USING DIS FOR LINKING SIMULATION AND ANIMATION IN SIMULINK AND VRML

Ole Ravn and Nils Andersen

Department of Automation, Technical University of Denmark,
Building 326, DK-2800 Lyngby, Denmark,
E-Mail: or@iau.dtu.dk

Abstract: The paper describes the techniques enabling the control system designer to easily and seamlessly integrate visualization into simulations studies. The visualization module has been evaluated by doing a prototype implementation of vizualising the docking of an Autonomous Underwater Vehicle to a docking station. The paper also explains the underlaying simulation centered architecture that utilizes the IEEE DIS standard (Distributed Interactive Simulation). *Copyright ©2000 IFAC*

Keywords: visualization, simulation, modelling.

1. INTRODUCTION

Visualization of scientific data has been an active research area for a number of years and much progress have been made in the area of for instance visualization of large data set and visualization of mechanical system. 3D visualization of scientific data is very helpful in the understanding of complex measurements.

In the area of control engineering visualization of simulation results has been mainly time plots. 3D visualization has been of less importance in this area but visualization models has opened a new potentially beneficial area using established techniques from civil and mechanical engineering as well as architecture.

The area of animation is not yet as advanced as the visualization of static data, but using CAD systems and with the introduction of standards, more powerful computers and commonly available tool for building virtual worlds. Much effort has been put into areas like multi-user virtual worlds for games, simulators for ships, planes, battlefields etc and virtual manufacturing. Substantial progress again partly driven by the raising power of desktop computing has been made in the area.

Some work has been done in the area of visualization and animation in connection with control engineering applications, both simple 2D animations and more advanced 3D animations. However much effort has to be invested in the model building and in the non standard integration into the simulation environment. There is a large potential for using animations for better understanding the performance of a system especially for non control engineers where a simple visualization combined with the more traditional time plot often will give a more immediate impression. Our aim has been to make an easy to use yet powerful and versatile concept and toolset for integrating 3D animations in a standardized way in a well known simulation environment but with a modest added design effort. Our goal has been that only a little extra effort should be needed compared to the effort of creating the simulation model. This would make it possible to build animation model as a standard part of simulation models enhancing the understandability of their results.

Ease of use and the integration into well-known simulation environments has also been major concerns.

The use of standard tools and protocols like DIS (Brutzman, 1999), java and VRML (Carey and Bell, 1997) helps to make the concept and toolset more hardware platform independent while still utilizing the existing tools.

In section 2 the underlying DIS (Distributed Interactive Siimulation) protocol is described with special focus on the area of visualization and animation. This forms the basis for section 3 giving a detailed description of the concept, methodology and the prototype implementation of the toolset. Section 4 describes the application of the toolset for visualising the docking of an AUV (Autonomous Underwater Vehicle and section 5 gives conclusions.

2. DISTRIBUTED INTERACTIVE SIMULATION (DIS)

Up through the nineties, the task of simulating complex scenarios has received increasing interest. Especially the US Department of Defence has been an important player in this field, triggered by an interest to run large-scale battle scene simulations involving large numbers of entities distributed over the Internet. This work led in the mid-eighties to the SIMNET system, which ten years later was replaced by the more advanced DIS standard, see (Brutzman, 1999). The standard describes how a simulation is distributed and managed over a network of computers, defining protocols for communication and standards for the appearance and behaviour of simulated entities. Although DIS started as a military standard, its nice characteristics quickly attracted software developers from the computer game industry as well as academic researchers in mobile robotics. Some of the key features of DIS that contributed to its popularity are described below.

2.1. Distribution

No central computer controls the entire simulation exercise. Instead DIS uses a distributed simulation approach in which the responsibility for simulating the state of each entity rests with separate simulation applications residing in host computers connected via a network. As new host computers are added to the network, each computer brings its own resources. A simulation can therefore be extended without introducing the problems or bottlenecks a central computer inevitably causes.

A so-called *simulation manager* does provide some central control of the simulation, however. The simulation manager handles functions such as start, pause, restart, maintenance, shutdown and collection and distribution of certain types of data.

2.2. Interaction

Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities. If a user operates some simulated or actual equipment, the application is responsible for modelling the resulting actions of the entity using a simulation model. The application is responsible for sending messages to others to inform them of any observable actions. Likewise, it is responsible for interpreting and responding to messages of interest from other simulation applications and maintaining a model of the state of entities represented in the simulation exercise.

The concept of simulation applications controlling specified subsets of the simulation, makes the simulation exercise quite modular. Replacing real and simulated hardware is easy and has little or no impact on the remaining parts of the simulation. Also, adding or replacing user interfaces to specific entities becomes quite painless.

2.3. Communication Protocol

A standard protocol is used for communicating entity state data. Each simulation application communicates the state of the entity it controls/measures (location, orientation, velocity, articulated parts position, etc.) to other simulations on the network. The receiving simulation application displays relevant parts of these data to the user as required by the individual simulation.

The protocol defines the exchange of data through packages known as protocol data units (PDUs). A total of 37 PDUs are defined in the standard, describing entity information, simulation management, logistics, radio communications and military specific functions such as weapon launching and electromagnetic emission. Definitions of these PDUs and the data types used in these, can be browsed on-line (DIS).

2.4. Dead Reckoning

Dead reckoning algorithms are used to reduce communications processing. A method of position/orientation estimation, called dead reckoning, is used to limit the rate at which simulation applications must issue updates for an entity. Each application maintains a dead reckoning model of its entity. The dead reckoning model represents the view of that entity by other simulations on the network and is an extrapolation of its position and orientation using a specified dead reckoning algorithm. On a regular basis, the simulation application compares the internal model of its entity to the dead reckoning model. If the difference between the two exceeds a predetermined threshold, the simulation application will update the dead reckoning model using the information from the internal model. At the same time, updated information is sent to other simulation applications on the network so that they can update their dead reckoning model of the entity. By using dead reckoning, simulations are not required to report the status of their entities as often.

3. THE METHODOLOGY AND TOOLKIT

The proposed methodology consists of two distinct phases: The model development phase and the model/animation run-time phase. In the first of the two phases the simulation model is build and tested. In the prototype implementation this is should be done in Simulink, but basically the ideas should work with any modelling/simulation system. Furthermore the animation model is build and tested this could also be done in several environments, in this case the Cosmo Worlds system is used producing a VRML 2.0 description of the system and scene. The specification of linking between simulation variables and movable parts in the VRML world is now done using DIS.

The second phase could now be entered by starting the simulation and specifying the viewing parameters to the VRML browser. Note that the data for the animation/visualization could come from either the simulator or the real-world experiment.

A third phase could be introduced in order to separate the simulation and visualization, this would be helpful when considering large and complex models where is simulation would run considerably slower than real time. In this case the simulation is performed off line and the animation is then performed separately. This will make user interaction with the simulation through the animation browser difficult if not impossible. This kind of user interaction not just with the animation parameters like lighting and camera position but also with the simulated system is one of the very appealing features of the whole concept. However when considering long time simulation or large data sets from i.e. sea trails with an Autonomous Underwater Vehicle it could be useful to have some kind of fast forward or reverse button to be able to focus attention to the interesting parts of the animation. Figure 1 shows the different phases and their relations. As shown it could even be considered integrating the modelling phase with the two others allowing the user to add parts to or modify the simulated system on-line. This is outside the scope of this paper.

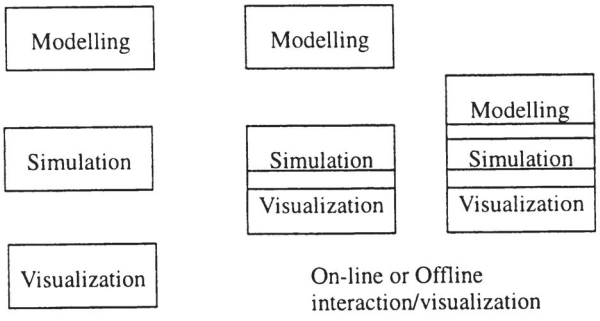

Fig. 1. Phases in the use of visualization/simulation framework

A automatic or user adjustable variable level of detail in the simulation and animation could be useful when moving through large sets of data or simulating a long period of time faster than real time. Both the time resolution, the complexity of the simulated model in the simulation as well as the spacial resolution in the animation are parameters to consider when designing this kind of system.

The issue of seamless integration with hardware in the loop simulation and real system components mentioned earlier is also important. One view of the situation is shown in figure 2.

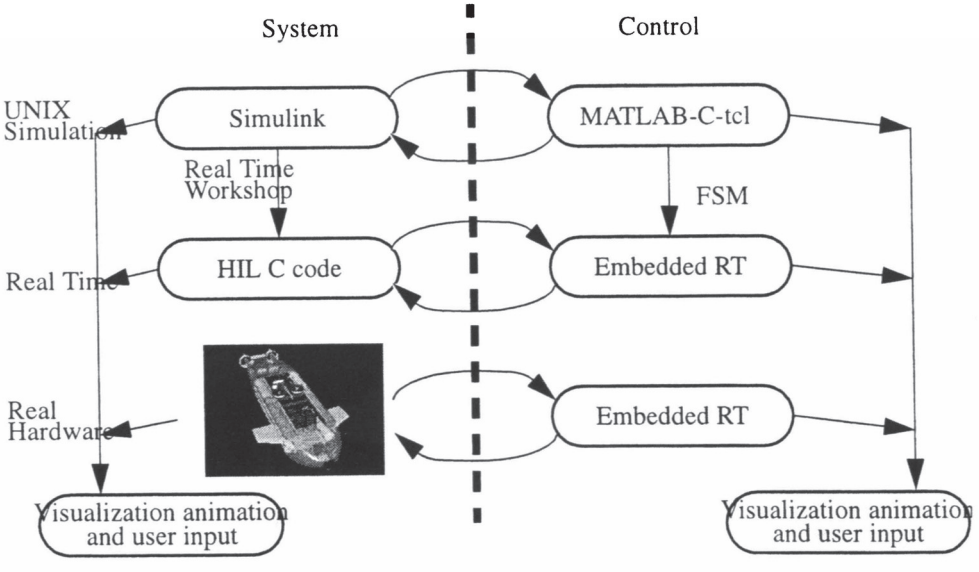

Fig. 2. Hardware-in-the-loop simulation.

Three levels are distinguished: an off line simulation level were the system is simulated in for instance Simulink and the control algorithm is implemented using MATLAB/C/tcl or any other suitable language. On the second level called the real time level the system is moved to a HIL (Hardware in the loop) platform where the code is generated by RTW (Real Time Workshop) and the controller is prototyped and running on a suitable real time platform. The third level is called the real hardware level and includes the real system hardware and the real system controller. The levels could be mixed in different ways in the different phases of the controller design.

It is important to note that the visualisation/animation and input actions are split in two distinct parts in order to make the transition between different levels simple. On one side the visualization of the system which is the one being considered in this paper and on the other hand the similar block of the controller. The strict separation into system and controller is equally important for the easy transition between levels.

To be able to evaluate the concept and methodology outlined above a prototype implementation has been made using the components shown in figure 5. The toolset is called VRML animation toolset for Simulink and uses java, DIS and TCP/IP sockets for communication between Simulink and the VRML model. The use of Simulink enables the use of Real Time Workshop to make the translation from simulation to real-time code.

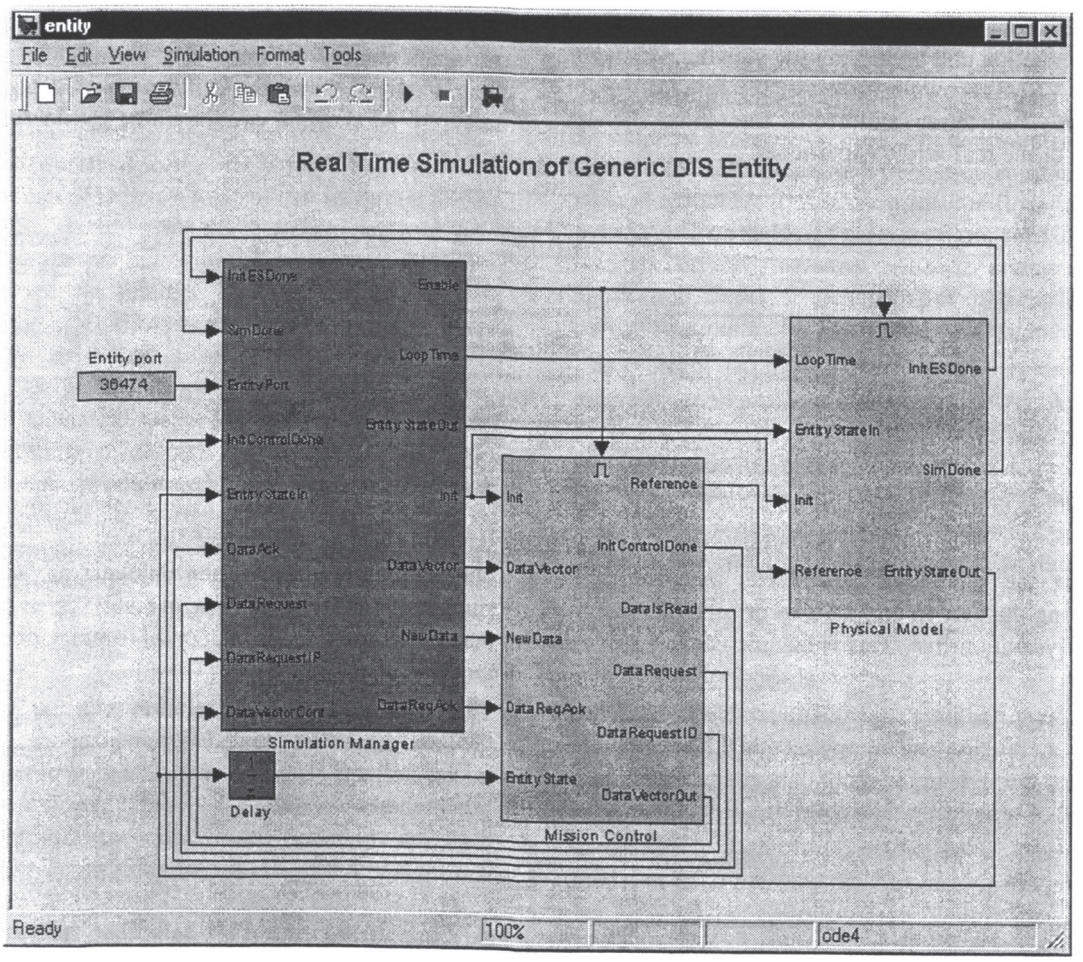

Fig. 3. Simulink model of an AUV with the VRML animation block.

4. THE DOCKING SIMULATION

The implementation of the docking simulator was greatly inspired by the DIS standard and does adhere to this almost completely. A central user interface developed in Java, acts as the simulation manager, permitting the user to control the flow of the simulation by the use of start, stop and pause buttons. Further, the user is allowed to control which entities should take part in the simulation by defining IP and port numbers for all entities and specifying which geometrical models (VRML files) to use. The user interface currently allows five different entities to join the simulation but can easily be enhanced to handle more.

Three different simulation applications have been defined. An AUV entity, which is either a simple zero-order model of an underwater vehicle or an accurate model of the submarine Martin. A DS entity that simulates a docking station and a GDS entity, which is a visualisation of the estimate of the position of the docking station, held by the AUV. The final element in the simulation is the visualiser, which also communicates by exchanging DIS packages with the user interface. An overview of the components is shown in Figure 5.

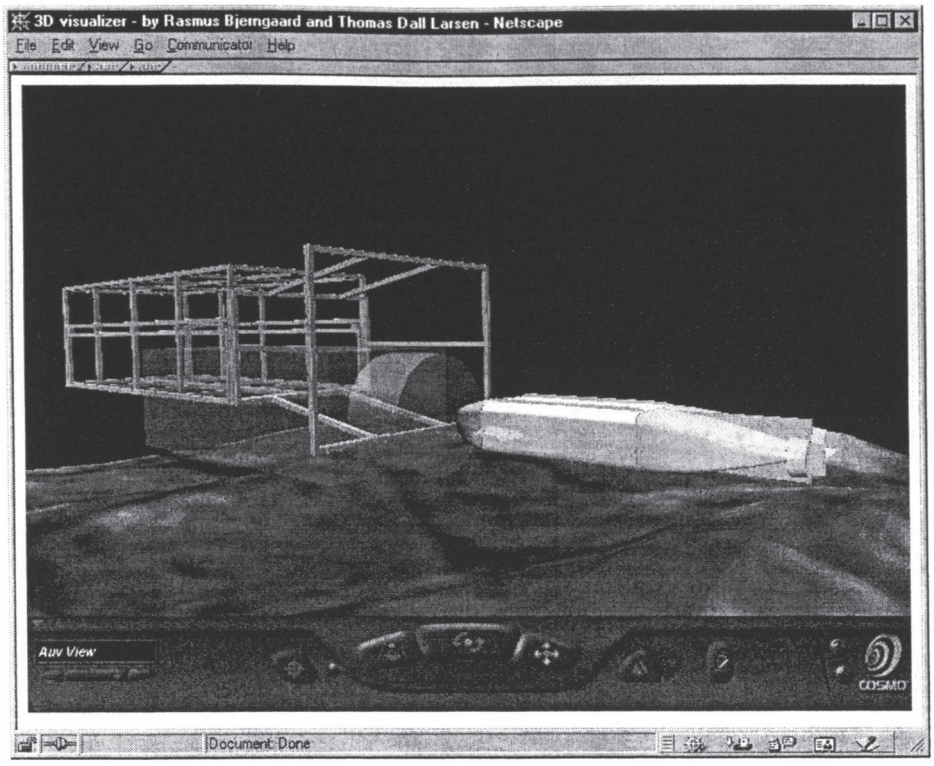

Fig. 4. Screen shot of the visualization of the AUV. The 'ghost' docking station show the estimated location of the docking station by the AUV.

4.1. Accelerated time

The DIS standard requires that all entities in a simulation run at real time. In order for a distributed simulation to make sense, it is of course necessary that time evolves consistently for all involved simulation applications and that all applications respond in a timely manner. There is, however, in general no need for simulated time to be the same as the real world time. In some situations it can be useful to accelerate the simulation, for instance to speed up uninteresting parts of the simulation. In other situations, very complex and computationally demanding models can be used, making it necessary to run the simulation slower than real time. Therefore, it is chosen to augment the DIS standard with an acceleration factor indicating the speed of the simulation. By choosing this to equal one, a real time DIS simulation is run. Choosing a different value causes the (non-DIS) simulation to run exactly as a DIS simulation but at a different speed.

4.2. Elements and tools

The ideas described above contains much flexibility with respect to choosing the actual parts of the implementation. This is intentional as the emergence and demise of specific software products and the change through new version is bound to have a strong impact on the actual implementation. Some generic components of a meta toolset has been distinguished and the elements needed are mapped onto the current available products to make up the prototype implementation used.

The elements are

- VRML standard specification language in version 2.0 facilitates animation of dynamic worlds and both browsers and modellers are available.
- Cosmo Worlds for modelling the virtual world in native VRML.
- Cosmo Player for viewing the world in the animation phase.
- DIS as the 'glue' communication layer based on sockets for communication between simulation and animation. Adds flexibility as the simulation and animation could be on separate hosts. The communication across the sockets are pose data.
- Java for scripting the interface from the VRML World to the sockets.
- Simulink for simulation using C s-functions for socket communication. This makes it possible to use RTW to automatically generate real-time code for the prototype controller.

The specific elements of the prototype implementation are not vital for the concept, other elements could used if needed. However the selected elements gives optimal flexibility and minimize the amount of specialized code needed.

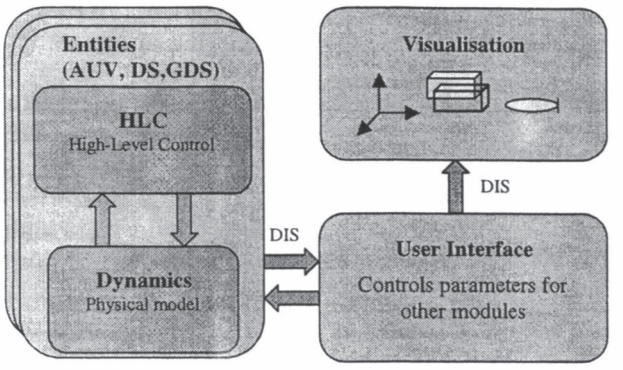

Fig. 5. Elements in a docking simulation.

5. CONCLUSION

In this paper the ideas behind and the design considerations for the VRML animation toolset for Simulink presented. The need for animation in control engineering is emphasized and that this should be at a reasonable cost compared to the rest of the modelling effort. The prototype implementation is presented and the each element is discussed. The extensive use of standard readily available software products such as Simulink, Cosmo Player, Cosmo World, DIS and java as glue code. The use of sockets for communication adds flexibility to the solution by enable multi computer systems and widening the number of possible platforms (UNIX, Windows, OS-9 etc.) for executing the system.

ACKNOWLEDGEMENTS

The authors gratefully acknowledges the support of EU Euro-Docker contact no. MAS3-CT97-0084.

REFERENCES

Brutzman, D. (1999). Home page of distributed interactive simulation dis-java-vrml working group. http://www.stl.nps.navy.mil/dis-java-vrml/AnnotatedReferences.html#DIS.

Carey, R. and Bell, G. (1997). *The Annotated VRML97 Reference Manual.*

The DIS Data Dictionary: http://SISO.sc.ist.ucf.edu/dis/dis-dd/pdu/index.htm.

MATLAB (1998). *Using MATLAB Graphics*. The Mathworks Inc,.

McLure, D. P. (1998). Client/server programming with VRML.http://www.webresource.net/vrml/articles/ScriptSample/vrmlScriptArticle.html.

Schmid, C. (1999). Virtual control laboratory. http://www.esr.ruhr-uni-bochum.de/VCLab.

TWM (1995). Simulink animation blockset make development easier, an it's free. *MATLAB News and notes.*