Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

AN ARCHITECTURE FOR DISTRIBUTED SIMULATION GAMES

Stijn-Pieter A. van Houten
Peter H.M. Jacobs

Section of Systems Engineering
Delft University of Technology
Delft, THE NETHERLANDS

ABSTRACT

In this paper we present an architecture for internet-mediated simulation games. The challenge was to use today's state of the art technologies to provide a simulated environment for decision making for which the users are trained. Current technologies provide a means to construct a more realistic environment and to embed algorithms for operational decision making. The services, which span the architecture, consist of an embedded simulator, communication using messages, scenario design and control, content management, human and simulated players, and game administration. We implemented a supply chain simulation game as a proof of concept. It consisted of a server-side simulation and communication core combined with objects representing the players. Human players and the game administrator used user interfaces to connect to the server via the Internet. Further research will focus on scenario management and algorithms for simulated decision making.

1 INTRODUCTION

Simulation has long been established as a useful method for inquiry into complex, ill-structured problem situations. One of its related fields is gaming, in which the tools are called simulation games.

In this paper we argue that most currently used simulation games have not incorporated the concepts, technologies and recipes developed throughout the last decade of web-enabled IT development. Hence such games miss several valuable properties, as we show in section 2.

We recognized that there was a need to develop an architecture on which distributed simulation games can be built. Using a service-based approach and the concept of object-oriented design, we focused on a loosely coupled aggregation of generic and end-user specific services.

2 DRIVERS AND REQUIREMENTS FOR THE ARCHITECTURE

The drivers and requirements for the architecture are introduced in this section. The drivers and the value of these drivers are discussed in subsection 2.1. In subsection 2.2 we continue with the requirements that are important for the services which span the architecture and the simulation games based on it.

2.1 Drivers for the Architecture

In order to understand the value of an architecture for gaming, we will first focus on its drivers. Gaming, or interactive simulation is used as a method for inquiry to gain insight or create awareness of a certain problem situation, for learning, or for training. In contrast to simulation, gaming actually involves human decision making. This potentially increases the level of understanding achieved, acceptance of results, and the accuracy of the model and its outcome.

The disadvantages of human involvement are a that the validation of model outcomes becomes more complex, there is a lack of reproducibility and the amount of time invested in decision support increases. As argued in the introduction, technology is considered to be the driver for the development of a multi-player, distributed architecture for gaming. The question is: Why? To what extent have the technological developments of the last decade improved understanding and acceptance?

The first driver is that incorporating state of the art web-enabled technologies in simulation gaming provides players with a simulated environment for decision making which actually resembles the environment for which they are trained.

A second driver is that incorporating these technologies in simulation gaming provides opportunities to construct a more complex and realistic environment. If simulation games are accessible over the web, various complexities which are normally only encountered in real life, such as

cultural, time and language barriers can now actually be incorporated in a scenario. A realistic number of players, products, duration, and relations can be tested.

A third driver is to embed algorithms for operational decision making in simulation games. For example, we can now develop a scenario for auction based acquisition of transport by distributors which allows transport domain experts to embed their competitive planning algorithms.

2.2 Requirements for the Architecture and Simulation Games

When we consider simulation games, there are three "U"'s that are important: the usefulness of the tools and methods, for example the value that they add to the goal of a simulation game. Then thee is their usability, for example the mesh between users, processes and technology. And finally their usage, for example their flexibility, their adaptivity, and their suitability to the context of the problem environment(Keen and Sol 2004). We explore a summarized enumeration of requirements in the following subsections.

2.2.1 Usefulness

The first requirement is that we need to have the opportunity to choose which players in a game are controlled by human decision making and which players are controlled by computational algorithms. Fulfilling this requirement enables us, for example, to use simulated players to model players which are not feasible for human control, to increase complexity, or to increase dynamics. Eventually this leads to more valuable games. Furthermore, by providing a hybrid form, we are able to model a player in a game and let parts of its decision making processes be controlled by a human, and parts by the algorithms.

Depending on the complexity of the problem at hand, it is useful to support long ($>$ 1 day) playing times. This enables players to grasp the complexity better and hence to support better decision making. Persistency of data and a scenario service suitable for long time control are needed to support long games. Furthermore, the scenario service is useful to support complex games, which often involves a lot of administration and actions while playing.

2.2.2 Usability

One of the big advantages of a service-based architecture is that it is possible to use the presentation layer, that is user interfaces placed on top of generic services. This could make it possible for game designers to design and deploy games more easily using the generic services below the representation layer. To support this we need services which, for example, take care for distributed usage or the communication between players. Requirements which further come to mind are the deployment of the game, security scenario design (the scenario service), and the design and tuning of algorithms.

2.2.3 Usage

Scalability is needed to support variable amounts of players.

Persistency is needed to support the saving of data when a (human) player is deliberately not on-line, or when the connection with a player is unexpectedly lost. Saving occurs at fixed and/or flexible points in time. Furthermore persistency provides users with a possibility to go back to earlier moments in a game, and replay certain phases.

Other requirements are related to reliability, robustness, credibility and adaptivity. Reliability and robustness require a system which is unlikely to break or fail. Therefore we need a stable server and to test the whole architecture thoroughly. Credibility includes, for example, support the players native language and a description of the context of an environment. The focus of adaptivity is managing different scenarios and changing business logic or strategies during game play.

2.3 Existing Games

The games that we have found until now most of the times provide a subset of above mentioned requirements, for example distributed usage and long playing times. Often, these game are well suited to the goal they were designed for, but due to their 'lock-in' to a certain problem domain or environment, they are less usable for applications that require a different setting, for example a different scenario, changes in business logic, changing numbers of players etc.

3 AN ARCHITECTURE FOR DISTRIBUTED SIMULATION GAMES

An architecture for distributed, multi-player simulation gaming is introduced in this section. A brief introduction to systems, sub-systems, layered systems and vertically divided systems is given in subsection 3.1.

We continue with an overview of the proposed architecture in subsection 3.2. Several services are described which coherently foresee in the prescribed requirements. A description of the technologies used in a reference specification of the architecture is given in subsection 3.3.

3.1 Designing Information Systems

Taking into account the requirements of operability, modularity, validity, communicabilities, etc, it became an obvious choice to use Object-Orientation as our meta-model for system description. A first set of design principles directly resulted from this choice. These are referred to as

the principles of Object-Orientation of which the principles of delegation, encapsulation and late binding are good examples. A more detailed description of these principles can be found in (Lee and Tepfenhart 2002).

Although these principles form a good basis for design, they tend to prescribe the structure of individual objects and relations instead of the design of complex, dynamic clusters of objects. So the question is: What kind of strategy should we apply to the division of systems into sub-systems?

Each major component of a system is called a sub-system. Each sub-system should deal with a separate subject matter called a domain. Lee and Tepfenhart (2002) define a domain as a separate real, hypothetical, or abstract world that has its own terminology and specific semantic meaning.

Now we know the basis for dividing a system into sub-systems, the next question concerns the relation between a sub-system and the rest of the system. This relation can be peer-to-peer (P2P) or hierarchical.

In an autonomous P2P relation, either side may have access to the other's services. This relation leads to vertically partitioned systems which divide the system into several independent and weakly coupled sub-systems.

The opposite relation is the hierarchical relation between a sub-system and the rest of the system. This leads to a layered system which is an ordered set of sub-systems in which each of the sub-systems is built in terms of the ones below it and provides the basis for building the ones above it.

3.2 Services of the Architecture

An overview of the architecture and its services is given in figure 1. Each of the services is briefly described in subsections 3.2.1 to 3.2.7. The human players' and game administrator user interface services have a P2P relation with the other services, which are all located within a server. However, it is very well possible that the content management system for example, is placed on a different server to manage server load etc. The services are composed of multiple sub-systems in a hierarchical way.

3.2.1 Simulator

The architecture uses a discrete event simulator as the events which take place in a simulation game are events that can be pointed out as a single event at a certain point in time. Common functionalities of a simulator are control commands, like start, stop and changing the simulator speed. These controls enable us to distort time to help emphasize and control the attention and focus of the players.

3.2.2 Message Server

The message server is used to take care of the communication between players in a game. The message server consists of a number of sub-services. It enables distributed communication that is loosely coupled, it manages persistency of messages, it supports peer-to-peer and publish/subscribe mechanisms, priority, acknowledgement, asynchronous and synchronous messaging, and life-time of messages.

3.2.3 Scenario

The scenario service consists of a number of sub-systems. This service communicates with all other components of the architecture. The service is able to schedule events on the simulator, send messages to users or refresh scores of users in the content management system.

The scenario service supports extended games, and complexity by automating a lot of activities a game administrator usually has to manage. Furthermore it supports the scenario service design and management of scenario's. For example, the service is able to save the state of multiple scenario's and in this way make it possible to compare scenario's and the decisions taken in a detailed way.

3.2.4 Simulated Players

The simulated players service incorporates the behavior of players. To do this we use algorithms for operational decision making. This service is used to improve the complexity and dynamics of a game and hence increase its value. It furthermore can be used to test all kinds of algorithms.

3.2.5 Human Players

The human players service, incorporates representations of players which humans are able to control. Representations of these players exist within the server. However, a human is able to connect via the Internet with its server-side representation using an user interface. Using the user interface, a human is able to take decisions and participate in the game. The user interface is furthermore used to support communication with other players in a game. A user interface supports formal 'business' messages and provides possibilities to send for example e-mail-like messages for a more informal messages.

3.2.6 Game Administrator

The game administrator service supports an administrator in managing and controlling a game. The administrator is the one person that can view the data of all the users and apply changes. Its primary functions are to take care of the smooth progression of a game and to solve all the kinds of problems

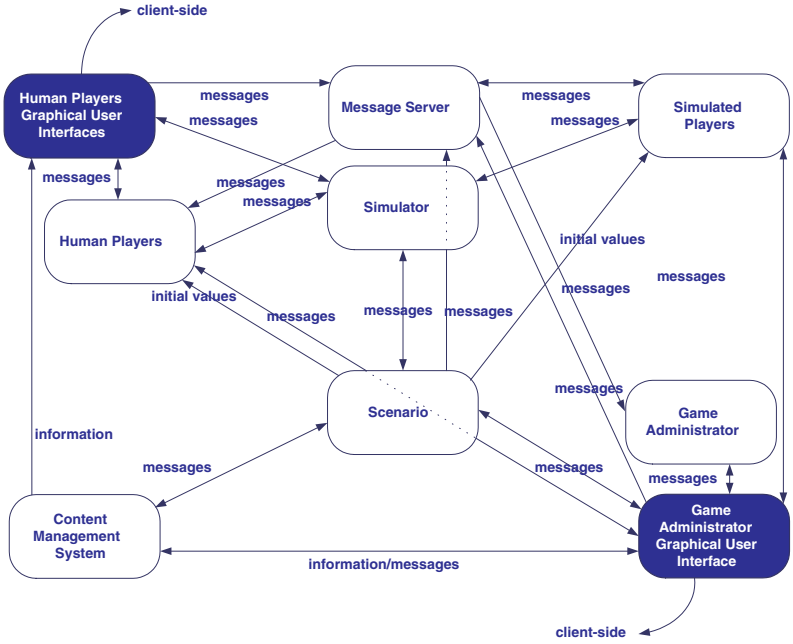

Figure 1: Overview of the Architecture

that may arise. An administrator is able to subscribe to all kinds of (critical) events and to communicate with users.

An administrator is not supposed to schedule all kinds of events, see also scenario service, to make sure a certain scenario is followed. However, an administrator is able to schedule events if this is desired. Just as with a human player, an administrator uses an user interface to connect to the service.

3.2.7 Content Management System

Besides managing content, the content management system (cms) is also used as a web portal. A separate section of the cms is used for each game, as each game has a different story-line and role descriptions. The sections that each player has access to may also differ. Access to these separate sections is managed based on user credentials. The cms is able to provide story-lines, role descriptions, in between scores, automatically updated by the scenario service, and a help and frequently asked questions section. It is furthermore used as a debriefing service.

3.3 Technologies and Tools Used for Implementation

Before we come to the proof-of-concept in section 4, we conclude this section with an overview of the technologies used to specify the above sub-systems and services.

The reference implementation is a J2EE system which is hosted on a JBOSS & Tomcat combination. The simulation core was provided by the DSOL (Jacobs, Lang and Verbraeck 2002) suite for simulation which was accessed through RMI. A combination of HTML and JSP pages provided most of the content to the actual players. Persistency of the game was provided by MySQL and game scenarios were parsed from locally accessible XML-files. The Java Message Service is used for the message service.

4 A PROOF OF CONCEPT: SUPPLY CHAIN GAME

In this section we describe the game we designed as a proof of concept of the feasibility of our architecture. We implemented a supply chain simulation game.

We chose to model a supply chain as it is difficult to teach students and others, for example people from business organizations, all the aspects involved in managing supply chains. It is almost impossible to get a clear view of, and to predict all the effects of, decisions made in a supply chain. Furthermore, ever-increasing complexity and accelerating changes are faced by organizations in supply chains. This brings with it a need for games that can be used to transmit the knowledge and skills needed for this kind of environment (Lainema and Makkonen 2003).

We used a spot buy market for the supply chain. The spot market was formed of multiple yellow pages which kept track of sellers for certain products. Buyers interested in a product contacted a yellow page to retrieve a list of sellers. Then they could contacted these sellers and start the process of acquiring the products. The supply chain consisted of a number of simulated and human players. The customers and suppliers were all simulated. Simulated and human players were used for retailers and manufacturers.

A lot of the functionalities described for the scenario and game administrator services have not yet been implemented, personalization of content for players using the content management system was also not implemented. The game was played using the Internet as the connecting medium. The simulator, and other objects were instantiated on a server. An impression of the user interface used for a manufacturer in the game is given in figure 2.

5 CONCLUSIONS

This paper gives a description of an architecture for internet-mediated simulation games. A Java™based implementation of the services which span the architecture resulted in a "ready to use" design of a simulation core and multiple remote user interfaces.

The architecture presented in this paper shows the various possibilities for this type of architecture with its human and simulated players, and the value this has for modeling complex, multi-actor environments.

Several topics remain for further research and implementation. First, the implementation of the architecture presented in this paper has not yet resulted in a completed set of services that are ready to use for the life-cycle of a simulation game. Furthermore we have found that the management of a scenario and a game require more attention and skills than initially foreseen. For example, the visualization of several processes like initializing a scenario, for example type of players, their parameter values, behavior, and the visualization of control during a game require further research.

A second area requiring further research is the complexity of simulated players, as their complexity increases with the level of detail in a game. Research on the algorithms used for operational decision making remains a challenge.

REFERENCES

Jacobs, P. H. M., N. A. Lang, and A. Verbraeck. 2002. D-SOL; A distributed java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 793-800.

Keen, P. W. G., and H. G. Sol. 2004. *Rehearsing the Future*. To appear.

Lainema, T., and P. Makkonen. 2003. Applying constructivist approach to educational business games: Case realgame. *Simulation & Gaming* 34 (1): 131–-149.

Lee, R. C., and W. M. Tepfenhart. 2002. *Practical object-oriented development with uml and Java™*. Prentice Hall, Upper Saddle River.

AUTHOR BIOGRAPHIES

STIJN-PIETER A. VAN HOUTEN is a Ph.D. student at Delft University of Technology. His research is focused on services for decision support environments, specializing in interactive distributed simulation. His e-mail address is <s.p.a.vanhouten@tbm.tudelft.nl> and his web page is <www.tbm.tudelft.nl/webstaf/stijnh>.

PETER H.M. JACOBS is a Ph.D. student at Delft University of Technology. His research is focused on the design of simulation and decision support services for the web-enabled era. His previous experience at the iForce Ready Center, Sun Microsystems (Menlo Park, CA), and his engineering education at Delft University of Technology gave rise to his interest in this research field. His e-mail address is <p.h.m.jacobs@tbm.tudelft.nl>, and his web page is <www.tbm.tudelft.nl/webstaf/peterja>.

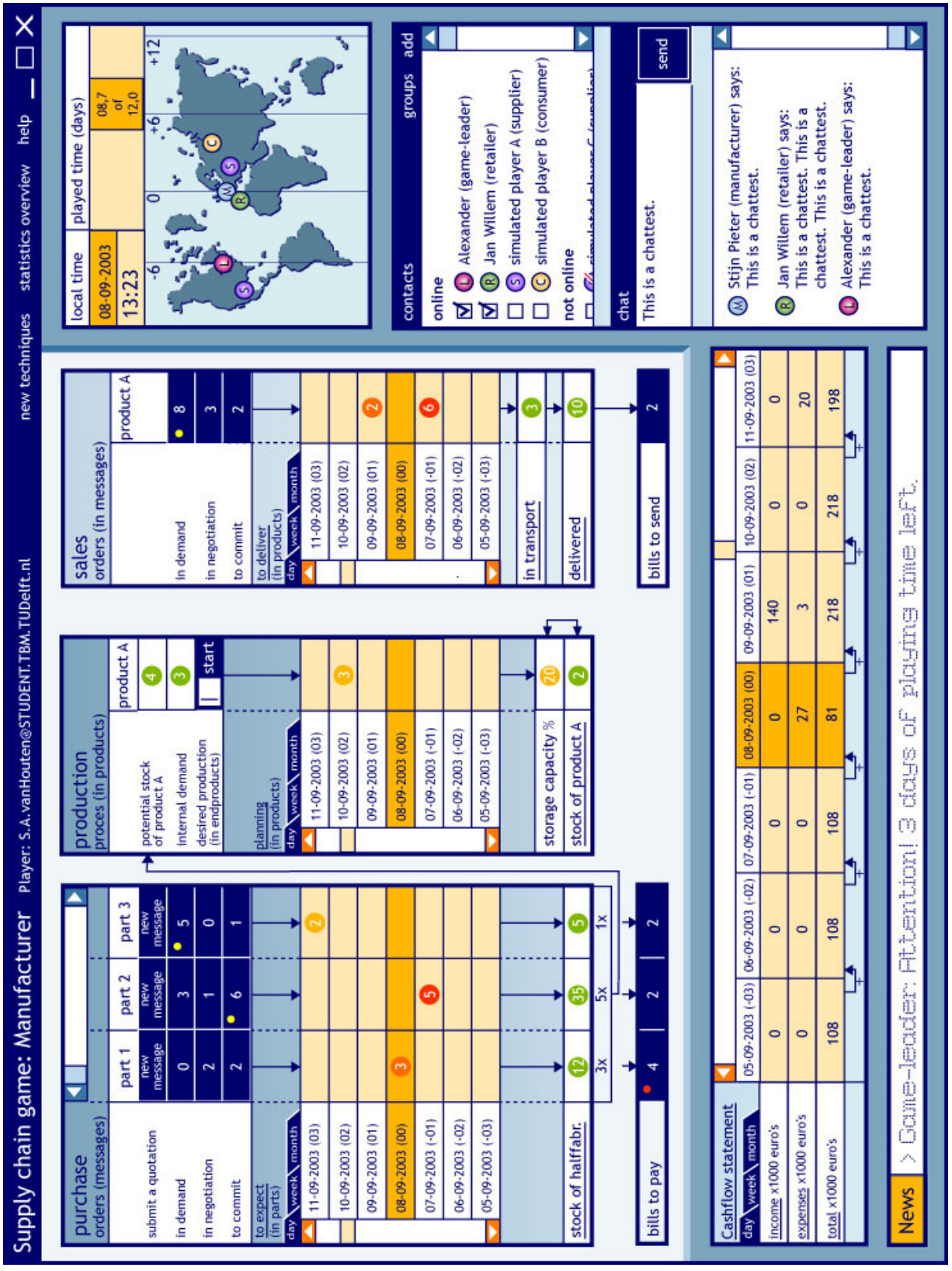

Figure 2: A Proof of Concept Implementation for a Supply Chain Game