An Architecture for Hosting Management Flight Simulators on the World Wide Web

Magne Myrtveit, Powersim AS
David Bridgeland, Powersim Corporation

The need for simulation software installation can be a barrier to the usage of a management flight simulator. MFSes are often used once or twice each by a large number of people. Installing special software on each individual's PC can be impractical. Furthermore, MFSes are convenient to support just-in-time learning. Requiring software installation between the need and the satisfaction of that need is a barrier to just-in-time.

The world wide web is a natural platform for hosting an MFS. A user can point a standard web browser—like Netscape Navigator or Microsoft IE—to a URL that contains a management flight simulator. No special software is needed. The user interface for the simulator is downloaded transparently. Once downloaded, the MFS user interface communicates with a simulation model running on a server.

Hosting management flight simulators on the web also provides a way of mainstreaming system dynamics in the business community, removing SD from the ghetto of learning labs, and placing it as part of a standard business environment, the same environment that today hosts customer databases, human resource policies, technical support information, and other information.

However, hosting a reliable MFS on the web poses several technical problems. The most basic requirement is that communication between the client and server must be supported, with information about decisions sent from the clients to the server, and information about the state of the model sent from the server to the client. The latter should include support for real time user interface controls driven by datastreams from the simulation model, allowing animations at the client side.

Furthermore, this communication should be transparent on the client side. Building a user interface for an MFS is challenging enough without having to write network communication software. The client software should interact with local proxy classes that hide the details of the communication.

Multi-user games are becoming increasingly important, to simulate different functions in an organization, different positions in a supply chain, or different competitors. The web is a good place to host a multi-user game, because it naturally supports multiple users in different locations. But multi-user games puts additional requirements on the simulation server software. In particular, there must be support for the process of choosing a role in a game, for joining a game in progress, and for leaving a game. In addition the different users may be connected to the server with different bandwidths and latencies. The server software must be able to distinguish between a client who has left an active simulation, and a client who has slower responses because she is an ocean away.

There are several different methods of handling time in a multi-user game. For example, one alternative is to have time advance while decisions are made, without anyone able to pause the simulation. Another alternative is to have regular cyclic pauses for decision-making, for example every year or every quarter. A third alternative is to allow any user to pause the simulation, with that user responsible for resuming it when done. The simulation server software needs to support user interfaces built with any of these alternatives.

The biggest constraint on simulation server software is that it must manage a potentially large number of simultaneous simulations. It is not uncommon for 100 or 1000 individuals to visit the same site on the web at the same time. Managing a scale of 100 or 1000 simultaneous simulations is a difficult challenge.

Inside a websim

Figure 1 shows the architecture of a typical "websim". Some of this architecture should be familiar: the client machine has a web browser like Netscape Navigator, and the server machine has a http server like Microsoft IIS. The web browser and html server communicate over the internet via a stream of URL requests from the client to the server, and http responses from the server to the client.

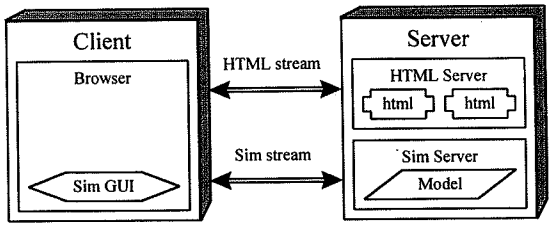

Figure 1: Architecture of a Websim

The model (shown in Figure 1 inside the Sim Server) represents the structure of the situation being simulated, a standard SD model. The Sim Server itself allows that model to be controlled: to be played, for parameters to be examined, and for parameters to be changed.

The sim GUI is a user interface to a simulation. In addition to handling all interaction with the user, the sim GUI does all other application tasks that are not part of controlling a simulation model. For example, if a local database is required, that will be part of the sim GUI. The sim GUI is written as executable content, typically in Java, but possibly in another medium like ActiveX.

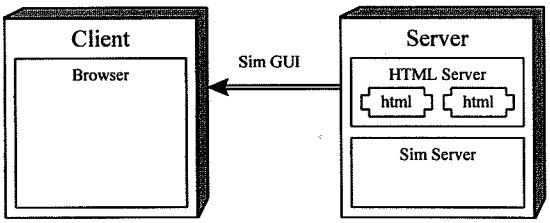

Figure 2: Invoking a Websim, the First Step

The communication between the sim GUI and the sim server takes place over the internet or intranet/extranet (of course), but it is convenient to talk about it as if it happened over a private communication channel, a "simulation stream". The simulation stream handles requests and control information from the client to the server, and provides responses and

asynchronous events from the server to the client. Each client has its own simulation stream.

The first step in invoking a websim is for the user to browse to a page that includes the Sim GUI. The html server then automatically downloads the Sim GUI executable content, as shown in Figure 2. If the right content is already available locally (e.g. is cached from previously), no download is necessary.

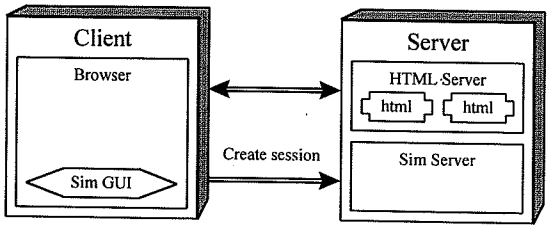

Figure 3: Loading a Model

Next the Sim GUI is executed. One of the first actions taken by the GUI is to open a simulation stream with the sim server, as shown in Figure 3, and ask the server to load and initialize the model—"creating a session" in the language of the sim server.

The result of that action is a websim that is ready to run, with all pieces in place. Subsequent commands are interpreted by the sim server. Notification of asynchronous events are also provided by the server to the sim GUI.

Building and buying pieces of a websim

Of course not everything shown in the preceding diagrams must be built by the websim creator. The simulation server is a standard application that is common to all websims. Powersim Metro Server is a sim server application that

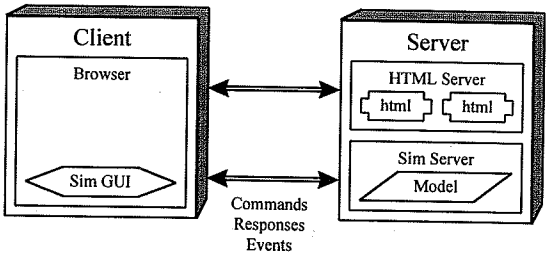

Figure 4: Websim ready to simulate

handles multiple clients and multiple models, as many as 200 simultaneous clients in our most recent tests.

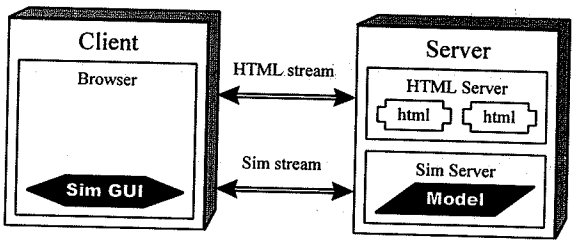

Figure 5: Two Pieces of the Websim that Must be Built

The Sim GUI middleware is also provided with Metro Server. The middleware hides all the communication details from the rest of the GUI. The rest of the GUI sees a set of local Java classes that make it seem as if the simulation is running locally. This frees the application programmer from the details of the communication and allows her to focus on how things should look to the user and what should happen to the model. Figure 5 shows the two pieces that must be provided: the model and that part of the Sim GUI that is not middleware.

Related work

The definition of the Model Interchange Format (MIF) (Myrtveit 1995) opens up the potential of using any SD modeling tool for creating models that can be run on the server side of a websim.

The technology described in this paper extends the local area network simulation technology (Davidsen and Myrtveit 1994) without introducing any new limitations—websims can be run over a LAN as well as a WAN.

Finally, this work can be seen as a extension of some pioneering websim work (Groessler 1996) that used CGI as middleware to a sim on a server.

References

Myrtveit 1995: *Models crossing the boundaries of tools*. Proceedings ISDC '95, Tokyo, Japan

Davidsen, Myrtveit 1994: *Der RÜTLI management simulator*. Proceedings ISDC '94, Stirling, Scotland

Groessler 1996: *Providing Simulation Models on the Internet*, Proceedings ISDC '96, Boston, USA