$, $Y0$, and $Z0$). The public interface defines the *INITIAL* method, which performs initial calculations, and the *ROTATE* method, which performs a rotation of the joint, given a rotation matrix. All the joints in the robot are *connected* to the next joint, except the manipulator, which is last. By encapsulating joints in objects, we can use them not only to describe PUMA robots but any other kind of robot.

The diagram to the right is a behavioral model, used to understand the flow of method invocations in the system. One could have an arbitrary number of these diagrams, with each one modeling a different situation. In our case, we show the behavior of the system when the main simulation invokes a rotation on the robot (method labeled 1). In particular, in the example shown, we wish to rotate the second joint by $\pi/2$ degrees. When a *PUMA260* object receives this message, it checks if the order parameter is appropriate—that is, if the number of rotations performed is equal to parameter *ORDER* minus 1. If this is the case, it invokes on itself method *ROT2*, which in turn invokes method *ROTATE* on the appropriate joint, using the appropriate rotation matrix (each joint is able to rotate through a different axis). When the *Joint* object receives this message, it performs the necessary operations and passes the message to the next joint connected to it until the manipulator is reached. This is repeated at each time step until the joints reach the desired rotation angle and then the next rotation can be executed.

6.2 Step 3: Coding the Model

In this step, we have to map the model produced by the previous step into the OOCSMP code. Using automatic tools, it is possible to automatically generate part of the OOCSMP code from the UML class diagram. The diagram may be completed with the OOCSMP code for the methods in the form of notes. As an example, Figure 6 shows a scheme of the OOCSMP code for the *Joint* class. *TINVERSE* is an OOCSMP function to invert a transformation matrix, using a well-known and more efficient algorithm than the usual inversion [31]. Details about the algorithms used to handle transformation matrices can also be found in Foley et al. [32].

6.3 Step 4: Adapting the Model to the Page

In this step, we create the main simulation models using the classes built in the previous step. This usually means creating instances of the classes, parametrizing the objects, and calling the appropriate methods inside the main simulation loop. Figure 7 shows the model for this page. It is a

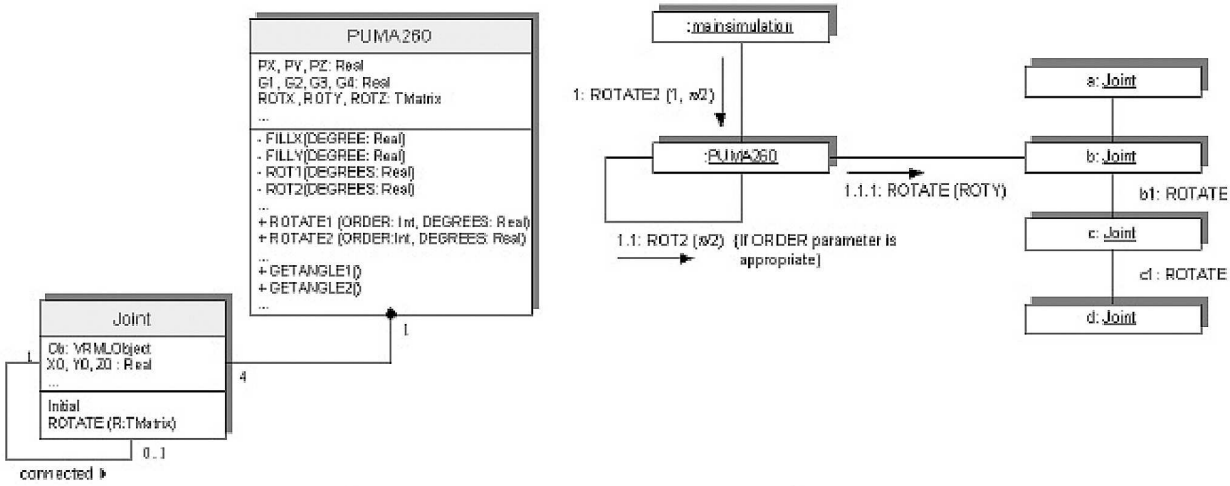

Figure 5. Modeling the robot arm using UML: (a) class diagram and (b) collaboration diagram

```
CLASS Joint
{
 Joint connected                                        * Pointer to the next Joint of the
                                                          robot
 VRMLobject ob := ''artic1.wrl''                        * VRML file containing the physical
                                                          shape
 DATA X0 := 0, Y0 := 0, Z0 := 0                         * Initial displacement of the joint
 DATA M[4;4], M[;] := 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1   * Initialization of local coordinate
                                                          matrix
 DATA INVM[4;4], X[4;4], XINV[4;4]                      * Auxiliary matrixes
 INITIAL                                                * Initial section
   M[0;3] := X0
   M[1;3] := Y0                                         * Initialize M with displacements
   M[2;3] := Z0
   TINVERSE(INVM, M)                                    * Calculate inverse

 2VRML                                                  * Convert matrix notation to fixed
                                                          angles notation
   Tx := M[0;3]                                         * Obtain displacements
   Ty := M[1;3]
   Tz := M[2;3]
   GRY:= ATAN2(-M[2;0],SQRT(M[0;0]*M[0;0]+M[1;0]*M [1;0])   * Obtain angle around y axis
   GRZ:= ATAN2(M[1;0]/COS(GRY), M[0;0]/COS(GRY) )       * Obtain angle around z axis
   GRX:= ATAN2(M[2;1]/COS(GRY), M[2;2]/COS(GRY) )       * Obtain angle around x axis
   INSW(1, , VRMLworld MOVE, Tx, Ty, Tz )               * Displace in VR world
   INSWW(1, , VRMLworld ROTATE, 1, 0, 0, GRX)           * Rotate around x in vrml world
   INSW(1, , VRMLworld ROTATE, 0, 1, 0, GRY)            * Rotate around y in vrml world
   INSW(1, , VRMLworld ROTATE, 0, 0, 1, GRZ)            * Rotate around z in vrml world

 ROTATE R[;]                                            * Apply a transformation matrix
   INSW(connected, , X := connected.M**INVM )           * Propagate the transformation
   INSW(connected, , TINVERSE (XINV, X) )
   INSW(connected, , connected.ROTATE ( (X**R)**XINV ) )
   M := R**M                                            * Apply the transformation
   INSW(connected, , TINVERSE (INVM, M))                * Calculate the inverse, if necessary
   2VRML
 }
```

Figure 6. Declaring joints as OOCSMP objects

very simple model that creates the robot and, in the main simulation loop (*DYNAMIC* section), makes it perform rotations on its second and third joints.

Note how, although both rotations are included in the main simulation loop, they are not performed by the robot at the same time. The order in which they should be performed is given by the first parameter of the method to rotate. As we explained in Figure 5b, before performing a certain rotation, the robot must check if all the previous ones (with lower order numbers) have been completed.

6.4 Step 5: Validating the Model

In this step, we check that the previous model is correct by compiling it and observing the output. When we compile Figure 7, C-OOL generates a VRML file containing the PUMA robot, which includes the code for the four joints. This file is the result of assembling the VRML files that contain the description of each *Joint* object. The compiler also produces three Java files, one for the *Joint* object, one for the *PUMA260* object, and another for the main simulation model. The validation is an iterative process in which we try rotations for each *Joint* object.

6.5 Step 6: Deciding Type and Position of the Output Forms

In this step, we complete and add output forms to present the user with richer information. In our example, we add a static VRML object to the VR panel (described in file "tool.wrl," the cone in Figure 9) and add a 2-D plot to the right of the 3-D panel to show the angles of the joints. The modifications to Figure 7 are shown in Figure 8 (the first six lines remain unchanged). It can be observed that some methods of the *PUMA260* object (*GETANGLE1* to *GETANGLE4*) are invoked to get the angles of the joints, which are plotted in the 2-D graphic (line 12).

A picture of a moment of the simulation is shown in Figure 9. The top window to the right (it pops up by clicking on the *p* button in the main panel) can be used to change some parameters of the robot, such as joint angles, velocity, position, and so on. Notice that the VRML panel has controls that permit the user to explore different viewpoints of the simulation visualization.

The simulation and the visualization processes are executed in different threads, as each output form creates its own thread. The synchronization of the simulation engine with the Java-based output forms is straightforward as the simulation engine only has to send update messages to the output forms at each communication interval. This is set in the simulation model (variable *PLdelta*; see line 13 in Figure 8) as sometimes we are not interested in updating the output forms at each time step, or it would be too costly. In the example in Figure 8, we update the output forms at each time step—that is, variables *delta* and *PLdelta* have the same value.

6.6 Step 7: Adding Multimedia Elements

In this step, we can include multimedia elements in the model. The multimedia elements available in OOCSMP are image panels, dynamic text panels, video panels, and audio sequences. In this example, we add a text panel describing to the student the movements the robot is performing. We synchronize different texts with the robot movements in the simulation. Figure 10 shows the modifications to be done to Figure 8. Line 12 now locates the plot of the joint angles at the center of the main panel of the user interface. Line 13 adds the dynamic text panel below the plot panel. The first explanation is shown when the angle of the second joint is smaller than $\pi/2$ and is stored in a text file named "explain_2nd_joint.txt." The second explanation appears when the first condition is not met and is expressed using the *DEFAULT* keyword.

6.7 Step 8: Testing Outputs and Multimedia

In this step, we check if the outputs and multimedia elements synchronize as planned. If this is not the case, we go back to Step 6 (if it is a problem with the output that does not affect the multimedia elements) or to Step 7 (if it is a problem with a multimedia element).

6.8 Step 9: Describing the Course Page with SODA-1L

In this step, we describe the course page in which the simulation is going to be embedded using the higher level language layer SODA-1L. Figure 11 shows an excerpt of the necessary SODA-1L code to describe this page. The listing shows how to use SODA-1L to define the page title (line 7), some textual explanations (lines 8-10), a table with a caption and images (lines 12-15), and how to invoke the previously defined OOCSMP model (line 18). The *MODEL* instruction has some parameters to customize the user interface that C-OOL generates for the simulation. In the example, we are only setting the height and width of the simulation applet (400×400). General options to control the appearance of the user interface of all the simulations in a document can be provided in the SODA-2L script (which may be overwritten for particular models, such as in this case). The last line in Figure 11 is a macro (defined in file "macros.csm," included in line 5) that creates some text and a link to the OOCSMP and SODA source files needed for the page.

6.9 Step 10: Including the Page in the Course

Once we have described the page with SODA-1L, we can insert it in the course. For this purpose, the only thing we have to do is to modify the previous SODA-2L script by adding a reference to the page and recompiling the script. The recompilation updates the pages affected by the insertion and generates the necessary applets for the OOCSMP

```
[1]   INCLUDE  ''Puma260.csm''
[2]   TITLE PUMA260 robot simulation
[3]   PUMA260 p()
[4]   DYNAMIC
[5]     p.ROTATE2 ( 1, PI/2.0 )
[6]     p.ROTATE3 ( 2, -PI/2.0 )
[7]   TIMER FINTIM:= 60, delta := 0.1, PLdelta := 0.1
```

Figure 7. Using a PUMA robot

```
[7]    a1 := p.GETANGLE1
[8]    a2 := p.GETANGLE2
[9]    a3 := p.GETANGLE3
[10]   a4 := p.GETANGLE4
[11]   VRMLworld ''tool.wrl''
[12]   PLOT a1, a2, a3, a4, TIME
[13]   TIMER FINTIM := 60, delta := 0.1, PLdelta := 0.1
```

Figure 8. Adding outputs to Figure 7

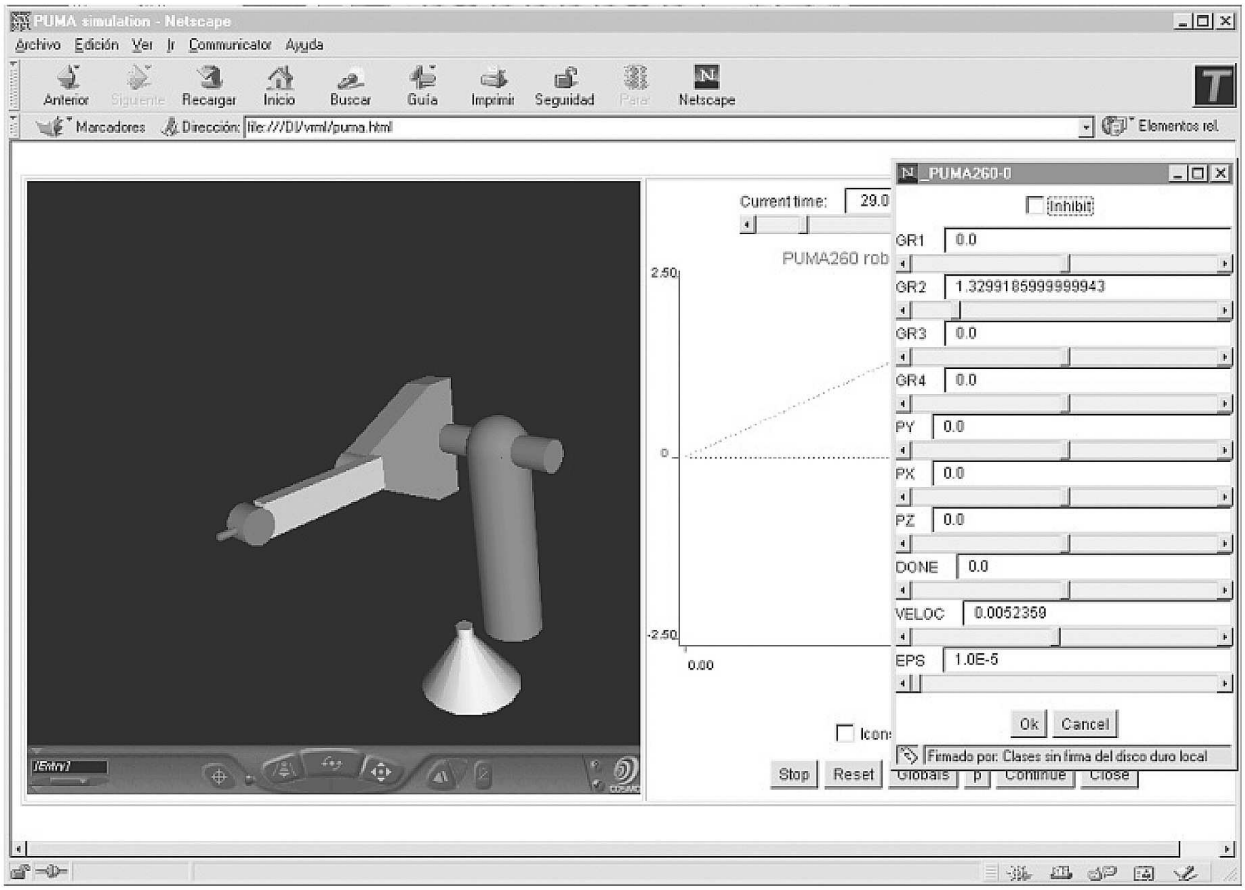

Figure 9. A moment in the execution of the previous model

```
[12]   PLOT [C], a1, a2, a3, a4, TIME
[13]   TEXTPANEL [S], a2 <= PI/2.0, ''explain_2nd_joint.txt'',
[14]   DEFAULT, ''explain_3rd_joint.txt''
[15]   TIMER FINTIM := 60, delta := 0.1, PLdelta := 0.1
```

Figure 10. Adding dynamic textual explanations to the models

```
[1]    * 1st page of the robotics subsection of the applications page
[2]    * AUTHOR Juan de Lara
[3]    * EMAIL Juan.Lara@ii.uam.es
[4]    * DATE 7/6/2002
[5]    INCLUDE ''macros.csm''
[6]    INCLUDE ''styles.csm''
[7]    TITLE An introduction to robotics
[8]    DESCRIPTION The word robot comes from the czech word robota which
[9]    DESCRIPTION means work. The Webster dictionary defines a robot as ''an automatic
[10]   DESCRIPTION device which performs functions usually assigned to human beings''.
[11]   ...
[12]   TABLE [1;2], [C,80],
[13]   ''\IMAGE 'robot22.jpg''',
[14]   ''\IMAGE 'arm1.jpg''',
[15]   ''A PUMA-560 robot, courtesy of MONASH University and Georgia Institute of technology''
[16]   DESCRIPTION A robot is usually composed by a chain of rigid elements connected by joints.
[17]   ...
[18]   MODEL [400;400], [C], ''puma.csm''
[19]   SHOWSODA
```

Figure 11. A scheme of the SODA-1L code for the page

models, thus making the maintenance process easy. An excerpt of the modified SODA-2L script is shown in Figure 12. In this script, we specify the authors' data (lines 2-4), declare that we are building an educational course in opposition to presentations and articles (line 7), select the default options for the compilation of the simulation models (line 8), and give a list of the pages in the course. In this script, we can define some variables (such as *AUTHOR*, *EMAIL*, etc.) that can be used in the pages described with SODA-1L.

When this script is compiled by C-OOL, it generates HTML files for each page of the document using the appropriate formatting styles and compiles the simulations, generating the Java and the VRML files. This recompilation implies that if a new page has been inserted into a course or a presentation, the links in the other pages are updated automatically.

7. Related Research

Hopkins and Fishwick's *rube* [33] is a similar approach to the ideas presented here. The idea of the system is to promote personalization in dynamic model structural representation and bridge the gap between arts and computer science. The authors can provide 3-D *metaphors* onto which simulations can be mapped. These metaphors are specified in VRML. *Rube* is based in the multimodeling tool OOPM [34], which describes a model by connecting components, and each one of them can be described in a different formalism. In OOCSMP, we are currently working on higher modeling layers whereby one can create models described in different formalisms (by using AToM^3 [27]), which can then be mapped onto OOCSMP. With respect to the visualization, our tools also allow a very easy interaction between 2-D and 3-D Java outputs, multimedia, and VRML outputs.

Most current approaches for building educational, interactive materials for the Internet are based on a direct programming in HTML, Java, and VRML. This low-level programming is precisely what we wanted to avoid with the work presented in this paper. As explained in Section 2, the use of an integrated environment for the design of these documents (both simulations and pages) offers many advantages: courses are easier to create, modify, reuse, test, and so on. For example, Roccetti, Salomoni, and Bonfigli [17] proposed an educational environment that can combine simulations written directly in Java and VRML. In a virtual engineering/science laboratory course at Johns Hopkins University (see http://www.jhu.edu/virtlab/virtlab.html), we find a similar approach: some small courses in different subjects are presented. The courses pages contain simulations written directly in Java and HTML, respectively. By

```
[1]   * SODA-2L script for the course on Robotics
[2]   AUTHOR Juan de Lara, Manuel Alfonseca
[3]   EMAIL Juan.Lara@ii.uam.es, Manuel.Alfonseca@ii.uam.es
[4]   WEBADDRESS http://www.ii.uam.es/~jlara, http://www.ii.uam.es/~alfonsec
[5]   INCLUDE ''macros.csm''
[6]   INCLUDE ''styles.csm''
[7]   COURSE  ''A course on Robotics'' BACKGROUND =''WHITE''
[8]   SIMULATIONS  --noFrame --noScaleWindow --noLeyenda --WIDTH= 500 --HEIGHT= 400
[9]   PAGE ''index.csm''
[10]  PAGE ''intro.csm''
[11]  ...
[12]  PAGE ''puma_page1.csm''
[13]  ...
```

Figure 12. An excerpt of the SODA-2L script

coding the simulations directly in Java, one may use output forms more adapted to the problem than by using a predefined output form provided by a high-level language. The disadvantage is that coding the simulation and the output forms directly in Java requires more effort.

Other approaches rely on the use of proprietary technologies to describe and execute the simulations. In Budhu [35], a course on civil engineering containing interactive multimedia simulations is presented. The course is based on Authorware and Flash (see http://www.macromedia.com). These are very powerful tools for graphical design, in which the user can design films and animate objects using the ActionScript language (a language in the style of Java). Our aim is not to compete with these well-established tools in which the focus is on general graphical design. In our system, the focus is on providing powerful constructs for simulation (blocks to solve PDEs, ODEs, etc.) and to complement these with visualization and multimedia facilities. The graphical capabilities of Authorware and Flash are much higher, but if one tries to build an advanced simulation course with these tools, all the numerical methods to solve PDEs (finite element methods, finite difference methods, mesh generation techniques, etc.) and ODEs (Runge-Kutta, Adams, Simpson, etc.) would have to be coded by hand using ActionScript. We provide these libraries, which can be accessed using OOCSMP constructs in a declarative way. Moreover, if the user really needs efficiency in the simulations (rather than building an educational course), the C-OOL compiler can be used to generate C++ code (instead of Java) as we provide a C++ version of all the numerical libraries.

In Schmid [2], in the context of the DynaMit project (http://dynamit.esr.ruhr-uni-bochum.de), a Web-based learning framework was built that allows building tutorials, exercises, and virtual experiments. The system relies on plug-ins such as MATLAB/SIMULINK/MAPLE, VRML, Graphics, and the ToolBook Neuron. Our aim was to develop a system in which models are easily integrated with the document pages, allowing for the easy maintenance of the documents and the reuse of models and pages. A side effect of using a compiler able to generate Java code for the simulations is that the simulation user does not have to download and install a plethora of different plug-ins to execute the simulations.

With respect to the robotic applications, the Department of Electrical Engineering and Information Technology of the University of Hagen has developed several (very impressive) VRML simulations of robots and has included them in a course on robotics (see http://prt.fernuni-hagen.de/pro/richodl/richodl.html and http://www.geocities.com/ResearchTriangle/Lab/8585/robot/robot.html). These simulations have been coded directly in VRML and Java. We provide a high-level simulation language, combined with other layers, to describe the course pages, with no need to program in Java and HTML. By building these simulations directly in Java and VRML, the level of customization is greater, but the effort required is bigger.

8. Conclusions and Future Work

In this paper, we have presented a simulation system that, starting from a high-level description of the models, is able to generate programs executable from the Internet. The programs can combine different output forms, including multimedia elements, VRML panels, and other Java-based outputs. The system makes easy the integration of such simulations in Web documents such as courses, articles, or presentations. It has been used mainly for educational purposes. Some simulations generated with it have been used in a university course that is being taught through the Internet. OOCSMP is also being used as a modeling and simulation tool in a predoctoral course on simulation taught at the Universidad Autónoma de Madrid.

As an example of the use of VRML panels, the simulation of a PUMA260 robot has been presented. The use of VRML makes the simulations more realistic, being a good complement to the laboratory classes. In fact, sometimes it can be the only possible alternative because experiments with the real system would be expensive (such as in the case of the robot) or impossible (such as in the case of the solar system). A procedure to build documents for the Web containing simulations has been followed to

embed the PUMA260 simulation in an existing educational course. The system and the procedure improve productivity and emphasize important points in Web engineering, such as maintainability, easy testing, and standardization of the user interface. By using the system, the course designer avoids having to program in low-level languages such as Java, HTML, or VRML, thus reducing drastically the developing time and the effort to integrate the applets with the VRML models.

We are currently working on an OOCSMP interpreter [18] that would allow the student to change dynamically the simulation model during execution. In our example, this means that the student would be able to plan the robot actions. In the future, we want to work on a planning system for the robot, which would be able to generate OOCSMP code. The simulation could also be improved by adding sensors to the robot to detect and manipulate physical objects or to make it more realistic, bearing in mind forces, accelerations, and so forth.

Currently, it is possible to add new objects during the simulation execution [23]. For example, in the case of the simulation of the inner solar system, it would be possible to add a new planet at runtime. But this is not possible if we are using a VRML panel. The system could be extended to allow this (the EAI interface permits creating new VRML nodes from strings, for example). It would also be interesting to extend OOCSMP to define actions to happen when the user manipulates the simulation objects in the VRML world. This would require linking the simulation variables from the VRML world to the simulation engine. Currently, they are linked in one direction only, from the simulation engine to the VRML world.

A further step would be the extension of our tool to build virtual laboratories in which the students can make experiments and appear as avatars. Ideally, this environment should permit interaction and communication between students in a way similar to cooperative learning environments. Among other things, this would require immersing all the simulation controls inside the VRML world, even those at present in the Java panels. In this direction, we are currently working on integrating our system with the FACT framework [36], which allows the construction of collaborative applications, analysis of the learning process, and collaborative tutoring.

Together with the Modeling Simulation and Design Lab at McGill University, we are working on a graphical environment (called AToM^3) for the construction of the simulations and Web documents. The approach we are taking is using meta-modeling [27] to define the formalisms we are interested in and, from this meta-information, generating a tool to process that formalism. We are also planning to incorporate the possibility of designing VRML worlds or at least allow an easy interaction with some VRML development tool. Additional examples and courses generated with our system can be found at the OOSMP homepage (http://www.ii.uam.es/~jlara/investigacion).

9. Acknowledgment

We thank all the anonymous referees and Jean-Sebastien Bolduc for their accurate and constructive comments that greatly improved the paper. This work has been sponsored by the Spanish Ministry of Science and Technology (MCYT), project number TIC2002-01948.

10. References

[1] Page, E. H., A. Buss, P. A. Fishwick, K. Healy, R. E. Nance, and R. J. Paul. 2000. "Web-based simulation: Revolution or evolution? *ACM Transactions on Modeling and Computer Simulation* 10 (1): 3-17.

[2] Schmid, Ch. 1999. A remote laboratory using virtual reality on the Web. *SIMULATION* 73:13-21.

[3] de Lara, J., and M. Alfonseca. 2000. Using simulation and virtual reality for distance education. *SIIE'2000*, November, Puertollano, Spain.

[4] Scholz-Reiter, B., W. Echelmeyer, T. Hamann, and J. Hoheisel. 2002. Games for engineering education. *Proceedings of 16th European Simulation Multiconference ESM'2002*, Darmstadt, Germany, pp. 391-4.

[5] Bryson, S. 1996. Virtual reality in scientific visualization. *Communications of the ACM* 39 (5): 62-71.

[6] Bredeweg, B., and R. Winkels. 1998. Qualitative models in interactive learning environments: An introduction. *Interactive Learning Environments* 5:1-18.

[7] de Jong, T., ed. 1991. Computer simulations in an instructional context [Special issue]. *Education and Computing*, Vol. 6. Amsterdam: Elsevier Science.

[8] Schank, R. C., and C. Cleary. 1995. *Engines for education*. Hillsdale, NJ: Lawrence Erlbaum.

[9] de Lara, J., and M. Alfonseca. 2001. Constructing simulation-based Web documents. *IEEE Multimedia*, January-March, 42-9.

[10] Lorenz, P., H. Dorwarth, K. C. Ritter, and T. J. Schriber. 1997. Towards a Web-based simulation Environment. *Proceedings of the 1997 Winter Simulation Conference, Society for Computer Simulation*, pp. 1338-44.

[11] Fishwick, P. A. 1996. Web-based simulation: Some personal observations. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, pp. 772-9.

[12] Alfonseca, M., J. de Lara, and H. Vangheluwe. 2001. Web-based simulation of systems described by partial differential equations. *Proceedings of the Winter Simulation Conference*, Arlington, VA, pp. 629-36.

[13] Bajaj, C., and S. Cutchin. 1997. Web based collaboration aware synthetic environments. *Proceedings of the TeamCAD Gvu/Nist Workshop on Collaborative Design*, pp. 143-50. See also SHASTRA Web page at http://www.ticam.utexas.edu/CCV/projects/shastra

[14] Bidarra, R., E. Van Den Berg, and W. F. Bronsvoort. 2001. Interactive facilities for collaborative feature modeling on the Web. *Proceedings of the 10th Portuguese Conference on Computer Graphics*, Lisbon, Portugal. See also http://www.cg.its.tudelft.nl/~eelco/publications.html

[15] Kuljis, J., and R. J. Paul. 2001. An appraisal of Web-based simulation: Whither we wander? *Simulation Practice and Theory* 9:37-54.

[16] Healy, K. J., and R. A. Kilgore. 1997. Silk: A Java-based process simulation language. *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, pp. 475-82.

[17] Roccetti, M., P. Salomoni, and M. E. Bonfigli. 2001. A design for a simulation-based multimedia learning environment. *SIMULATION* 76 (4): 214-21.

[18] Alfonseca, M., J. de Lara, and G. Montoro. 2002. Teaching partial-differential equations through the Internet: An interactive

approach. *Proceedings of the 16th European Simulation Multiconference, ESM'2002*, Darmstadt, Germany, pp. 395-9.
[19] IBM Corp. 1972. *Continuous System Modelling Program III (CSMP III) and Graphic Feature (CSMP III Graphic Feature) general information manual*. Ontario, Canada: IBM.
[20] Alfonseca, M., E. Pulido, R. Orosco, and J. de Lara. 1997. OOCSMP: An object-oriented simulation language. *ESS'97*, Passau, pp. 44-8.
[21] Alfonseca, M., and J. de Lara. 2002. Simulating evolutionary agent colonies with OOCSMP. *Proceedings of the 17th ACM Symposium on Applied Computing (SAC'2002), AI and Computational Logic*, March, Madrid, Spain, pp. 11-5.
[22] Myers, B. A., E. Borison, A. Ferrency, R. McDaniel, R. C. Miller, A. Faulring, B. D. Kyle, P. Doane, A. Mickish, and A. Klimovitski. 1997. The Amulet v3.0 reference manual. Technical Report No. CMU-CS-95-166-R2 and Human Computer Interaction Institute Technical Report No. CMU-HCII-95-102-R2, Carnegie Mellon University School of Computer Science.
[23] Alfonseca, M., J. de Lara, and E. Pulido. 1999. Dynamical object generation during the execution of continuous simulation models. *Argentine Symposium on Object Orientation 1999 (ASOO 99)*, September, Buenos Aires, Argentina.
[24] Alfonseca, M., J. de Lara, and E. Pulido. 1999. Semiautomatic generation of Web courses by means of an object-oriented simulation language. *SIMULATION* 73:5-12.
[25] Leonardo, L. 1997. *Using Netscape LiveConnect: Special edition*. Indianapolis, IN: Que Corporation. See also Netscape ftp site: ftp.netscape.com in pub/sdk/plugin/windows/oct_21_97
[26] Hartman, J., and J. Wernecke. 1996. *The VRML 2.0 handbook: Building moving worlds on the Web*. Reading, MA: Addison-Wesley.
[27] de Lara, J., and H. Vangheluwe. 2002. AToM^3: A tool for multiformalism modelling and meta-modelling. *Lecture Notes in Computer Science* 2306:174-88. See the AToM^3 homepage: http://atom3.cs.mcgill.ca/
[28] de Lara, J., H. Vangheluwe, and M. Alfonseca. 2002. Using metamodelling and graph grammars to create modelling environments. *Electronic Notes in Theoretical Computer Science* 72 (3). Also presented in the Graph Transformations and Visual Modelling Techniques (GT-VMT) workshop at the 1st International Conference on Graph Transformations, October, Barcelona, Spain. Available: http://www.elsevier .nl/locate/entcs/volume72.html
[29] Fu, K. S., R. C. Gonzalez, and C. S. G. Lee. 1987. *Robotics: Control, sensing, vision, and intelligence*. New York: McGraw-Hill.
[30] Booch, G., J. Rumbaugh, and I. Jacobson. 1999. *The unified modelling language user guide*. Reading, MA: Addison-Wesley. See also the UML homepage at http://www.uml.org
[31] Craig, J. J. 1989. *Introduction to robotics*. 2d ed. Reading, MA: Addison-Wesley.
[32] Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. 1993. *Computer graphics principles and practice*. 2d ed. Reading, MA: Addison-Wesley.
[33] Hopkins, J. F., and P. Fishwick. 2002. The *rube*^{TM} methodology for 3-D software engineering. *Lecture Notes in Computer Science* 2269:368-80.
[34] Lee, K., and P. Fishwick. 1999. OOPM/RT: A multimodeling methodology for real-time simulation. *ACM Transactions on Modeling and Computer Simulation* 9 (2): 141-70.
[35] Budhu, M. 2001. Enhancing instructions using interactive multimedia simulations. *SIMULATION* 76 (4): 222-31.
[36] Mora, M. A., and R. Moriyón. 2001. Collaborative analysis and tutoring: The FACT framework. *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'01)*, Madison, WI, pp. 82-5.

Juan de Lara *is an assistant professor in the Computer Science Department of the Escuela Politécnica Superior (Higher Polytechnical School) at the Universidad Autónoma de Madrid, where he teaches software engineering. He received his PhD in computer science from this university in 2000 with a thesis on Web-based simulation, which received a special award for the best thesis presented at the school. He also holds a technical engineering degree in computer science (1994, top of the class award) and an engineering degree in computer science (1996). He has published more than 50 technical papers in areas such as Web-based simulation, agent-based simulation, software verification, and multiparadigm modelling. In the latter area, he collaborates with the Modelling Simulation and Design Lab (headed by Professor Hans Vangheluwe) at McGill University, where he spent one year doing postdoctoral research.*

Manuel Alfonseca *holds a doctorate in electronics engineering (1972) and computer science (1976), with both degrees obtained from the Universidad Politécnica of Madrid. He teaches and does research at the Department of Computer Science of the Universidad Autónoma of Madrid, where he is director of the Higher Polytechnical School. Previously, he was a senior technical staff member at the IBM Madrid Scientific Center, where he worked from 1972 to 1994. He is a member of the SCS, the New York Academy of Sciences, the IEEE Computer Society, the ACM, the British APL Association, the IBM TEC, and the Spanish Association of Scientific Journalism. He has published papers and books on computer languages, simulation, complex systems, graphics, artificial intelligence, object orientation, and theoretical computer science, as well as popular science and juvenile literature.*