SimTJS: A Model for Developing Serious Games for Training
SimTJS: Uma Arquitetura para o Desenvolvimento de Jogos Sérios para Treinamento

Daniel Trindade, Peter Dam, Alberto Raposo
Tecgraf, Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brasil
{*danielrt, peter, abraposo*}*@tecgraf.puc-rio.br*

Ismael Santos
CENPES
Petrobras
Rio de Janeiro,Brazil
ismaelh@petrobras.com.br

***Abstract*—Industrial environments require well trained professionals to handle equipments which are usually complex or operate in risky situations. The training these professionals receive many times is only theoretical, excluding the practical part due to costs and possible risks that training may pose. To solve this, there has been an increasing interest in using *serious games* that mock real scenarios and allow professionals to perform operations that previously would only be possible in a real environment. In this work a model is proposed for the development of this kind of application, called *SimTJS*, which focuses on industrial plant environments. The components that compose the model are discussed, as well as examples that make use of it.**

***Resumo*—Ambientes industriais requerem profissionais bem treinados para lidar com equipamentos muitas vezes complexos ou até situações de risco. A capacitação desses profissionais muitas vezes se dá de forma apenas teórica, deixando a parte prática de fora devido aos custos e aos possíveis riscos que um treinamento prático pode oferecer. Por isso há um interesse crescente na utilização de *jogos sérios* que imitam cenários reais, permitindo que os profissionais realizem operações que antes só seriam possíveis em um ambiente real. Nesse trabalho é proposta uma arquitetura para esse tipo de aplicação, o *SimTJS*, como foco em ambientes de plantas industriais. São discutidos os componentes que compõem a arquitetura, além de exemplos de utilização da mesma.**

***Keywords*-Jogos Sérios; Framework; Interação 3D**

***Keywords*-Serious Games; Framework; 3D Interction**

I. Introdução

Empresas gastam anualmente quantias consideráveis em cursos de capacitação e treinamento para seus funcionários. Boa parte desses cursos é teórica, mas para alguns casos é essencial a realização de cursos práticos, que possam ensinar aos funcionários como realizar operações específicas.

Reproduzir uma situação real para estes cursos pode ser proibitivo devido aos custos e riscos, e como solução para esse tipo de problema, o uso de *jogos sérios* tem se tornado cada vez mais comum [1]. *Jogos sérios* são jogos usados para outros propósitos que não o entretenimento [2]. Nesse tipo de jogo, a ideia é usar os recursos avançados da tecnologia de jogos para permitir que as pessoas tenham outras formas de aprender [3].

O conceito de *jogo sério* pode ser aplicado em diversas áreas. Como exemplo o aplicativo *Nurse Training* [4] permite que alunos de enfermagem coloquem em prática conceitos aprendidos sem o risco de lesionar pacientes reais. Na área de segurança, aplicativos como o *REVAS Process* [4] e *RescueSim* [5] usam simulações em ambientes virtuais para treinamento.

A maioria dos sistemas encontrados, no entanto, são produtos finais que não permitem que o usuário crie novos treinamentos. Como solução para esse problema, esse artigo propõe o *SimTJS* (*Simulação e Treinamento para Jogos Sérios*), uma nova arquitetura para a criação de jogos sérios.

Nas próximas seções, serão apresentados os elementos que compõem a arquitetura do *SimTJS* e também exemplos de uso da arquitetura em aplicações de treinamento para a indústria de óleo e gás.

II. A arquitetura *SimTJS*

A arquitetura do *SimTJS*, representada na figura 1, é baseada no modelo *cliente-servidor*: um servidor concentra funcionalidades que podem ser acessadas por diferentes clientes. A forma como esses clientes são implementados independe do servidor, desde que sejam respeitados os protocolos de comunicação.

A. Executor e Estado Compartilhado

O Executor é responsável por garantir a interação correta entre os diferentes módulos que compõem o *SimTJS*, assim como o fluxo de execução do jogo. Ele recebe e executa comandos provenientes de outros módulos e das aplicações clientes, garante a corretude do estado das variáveis definidas e executa os módulos da Máquina de Jogo e simuladores associados na Interface de Simuladores.

O Executor contém também um espaço de memória que pode ser acessado pelos clientes e módulos do *SimTJS*, o *estado compartilhado*, que contém tuplas do tipo "$chave = valor$". Tanto os módulos quanto os clientes podem criar

978-1-4799-4261-9/14 $31.00 © 2014 IEEE
DOI 10.1109/SVR.2014.50

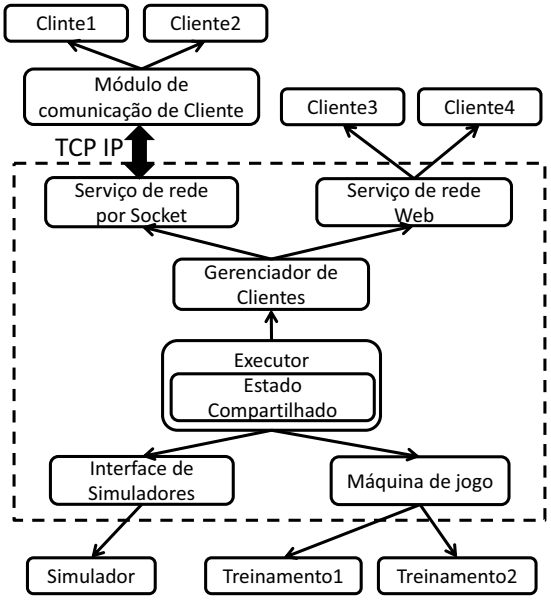

Figura 1. Arquitetura SimTJS.

novas variáveis no estado compartilhado, que ficam automaticamente disponíveis para os outros componentes do *SimTJS*.

B. Gerenciador de Clientes

O Gerenciador de Clientes controla o fluxo de comunicação entre os clientes e o servidor *SimTJS*.

Quando um cliente se conecta ao servidor, ele deve mandar uma mensagem de "registro de cliente". Existem dois tipos permissão: $administrador$ e $aluno$. O cliente $administrador$ tem o poder de iniciar simulações e treinamentos, e o $aluno$ é o que participa ativamente do treinamento.

O *SimTJS* fornece dois tipos de acesso: socket via TCP/IP e web via webservices. A comunicação é feita através de protocolos que permitem aos clientes enviar comandos para o servidor e, mais importante, sincronizar seus estados locais com o estado compartilhado do servidor.

C. Interface de Simuladores

A interface de simuladores permite a integração de diferentes tipos de simuladores ao *SimTJS*. Os simuladores tem como função modelar o funcionamento dos sistemas presentes na planta. Por exemplo, durante um treinamento pode ser necessário que o aluno realize a abertura de uma válvula. Essa ação desencadeia mudanças nas propriedades do equipamento usado (ou ainda de outros equipamentos), as quais são calculadas pelos simuladores acoplados ao *SimTJS*.

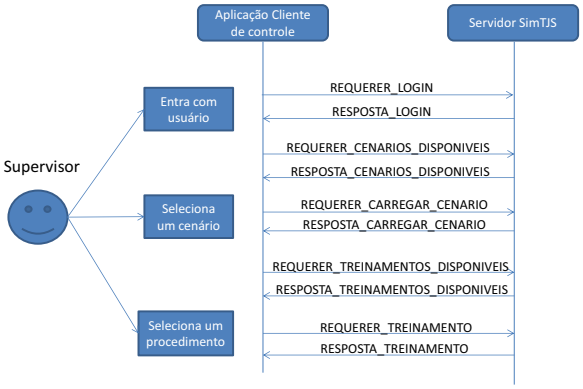

Figura 2. Protocolo de início de treinamento.

D. Máquina de Jogo

A máquina de jogo é o componente do *SimTJS* responsável pela execução e aplicação das regras de jogo definidas no treinamento. Um treinamento é um conjunto de operações tipicamente realizadas em ambientes de plantas industriais.

1) Protocolo de Início de Treinamento: O processo começa com a requisição de entrada no sistema. O servidor *SimTJS* recebe a requisição e estabelece a conexão. Quando um cliente $administrador$ se conecta, o servidor *SimTJS* envia uma lista dos cenários disponíveis. Quando o supervisor seleciona um dos cenários disponíveis, o servidor *SimTJS* envia aos clientes $alunos$ uma mensagem requerendo o carregamento do cenário. Uma vez ativado o cenário, o servidor envia uma mensagem ao cliente $administrador$ com a lista de treinamentos disponíveis para o cenário carregado. A partir dessa lista, o supervisor requisita o início de um treinamento ao servidor. É possível observar esta sequência na figura 2.

2) Suporte a Multiusuário: No *SimTJS* cada cliente carrega o cenário em execução no servidor. Esses clientes possuem sua própria instância do estado de variáveis, na qual escrevem seu atual estado de execução. Os clientes sincronizam seus estados com um estado compartilhado, mediado pelo servidor *SimTJS*.

3) Protocolo de Sincronização de Estado: Para garantir a corretude das funcionalidades de cooperação entre usuários, os estados presentes em cada cliente devem estar sincronizados. Mais ainda, a estratégia de sincronização adotada deve impedir a ocorrência de atrasos [6]–[8]. A figura 3 ilustra o processo de conexão e sincronização entre clientes e o servidor *SimTJS*.

III. Clientes

Utilizando o módulo de comunicação de cliente, é possível criar um cliente em qualquer linguagem ou engine utilizando comunicação por TCP. Além disso, é possível

Figura 3. Protocolo de sincronização do estado.

criar clientes que acessem o estado compartilhado através de serviço web. Desta maneira foram criados três clientes diferentes, onde cada um desempenha um papel diferente.

A. *Cliente Gráfico*

Como mencionado antes, um cliente gráfico pode ser criado usando qualquer motor gráfico, sendo que a única exigência é que a comunicação deve ser feita com o servidor *SimTJS* usando o módulo de comunicação de cliente. O trabalho principal na modelagem de um cliente resume-se, portanto, a criar formas de traduzir as ações realizadas no mundo gráfico em variáveis que possam ser armazenadas no estado compartilhado do servidor.

Como exemplo, foram modelados cenários de uma plataforma de extração de petróleo (figura 4), de uma usina termoelétrica (figura 5) e de uma usina fotovoltaica (figura 6). Elementos interagíveis como portas, válvulas e botões são mapeados em variáveis que indicam seu estado e podem influenciar diretamente no comportamento dos outros módulos do *SimTJS*. A abertura de uma válvula pode provocar uma mudança de estado na máquina de jogo ou iniciar um passo de uma simulação.

Quando disponíveis, ambientes imersivos podem ser usados para aumentar o nível de realismo do treinamento. Para suportar esse tipo de ambiente, os clientes gráficos implementados fazem uso do conjunto de bibliotecas providas pelo *LVRL* [9].

Para que um usuário possa completar um treinamento podem ser necessárias ferramentas específicas ao negócio relacionado. Em situações reais, operadores podem precisar usar ferramentas e equipamentos de proteção individual. A figura 7 ilustra uma forma de se usar esses conceitos em um cliente: o usuário tem acesso a uma barra de ferramentas, onde ele pode selecionar itens como trena ou lanterna e verificar quais equipamentos de proteção ele está usando.

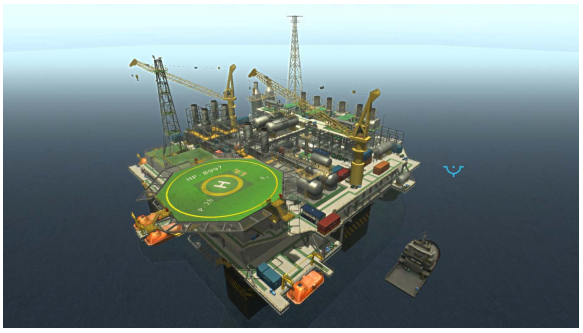

Figura 4. Cenário contendo uma plataforma de exploração de petróleo.

Figura 5. Cenário contendo uma usina termoelétrica.

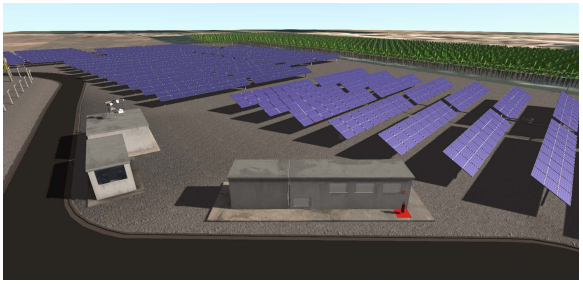

Figura 6. Cenário contendo uma fotovoltaica.

Figura 7. Barra de ferramentas

Certas tarefas podem necessitar também do uso de cooperação entre dois ou mais operadores. O *SimTJS* permite que os operadores conversem entre si via $chat$, simulando a presença de $walkie - talkies$ (ver figura 8). Através desse recurso o usuário pode também receber as tarefas que deve cumprir.

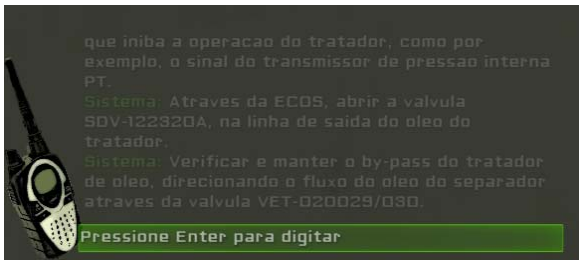

Figura 8. Ferramenta de comunicação

Figura 9. Formulário web em dispositivo móvel

B. Cliente Administrador

O cliente $administrador$ tem como função permitir que um usuário do tipo supervisor controle e tenha informações visuais sobre o andamento de um treinamento. Assim como os clientes gráficos, eles utilizam o módulo de comunicação de cliente para se conectar e mandar operações para o servidor.

Através deste cliente um supervisor pode dar início a uma sessão de treinamento. Internamente, o módulo de comunicação de cliente usa o protocolo de comunicação descrito na figura 2.

C. Cliente Web

Além do módulo de comunicação de cliente desenvolvido usando TCP/IP, é possível se conectar ao servidor *SimTJS* via interface web. Como exemplo, foi criado um cliente para submissão de relatórios. Enquanto navega no cenário, o usuário pode preencher um relatório em um dispositivo móvel (figura 9), que poderá ser submetido e avaliado pelo *SimTJS*, retornando um resultado para o usuário.

IV. Conclusões e trabalhos futuros

Este artigo apresentou o *SimTJS*, uma arquitetura voltada ao desenvolvimento de jogos sérios. Por não estabelecer quais tecnologias devem ser usadas nos clientes, a implementação dos clientes é livre: ele pode ser um ambiente virtual 3D usando um motor gráfico qualquer, um jogo 2D, pode-se usar realidade virtual para aumentar a imersão. O *SimTJS* possibilita também o uso de múltiplos simuladores acoplados.

Como trabalhos futuros, pretende-se estender a máquina de jogo com a criação de um sistema de pontuação para os treinamentos. Ainda, do lado das funcionalidades para clientes administradores, serão criadas formas de se obter relatórios sobre os treinamentos realizados, além de mais ferramentas para controle dos treinamentos.

Referências

[1] D. R. Michael and S. L. Chen, *Serious Games: Games That Educate, Train, and Inform*. Muska & Lipman/Premier-Trade, 2005.

[2] T. Susi, M. Johannesson, and P. Backlund, "Serious Games: An Overview," Web at GamesLearningSociety, GLS University of Wisconsin-Madison, Tech. Rep., Feb. 2007.

[3] B. Sawyer and D. Rejeski, "Serious games: Improving public policy through game-based learning and simulation," http://www.seriousgames.org/images/seriousarticle.pdf, 2002, acessado em: 11-02-2014.

[4] D. Digitally, "3d serious games and simulations," http://www.3dseriousgamesandsimulations.com, acessado em: 11-02-2014.

[5] VSTEP, "Vstep simulation and virtual training: The next best thing to real life!" http://www.vstep.nl/, acessado em: 11-02-2014.

[6] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, March 2004, pp. –107.

[7] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '02. New York, NY, USA: ACM, 2002, pp. 23–29. [Online]. Available: http://doi.acm.org/10.1145/507670.507674

[8] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, "Peer-to-peer-based infrastructure support for massively multiplayer online games," in *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, Jan 2007, pp. 763–767.

[9] L. Teixeira, D. Trindade, M. Loaiza, F. G. d. Carvalho, A. Raposo, and I. Santos, "A vr framework for desktop applications," in *Proceedings of the 2012 14th Symposium on Virtual and Augmented Reality*, ser. SVR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 10–17. [Online]. Available: http://dx.doi.org/10.1109/SVR.2012.27