UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROJETO DE UM FRAMEWORK PARA VISUALIZAÇÃO E CONTROLE DE SIMULAÇÕES DISTRIBUÍDAS EM DIFERENTES PLATAFORMAS DE SOFTWARE E HARDWARE

Fábio Minoru Iwasaki

SÃO CARLOS
MAIO/2008

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

I966pf

Iwasaki, Fábio Minoru.
Projeto de um framework para visualização e controle de simulações distribuídas em diferentes plataformas de software e hardware / Fábio Minoru Iwasaki. -- São Carlos : UFSCar, 2008.
98 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2008.

1. Simulação por computador. 2. Aplicações distribuídas. 3. Visualização 3D. 4. X3D (Linguagem de programação). I. Título.

CDD: 004.36 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Projeto de um framework para visualização e controle de simulações distribuídas em diferentes plataformas de Software e hardware"

FÁBIO MINORU IWASAKI

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Regina Borges de Araújo
(Orientadora - DC/UFSCar)

Prof. Dr. Ednaldo Brigante Pizzolato
(Co-orientador – DC/UFSCar)

Profa. Dra. Wu Shin Ting
(DCA/FEEC/UNICAMP)

Prof. Dr. Márcio Merino Fernandes
(DC/UFSCar)

São Carlos
Maio/2008

RESUMO

Simulações em computadores podem ser usadas para avaliar futuros riscos e auxiliar na tomada de decisões e podem ser modeladas para execuções rápidas para prever o estado de um sistema do mundo real ou podem ser modeladas para execuções com tempo indeterminado que permita interações humanas em um ambiente virtual para treinamento. Para facilitar a modelagem e implementação dessas simulações, vários pacotes comerciais de software estão disponíveis, porém tecnologias proprietárias são limitadas quando há necessidade reutilizar uma determinada implementação em conjunto com outras soluções. Diante da necessidade de interoperabilidade, o padrão High Level Architecture (HLA) define uma infra-estrutura de comunicação para compartilhamento de dados e sincronização entre modelos de simulação distribuídos. Utilizando o HLA como base, este trabalho define um framework para gerar uma aplicação de controle e gerenciamento utilizando gráficos tridimensionais para visualização com o padrão Extensible 3D (X3D). O X3D especifica um ambiente de execução para apresentação de conteúdo 3D e define uma linguagem e um conjunto de componentes para descrever esse conteúdo. Utilizando os padrões HLA e X3D e tecnologias como serviços Web, o framework deste trabalho apresenta uma solução para visualização e controle de simulações distribuídas em diferentes plataformas de software e hardware.

ABSTRACT

Computer simulation can be used to assess future risks and to support decision making and can be modeled either for quick runs to predict the state of a real-world system or for indeterminate runtime to allow human interaction in a virtual environment for training. To facilitate the modeling and implementation of those simulations, many commercial software suites are available, but proprietary technologies are limited when there is the need to reuse a particular implementation together with other solutions. Faced with the need for interoperability, the High Level Architecture (HLA) standard defines a communication infrastructure for data sharing and synchronization among distributed simulation models. Using the HLA as its basis, this work defines a framework to generate an application for simulation control and management using tridimensional graphics for visualization through the Extensible 3D (X3D) standard. The X3D specifies a runtime environment for 3D content presentation and defines a language and a set of components to describe that content. By using the HLA and X3D standards integrated into technologies such as Web services, this framework provides a solution to visualization and control of distributed simulations in different software and hardware platforms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BOOM	Binocular Omni-Oriented Monitor
CAVE	Cave Automatic Virtual Environment
CORBA	Common Object Request Broker Architecture
CRT	Cathode Ray Tube
CVE	Collaborative Virtual Environment
DDM	Distributed Data Management
DIS	Distributed Interactive Simulation
DLC	Dynamic Link Compatible
DOM	Document Object Model
DVE	Distributed Virtual Environment
FDD	Federation Document Data
FED	Federation Execution Details
FOM	Federation Object Model
HLA	High Level Architecture
HMD	Head Mounted Display
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers

IIOP	Internet Inter-Orb Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
LBTS	Lower Bound Time Stamp
LCD	Liquid Crystal Display
LCOS	Liquid Crystal on Silicon
LP	Logical Process
MOM	Management Object Model
MUD	Multi-User Dungeon
OLED	Organic Light-Emitting Diode
OMT	Object Model Template
OOAD	Object-Oriented Analysis and Design
P2P	Peer-to-Peer
PADS	Parallel and Distributed Simulation
RMI	Remote Method Invocation
RTI	Runtime Infrastructure
SAI	Scene Access Interface
SISO	Simulation Interoperability Standards Organization
SOAP	Simple Object Access Protocol
SOM	Simulation Object Model
TCP/IP	Transmission Control Protocol/Internet Protocol

TSO	Time Stamp Ordered
UDDI	Universal Description, Discovery and Integration
VRML	Virtual Reality Modeling Language
WAN	Wide Area Network
WS	Web Services
WSDL	Web Services Description Language
X3D	Extensible 3D
XML	Extensible Markup Language
XMSF	Extensible Modeling and Simulation Framework
XSLT	Extensible Stylesheet Language Transformations

SUMÁRIO

1. INTRODUÇÃO

Novas tecnologias facilitam a vida moderna, mas ao mesmo tempo introduzem novos riscos. Por exemplo, na ocorrência de algum acidente em uma planta industrial, produtos químicos podem liberar gases tóxicos criando um ambiente altamente perigoso para as equipes de resgate como bombeiros e paramédicos. Para diminuir os danos causados na ocorrência de um acidente e aumentar as chances de sobrevivência de todas as pessoas envolvidas, o treinamento e planejamento em ambientes virtuais é uma alternativa que permite prever possíveis cenários onde essas equipes poderão atuar.

Ambientes virtuais são mundos simulados onde os vários participantes podem interagir uns com os outros para a realização de atividades cooperativas ou competitivas. Treinamento, ensino à distância e entretenimento são apenas algumas das possíveis utilizações para esses ambientes que precisam de uma infra-estrutura robusta para simulação e comunicação. O desenvolvimento de simulações distribuídas é uma tarefa custosa, que pode consumir muito tempo e muitos recursos. Assim, iniciativas para facilitar a interoperabilidade entre simulações foram propostas e adotadas pela comunidade de desenvolvedores de simulações distribuídas. Uma das iniciativas de maior sucesso foi a especificação do padrão High Level Architecture (HLA). O HLA define um arcabouço de propósito geral para simulações distribuídas que especifica todos os mecanismos necessários para a interoperabilidade e reutilização de simulações distribuídas. Utilizando o HLA e outros padrões abertos de tecnologias, é possível desenvolver uma complexa aplicação distribuída de ambientes virtuais.

Gráficos tridimensionais são mais intuitivos que gráficos bidimensionais para visualização de simulações, pois podem reproduzir na tela do computador imagens semelhantes ao que vemos no mundo real (ou seja, com maior fidelidade), em contraste com a utilização de representações através de símbolos ou figuras que precisam ser interpretadas. Um exemplo disso é os sistemas de informação geográfica onde a reprodução gráfica em três dimensões do relevo de um terreno apresenta prontamente as informações sobre as mesmas altitudes que estariam representadas em duas dimensões por curvas de nível. Mesmo com essa vantagem, a visualização tridimensional é custosa, tanto computacionalmente (pois envolve uma grande quantidade de

cálculos e operações matemáticas) como na criação de conteúdo (modelos geométricos, texturas, animações). O X3D consiste de uma linguagem para descrever mundos tridimensionais e um ambiente de execução que já é implementado em várias plataformas de software e hardware (incluindo os dispositivos de mão). Vários conteúdos descritos em X3D podem ser usados e reusados em conjunto para formar um mundo virtual tridimensional complexo.

Com o objetivo de implementar um sistema robusto para ambientes virtuais complexos, a adoção de padrões abertos é a melhor alternativa para se garantir sua manutenibilidade, pois conforme os padrões evoluem, novas características podem ser atribuídas a esse sistema. Essa preocupação surge ao analisar sistemas existentes para a criação de ambientes virtuais que são mantidos por tecnologias proprietárias ou por um desenvolvedor específico. Apesar de soluções comerciais fornecerem ferramentas que facilitam a implementação de cenários específicos de simulação, podem ser pouco flexíveis quando se quer interoperabilidade e reutilização.

1.1. MOTIVAÇÃO E OBJETIVOS

Este trabalho define um arcabouço de software, ou framework, que faz parte de um projeto de gerenciamento de emergências do Laboratório de Realidade Virtual em Rede (LRVNet) do Departamento de Computação da Universidade Federal de São Carlos. A Figura 1 apresenta uma visão geral desse projeto para criação de ferramentas baseado em padrões e integração de serviços e sistemas que envolvem principalmente simulação, monitoramento e interpretação de contexto.

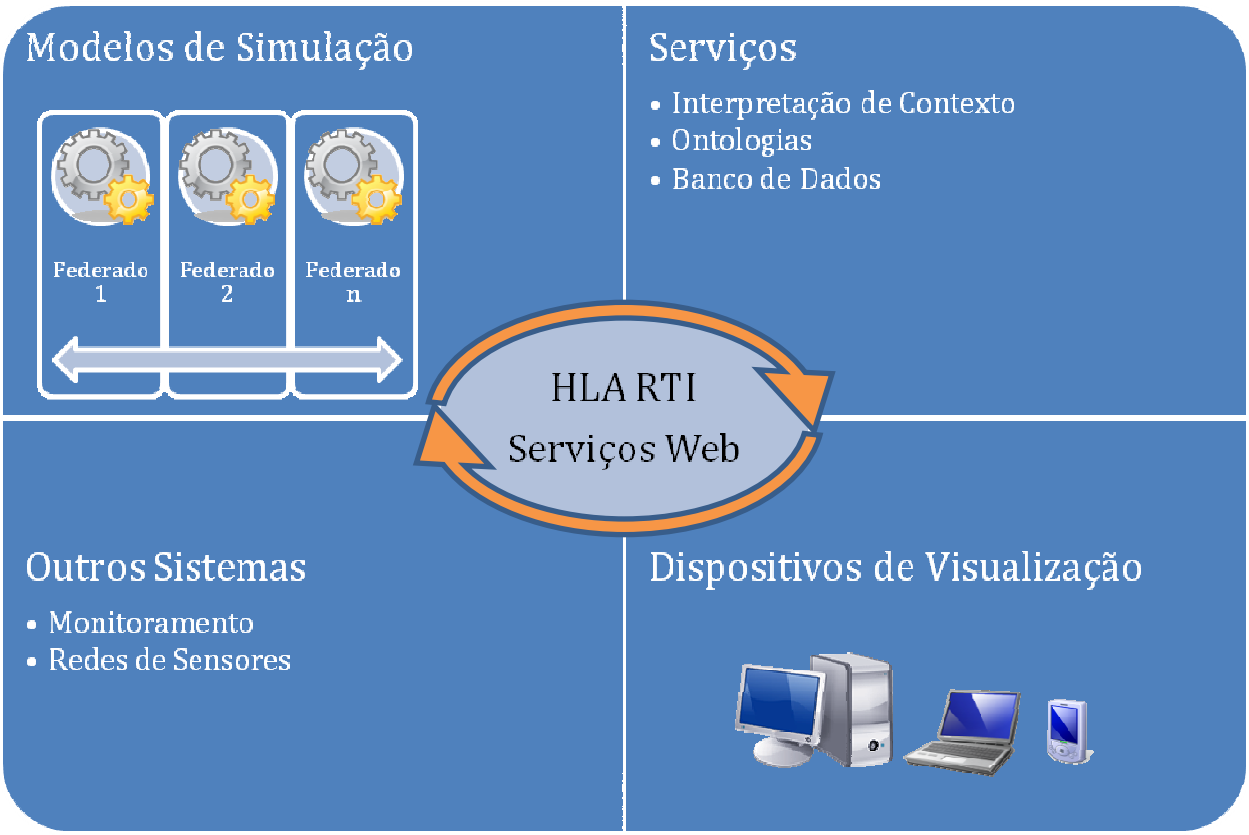


Figura 1. Visão Geral do Projeto de Gerenciamento de Emergências do LRVNet.

O projeto de gerenciamento de emergências envolve sistemas para monitorar ambientes que utiliza ontologias para interpretação de contexto. Na ocorrência de algum incidente, os dados capturados do ambiente monitorado podem ser utilizados para executar simulações para auxiliar na tomada de decisões. Esses dados são armazenados e podem ser utilizados para reproduzir o incidente em um ambiente virtual para análise ou treinamento.

O desenvolvimento de ambientes virtuais tridimensionais é um processo custoso que requer basicamente: uma infra-estrutura de comunicação, uma simulação do ambiente virtual e mecanismos de visualização e controle da simulação. Atualmente poucas ferramentas de visualização e controle estão disponíveis na forma de software livre. Tipicamente, essas ferramentas fazem parte de sistemas proprietários de alto custo. Assim, o objetivo principal deste trabalho é o desenvolvimento de um framework para ambientes virtuais, definindo uma

arquitetura e componentes reusáveis que possam ser integrados a outros sistemas utilizando tecnologias e padrões abertos na sua implementação.

1.2. ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta os principais conceitos de simulações em computador, as variadas áreas de simulação e as características do padrão HLA para simulações distribuídas. No capítulo 3 são apresentados conceitos de visualização e realidade virtual, além de tecnologias que suportam aplicações dessas áreas, principalmente o padrão X3D. No capitulo 4 são explicado os detalhes do framework desenvolvido neste trabalho com a implementação e análise de um protótipo. As conclusões deste trabalho estão no capítulo 5 com propostas de extensões futuras.

2. SIMULAÇÕES DISTRIBUÍDAS COM HIGH LEVEL ARCHITECTURE E WEB SERVICES

2.1. CONSIDERAÇÕES INICIAIS

Nesta seção serão feitas revisões de vários conceitos relacionados à área de simulação em computadores e serão descritas as principais características do padrão High Level Architecture utilizado neste trabalho. Várias comunidades diferentes de pesquisas sobre simulação, que se organizaram para atender a objetivos específicos, também serão descritas a seguir. As características das simulações de interesse deste trabalho são:

a) dinâmicas: O estado do sistema se altera no decorrer do tempo;
b) discretas: As alterações do estado do sistema são resultados de eventos em pontos discretos do tempo;
c) estocásticas: Caracterizadas pelo comportamento não-determinístico (aleatoriedade).

2.2. SIMULAÇÃO EM COMPUTADOR

Simulação é definida como a imitação da operação de algum processo ou sistema do mundo real sobre o tempo (Banks 1999; Goldsman 2007). Uma simulação envolve a geração de uma história artificial do sistema e a observação dessa história para extrair inferências com respeito às características operacionais do sistema real que é representado. Podemos ressaltar, dentre as vantagens em se utilizar simulações, a possibilidade de explorar exaustivamente os mecanismos de um sistema para um melhor entendimento de seus problemas existentes e a prevenção de problemas futuros. Também pode ser uma alternativa efetiva e eficiente para treinamento quando modelada com esse propósito. Porém, algumas das suas grandes desvantagens são a quantidade de tempo e os custos necessários para se modelar uma simulação

que pode acabar sendo utilizada de maneira inapropriada ou gerar resultados difíceis de serem interpretados; o modelo que não for devidamente validado perante o sistema real pode resultar na implementação de uma simulação onde os resultados são errôneos e causam confusão.

Simulações em computadores são utilizadas em uma grande variedade de contextos práticos, tais como nas áreas de treinamento militar, de entretenimento, industriais, acadêmicas e de pesquisa.

Algumas definições comumente aceitas de termos utilizados com freqüência (Carson 1993, 2004; Harrel 1998; Banks 2000) serão listadas a seguir para esclarecer alguns conceitos de modelagem e simulação:

a) modelo e sistema: Um modelo é simplesmente a representação de um sistema ou um processo escolhido. Um sistema implementa um conjunto de processos coordenados que convertem entradas em saídas. Um processo preocupa-se com o fluxo lógico de entidades através de um conjunto de atividades (o que acontece com as entidades) enquanto o sistema é baseado no fluxo físico de entidades por determinados pontos de processamento (como, onde e quando);
b) variáveis de estado do sistema: Formam a coleção de todas as informações necessárias para definir o que acontece no sistema em um determinado momento;
c) eventos e notificações de eventos: Um evento em um modelo corresponde a algo que ocorre no sistema real que pode mudar o estado desse sistema. Eventos ocorrem em um determinado instante de tempo. Esse tempo é registrado durante a notificação de eventos e colocado em uma lista. Essa lista é a lista de eventos que é gerenciada pela simulação;
d) entidades e atributos: Entidade é um objeto ou elemento do sistema que precisa ser explicitamente definido. Uma entidade pode ser dinâmica e “mover-se” pelo sistema ou estática e prestar algum tipo de serviço para outras entidades. Cada entidade pode possuir valores locais que são atributos pertinentes somente a essa entidade;
e) recursos: São entidades que provêm serviços para entidades dinâmicas;
f) processamento de listas: Entidades podem ser colocadas em listas ordenadas para serem gerenciadas. Essas listas são filas nas quais as entidades são forçadas a esperar pela liberação de um recurso ou por alguma outra condição do sistema;

g) atividades e atrasos: Uma atividade é uma duração de tempo conhecida antes de esta mesma ser iniciada (um intervalo de tempo definido no modelo da simulação). Atraso é uma duração indefinida causada por uma combinação de condições do sistema como, por exemplo, a espera de uma entidade por um recurso. Simulações discretas contêm atividades que causam o avanço do tempo. O início ou fim de uma atividade ou atraso é um evento.

2.2.1. Simulações Paralelas e Distribuídas

A comunidade de simulação paralela e distribuída (*Parallel and Distributed Simulation* ou PADS) realiza pesquisas para desenvolver tecnologias de alto desempenho para reduzir o tempo de execução de simulações de larga-escala. Os programas de simulações paralela e distribuída são baseados no mesmo princípio de que uma tarefa complexa pode ser dividida em tarefas menores que são executadas simultaneamente através de um mecanismo de coordenação. A arquitetura computacional é a diferença básica entre simulações paralelas e simulações distribuídas (Chandy e Misra 1979; Fujimoto 1989, 2001). Uma simulação paralela é um programa executado por computadores que utilizam multiprocessadores ou por computadores interligados por um barramento externo de alta velocidade com memória compartilhada. Em um programa de simulação distribuída os processos se comunicam através de mensagens, não existem variáveis compartilhadas e não existe um processo central de controle. Computadores que fazem parte de uma simulação distribuída podem possuir configurações de software e hardware distintas e são interconectados através de uma rede de comunicação local (LAN) ou redes geograficamente distribuídas (WAN) como a Internet.

2.2.2. Simulações Interativas Distribuídas

A comunidade de simulação interativa distribuída (*Distributed Interactive Simulation* ou DIS) possui pelo menos um objetivo em comum com a PADS: a execução de programas de simulação em múltiplos processadores interligados por uma rede. Diferente das pesquisas da PADS onde se busca desempenho para executar simulações no menor tempo possível, os trabalhos da DIS visam aplicações militares para treinamento de pessoal distribuído geograficamente criando ambientes virtuais nos quais eles possam interagir em tempo-real em preparação para situações do campo de batalha (Hofer e Loper 1995; Fullford 1996). As tecnologias desenvolvidas têm como objetivo criar uma infra-estrutura para interoperabilidade entre vários tipos de modelos de simulação misturando diferentes entidades como as de simuladores (com interação humana), plataformas operacionais, sistemas de teste e avaliação e agentes computacionais. Além das aplicações militares, a mesma tecnologia tem sido aplicada em áreas comerciais como entretenimento, controle de tráfego, preparação para emergências (como incêndios e terremotos), entre outros.

2.2.3. Simulações On-Line

Simulação on-line, também chamado de simulação em tempo-real, é uma classe de simulações onde o modelo da simulação é remotamente acessado através de uma rede de comunicação e inicializado com dados representando o estado atual do sistema real para uma execução rápida onde os resultados são previsões utilizadas para tomadas de decisões (Davis 1998). As áreas de aplicações típicas são as de suporte para decisões pró-ativas em agendamento em tempo-real de sistemas de manufatura (Katz e Mannivannan 1993; Harmonsky 1995; Sivakumar 1999; Gupta et al. 2002; Chong et al. 2002; Glinsky e Wainer 2002). Outras áreas de aplicação são simulações de controle de tráfego de automóveis (Mazur et al. 2004) e simulações de fluxo de pessoas em prédios (Hanisch et al. 2003).

2.2.4. Simulações Distribuídas Ad-Hoc

Uma simulação distribuída ad-hoc é uma coleção de simulações on-line autônomas, ou processos lógicos (*Logical Processes* ou LPs), que modelam um sistema operacional (Fujimoto 2007). A quantidade de LPs que formam a simulação pode variar durante a execução. Cada LP pode modelar uma porção do ambiente físico e obter informações através de sensores embarcados, por exemplo. Possíveis aplicações para essa categoria de simulação estão no controle de tráfego de automóveis, onde cada veículo possui um simulador para uma porção do tráfego e trocam dados com outros simuladores (veículos e infra-estrutura). Tomando como base esse exemplo, ao contrário da simulação distribuída tradicional, onde LPs são atribuídos para regiões sem sobreposição, na simulação distribuída ad-hoc porções do modelo do sistema podem ser duplicados ou simplesmente não serem modelados.

2.3. HIGH LEVEL ARCHITECTURE

O padrão IEEE 1516-2000 High Level Architecture (HLA) para simulações distribuídas foi inicialmente desenvolvido pelo Departamento de Defesa dos Estados Unidos e define um arcabouço de propósito geral para simulações distribuídas que tem como objetivo a interoperabilidade e reutilização dessas simulações. Uma simulação ou um conjunto de simulações existentes podem ser reutilizados no desenvolvimento de novas simulações. A reutilização refere-se à adaptação de componentes como, por exemplo, idéias, linhas de códigos ou toda a implementação. Além da redução do custo do desenvolvimento, a intenção do HLA é permitir que simulações projetadas com propósitos diferentes possam fornecer serviços e trocar dados entre si para atingir um objetivo em comum. O compartilhamento efetivo e a interpretação consistente de dados são os principais elementos da interoperabilidade.

No HLA, uma simulação distribuída (um sistema composto de modelos de simulações distribuídos) é chamada de federação. Cada modelo de simulação ou aplicação individual que faz parte da federação é chamado de federado. Um federado pode ser uma simulação em computador,

mas pode ser também um dispositivo físico, um coletor passivo de dados para visualização ou uma interface para um participante humano.

Outros termos essenciais definidos pelo HLA utilizados neste texto são:

a) classe de objeto: Descreve tipos de entidades que podem persistir;
b) atributo: É uma característica nomeada de uma classe ou instância de objeto;
c) instância de objeto: É um objeto individual criado a partir de uma classe;
d) classe de interação: Descreve tipos de eventos;
e) parâmetro: É uma característica nomeada de uma classe ou instância de interação;
f) instância de interação: Um evento específico;
g) publicação: É a declaração dos dados que um federado poderá disponibilizar para federação;
h) subscrição: É a declaração de interesse de um federado em algum dado que pode ser publicado na federação;
i) registro: É o anúncio da existência de uma nova instância de objeto de um federado para federação;
j) descoberta: É a tomada de conhecimento da existência de uma instância de objeto registrado por algum federado;
k) remoção: É a remoção de uma instância de objeto da federação;
l) atualização: É o anúncio de uma mudança de valor de um atributo de uma instância de objeto;
m) reflexão: É a tomada de conhecimento da mudança de valor (atualização) de um atributo de uma instância de objeto pertencente à algum federado.

Os federados de uma mesma federação comunicam-se entre si através da infra-estrutura de tempo de execução (*Runtime Infrastructure* ou RTI). O RTI é efetivamente um sistema operacional distribuído que oferece serviços para interação entre federados e para o gerenciamento da federação. Os componentes do HLA são os federados, o RTI e a especificação de interface que define uma maneira padronizada para os federados se conectar ao RTI. A Figura 2 ilustra os componentes do RTI. O RtiExec (*RTI Executive*) é um processo global que é executada em alguma plataforma e gerencia múltiplas execuções de federações. O FedExec

(*Federation Executive*) é um processo criado por um federado para gerenciar uma federação. O libRTI (*RTI Library*) fornece o acesso aos serviços do RTI para a implementação dos federados.

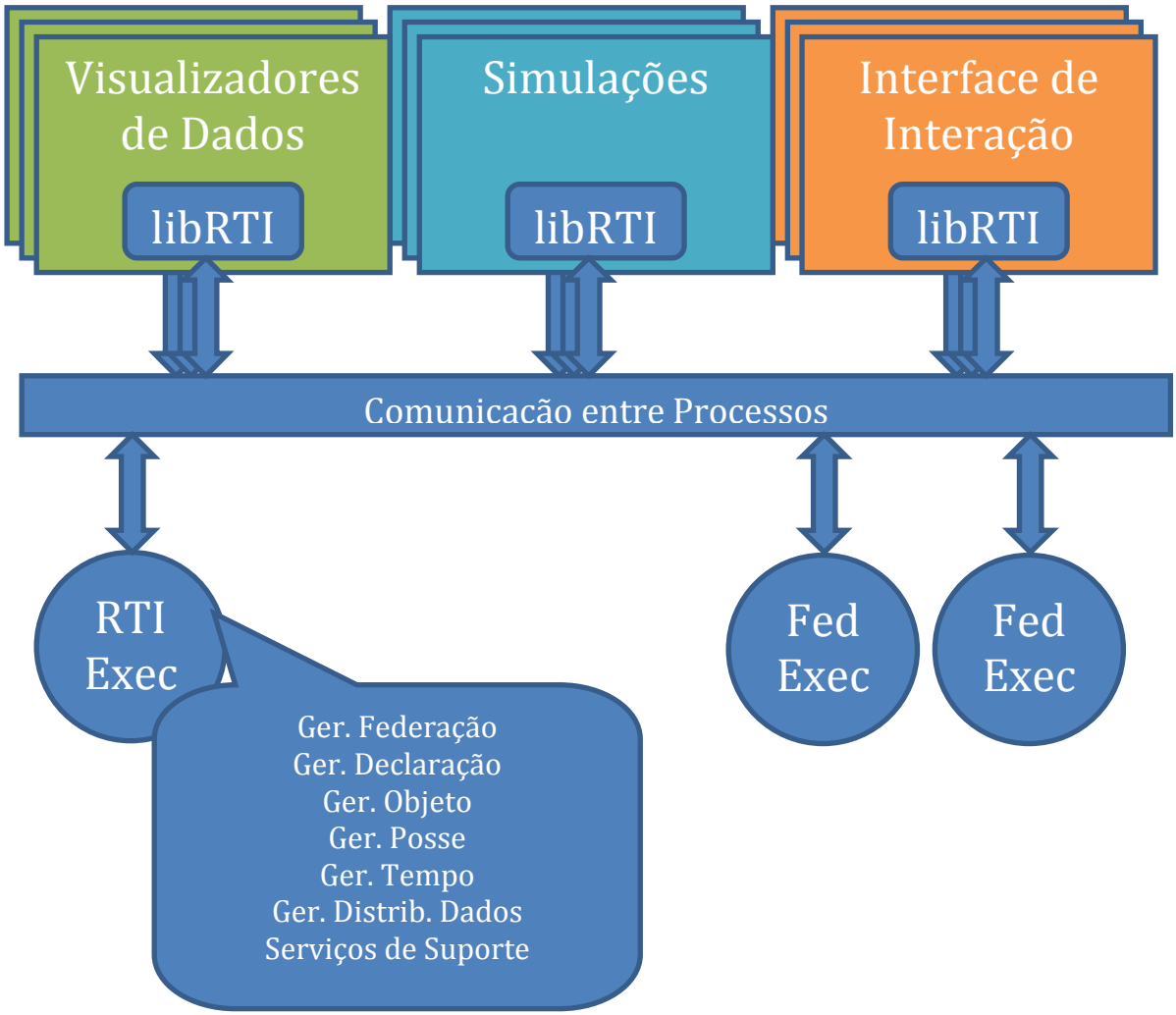


Figura 2. Componentes do RTI.

O HLA é definido por três componentes: as regras do HLA, a especificação de interface e a especificação do documento base de modelo de objeto (*Object Model Template* ou OMT) que serão descritos nas próximas seções.

2.3.1. Framework e Regras do HLA

Define os princípios da arquitetura e as regras de funcionamento. Existem cinco regras da federação e cinco regras do federado.

As cinco regras da federação são:

a) federações deverão possuir seus respectivos modelos de objeto da federação (*Federation Object Model* ou FOM, definição na seção 2.3.2.1) documentados em conformidade com o OMT;
b) em uma federação, toda representação de instâncias de objetos associados à simulação deverá estar no federado e não no RTI;
c) durante uma execução da federação, toda troca de dados do FOM será feita através do RTI;
d) durante a execução da federação, todos os federados irão interagir com o RTI em conformidade com a especificação de interface do HLA;
e) durante a execução da federação, um atributo de uma instância de um objeto poderá pertencer a apenas um único federado em um dado momento.

As cinco regras para o federado são:

a) federados devem possuir seus respectivos modelos de objeto de simulação (*Simulation Object Model* ou SOM, definição na seção 2.3.2.1) documentados em conformidade com o OMT;
b) federados poderão atualizar ou refletir atributos de objetos e enviar ou receber interações conforme especificados em seus respectivos SOMs;
c) federados poderão transferir ou aceitar posse de atributos conforme especificados em seus respectivos SOMs;
d) federados devem ser capazes de variar as condições sob as quais ocorrem as atualizações de atributos conforme especificados em seus respectivos SOMs;
e) federados devem ser capazes de gerenciar seus respectivos tempos locais de maneira que permita a coordenação de troca de dados com outros membros da federação.

2.3.2. Documento Base para Modelo de Objeto do HLA

Os documentos base de modelos de objetos (*Object Model Template* ou OMT) do HLA são descrições de elementos da federação ou do federado que podem ser compartilhados. O HLA requer a documentação de cada federação e de cada federado utilizando o modelo padrão da especificação para facilitar o reuso através do compartilhamento da informação. É importante tomar nota que o OMT, apesar de ser parcialmente baseado nas metodologias de análise e projeto orientado-a-objeto (*Object-Oriented Analysis and Design* ou OOAD), não implementa um paradigma genuinamente orientado-a-objeto. Um modelo OOAD descreve uma abstração de um sistema para melhor entender esse sistema. Em OOAD objetos são definidos em termos de seus atributos, operações e relacionamento com outros objetos. No HLA, o OMT focaliza-se nos requerimentos para troca de informação entre os componentes de simulação da federação, limitando a especificação para objetos, atributos, interações, parâmetros e apenas relacionamentos hierárquicos.

O OMT é composto pelos seguintes componentes que devem ser completados na especificação do modelo de objeto de federações ou de federados individuais:

a) tabela de identificação do modelo de objeto: Associa informações de identificação importantes com o modelo de objeto do HLA;
b) tabela de estrutura de classes de objeto: Registra os identificadores de todas as classes de objetos dos federados ou das federações e descreve as relações entre classes e subclasses;
c) tabela de estrutura de classes de interação: Registra os identificadores de todas as classes de interação dos federados ou das federações e descreve as relações entre classes e subclasses;
d) tabela de atributos: Especifica características de atributos de objetos em uma federação ou em um federado;
e) tabela de parâmetros: Especifica características de parâmetros de interações em uma federação ou em um federado;
f) tabela de dimensões: Especifica dimensões para filtrar instâncias de atributos e interações;

g) tabela de representação do tempo: Especifica a representação de valores de tempo;
h) tabela de rótulos fornecidos pelo usuário: Especifica a representação de rótulos (*tags*) usados nos serviços do HLA;
i) tabela de sincronização: Especifica a representação e tipos de dados usados nos serviços de sincronização do HLA. Federados são sincronizados através de um mecanismo do RTI que usa pontos de sincronização e na tabela de sincronização devem ser especificadas as condições sob quais esses pontos ocorrem;
j) tabela de tipos de transporte: Descreve os mecanismos de transporte usados. HLAreliable e HLAbestEffort são dois tipos que devem ser oferecidos por todos RTIs. Esses tipos e outros devem ser detalhados conforme a utilização requerida pela federação;
k) tabela de chaveamento (*switches*): Especifica os parâmetros iniciais de configuração usadas pelo RTI que podem ser habilitados ou desabilitados;
l) tabela de tipo de dados: Especifica detalhes da representação de dados no modelo de objeto;
m) tabela de anotação: Acrescenta maiores explicações sobre qualquer item da tabela OMT;
n) dicionário do FOM/SOM: Define todos os objetos, atributos, interações e parâmetros usados no modelo de objeto do HLA.

2.3.2.1. Modelos de Objetos: FOM, SOM e MOM

O HLA especifica o modelo de objeto da federação (*Federation Object Model* ou FOM) e o modelo de objeto da simulação (*Simulation Object Model* ou SOM) que devem estar no formato OMT. O FOM descreve um conjunto de objetos, atributos e interações que são compartilhadas na federação. O SOM descreve o federado em termos de tipos de objetos, atributos e interações que podem ser oferecidos a futuras federações.

O modelo de objeto de gerenciamento (*Management Object Model* ou MOM) consiste em um conjunto de classes e interações predefinidas que possibilitam a implementação de um

federado gerente para monitorar e controlar aspectos da federação usando os serviços padrões do RTI. O MOM utiliza o formato e sintaxe do OMT para definir esses elementos de controle e informação.

O MOM consiste de duas classes de objetos:

a) A HLAmanager.HLAfederate: Fornece informações sobre um federado específico;
b) A HLAmanager.HLAfederation: Fornece informações sobre uma federação em execução.

As interações do MOM estão divididas em quatro hierarquias:

a) HLAadjust: Podem ser instanciadas pelos federados para controlar aspectos da federação, dos federados e do RTI como a sincronização da atualização de atributos, posse de atributos e serviços de registro de atividades;
b) HLArequest e HLAreport: Informam sobre publicações, subscrições, posse, atualização, reflexão e alertas;
c) HLAservice: permite a invocação de serviços e controle sobre outros federados.

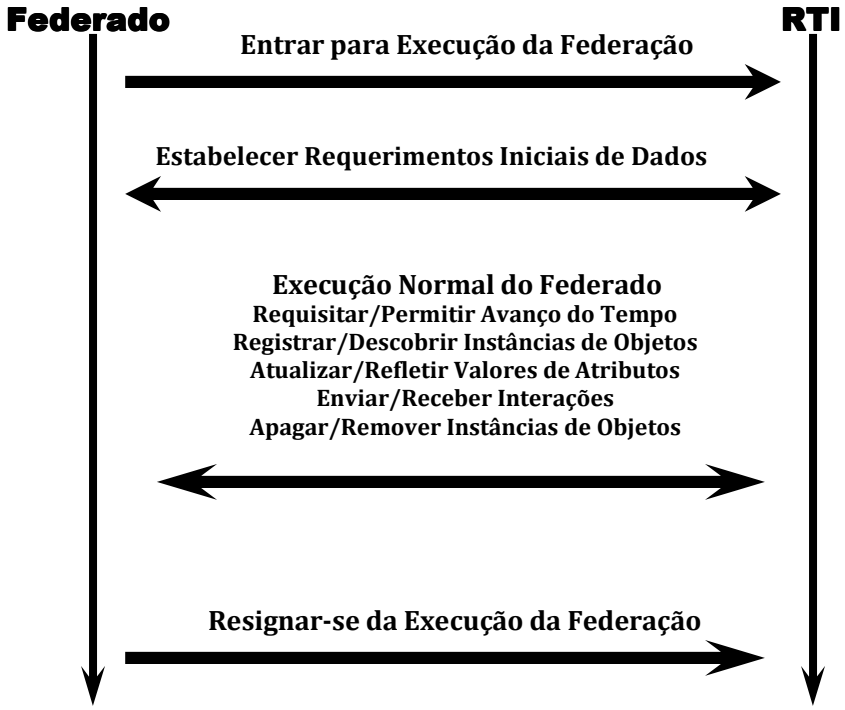
Além das classes de objetos e interações predefinidas, o MOM permite sua extensão através de subclasses, atributos ou parâmetros que podem ser utilizados para monitorar e controlar dados específicos através da aplicação.

2.3.3. Interface do Federado do HLA

A especificação de interface do HLA padroniza como os federados acessam os serviços do RTI. São definidas APIs em diversas linguagens de programação que incluem C++ e Java. A versão 1.3 do HLA (anterior à padronização IEEE) também definia as APIs para Ada95 (padrão ISO/ANSI da linguagem de programação Ada) e para CORBA IDL (mais detalhes sobre CORBA na seção 2.6).

Seis grupos de serviços do RTI fazem parte da especificação de interface: gerenciamento de federação, gerenciamento de declaração, gerenciamento de objeto, gerenciamento de posse,

gerenciamento de tempo e gerenciamento de distribuição de dados. O ciclo de vida de um federado e o seu relacionamento com o RTI é representado na Figura 3.



Fonte: Adaptado de Std. 1516.1-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification.

Figura 3. Visão Geral do Relacionamento entre o Federado e o RTI

Além desses seis grupos de serviços, existem os serviços de suporte, que são dependentes da implementação como a codificação e decodificação de identificadores (*handles*), manipulação de regiões, dar inicio ou terminar a execução do RTI, entre outros.

2.3.3.1. Gerenciamento de Federação

O gerenciamento de Federação (*Federation Management*) oferece serviços para criar ou destruir uma execução da federação, permitir a entrada ou saída de federados em federações existentes e interromper ou dar continuidade a uma execução (Tabela 1).

Tabela 1. Resumo dos Serviços do Gerenciamento de Federação.

AÇÃO	SEMÂNTICA
Criação	"Vamos jogar."
Entrada	"Eu também quero jogar."
Gravação	"Vamos gravar nosso estado."
Pausa	"Espere por um momento."
Saída	"Estou deixando o jogo."
Destruição	"Vamos finalizar o jogo."

2.3.3.2. Gerenciamento de Declaração

O gerenciamento de declaração (*Declaration Management*) oferece serviços que suportam a coordenação da troca de dados entre os federados baseado em interesse (Tabela 2). Os federados precisam fornecer as informações necessárias sobre as classes de objetos e seus atributos e sobre as classes de interações que desejam publicar ou subscrever durante a execução da federação.

Tabela 2. Resumo dos Serviços do Gerenciamento de Declaração.

AÇÃO	SEMÂNTICA
Subscrição	"Eu quero receber informações sobre isto." "Eu não quero mais receber informações sobre isto."
Publicação	"Estas são as informações que irei enviar."
Controle	"Atenção, alguém quer informações sobre aquilo."

2.3.3.3. Gerenciamento de Objeto

O gerenciamento de objeto (*Object Management*) oferece serviços para criação ou remoção de instâncias de objetos, atualização ou reflexão de atributos de instâncias de objetos e o envio ou recebimento de interações (Tabela 3).

Tabela 3. Resumo dos Serviços do Gerenciamento de Objeto.

AÇÃO	SEMÂNTICA
Registro de Objeto	"Eu tenho um novo tanque."
Atualização de Atributo	"Um de meus aviões mudou de direção."
Envio de Interação	"Vôo 501 pedindo permissão para aterrissar."
Remoção de Objeto	"Um caminhão saiu da visão."
Alteração do Tipo de Transporte	"O nível de combustível deste carro não é importante."
Alteração do Tipo de Ordenação	"A chegada dos suprimentos de comida pode ser notificada fora de ordem."

2.3.3.4. Gerenciamento de Posse

O gerenciamento de posse (*Ownership Management*) oferece serviços de suporte para a transferência dinâmica da posse de objetos e de atributos de objetos entre os federados durante uma execução (Tabela 4). O federado que tem posse de um atributo de um objeto tem o direito exclusivo de atualizar esse atributo.

Tabela 4. Resumo dos Serviços do Gerenciamento de Posse.

AÇÃO	SEMÂNTICA
Negação	"Eu não posso mais rastrear este avião."
Aquisição	"Obrigado, eu aceito a responsabilidade sobre este carregamento."
Pesquisa	"Quem está gerenciando este caminhão?"

Os métodos de troca de posse utilizam o modelo *push/pull*:

a) *push*: Um federado pode tentar se livrar da responsabilidade de um ou mais atributos de uma instância de objeto, mas não pode forçar outros federados a adquirir essa posse.
b) *pull*: Um federado pode tentar adquirir a responsabilidade de um ou mais atributos de uma instância de objeto, mas não pode usurpar essa posse de outros federados.

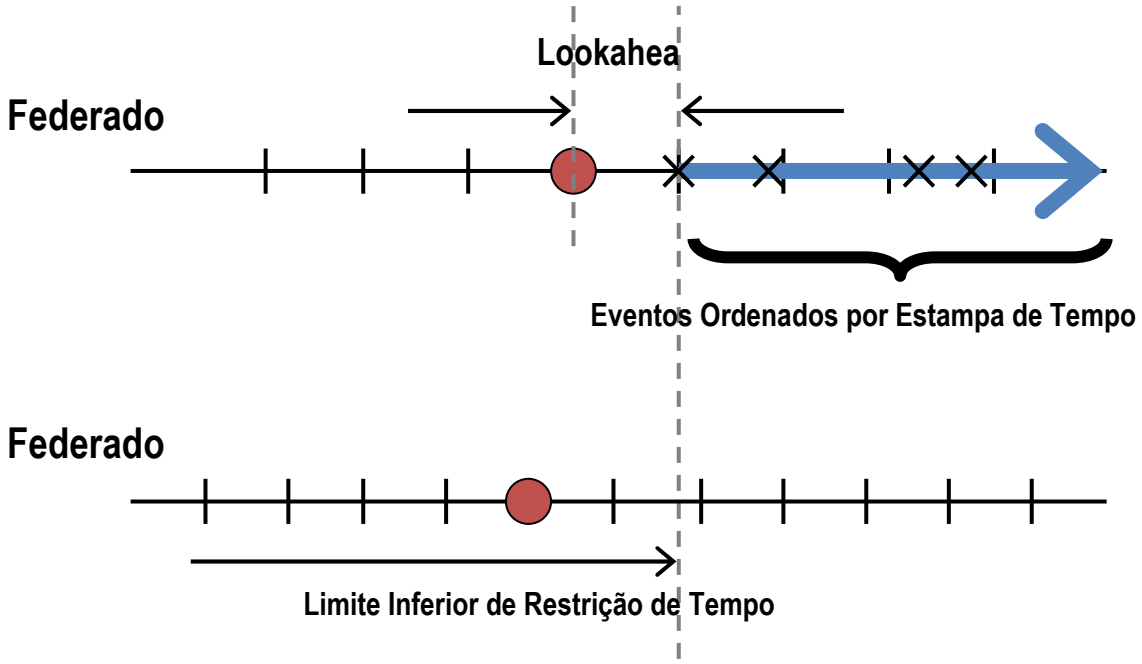
O privilégio para remover é um atributo especial que existe em todas as instâncias de objetos. O federado que tem posse desse atributo tem o direito de remover a respectiva instância de objeto. A posse do privilégio para remover pode ser trocado entre os federados como qualquer outro atributo.

2.3.3.5. Gerenciamento de Tempo

O gerenciamento de tempo (*Time Management*) oferece serviços que coordenam o avanço do tempo e a sincronização da troca de dados da simulação em tempo de execução.

O HLA suporta uma variedade de políticas de gerenciamento de tempo e antecipa a interoperabilidade entre federados que adotam diferentes políticas. Na federação o tempo sempre avança, mas a percepção de tempo corrente pode variar entre os federados participantes (cada federado gerencia seu tempo local).

Em algumas situações, o andamento de um federado "regulador" (*regulating*) pode restringir o andamento de um federado "constrito" (*constricted*). Na Figura 4 é apresentado um exemplo com um federado regulador e um federado constrito onde o tempo corrente de cada federado é representado pelos círculos em um eixo do tempo. O *lookahead* é o menor intervalo de tempo em que pode ocorrer um evento ordenado por estampa de tempo (*Time-Stamp-Ordered* ou TSO) gerado por um federado regulador. Todo federado constrito tem uma estampa de tempo de limite inferior (*Lower Bound Time Stamp* ou LBTS) que especifica quando será recebido o próximo evento TSO. Um federado constrito não pode avançar seu tempo além do LBTS.



Fonte: Adaptado de High Level Architecture Run-Time Infrastructure Programmer's Guide RTI1.3 Version6.

Figura 4. Federado Regulador e Federado Constrito.

2.3.3.6. Gerenciamento de Distribuição de Dados

O gerenciamento de distribuição de dados (*Data Distribution Management* ou DDM) oferece serviços que suportam o roteamento eficiente de dados entre os federados durante a execução da federação. Os federados podem definir regiões em um espaço de roteamento conforme definido na tabela de dimensões para filtrar os dados recebidos através das subscrições ou enviados pelas publicações. Ao associar regiões a atributos de objetos e interações, somente haverá troca de dados entre os federados se houver intersecção entre suas regiões de publicação e de subscrição conforme o exemplo ilustrado na Figura 5. Neste exemplo são utilizadas regiões tridimensionais onde um federado #1 definiu uma região de publicação "A" e um federado #2 definiu uma região de subscrição "B", ambos para uma mesma classe de objeto. A intersecção das regiões A e B garantem que as atualizações enviadas pelo federado 1 serão recebidas e refletidas no federado 2.

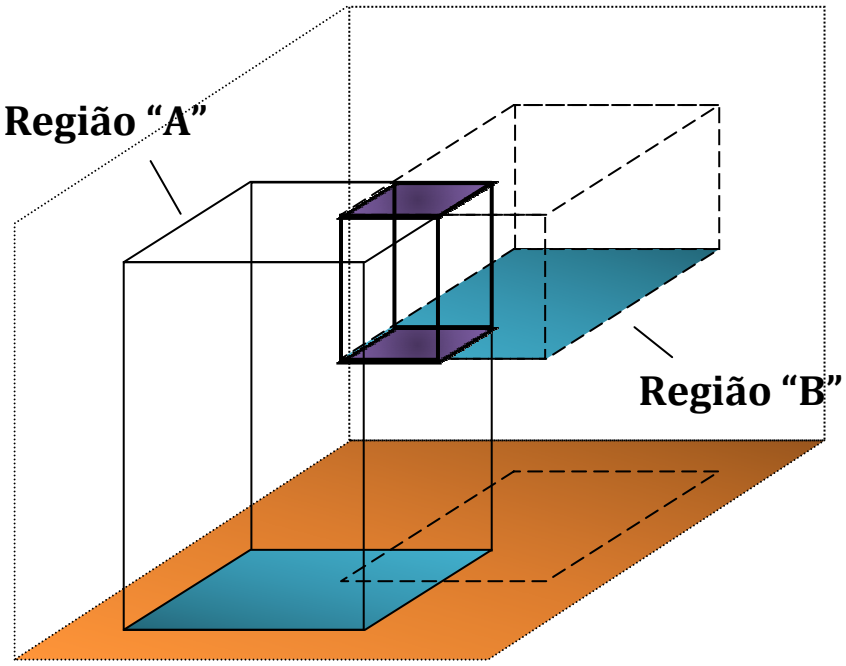


Figura 5. Exemplo de Espaço de Roteamento Tridimensional

2.3.4. HLA API Compatível com Ligação Dinâmica

Apesar dos objetivos do HLA serem interoperabilidade e reuso de modelos de simulação, a implementação dos federados e da federação dependem das implementações atualmente disponíveis do RTI, em sua maioria em C++ e Java. A limitação da portabilidade do código que era possível apenas pelos esforços voluntários dos fornecedores do RTI foi resolvida com a padronização da API do HLA compativel com ligação dinâmica (*Dynamic Link Compatible* ou DLC) pela Simulation Interoperability Standards Organization (http://www.sisostds.org/). O DLC possibilita a troca entre diferentes implementações do RTI sem precisar recompilar ou religar o federado desde que eles estejam na mesma plataforma (ou em plataforma compatível).

2.4. SIMULAÇÃO BASEADA NA WEB E SERVIÇOS WEB

Mais do que uma fonte de informações, a rede mundial de computadores (World *Wide Web* ou simplesmente Web) tem demonstrado ser uma poderosa plataforma de aplicações (O'Reilly 2005). Uma aplicação Web (*webapp*) é uma aplicação acessada por um navegador Web (browser) em uma rede como a Internet. As webapps tornaram-se populares pela ubiqüidade do browser como cliente, pois as aplicações não precisam ser instaladas ou atualizadas nos milhares de computadores que podem ter acesso a elas.

Simulação baseada na Web é uma idéia que representa a conexão entre a Web e a área de simulação (Fishwick 1996). O impacto das tecnologias Web para acessar e executar programas legados de simulação remotamente através do browser e o impacto para modelagem e simulação distribuídas são algumas das principais questões abordadas por Page (1998).

Tecnologias baseadas na linguagem de programação Java foram extensivamente utilizadas desde as primeiras abordagens de modelagem e simulação distribuída na Web (Kuljis e Paul 2000, 2003). Os trabalhos realizados no Extensive Modeling and Simulation Framework (XMSF) prometem a interoperabilidade entre simulações legadas e no padrão HLA através de serviços Web (*Web Services*) e futuramente através da computação em grade (Brutzman et al. 2002; Pullen et al. 2005).

Serviços Web oferecem um arcabouço extensivo para interação entre aplicações implementadas em diferentes linguagens e operando em diferentes sistemas através de protocolos Web e padrões abertos XML (Curbera et al. 2002). A especificação desse arcabouço é divida em três áreas: protocolos de comunicação, descrição de serviços e descoberta de serviços. Os padrões da indústria mais utilizados na implementação dos serviços são: o protocolo SOAP (*Simple Object Access Protocol*) para comunicação, WSDL (*Web Services Description Language*) para descrição formal de serviços e o diretório UDDI (*Universal Description, Discovery and Integration*) para registro e descoberta.

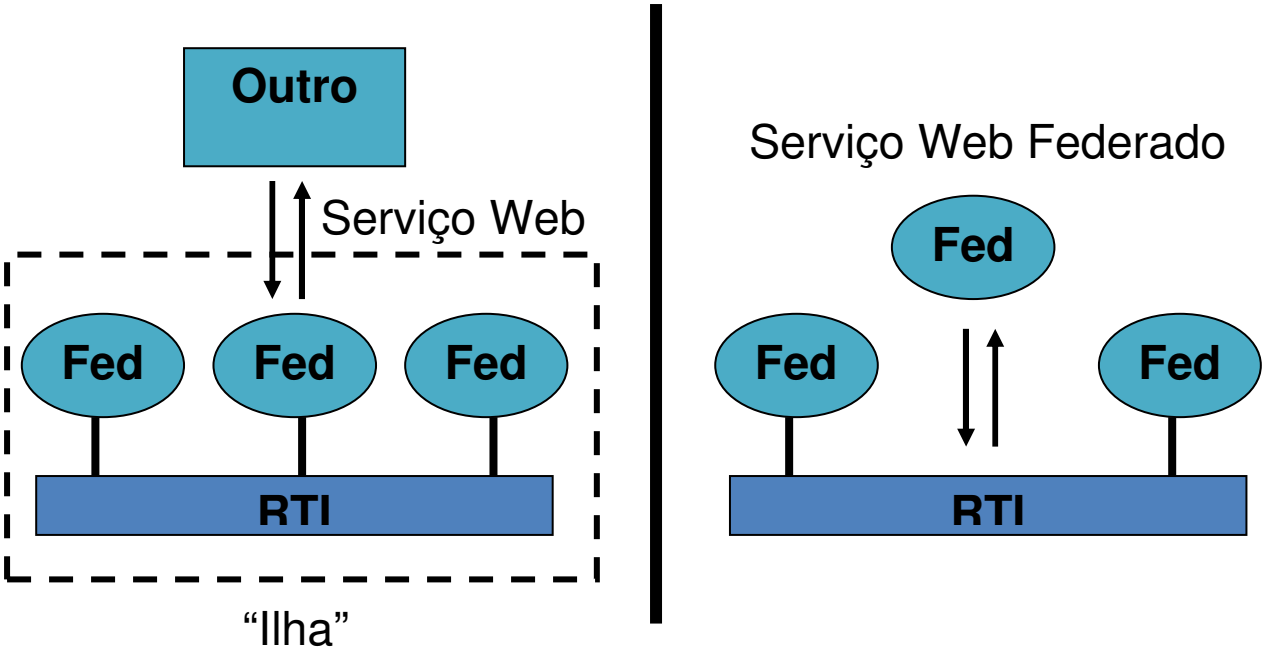
Com os serviços Web, aplicações podem fazer requisições de serviços através de uma WAN usando o protocolo HTTP para trocar mensagens formatadas em XML. Já existem muitos arcabouços de software para gerar o código do provedor e do consumidor de serviços a partir do

WSDL para várias linguagens de programação, entre elas C++, C#, Java, J2ME, Visual Basic, ADA, PL/1, Fortran e COBOL.

2.5. EVOLUÇÃO DO HLA: HLA-EVOLVED

A especificação do HLA está sendo revisada para incluir uma nova API baseada em serviços Web (HLA WSDL API). Essa revisão, o IEEE 1516-2008 tem sido chamada de HLA-Evolved (Möller e Dahlin 2006; Möller e Löf 2006). Apesar da padronização do DLC HLA API resolver problemas de portabilidade do código do federado entre diferentes implementações do RTI, a interoperabilidade entre diferentes plataformas ainda depende de funcionalidades oferecidas pelos fornecedores do software. Com crescente uso, os serviços Web oferecem outro nível de interoperabilidade entre sistemas que vem complementar o HLA.

Os usos de serviço Web com HLA, como os descritos no XMSF, permitem que outros sistemas acessem os dados da federação através de um federado que atua como uma "ponte" (*proxy*). Esse federado-ponte define serviços específicos em conjunto com outros federados formando uma "ilha" (Figura 6). Com a padronização da HLA WSDL API, federados podem ser implementados para acessar a federação pelo RTI usando serviços Web. Ao contrário da abordagem usando ilha, o federado baseado na Web tem acesso a todos os serviços do RTI e aos dados da federação.



Fonte: Adaptado de Möller e Dahlin (2006).

Figura 6. Acesso ao RTI Usando Serviços Web.

Apesar do HLA com serviços Web ter menor desempenho em comparação com a API em C++ ou em Java, o seu uso permite o acesso de diferentes aplicações em diferentes dispositivos por WAN. Com o WSDL é possível gerar códigos em diferentes linguagens de programação, deixando transparente para a federação se o federado foi implementado em C++, em Java ou em qualquer outra linguagem.

2.6. CORBA, JAVA RMI E HLA

CORBA define a Interface Definition Language (IDL), uma linguagem para especificar as interfaces dos objetos compartilhados. Quando o IDL é compilado, são gerados o esqueleto para a implementação do objeto compartilhado no servidor e os stubs para os clientes acessarem esse objeto no servidor através de referências. O Object Request Broker (ORB) é o principal

componente da arquitetura do CORBA que fornece os mecanismos para a comunicação entre o cliente e o servidor.

No artigo de Buss e Leroy (1998) são realizadas comparações entre três arquiteturas de suporte para a execução de simulações distribuídas: HLA, CORBA e RMI. Para a comparação, três elementos básicos de arquiteturas distribuídas foram analisados: linguagens de programação suportadas, hardware e sistema operacional, e protocolo de rede utilizado. A arquitetura do HLA, conforme foi discutida anteriormente, não especifica nenhuma linguagem para implementação (somente as interfaces nas linguagens de programação mais comuns) e nem um protocolo de comunicação específico. Em comparação, RMI é implementada em Java e o protocolo utilizado é o TCP/IP. CORBA define o seu próprio protocolo, o Internet Inter-Orb (IIOP) que é implementado sobre o TCP/IP. CORBA e RMI são recomendados para aplicações distribuídas genéricas enquanto o HLA é recomendado para simulações distribuídas, principalmente para aquelas que envolvem modelos de simulação legados escritos em diferentes linguagens.

O RTI possui requisitos de operações em tempo-real para simulação distribuidas que não foram cumpridas em implementações utilizando CORBA (Schug et al. 1997; Au 2000) por causa do aumento no overhead de processamento e latência na comunicação.

2.7. CONSIDERAÇÕES FINAIS

Neste capítulo foram revisados alguns conceitos importantes de simulações distribuídas e foram apresentadas as principais características do padrão HLA. Uma simulação modelada conforme as especificações do HLA pode ser reutilizada para operar em conjunto com outras simulações e aplicações para compor uma complexa simulação distribuída. Cada federado pode se dedicar a processar um determinado aspecto dessa simulação distribuída e disponibilizar apenas os dados resultantes de sua computação que sejam relevantes para outros participantes da mesma federação. Além disso, não é preciso que exista uma interface específica para coleta e visualização de dados em cada federado. Um único federado dedicado para essa tarefa pode coletar dados de toda federação de maneira transparente através dos serviços do HLA para gerar a

visualização desses dados. Também é possível que um federado controle remotamente alguns aspectos de outros federados definindo interações para as quais eles devem respondem.

A possibilidade de separar os aspectos computacionais de uma simulação e suas interfaces de visualização e de interação é um fator determinante para a escalabilidade de um sistema distribuído. A infra-estrutura oferecida pelo RTI fornece todos os mecanismos necessários para a interoperabilidade entre as simulações e aplicações heterogêneas, o que permite o foco no desenvolvimento da simulação em si.

A existência do RTI como um processo central para coordenar a troca de dados entre os componentes de uma simulação distribuída contradiz os conceitos apresentados no início deste capítulo. Porém, a especificação do HLA apenas define os serviços que devem ser oferecidos pelo RTI e não define nenhuma linguagem de programação ou tecnologia para sua implementação. A escolha da implementação do RTI é totalmente dependente do desenvolvedor. Por ser uma tecnologia dedicada, existem poucas implementações de uso livre disponíveis para o público em geral. Nas implementações em código-aberto ou de software livre disponíveis, o RTI funciona como um processo central em máquinas com multiprocessadores ou distribuídas em rede. Para fins de pesquisa, podemos assumir a existência de RTIs que operam distribuidamente, pois esse também é um tópico de pesquisa e desenvolvimento do padrão.

Uma questão importante de simulações distribuídas é a visualização das mesmas. O próximo capítulo apresenta os principais aspectos da visualização.

3. VISUALIZAÇÃO 3D BASEADA NA WEB COM EXTENSIBLE 3D

3.1. CONSIDERAÇÕES INICIAIS

Nesta seção serão feitas revisões sobre conceitos de visualização e as técnicas para visualização de simulações e ambientes virtuais. Também serão apresentadas as principais características do padrão Extensible 3D e suas funcionalidades para visualização e interação com simulações. Os conceitos de visualização a serem apresentados ressaltam a importância na elaboração da estratégia para extrair dados de uma simulação e como apresentar esses dados de maneira eficiente ao usuário. A especificação da visualização (maneira como se apresenta os dados) pode ser mais eficiente se houverem mecanismos que permitam ao usuário adaptá-la conforme o seu nível de especialização.

3.2. VISUALIZAÇÃO E REALIDADE VIRTUAL

Um grande volume de dados pode ser adquirido de diversas fontes como câmeras e sensores ou geradas a partir de complexas simulações em computadores. A ciência da visualização é caracterizada pela transformação desses dados em representações visuais através da computação gráfica (McCormick 1988). A visualização desses dados permite a detecção de padrões e características interessantes de maneira efetiva e eficiente graças às capacidades únicas do sistema visual humano.

Além de um resultado visual, o termo visualização também pode se referir à disciplina de pesquisa, a uma tecnologia ou a uma técnica especifica. Quando consideramos a visualização como uma tecnologia, podemos medi-la conforme sua efetividade e eficiência. Em outras palavras, a visualização deve fazer o que se propõe a fazer, utilizando-se a menor quantidade de recursos possíveis (van Wijk 2004). Percebe-se que não podemos julgar a visualização em si e que é preciso levar em conta o contexto na qual ela se insere.

A realidade virtual tem um grande impacto como interface de visualização e interação com simulações (Bryson 1996). A realidade virtual pode ser utilizada com naturalidade para interagir com visualizações tridimensionais geradas a partir de um grande volume de dados de natureza espacial.

3.2.1. Visualização Científica e Visualização de Informação

Citando Mackinlay (2004):

> *"A visualização se divide grosseiramente em duas áreas dependendo se dados físicos estão envolvidos. Visualização científica (...) focaliza-se nos dados físicos como o corpo humano, a terra, moléculas e assim por diante. Visualização da informação focaliza-se nos dados abstratos ou não-físicos, como textos, hierarquias e dados estatísticos." (tradução livre)*

Para complementar com Gershon e Eick (1995), a visualização da informação é um processo de transformação de dados e informações, que não são inerentemente espaciais, em uma forma visual, que permite ao usuário observar e entender essa informação. Em contraste, a visualização científica freqüentemente focaliza-se em dados espaciais gerados por processos científicos.

Apesar dessas definições, Rhyne (2003) discute se realmente existe a necessidade da divisão da visualização em duas áreas distintas já que "visualização da informação não é não-científica e visualização científica não é não-informativa", uma conotação infeliz quando justapostas (Munzner 2002).

3.2.2. Problemas e Desafios na Visualização

Heckbert (1987) e Blinn (1998) apontaram 10 problemas não resolvidos da visualização. Esses problemas envolviam questões de eficiência algorítmica e de capacidade de hardware e também preocupações com a organização da comunidade de pesquisa. Hibbard (1999) criou uma nova lista, com problemas classificados nas categorias de:

a) qualidade visual: Gráficos de alto desempenho e integração da realidade virtual;
b) integração: Integração da visualização com outras tecnologias computacionais;
c) informação: Como representar a informação visualmente;
d) interações: Interações do usuário com as visualizações através de gestos e reconhecimento de voz;
e) abstrações: Facilitar a construção e o uso de sistemas de visualizações.

Johnson e Sanderson (2003) fazem uma discussão de como visualizações tridimensionais não representam erros e incertezas. Variações do resultado de uma técnica ou tecnologia para outra são percebidas somente quando comparadas umas com as outras e os erros e incertezas podem surgir:

a) na aquisição (erro de medida instrumental);
b) no modelo (matemático ou geométrico);
c) na transformação (erros introduzidos na reamostragem, na filtragem, na quantização e na mudança de escala);
d) na visualização.

Esses erros e incertezas são críticos na reprodução tridimensional de dados em áreas de aplicações médicas e geoespaciais, por exemplo. A partir disso, uma nova lista foi elaborada por Johnson (2004), que além da representação dos erros e incertezas discute as seguintes questões:

a) pensar na ciência;
b) efetividade quantitativa;
c) percepção;
d) hardware gráfico;

e) interação humano-computador;
f) visualização global/local;
g) ambientes para solução de problemas integrados;
h) visualização multi-campo;
i) integração da visualização científica e da informação;
j) detecção de características;
k) visualização dependente do tempo;
l) visualização escalável e distribuída baseada em grade;
m) abstrações visuais;
n) teoria da visualização.

O que se observa nas listas dos problemas da visualização que os autores compilaram durante os anos é que as preocupações com a otimização de algoritmos de renderização é colocado em segundo plano diante das aplicações com as novas tecnologias de hardware e software conforme vão surgindo. Apesar de ainda existir a preocupação no desempenho gráfico, a visualização efetiva e eficiente é medida pela incorporação de outros sistemas e pelas facilidades disponibilizadas para o usuário.

3.2.3. Simulação e Visualização Interativas

Com o aumento da complexidade de simulações um enorme volume de dados é gerado para ser analisado e visualizado. Tradicionalmente, a análise e visualização de dados é uma etapa que vem após o término de uma simulação. Dessa forma, possíveis erros que invalidarão os resultados serão somente detectados após o processamento da simulação. Surge então o desejo de visualizar os dados e a possibilidade de alterar os parâmetros durante a simulação. Johnson et al. (1999) apontam duas abordagens para a interatividade: algoritmos eficientes para apresentação dos dados e acesso eficiente aos dados.

De acordo com Marshall et al. (1990), técnicas de visualização são classificadas de acordo com o grau de integração entre a execução da simulação e a visualização das imagens resultantes: pós-processamento (*post-processing*), monitoramento (*tracking*) e controle (*steering*).

No pós-processamento a visualização é gerada após o final da simulação. O monitoramento permite a visualização durante a simulação, mas é necessário interromper a simulação para alteração de parâmetros. O controle permite a alteração dos parâmetros da simulação durante a visualização.

Os conceitos de monitoramento e controle se aplicam para as simulações interativas distribuídas onde há a necessidade de acompanhar um treinamento ou participar diretamente dele. Já o pós-processamento é comum nas simulações paralelas e distribuídas, principalmente nas simulações on-line onde suas execuções precisam ser rápidas.

3.2.4. Ambientes Virtuais e Realidade Virtual

Um ambiente virtual é um mundo altamente gráfico gerado por computador com o qual o usuário pode interagir.

A realidade virtual é tipicamente retratada como um meio, como um telefone ou um televisor, ou seja, definido em termos de uma coleção de hardware tecnológico. Em termos da experiência humana, realidade virtual é definida pelo conceito da presença, ou seja, a sensação de estar em um ambiente (Steuer 1992; Schubert et al. 1999; Ijsselsteijn et al. 2000). A presença é um efeito psicológico.

Ainda, Steuer (1992) define o termo telepresença como a experiência da presença em um ambiente apresentado através de um meio de comunicação, em detrimento do ambiente físico no qual alguém se encontra. Isso também é conhecido como suspensão da incredibilidade, a habilidade de ignorar a interface e concentrar na aplicação.

Nessa linha de pensamento, apesar da realidade virtual enfatizar o uso de gráficos 3-D de alta resolução e equipamentos de alto desempenho, existem as chamadas realidades virtuais baseadas em texto (Hand 1994), conectadas à Internet e também conhecidas como MUD (*Multi-User Dungeon/Domain/Dimension*). Como o nome diz, o serviço oferecido é semelhante aos das salas de bate-papos, mas permite uma maior interação entre os usuários através de mecanismos textuais.

Imersão é o termo utilizado para o grau de estímulo visual que uma interface de realidade virtual oferece. Esse termo é mais adequado para as realidades virtuais nas quais imagens geradas por software cercam ou submergem os participantes (Schubert 1999). O grau de imersão varia conforme os dispositivos de interação como capacetes e luvas. De acordo com Bryson (1992) é a interface e não o conteúdo que caracteriza a realidade virtual.

As pesquisas no campo da realidade virtual dividem-se em quatro paradigmas baseados a princípio no dispositivo de visualização utilizado que pode ser:

a) monitores dos tipos CRT (tubo de raios catódicos), LCD (monitor de cristal líquido), LCoS (cristal líquido em silício) ou OLED (diodos orgânicos emissores de luz);
b) visor montado em capacete (*Head Mounted Display* ou HMD): Capacete ou óculos que contém uma ou duas pequenas telas CRT, LCD, LCoS ou OLED que cobrem os olhos do usuário. Um dispositivo para rastrear os movimentos da cabeça do usuário é usado para refletir a localização e orientação na simulação;
c) monitor binocular onidirecional (*Binocular Omni-Oriented Monitor* ou BOOM): Semelhante ao HMD, mas é suspenso por um braço articulado que mede a posição e orientação no espaço;
d) caverna digital (*Cave Automatic Virtual Environment* ou CAVE): ambiente de visualização onde as imagens são projetadas em volta do usuário, nas paredes e até mesmo no chão de uma sala.

Cinco principais questões são apresentadas por Cruz-Neira et al. (1992) sobre imersão:

a) campo de visão: É a extensão angular que um observador enxerga sem mover a cabeça;
b) panorama: É a habilidade do dispositivo de exibição envolver o observador;
c) perspectiva centrada no observador: Depende da velocidade e da precisão em detectar a localização do observador;
d) representação física e do corpo: Em interfaces do tipo HMD e BOOM a representação do corpo é explicita, ou seja, precisa ser representada geometricamente pela simulação no ambiente. No caso do CRT e CAVE, a representação é implícita, pois o corpo é fisicamente visível e não precisa ser renderizado;

e) intrusão: Depende do grau com que a interface restringe os sentidos, isolando o usuário do ambiente real.

Ainda em Cruz-Neira et al. (1992) é avaliada a utilidade da realidade virtual como ferramenta através da efetividade da visualização, onde são abordadas cinco questões:

a) acuidade visual: É a qualidade da resolução combinada com o campo de visão de uma interface de realidade virtual;
b) linearidade: Verifica a existência de distorções da imagem devido à menor resolução em algumas áreas do dispositivo de visualização;
c) olhar em volta: É a possibilidade de se mover em torno de um objeto para visualizá-lo em diferentes ângulos;
d) refinamento progressivo: É a habilidade de aumentar dinamicamente a resolução de um modelo quando o usuário não está em movimento;
e) colaboração: A possibilidade de permitir vários usuários em um mesmo ambiente sem modificar a aplicação.

As questões sobre imersão e visualização apresentadas anteriormente também são discutidas com maiores detalhes em Brooks Jr. (1999), onde são analisadas as tecnologias essenciais para realidade virtual.

3.2.5. Ambientes Virtuais Distribuídos

Ambientes Virtuais Distribuídos (*Distributed Virtual Environments* ou DVEs) são mundos gerados por computador no qual um usuário pode se conectar e interagir com outros usuários remotamente através de uma rede de comunicação. O termo Ambiente Virtual Colaborativo (*Collaborative Virtual Environment* ou CVE) também é utilizado quando um sistema é modelado especificamente para dar suporte a atividades cooperativas e colaborativas em ambientes virtuais (Churchill e Snowdon 1998).

Waters e Barrus (1997) discutem algumas características chaves de DVEs:

a) DVEs devem permitir grupos de usuários separados geograficamente a interagirem uns com os outros em tempo real;
b) usuários devem imergir em um ambiente tridimensional (visual e auditivo) e serem representados por um avatar. Um avatar pode ser uma representação gráfica gerada pelo computador, um vídeo do usuário ou uma combinação de ambos;
c) os usuários, além de interagirem uns com os outros, poderão interagir com simulações de computadores;
d) a interação entre os usuários deve ocorrer através de mecanismos naturais como a fala;
e) um DVE, mesmo em uso, poderá ser alterado e crescer dinamicamente, aceitando contribuições de objetos e estruturas de variadas fontes;
f) o sistema deverá ser executado em hardwares de baixo custo para permitir o acesso a um publico maior.

Basicamente existem três tipos de arquiteturas de comunicação para sistemas DVE: cliente-servidor, peer-to-peer e híbrida. Conforme as comparações realizadas no trabalho de Ng et al. (2003), os sistemas NPSNET (Falby et al. 1993), DIVE (Carlsson e Hagsand 1993) e MASSIVE-2 (Greenhalgh e Benford 1995, 1997) são implementados com a arquitetura peer-to-peer onde todos os clientes do sistema se comunicam diretamente uns com os outros para a entrega rápida de informações. Já o MASSIVE-3 (Greenhalgh et al. 2000) é implementado com a arquitetura cliente-servidor para manter a persistência do mundo virtual e a consistência dos dados. E, combinando as vantagens da arquitetura peer-to-peer e cliente-servidor, sistemas como o SPLINE (Waters et al. 1997) é implementado com a arquitetura hibrida.

3.2.5.1. Modelo Cliente-Servidor

O modelo cliente-servidor é uma arquitetura na qual um processo ou um computador na rede assume o papel de cliente ou de servidor. Um servidor se dedica a oferecer recursos e responder às requisições dos clientes. Os clientes não compartilham seus recursos.

As vantagens desse modelo são:

a) segurança e maior controle, pois somente usuários com permissão podem ter acesso aos recursos e alterar os dados que se encontram nos servidores;
b) facilidade para manutenção e atualização dos recursos.

As desvantagens mais conhecidas são:

a) congestionamento quando o número de requisições é alto;
b) clientes não têm acesso aos recursos quando o servidor não está disponível.

3.2.5.2. Modelo Peer-to-Peer

Nas arquiteturas Peer-to-Peer (ou P2P) é estabelecida uma rede virtual de computadores onde todos os pares (*peers*) possuem as mesmas responsabilidades ou capacidade de troca de informação direta entre elas. Diferente do modelo cliente-servidor, um participante da rede P2P pode assumir o papel de cliente ou servidor em determinadas situações ou cliente e servidor simultaneamente (*servent*). É classificada como uma rede P2P "pura" aquela em que todos pares podem compartilhar serviços para o funcionamento da rede e a saída de um par não interrompe o acesso a esses serviços (Schollmeier 2001). Quando algum recurso é centralizado para aumentar a eficiência de uma rede P2P, essa rede é classificada como "híbrida". Uma das grandes desvantagens da rede P2P pura é a complexidade dos mecanismos de buscas por um determinado recurso. A rede P2P híbrida busca combinar as vantagens do modelo P2P e o modelo cliente-servidor. Para aumentar o desempenho das pesquisas na rede, a abordagem mais comum é a centralização de um recurso como um servidor para indexação (Yang e Garcia-Molina 2001).

3.3. EXTENSIBLE 3D

Extensible 3D (X3D) é um padrão aberto da ISO (*International Organization for Standardization*) mantido pela Web 3D Consortium (http://www.web3d.org/) para distribuição de conteúdo 3D interativo, sucessor do VRML (*Virtual Reality Modeling Language*). O VRML é

um formato de arquivo utilizado para descrever geometrias tridimensionais e foi muito popular para a criação de mundos virtuais hospedados em páginas da Web e renderizados através de plugins ou navegadores 3D (*browsers 3D*). A especificação X3D não define somente os aspectos visuais, mas também determina o comportamento dos objetos 3D descritos.

Bullard (2003) discute a importância do VRML e do X3D:

a) manipular modelos 3D para análise é mais intuitivo do que interpretar imagens 2D;
b) VRML é usada para simulações 3D em tempo-real;
c) VRML não é uma sopa de rótulos;
d) o principal objetivo do VRML/X3D é a interoperabilidade entre aplicações 3D.

O termo VRML foi cunhado em 1994 e tinha como objetivo descrever na Web mundos tridimensionais da mesma forma que o HTML descrevia formulários bidimensionais. A primeira especificação definiu uma API e formato de arquivo semelhante ao do Open Inventor, uma API de gráficos 3D da Silicon Graphics. O Open Inventor era uma camada de nível mais alto para programação utilizando OpenGL que fazia uso do grafo de cena. O VRML e o X3D também utilizam o grafo de cena que é uma estrutura de dados onde os nós representam entidades ou objetos na cena.

A sintaxe original do VRML 1.0 e VRML97 são semelhantes ao da linguagem de programação C e C++ no sentido em que usa os caracteres de abre e fecha chaves para definir blocos. No X3D essa sintaxe ainda é válida, mas é recomendada a codificação em XML. A adoção do XML justifica-se pela necessidade de:

a) interoperabilidade com a Web;
b) incorporação padronizada de novas tecnologias gráficas;
c) mesmo conteúdo executando da mesma maneira em navegadores diferentes;
d) um padrão que evolui com freqüência.

3.3.1. Conceitos e Visão Geral do Ambiente de Execução

A seguir serão apresentados alguns detalhes do ambiente de execução do X3D e as funcionalidades necessárias para montar uma cena usando os componentes básicos.

3.3.1.1. Hierarquia de Transformação de Nós

O grafo de cena do X3D é um grafo direto acíclico. A especificação define diversos tipos de nós organizados em grupos de componentes que incluem primitivas geométricas, propriedades da aparência, som e propriedades do som e diversos tipos de nós de agrupamento. Os nós armazenam dados em campos e podem conter campos específicos que podem conter outros nós que também poderão conter mais nós, formando uma hierarquia. Existem nós de agrupamentos específicos que contém informações espaciais que quando modificadas afetam todos os seus nós descendentes.

Na figura 7 é ilustrado um exemplo de um grafo de cena. As esferas representam nós e os quadrados representam campos de nós, nomeados conforme a especificação. Nesse exemplo é descrito uma esfera vermelha de 2.5 unidades de raio e um cubo azul de duas unidades de aresta (valor padrão). O cubo está rotacionado 45 graus (0.54 em radianos) no eixo y. A esfera está transladada duas unidades no eixo x e uma unidade no eixo y referente à raiz que é o primeiro nó de transformação. O cubo está transladado três unidades no eixo x referentes ao segundo nó de transformação. Os nós de transformação não são elementos visuais, mas são afetados pelas transformações na hierarquia. No total, o cubo foi transladado cinco unidades no eixo x e uma unidade no eixo y a partir da raiz.

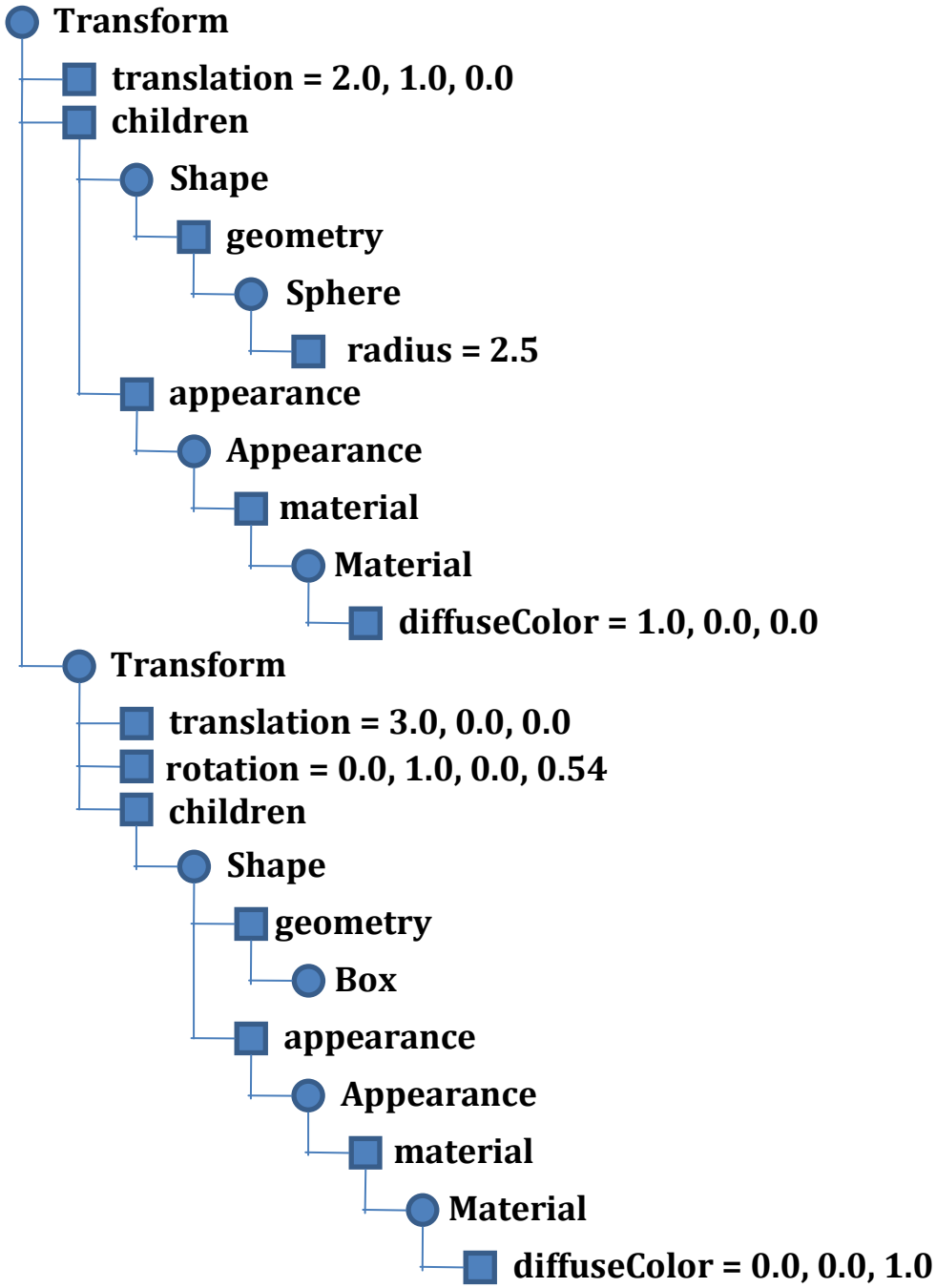


Figura 7. Representação da Hierarquia do Grafo de Cena

A seguir, as Figuras 8 e 9 mostram o código que gera o grafo da Figura 7.

```
<Transform translation="2 1 0">
  <Shape>
    <Appearance>
      <Material diffuseColor="1 0 0"/>
    </Appearance>
    <Sphere radius="2.5"/>
  </Shape>
  <Transform rotation="0 1 0 0.54" translation="3 0 0">
    <Shape>
      <Appearance>
        <Material diffuseColor="0 0 1"/>
      </Appearance>
      <Box/>
    </Shape>
  </Transform>
</Transform>
```

Figura 8. Descrição da Cena no Formato X3D (XML).

```
Transform {
  children [
    Shape {
      geometry Sphere {
        radius 2.5
      }
      appearance Appearance {
        material Material {
          diffuseColor 1 0 0
        }
      }
    }
    Transform {
      children [
        Shape {
          geometry Box {}
          appearance Appearance {
            material Material {
              diffuseColor 0 0 1
            }
          }
        }
      ]
      rotation 0 1 0 0.54
      translation 3 0 0
    }
  ]
  translation 2 1 0
}
```

Figura 9. Descrição da Cena no Formato VRML Clássico.

3.3.1.2. Modelo de Eventos

Alguns nós geram eventos em resposta a mudanças no ambiente ou interações do usuário. Quando um evento é gerado, ele é enviado através de uma rota definida pelo usuário para outros nós que podem gerar mais eventos ou alterar a estrutura do grafo da cena.

Os campos de um nó podem ser campos de saída ou campos de entrada. Um evento é gerado quando um valor é atribuído a um campo de saída de algum nó. Se houver uma ou mais rotas definidas a partir desse campo de saída, o valor desse campo será propagado a um ou mais campos de entradas de outros nós.

3.3.1.3. Nós Sensores, Nós Interpoladores e Nós de Roteiro e Prototipação

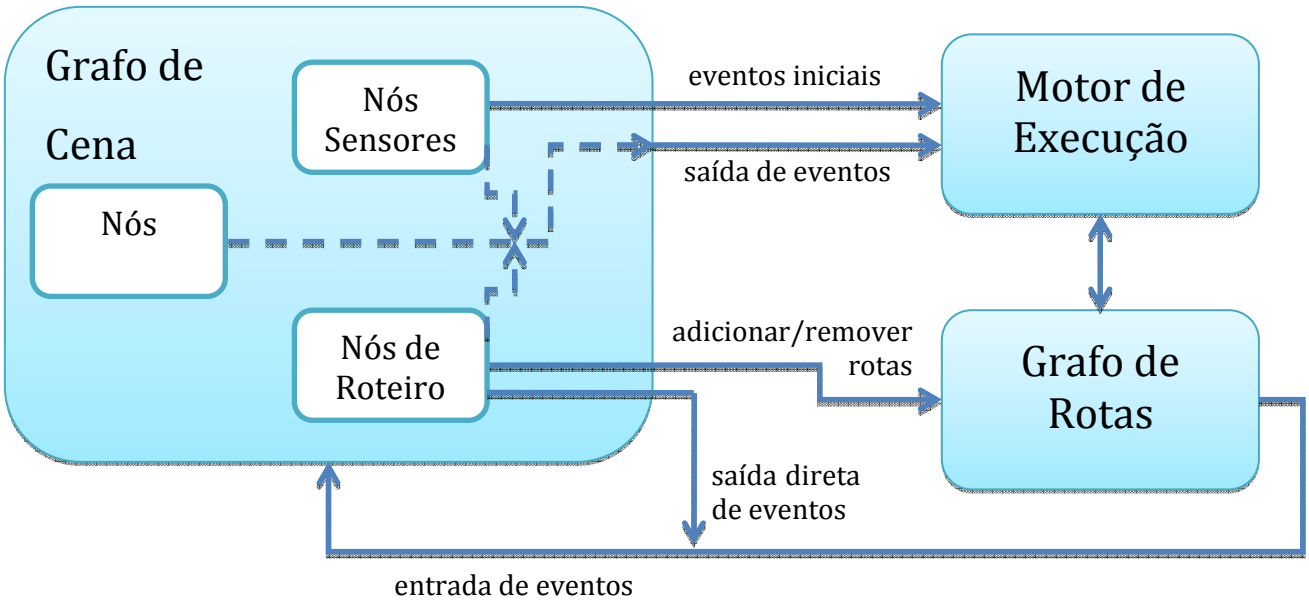
Nós sensores são as primitivas para animação e interação do usuário. Os nós sensores de tempo geram eventos conforme o tempo passa e podem ser usado em conjunto com nós interpoladores que realizam cálculos simples para animações. Outros sensores geram eventos quando ocorre interação do usuário no mundo virtual através de algum dispositivo de entrada. Esses eventos podem ser roteados para a ativação de nós de roteiro. Um roteiro pode ser escrito em Java ou EcmaScript para definir comportamentos arbitrários no mundo.

Prototipação é um mecanismo para encapsular e reusar um grafo de cena ou parte dela. Novos nós protótipos podem ser definidos através da combinação de nós existentes (básicos ou protótipos). Depois de definido, um protótipo deve ser declarado ou instanciado como qualquer outro nó para que ele faça parte da cena.

3.3.1.4. Modelo de Execução

Um evento inicial gerado por um sensor ou roteiro irá propagar até outros nós que poderão gerar eventos adicionais e assim continuar até que não haja mais rotas. Esse processo se chama cascata de eventos. As seguintes ações são tomadas conforme ilustradas na figura 10:

a) atualizar a câmera baseado na posição e orientação atual do ponto-de-vista;
b) avaliar entrada dos sensores;
c) avaliar rotas;
d) se algum evento for gerado nos passos b) ou c), voltar para b).



Fonte: Adaptado de ISO/IEC 19775:2004, X3D Abstract.

Figura 10. Modelo Conceitual de Execução.

Todos os eventos gerados em uma determinada cascata de eventos possuem o mesmo rótulo de tempo. Dessa forma um mesmo evento não será gerado novamente com o mesmo rótulo de tempo para evitar laços de repetições infinitas.

3.3.1.5. Interface de Acesso à Cena

Toda interpretação, execução e apresentação de conteúdo X3D ocorre através do navegador X3D. A apresentação ocorre em um mundo virtual por onde o usuário se move através de mecanismos navegacionais (como andar ou voar) disponíveis pelo navegador. O navegador disponibiliza também mecanismos limitados de interação com a cena através de nós sensores.

A interface de acesso à cena (*Scene Access Interface* ou SAI) é uma interface de programação para o acesso da cena X3D por aplicações externas ao navegador X3D para aumentar a capacidade de navegação e interação do usuário. Conceitualmente cinco tipos de acessos são possíveis na cena X3D através do SAI:

a) acesso as funcionalidade do navegador;
b) recebimento de notificações específicas do navegador;
c) envio de eventos para campos de entradas de nós que fazem parte da cena;
d) leitura dos últimos valores enviados dos campos de saídas dos nós;
e) ser notificado quando eventos alteram valores dos campos dos nós da cena;

3.3.2. Toolkits, Plugins e Applets

O conteúdo X3D é acessado exclusivamente via um navegador X3D. Esse navegador 3D pode ser embutido em uma página Web através de plugins ou applets que são programas que adicionam essa funcionalidade em navegadores Web. O navegador X3D pode ser embutido em outras aplicações utilizando APIs ou toolkits.

A tabela 5 compilada por http://cic.nist.gov/vrml/vbdetect.html compara características das principais implementações que suportam VRML e algumas também o X3D:

Tabela 5. Implementações que suportam VRML/X3D.

SOFTWARE	TIPO	SO			NAVEGADOR		X3D
		Win	Linux	Mac	IE	Firefox	
Cosmo Player	P	X			X	X	
Flux Player	P, S, T	X			X	X	X
Octaga Player	P, S	X	X	X	X	X	X
Cortona3D	P	X		X	X	X	
BS Contact	P, S	X			X	X	X
FreeWRL	P, S, T		X	X		X	X
OpenVRML	P, S, T		X	X		X	X
InstantPlayer	S	X	X	X			X
Xj3D	S, T	X	X	X			X
Orbisnap	S	X	X	X			
Demotride	S	X					X
BS Contact J	A	X	X	X	X	X	
Tipos: P – plugin, S – standalone, T – toolkit, A – applet							

Fonte: Adaptado de http://cic.nist.gov/vrml/vbdetect.html.

Os navegadores X3D não requerem licença para utilização, mas a maioria dessas tecnologias é distribuída comercialmente. Somente o Xj3D, OpenVRML e o FreeWRL são distribuídos sob licença GNU GPL (*General Public Licence*) e GNU LGPL (*Lesser General Public Licence*) para suas utilizações na implementação de aplicações que utilizam conteúdo X3D.

Alguns navegadores X3D implementam componentes específicos que aumentam as funcionalidades do X3D. Esses novos componentes são chamados extensões e apesar do conteúdo X3D que os utilizam ficarem dependentes de determinada implementação, eventualmente eles podem ser adotados pelo padrão e fazer parte da especificação. Dessa forma, as extensões servem como um campo de teste para novas funcionalidades a serem incluídas no padrão X3D.

3.3.3. Xj3D e Flux Player

O toolkit do Xj3D (http://www.xj3d.org/) e o navegador Flux Player (http://www.mediamachines.com/) são as únicas implementações código-aberto disponíveis atualmente que possibilitam a utilização do SAI em diferentes contextos. O Xj3D é um toolkit escrito em Java e com ele é possível embutir o navegador X3D em aplicações Java. Também é possível utilizar o Xj3D para distribuir conteúdo X3D através de Java Applets. O Flux Player é um plugin para os navegadores Internet Explorer e para o Firefox que implementa o SAI para acesso das cenas X3D via JavaScript. Com o Flux Player, é possível a implementação de webapps que utiliza conteúdo X3D.

As outras opções disponíveis para utilizar conteúdo X3D em aplicações são as plataformas comerciais: blaxxum platform (http://www.blaxxun.com/), Bitmanagement Software (http://www.bitmanagement.de/) e Octaga (http://www.octaga.com/). Essas plataformas oferecem algumas facilidades para criar ambientes virtuais distribuídos (Bouras et al. 2005).

3.3.4. Ajax3D

O Ajax3D é uma abordagem para criação de aplicações Web com conteúdo gráfico tridimensional utilizando-se das práticas do AJAX (Asynchronous JavaScript and XML) e do padrão X3D. Uma aplicação Web ou webapp é uma aplicação acessada por um navegador Web em uma rede como a Internet.

AJAX (Garret 2005) não é uma tecnologia, mas um conjunto de práticas para desenvolvimento de aplicações Web que consistem em:

a) uso de scripts no navegador para formatação e apresentação;
b) não recarregar toda a página para alteração do seu conteúdo;
c) uso de outros formatos de dados além do HTML;
d) interagir assincronamente com o servidor.

Atualmente o AJAX incorpora as seguintes tecnologias:

a) apresentação baseadas em padrões utilizando XHTML e CSS;
b) exibição dinâmica e interação utilizando o Document Object Model (DOM);
c) troca e manipulação de dados utilizando XML e XSLT;
d) obtenção de dados assíncrona utilizando XMLHttpRequest;
e) JavaScript para juntar tudo.

O XMLHttpRequest (McLellan 2005) é um objeto nativo do navegador Web que pode ser usado com JavaScript para a troca de dados assíncrona com um servidor Web. Requisições são feitas e respostas são recebidas sem que o usuário experimente interrupções visuais, pois o processo ocorre no plano de fundo e não precisa que toda página seja recarregada.

Ajax3D, por definição (Parisi 2006) é uma aplicação acessada por um navegador Web que:

a) usa o SAI para acessar uma cena X3D em tempo-real e;
b) usa o XMLHttpRequest para armazenar ou obter os dados de uma aplicação 3D e/ou usa o DOM para manipular o conteúdo de uma página Web em resposta a mudanças na cena X3D.

O DOM é um modelo de programação para programas interagirem com um documento Web. O SAI permite programas controlar uma cena X3D. Um código JavaScript executando em uma página Web pode interagir com o DOM e o SAI simultaneamente permitindo integrar elementos bidimensionais da página com mundos tridimensionais.

Além das técnicas AJAX e scripts externos, outra maneira de realizar comunicação entre mundos X3D pela rede são:

a) scripts internos: Utiliza classes implementadas em C++ ou Java embutidas em nós de scripts para comunicação com a rede. Essa abordagem não garante a portabilidade do código entre diferentes implementações dos navegadores X3D, pois nem todas suportam o uso de C++ ou Java internamente.
b) comunicação direta: um novo componente que foi recentemente adicionado à especificação do X3D mas também depende da implementação do navegador X3D para suporte.

3.4. CONSIDERAÇÕES FINAIS

As pesquisas no campo de visualização e de realidade virtual podem ser dividas em dois grupos: aquelas que estão preocupadas como utilizar a tecnologia e aquelas que buscam desenvolver essa tecnologia. Por exemplo, em conjunto com a visualização científica, a realidade virtual pode ser utilizada como interface natural para analisar e interagir com os resultados de uma simulação. O que se deve fazer para gerar uma visualização efetiva e eficiente de dados ou como elaborar uma interface intuitiva para o usuário são questões para o desenvolvimento da própria tecnologia.

É possível notar divergências quanto ao conceito de realidade virtual pelo uso ou não de interfaces imersivas. Muitas vezes não há distinção entre os termos realidade virtual e ambientes virtuais. Para este trabalho será adotada a noção da realidade virtual como o uso das interfaces naturais e ambiente virtual como um mundo simulado pelo computador.

Neste capítulo foi apresentada uma tecnologia importante relacionada diretamente com os conceitos de visualização e ambiente virtual: X3D. A tecnologia utilizada para gerar a visualização de uma simulação depende exclusivamente da natureza do que está sendo simulado, mas o padrão X3D pode ser utilizado de maneira flexível para gerar a visualização tridimensional de dados de uma grande variedade de simulações. O emprego do X3D possibilita a adaptação do conteúdo para visualização em diferentes dispositivos utilizando modelos geométricos de maior ou menor complexidade.

4. FRAMEWORK DE VISUALIZAÇÃO E CONTROLE DE SIMULAÇÕES DISTRIBUÍDAS

4.1. CONSIDERAÇÕES INICIAIS

O objetivo deste trabalho foi definir um framework com a arquitetura de aplicação, os componentes reutilizáveis e a geração de código necessários para facilitar o desenvolvimento de interfaces flexíveis de controle e gerenciamento de simulações de ambiente virtuais. Para garantir a manutenibilidade das aplicações que utilizem este framework, foram adotados os padrões High Level Architecture (HLA) para simulação distribuída com serviços Web e Extensible 3D (X3D) para visualização com gráficos tridimensionais na Web. A utilização de implementações em código-aberto desses padrões com tecnologias Web permite a implantação de simulações de treinamento ou apoio a decisão em diferentes plataformas de software e acessíveis através de dispositivos com capacidade computacional variada (desktops, laptops, palmtops, etc.).

Neste capítulo será apresentado o processo de será explicado os passos da geração de federados de visualização e de seus componentes pelo framework deste trabalho, batizado como "LAViE-3D". No final serão apresentados as descrições dos testes realizados e os resultados obtidos sobre um protótipo implementado usando o framework LAViE-3D. Algumas características e funcionalidades dos padrões HLA e X3D que não foram mencionadas anteriormente serão descritas com suas aplicações neste trabalho.

4.2. MAPEAMENTO DO FOM PARA X3D

O FOM é um componente indispensável para iniciar a execução de uma federação. Nele estão descritos todos os objetos e seus atributos e todas as interações e seus parâmetros que estão disponíveis nessa federação. A chamada de serviços requisitando algo não especificado no FOM

irá resultar em exceções de tempo de execução. O FOM pode ser descrito em um arquivo FED (*Federation Execution Details*) na versão 1.3 do RTI, com sintaxe semelhante a um código de programa Lisp. Na versão 1516 (especificação atual), o arquivo FOM é um documento XML chamado de FDD (*FOM Document Data*). A partir do FOM foi criado para este trabalho um mapeamento entre as classes de objetos e interações para nós protótipos X3D. O mapeamento gera um arquivo X3D com um grafo de cena estruturado para ser acessado pela aplicação de visualização. Os seguintes nós são utilizados:

a) protótipo: Definido em função de outros nós pré-existentes. A interface declara os campos de leitura e escrita e a implementação (ou corpo do protótipo) encapsula um número indeterminado de nós;
b) Script: Contém instruções para interagir com o grafo da cena;
c) Sensor: Detecta as ações do usuário;
d) Transform: Realiza transformações espaciais (rotação, translação, escala) em agrupamento de nós;
e) Inline: Faz ligação com outros arquivos X3D.

O arquivo X3D gerado pode ser modificado com editores XML comuns ou através de programas específicos disponíveis para criação de conteúdo X3D.

Uma classe de objeto é um encapsulamento dos dados que são trocados entre os federados. Algumas classes de objetos podem representar dados abstratos, dessa forma pode não haver uma representação geométrica adequada para visualização tridimensional. Como não é possível prever como uma classe de objeto deve ser representada tridimensionalmente, o nó protótipo gerado dessa classe deve ser editada manualmente. O nó Inline pode ligar com outros arquivos X3D contendo modelos geométricos, scripts de animação, elementos da interface de usuários, etc. O nó de script faz a ligação entre os atributos da classe de objeto com outros nós do grafo da cena. A Figura 11 ilustra o mapeamento.

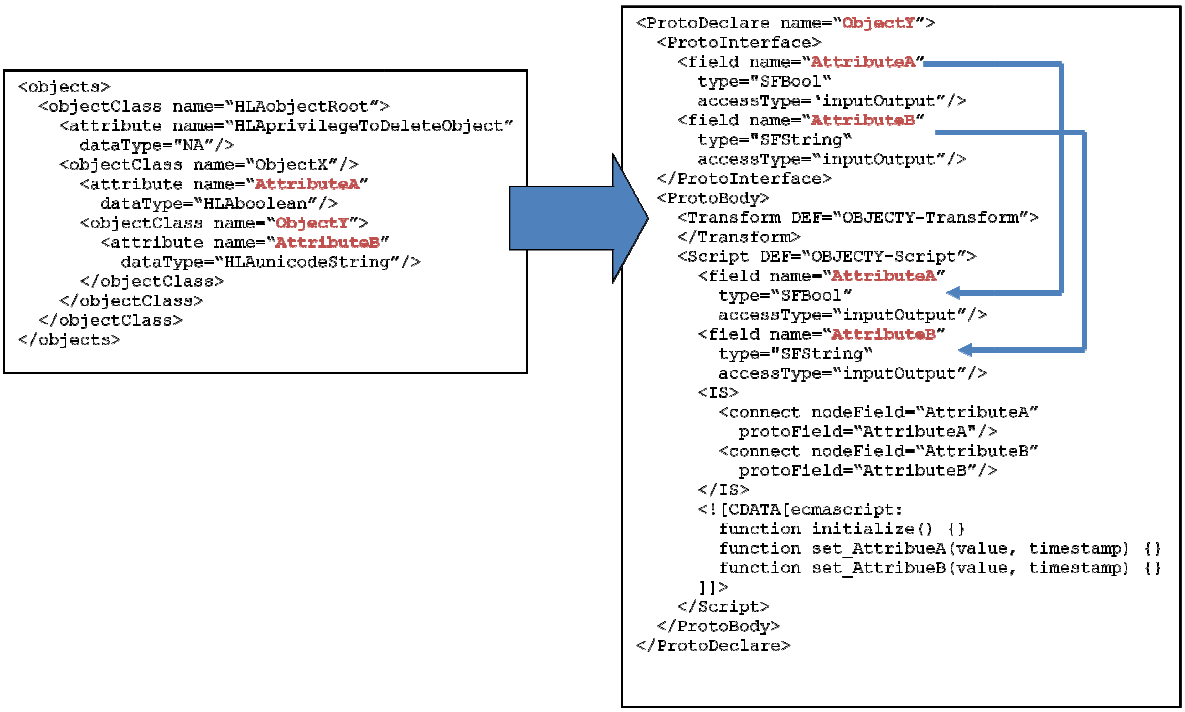


Figura 11. Mapeamento da Classe de Objeto para o Nó Protótipo

Uma classe de interação pode representar um comando que atue na execução de alguns federados. O nó protótipo para as classes de interação são definidos da mesma forma como foi definido para as classes de objetos (Figura 12).

No FOM, uma classe de objeto é definida pelo rótulo *objectClass* na seção delimitada pelo rótulo *objects*. Uma classe de interação é definida pelo rótulo *interactionClass* na seção delimitada pelo rótulo *interactions*. Subclasses são classes aninhadas em outras classes e como não é possível representar uma herança no X3D, define-se protótipos separados para a classe e suas subclasses. Para cada subclasse de objeto ou interação é gerado seu próprio nó protótipo com todos os atributos ou parâmetros de suas respectivas superclasses.

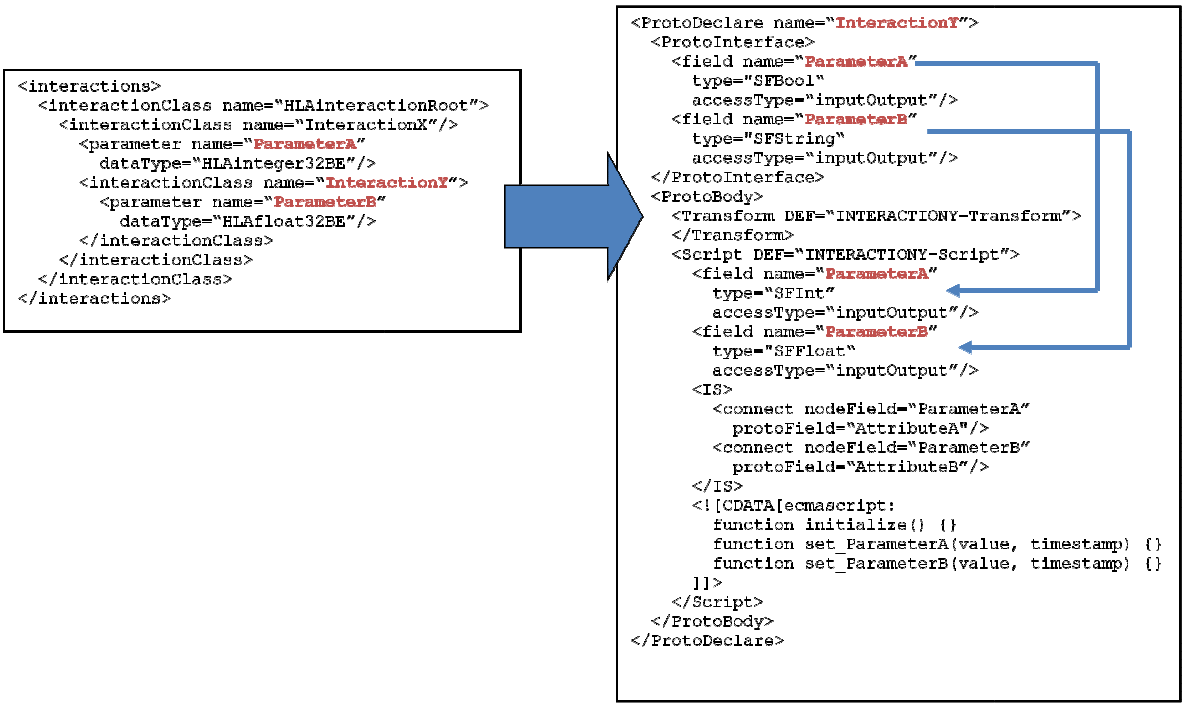


Figura 12. Mapeamento da Classe de Interação para o Nó Protótipo

4.3. IMPLEMENTAÇÃO DOS SERVIÇOS HLA COM SERVIÇOS WEB

Até a implementação deste trabalho a especificação do HLA-Evolved não foi concluída. Contudo, uma versão preliminar do HLA WSDL API foi disponibilizado pela SISO (*Simulation Interoperability Standards Organization*) que permitiu gerar o provedor e o consumidor dos serviços Web. O provedor dos serviços é um servidor Web que atua como um federado-ponte que encaminha as mensagens entre seus clientes (consumidores do serviço) e o RTI. O RTI utilizado neste trabalho é uma implementação código-aberto em Java, chamado Portico v0.8 (http://porticoproject.org/). O código em Java do servidor e dos clientes foi gerado no IDE Netbeans 6. Para gerar a implemantação dos serviços Web em Java, também é possível utilizar JAX-WS (*Java API for XML Web Services*) ou POJO (*Plain Old Java Objects*) nos frameworks do Apache CXF ou Apache Axis2 e no Eclipse Web Tools Platform.

O federado-ponte mantém uma conexão individual com o RTI usando a API do HLA em Java em cada sessão inicializada pelos federados baseados na Web (Figura 13).

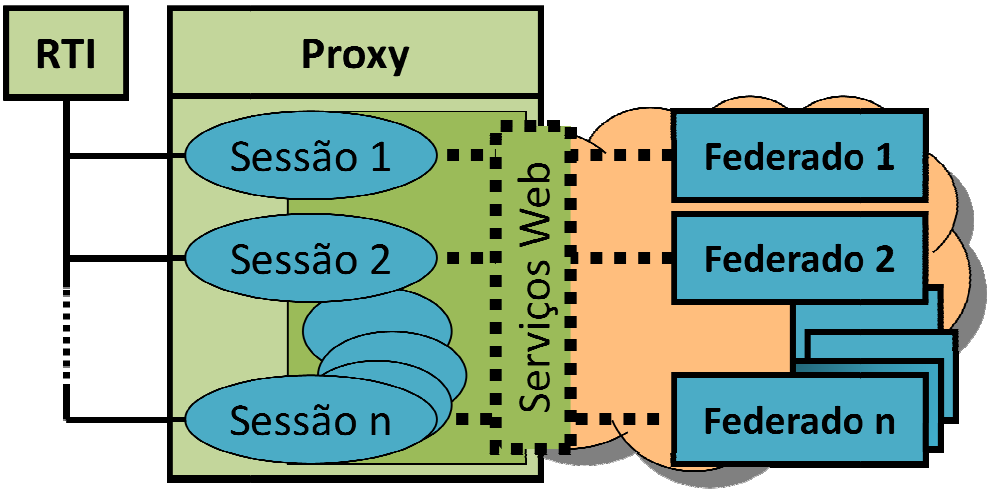


Figura 13. Implementação do Provedor de Serviços

Diferente das implementações usando a API do HLA em Java ou C++ onde se registra funções de retorno no federado para o recebimento de mensagens do RTI, a implementação usando serviços Web suporta apenas a comunicação requisição-resposta iniciada pelo cliente. O cliente precisa requisitar um serviço específico que retorna um número de mensagens do RTI que foram armazenadas no servidor.

Caso haja a necessidade de trocar a implementação do RTI e alterar o código do servidor, os clientes permanecem inalterados. A abordagem utilizada neste trabalho difere da abordagem em ilha pelo fato de utilizar a interface padrão do HLA para os serviços Web. Eventualmente estarão disponíveis implementações do RTI com acesso nativo via serviços Web, ou seja, sem o uso de um federado-ponte.

4.4. FEDERADO DE VISUALIZAÇÃO E CONTROLE

No framework LAViE-3D, juntamente com o arquivo X3D é gerado também o código do federado de visualização. O visualizador é uma webapp que estabelece a comunicação através do RTI com a federação e atualiza a cena X3D.

É possível implementar o visualizador com combinações de diferentes tecnologias e neste trabalho a estrutura da aplicação foi divida em três camadas: camada de comunicação, camada de aplicação do federado e camada de apresentação (Figura 14).

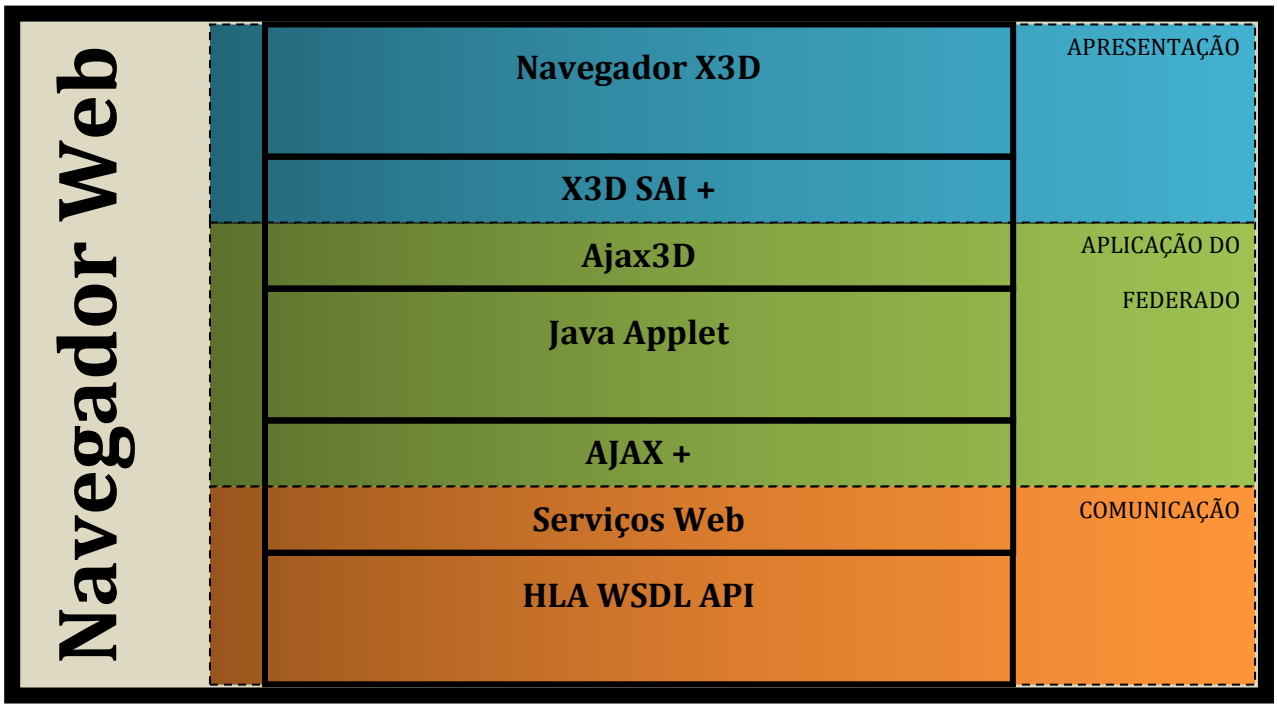


Figura 14. Estrutura do Federado de Visualização

4.4.1. Camada de Comunicação

A camada de comunicação é a implementação do consumidor de serviços Web, responsável em estabelecer conexão com o RTI através de um provedor de serviços Web. O seu código pode ser gerado a partir do HLA WSDL API e ser embutido em um applet Java ou utilizar as técnicas AJAX para fazer o encapsulamento das mensagens e requisitar os serviços usando o XMLHttpRequest.

4.4.2. Camada de Aplicação do Federado

A camada de aplicação do federado contém o código específico para a federação da qual o visualizador faz parte. Esta camada é responsável pela ligação entre a camada de comunicação com a camada de apresentação (Figura 15). Nesta camada são mantidas as instâncias locais das classes de objetos descobertos da federação e, se existir, as instâncias da qual o federado tem posse. Também é são feitas as ligações de cada instância local dos objetos com suas respectivas instâncias de nós protótipos na cena. As instâncias dos nós protótipos das classes de interações podem enviar interações para a federação através de *listeners* registrados em seus campos que recebem notificações das ações do usuário.

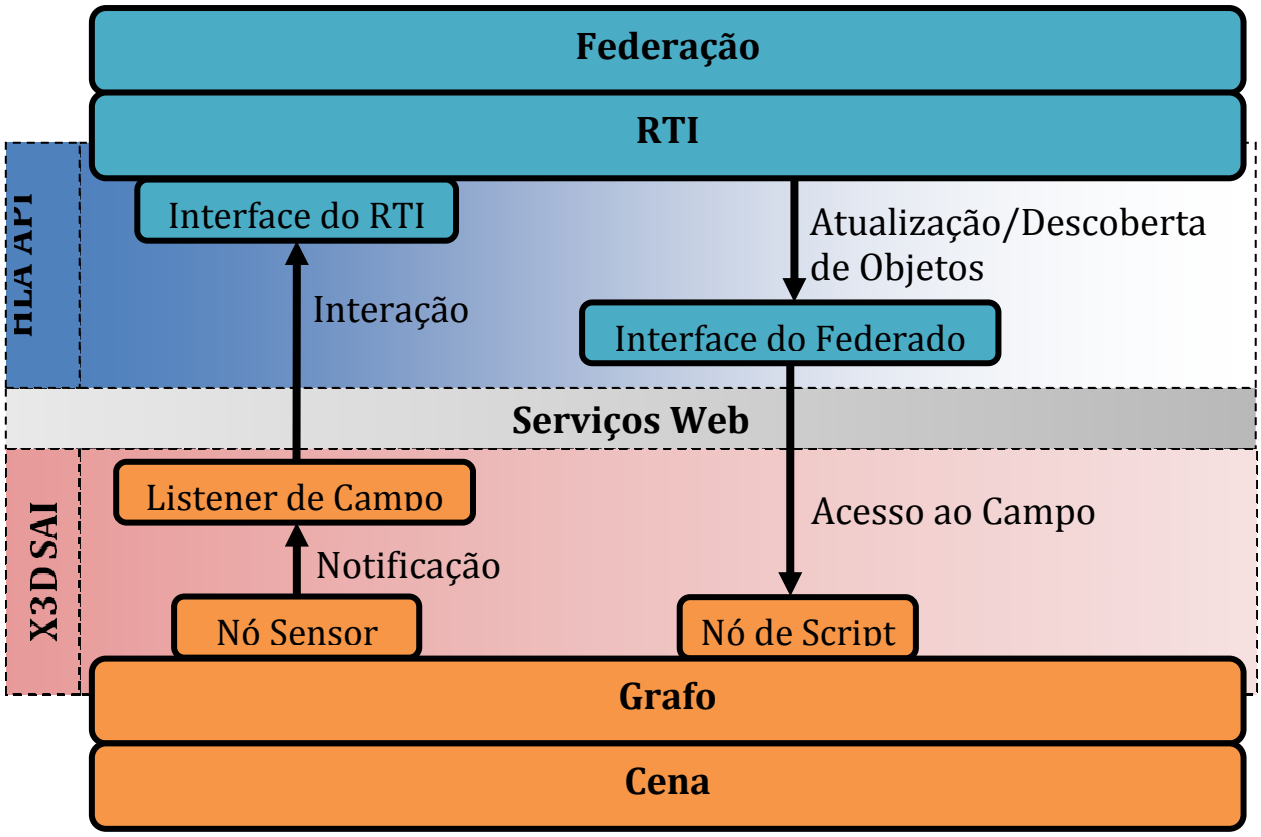


Figura 15. Ligação entre a Federação HLA e a Cena X3D.

4.4.3. Camada de Apresentação

A camada de apresentação é responsável pela renderização tridimensional da cena através de um navegador X3D. Nesta camada são definidas e registradas as funções de acesso aos campos de nós e as funções de retorno de eventos através do SAI. Até a conclusão deste trabalho os navegadores X3D que implementam o SAI foram o Xj3D e o Flux Player, em Java e Javascript respectivamente.

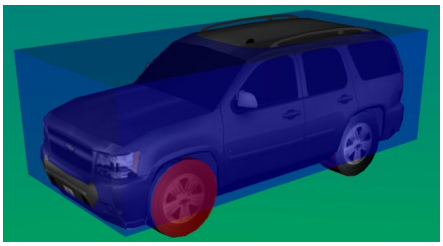
4.5. ANÁLISE DE DESEMPENHO

Para testar o desempenho da visualização de simulações distribuídas utilizando X3D e a comunicação através de serviços Web foi implementado uma simulação distribuída interativa em um ambiente virtual simples onde cada federado tem controle sobre um determinado número de carros.

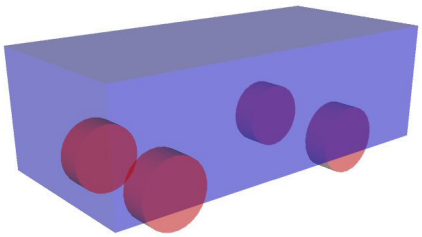
4.5.1. Implementação do Protótipo

O FOM define o carro como única classe de objeto existente na federação. Cada carro foi modelado com cinco corpos rígidos ligados para simulação física: uma caixa para o chassi e uma esfera para cada pneu. O modelo X3D utilizado para representar o carro (Figura 16) possui 1298 triângulos e 795 vértices. Esse modelo foi obtido da biblioteca SAVAGE (https://savage.nps.edu/Savage/), que disponibiliza uma grande variedade de modelos tridimensionais de veículos e ambientes em arquivos X3D que além da geometria contém informações em nós de metadados para uso em cenários de treinamento (Blais et al. 2001; Rauch 2006). Esses metadados são utilizados para descrever as características do modelo e no arquivo deste carro, por exemplo, informa suas dimensões, massa, velocidade máxima, etc. Esses

atributos foram adicionados na definição da classe de objeto do carro no FOM. Com o X3D é possível reutilizar vários modelos para compor o ambiente (Figura 17).



(a)



(b)

Figura 16. O Modelo Geométrico (a) do Carro Sobreposto com o seu Modelo Físico (b).

Figura 17. Usando a Biblioteca SAVAGE para Compor Ambientes Virtuais.

No código dos federados gerados foi adicionado um componente de simulação física de corpos rígidos (Figura 18) usando a API em Java do Open Dynamics Engine (ODE) disponível em http://www.odejava.org/. O motor de simulação física atua antes apresentação da cena calculando a posição e orientação dos carros da federação e do federado baseando-se em suas velocidades e direções, que pode resultar em colisões. Os atributos de velocidade e posição do volante são alterados pelo usuário através das classes de interações "acelerar" e "manobrar" respectivamente.

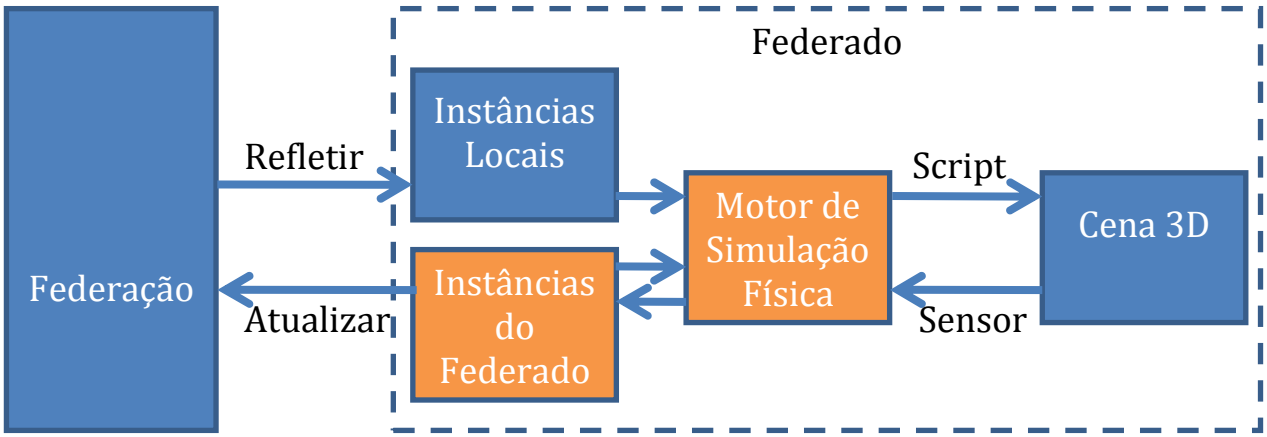


Figura 18. Inclusão do Motor de Simulação Física na Aplicação do Federado

Cada federado possui um motor de simulação física e executa uma simulação simultânea sobre todos os carros da federação. Ou seja, cada federado fornece para a federação esses três atributos dos carros que tem controle e cada federado executa sua própria simulação física baseado nos valores que recebe. Como o resultado de cada simulação pode ser diferente em cada federado, é preciso definir pontos de sincronização da federação com os serviços de gerenciamento de federação, onde todos os federados informam as posições e orientações dos seus carros.

Todos os testes foram realizados em computadores Core 2 Duo 2.4GHz com 2GB de RAM e placas de vídeo GeForce 7600 GS (512MB RAM) em LAN.

4.5.2. Desempenho do Visualizador

Duas implementações diferentes do visualizador foram testadas: uma com o navegador Xj3D embutido no applet e outro onde o applet acessa o navegador Flux Player com JavaScript. Ambas as implementações foram embutidas em uma página Web e hospedada no servidor Apache Tomcat 6 para acesso remoto.

Para este teste, cada federado registra 10 instâncias da classe de objeto carro e publicam e subscrevem para todos os seus atributos. Cada ciclo da simulação é computado o mais rápido possível em cada federado e avançam no tempo com o mesmo intervalo de tempo, porém todos os federados são reguladores e constritos, ou seja, estritamente sincronizados. A cada ciclo todos os federados enviam atualizações dos dados das suas instâncias para criar o maior tráfego de mensagens possível neste teste.

O gráfico na Figura 19 mostra a taxa de quadros por segundo (QPS) de renderização do navegador Xj3D conforme aumenta o número de carros na cena. A medição foi feita utilizando o aplicativo FRAPS (http://www.fraps.com/). O valor máximo foi limitado em 60 QPS com a ativação da opção de sincronização vertical da placa de vídeo. O valor mínimo é obtido quando todos os objetos estão visíveis na cena em relação ao ponto de vista do usuário. Após 100 carros as texturas deixaram de carregar para novos carros que eram incluídos na cena e com mais de 120 carros, a memória disponível para o applet esgotou (na configuração padrão é cerca de 60-90MB). Apesar de todas as instâncias do carro utilizarem o mesmo modelo e textura, cada carro incluído na cena através da instanciação de um protótipo carregava a imagem novamente na memória ao invés de fazer referência para essa imagem que já havia sido carregada anteriormente. Esse foi um problema de falta de otimização encontrado na implementação utilizada do navegador Xj3D.

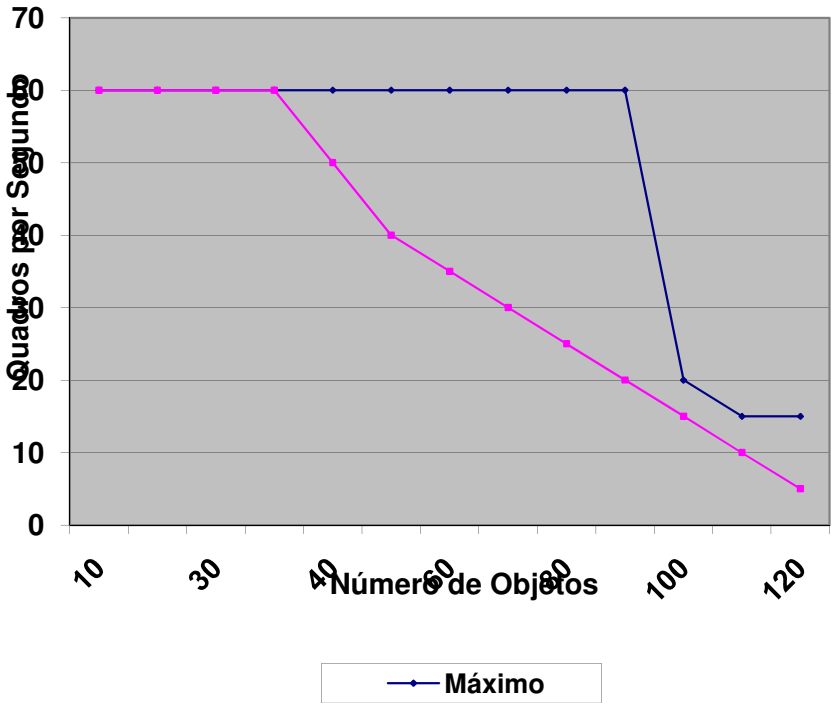


Figura 19. Taxa de Renderização do Visualizador

O tempo de processamento do motor de simulação física aumentava cerca de quatro vezes para cada federado que entrava na federação, diminuindo a taxa de atualizações da federação. O guia do usuário do Open Dynamics Engine (http://ode.org/ode-latest-userguide.html) havia previsto esse aumento linear no tempo de processamento como ocorreu nesta simulação. Uma taxa de renderização semelhante foi obtida com o visualizador sem o motor de simulação física.

Os testes realizados com o navegador Xj3D não foram possíveis de serem reproduzidos com o navegador Flux Player. Enquanto o visualizador usando o navegador Xj3D realizava sem problemas cerca de 500 atualizações por segundo da cena, o SAI do Flux Player não conseguiu manter 20 atualizações por segundo. Contudo, o Flux Player ainda pode ser empregado se a taxa de atualizações através do SAI for menor e a visualização depender de scripts internos executados pelo navegador.

4.5.3. Comparação de Desempenho do RTI

Utilizando a mesma federação foi realizada uma comparação de desempenho utilizando o RTI em Java e a camada de serviços Web construída sobre ele. Neste teste, cada federado registra apenas uma instância da classe de objeto carro e um ciclo de simulação consiste apenas em enviar atualização da sua instância e refletir os valores da federação antes de requisitar o avanço do tempo. As mensagens trocadas entre os federados e o RTI possuem tamanhos variados, já que algumas mensagens podem ser de requisição ou resposta de algum serviço ou podem ser de envio de um bloco grande de dados atualizados.

O gráfico da Figura 20 compara o tempo médio necessário para cada federado executar 1000 ciclos de simulação conforme mais federados entram para a federação. Como todos federados são reguladores e constritos, esses valores podem ser considerados como tempo de processamento de toda federação.

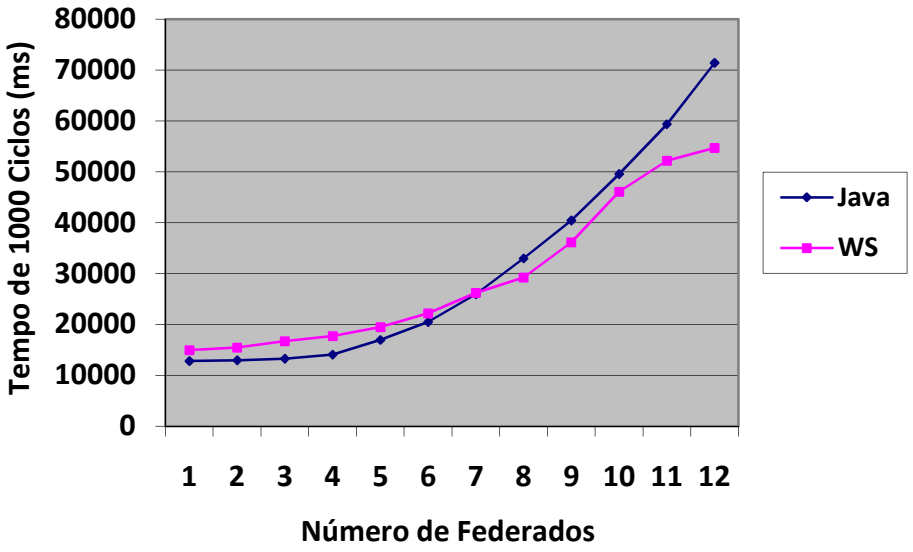


Figura 20. Comparação do Tempo de Processamento.

A figura 21 compara a quantidade média de mensagens que são trocadas entre os federados e o RTI. A implementação em Java do RTI utiliza o protocolo JSOP (*Java Serialized Object Protocol*) e a análise do tráfego na rede mostrou que cada mensagem era segmentada em vários pacotes pequenos (até 14 Bytes de dados), resultando em uma grande taxa de pacotes por segundo. As mensagens SOAP, apesar de serem grandes (a atualização de um carro gera uma mensagem de 2.860 Bytes) eram segmentadas em pacotes maiores (dois ou três pacotes). O aumento do tempo no processamento da federação foi resultado da demora do RTI para processar todos os pacotes (overhead de processamento) que atingiu uma taxa limite de cerca de 150.000 pacotes por segundo na implementação em Java o que explica porque a implementação com serviços Web obteve um melhor desempenho na federação acima de oito federados no gráfico anterior.

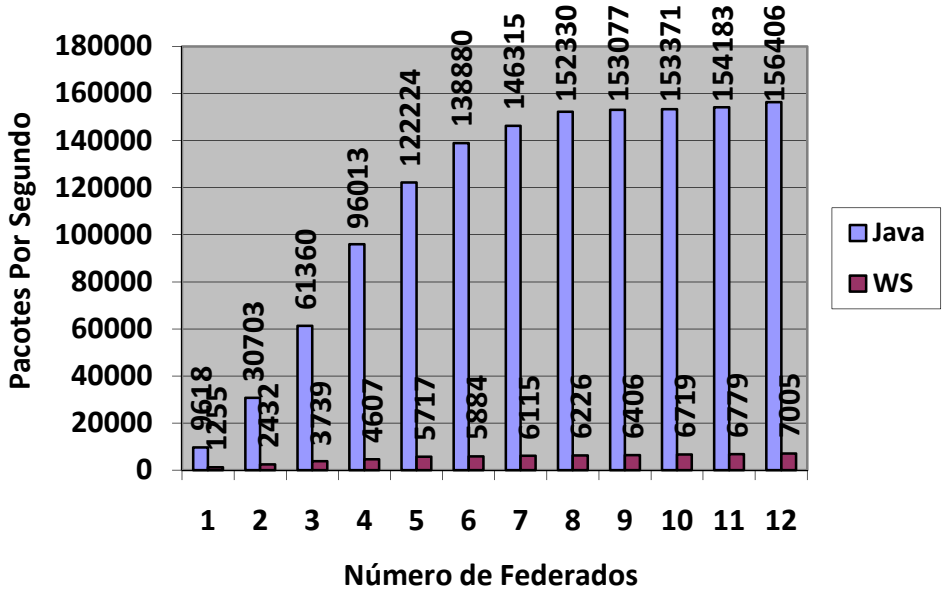


Figura 21. Comparação do Número de Pacotes.

Uma estratégia para reduzir o número de mensagens é simplesmente não enviar desnecessariamente valores de atributos que não foram atualizados para federação. Alguns middlewares para a API do HLA implementam esse tipo de gerenciamento no federado.

4.6. CONSIDERAÇÕES FINAIS

O framework deste trabalho consiste na geração de federados de visualização e controle a partir do FOM que descreve uma simulação distribuída no padrão HLA. Pode-se utilizar o FOM de uma simulação legada ou criar um novo FOM para gerar uma nova simulação como o protótipo que foi apresentado neste capítulo. O código do federado gerado para este protótipo foi modificado para incluir um componente de simulação física. Foi implementado de uma simulação distribuída interativa com carros o que permite elaborar estratégias para a implementação de um ambiente virtual para treinamento envolvendo pessoas. Destacam-se os seguintes pontos:

a) uma quantidade muito grande de federados reguladores baseados na Web pode atrasar toda federação, pois o tempo lógico não avançará se algum desses federados estiver atrasado. O motor de simulação física que foi usado no protótipo pode funcionar como uma forma de predição realizada pelo cliente para reduzir o tráfego de dados e evitar atrasos;

b) todo o comportamento do carro foi simulado pelo motor físico – uma velocidade de rotação foi atribuída nos eixos dos pneus para simular um motor que resultava em uma força que movia o corpo do carro. Em comparação, os avatares (representações dos participantes no ambiente virtual) podem ser implementados de maneira mais simples com um único corpo rígido e com comportamento controlado através de scripts, utilizando o motor físico apenas para auxiliar em alguns momentos como simular colisões o quedas, por exemplo;

c) A partir dos dados obtidos, observa-se a importância de utilizar o serviço de gerenciamento de distribuição de dados (DDM) do HLA para dividir o espaço de treinamento entre os federados. Cada federado se torna responsável em simular apenas uma região, facilitando a sincronização da federação.

5. CONCLUSÃO

A especificação do HLA fornece vários serviços necessários para comunicação e sincronização de modelos de simulação distribuídos. A princípio muitos desenvolvedores encontram dificuldades com a complexidade do HLA. O framework LAViE-3D tem como objetivo gerar todo o código necessário para a implementação da comunicação entre o federado e o RTI de maneira transparente.

O padrão X3D para distribuição de conteúdo tridimensional, com freqüentes adições de novas funcionalidades em sua especificação, mostrou ser uma alternativa viável aos diversos produtos comerciais disponíveis que não são padronizados. O uso do HLA com serviços Web é uma solução para conectar vários usuários em um mesmo mundo X3D.

O uso de serviços Web para conectar federados implementados e a utilização de outras tecnologias Web permitem a criação de diversas aplicações que podem ter acesso à federação através de diferentes dispositivos (de celulares a cavernas digitais).

Diferente dos trabalhos de Salisbury et al. (1999) e Hsia (2000) que utilizaram applets Java com a API Java3D para visualização 3D de simulações na Web com protocolos próprios de comunicação via *socket*, o framework LAViE-3D utilizaX3D com serviços Web. Na época em que esses trabalhos foram publicados, Java era uma das opções mais usada para desenvolvimento de webapps. Porém, com Java3D o conteúdo 3D é compilado juntamente com a aplicação, enquanto que no X3D esse conteúdo é separado da aplicação e interpretado em tempo de execução. Atualmente serviços Web é uma tecnologia mais viável para troca de dados em aplicações Web por questões de interoperabilidade e segurança.

5.1. CONTRIBUIÇÕES GERADAS

Este trabalho traz as seguintes contribuições:

a) utilização dos padrões HLA e X3D para criação rápida de diferentes cenários de treinamento em ambientes virtuais;
b) utilização dos padrões HLA e X3D para criação rápida de federados de visualização para diferentes cenários de treinamento em ambientes virtuais;
c) implementação de um framework que torna transparente as complexidades do HLA e gera federados de visualização e controle a partir do FOM;
d) geração de código para federados que gerenciam a comunicação com suas federações e apresentam os dados em uma cena X3D;
e) melhor compreensão dos padrões HLA e X3D;
f) análise aprofundada das implementações código-aberto do HLA e X3D;
g) viabilidade do uso de serviços Web em simulações distribuídas;
h) RTI como alternativa para estabelecer a comunicação entre sistemas heterogêneos.

5.2. LIMITAÇÕES E TRABALHOS FUTUROS

Este framework ainda não possui um ambiente de desenvolvimento próprio para a geração de código. Esse ambiente ainda precisa levar em consideração a integração com outros trabalhos do LRVNet, incluindo a criação de simulações no padrão HLA através de ontologias e a adaptação do arquivo X3D gerado para diferentes interfaces de usuário. Também é necessário definir uma interface específica em X3D para acessar aos objetos e interações do MOM.

O X3D define componentes de H-Anim (animação humanóide) e recentemente foi incluído o componente de física de corpos rígidos. O Xj3D já implementa o componente de física de corpos rígidos com o ODE. O uso desses componentes neste framework pode facilitar ainda mais a criação de simulações de treinamento.

Como ainda não foi especificada uma estratégia de comunicação para o padrão X3D, o uso de serviços Web com o padrão HLA pode ser uma solução para criação de ambientes multiusuários em mundos X3D. Maiores avanços na área de serviços em grade (computação em grade com serviços Web) poderão proporcionar o uso do padrão HLA para uma solução totalmente distribuída (RTI distribuído).

Padrões abertos como HTML e JavaScript continuam evoluindo para suportar novas aplicações na plataforma Web. Com a especificação do HTML5 (http://www.w3.org/html/wg/html5/), o novo elemento (*tag*) Canvas pode ser utilizado para apresentação conteúdo X3D. Uma implementação do Canvas já foi distribuída no navegador Web Firefox 3 (www.mozilla.com/firefox/) com extensões para utilização da API do OpenGL para gráficos tridimensionais. Alguns programadores já utilizaram essa extensão do Firefox 3 com a API do OpenGL para implementar em JavaScript um interpretador de arquivos X3D para apresentar seu conteúdo no Canvas (http://philip.html5.org/demos/canvas/3d/x3d/).

REFERÊNCIAS

AU, T. Andrew. Performance Issues of HLA Run Time Infrastructure based on CORBA. In: SIMTECT 2000 SIMULATION CONFERENCE AND EXHIBITION. 2000. Sydney, Australia. **Proceedings of the Simulation Technology and Training 2000**. 2000. 123-128.

BANKS, Jerry. Introduction to Simulation. In: 1999 WINTER SIMULATION CONFERENCE. 1999. Phoenix, Arizona, United States. **Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 1**. New York, NY, USA: ACM Press, 1999. 9-16.

BLAIS, C.; BRUTZMAN, D.; HORNER, D.; NICKLAUS, S. Web-Based 3D Technology for Scenario Authoring and Visualisation: The Savage Project. In: INTERSERVICE/INDUSTRY TRAINING, SIMULATION, AND EDUCATION CONFERENCE. 2001. Orlando, Florida, USA.

BLINN, James F. Transcription of Keynote Address at Siggraph ’98. **Computer Graphics**, v. 33, n. 1, p. 43-47, 1999.

BOURAS, Ch.; GIANNAKA, E.; TSIATSOS, Th. Advances in X3D Multi-User Virtual Environments. In: SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON MULTIMEDIA. 2005. Irvine, CA, USA. **Proceedings of the Seventh IEEE International Symposium on Multimedia**. Washington, DC, USA: IEEE Press, 2005. 136-142.

BROOKS, Frederick P. What’s Real About Virtual Reality. **IEEE Computer Graphics and Applications**, v. 19, n. 6, p. 16-27, 1999.

BRUTZMAN, D.; ZYDA, M.; PULLEN, J. M.; MORSE, K. L. **Extensible Modeling and Simulation Framework (XMSF) Challenges for Web-Based Modeling & Simulation**. 2002. Monterey, CA. Naval Postgraduate School.

BRYSON, Steve. Virtual Reality in Scientific Visualization. **Communications of the ACM**, v. 39, n. 5, p. 62-71, 1996.

BRYSON, Steve. Virtual Reality Takes on Real Physics Applications. **Computer in Physics**, v. 6, n. 4, 1992.

BULLARD, Len. **Extensible 3D: XML meets VRML**. Disponível em: http://www.xml.com/pub/a/2003/08/06/x3d.html. Acesso em: abril de 2008.

BUSS, Arnold; JACKSON, Leroy. Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI. In: 1998 WINTER SIMULATION CONFERENCE. 1998. Washington, D.C., United States. **Proceedings of the 30th Conference on Winter Simulation**. 1998. Los Alamitos, CA, USA: IEEE Press. 819-826.

CARLSSON, C.; HAGSAND, O. DIVE: A Multi-User Virtual Reality System. In: 1993 IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM. 1993. Seattle, Washington. **1993 IEEE Virtual Reality Annual International Symposium Proceedings**. 1993. 394-400.

CARSON, John. S. Introduction to Modeling and Simulation. In: 2004 WINTER SIMULATION CONFERENCE. 2004. Washington, DC, USA. **Proceedings of the 36th Conference on Winter Simulation**. Washington, DC, USA: WSC, 2004. 9-16.

CARSON, John. S. Modeling and Simulation Worldviews. In: 1993 WINTER SIMULATION CONFERENCE. 1993. Los Angeles, California, USA. **1993 Winter Simulation Conference Proceedings**. Los Angeles, California, USA: ACM Press, 1993. 18-23.

CHANDY, K. M.; MISRA, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. **IEEE Transactions on Software Engineering**, v. 6, n. 5, p. 440-452, 1979.

CHONG, Chin Soon; SIVAKUMAR Appa Iyer; GAY, Robert. Design, Development and Application of an Object Oriented Simulation Toolkit for Real-Time Semiconductor

Manufacturing Scheduling. In: 2002 WINTER SIMULATION CONFERENCE. 2002. San Diego, California. **Proceedings of the 34th conference on Winter simulation: exploring new frontiers**. San Diego, California: WSC, 2002. 1849-1856.

CHURCHILLl, E. F; SNOWDON, D. Collaborative Virtual Environments: An Introductory Review of Issues and Systems. **Virtual Reality: Research, Development and Applications**, v. 3, n. 1, p. 3-15, 1998.

CRUZ-NEIRA, Carolina; SANDIN, Daniel J.; DEFANTI, Thomas A.; KENYON, Robert V.; HART, John C. The CAVE: Audio Visual Experience Automatic Virtual Environment, **Communications of the ACM**, v. 35, n. 6, p. 64-72, 1992.

CURBERA, Francisco; DUFTLER, Matthew; KHALAF, Rania; NAGY, William; MUKHI, Nirmal; WEERAWARANA, Sanjiva. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. **IEEE Internet Computing**, v. 6, n. 2, p. 86-93, 2002.

DAVIS, Wayne J. On-Line Simulation: Need and Evolving Research Requirements. In: BANKS, Jerry. **Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice**. USA: Wiley-Interscience, 1998. 465-516.

FALBY, John S.; ZYDA, Michael J.; PRATT, David R.; MACKEY, Randy L. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. **Computers & Graphics**, v. 17, n. 1, p. 65-69, 1993.

FISHWICK, P. A. Web-Based Simulation: Some Personal Observations. In: 1996 WINTER SIMULATION CONFERENCE. 1996. Coronado, CA, USA. **1996 Winter Simulation Conference Proceedings**. Coronado, CA, USA: ACM Press, 1996. 772-779.

FUJIMOTO, Richard M. Parallel and Discrete Event Simulation. In: 1989 WINTER SIMULATION CONFERENCE. 1989. Washington, DC, USA. **Proceedings of the 21st Winter Simulation Conference**. Washington, DC, USA: ACM Press, 1989. 19-28.

FUJIMOTO, Richard M. Parallel and Distributed Simulation Systems. In: 2001 WINTER SIMULATION CONFERENCE. 2001. Arlington, VA, USA. **Proceedings of the 2001 Winter Simulation Conference**. Arlington, VA, USA: ACM Press, 2001. 147-157.

FUJIMOTO, Richard M.; HUNTER, Michael; SIRICHOKE, Jason; PALEKAR, Mahesh; KIM, Hoe; SUH, Wonho. Ad Hoc Distributed Simulations. In: 21ST INTERNATIONAL WORKSHOP ON PRINCIPLES OF ADVANCED AND DISTRIBUTED SIMULATION. 2007. San Diego, California, USA. **Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation**. Washington, DC, USA: IEEE Press, 2007. 15-24.

FULLFORD, D. A. Distributed Interactive Simulation: It's Past, Present, and Future. In: 1996 WINTER SIMULATION CONFERENCE. 1996. Coronado, CA, USA. **Proceedings of the 28th Conference on Winter Simulation**. Coronado, CA, USA: ACM Press, 1996. 179-185.

GARRETT, J. J. **Ajax: a new approach to web applications**. Disponível em: http://www.adaptivepath.com/publications/essays/archives/000385.php. Acesso em: abril de 2008.

GERSHON, N.; EICK, S. **Foreword**. In: 1995 IEEE SYMPOSIUM ON INFORMATION VISUALIZATION. 1995. Atlanta, Georgia, USA. **Proceedings of the 1995 IEEE Symposium on Information Visualization**. Atlanta, Georgia, USA: IEEE Press, 1995. vii-viii.

GLINKSY, Ezequiel; WAINER, Gabriel. Extensions: Performance Analysis of Real-Time DEVS Models. In: 2002 WINTER SIMULATION CONFERENCE. 2002. San Diego, California, USA. **Proceedings of the 34th Winter Simulation Conference: Exploring New Frontiers**. San Diego, California, USA: ACM Press, 2002. 588-594.

GOLDSMAN, David. Introduction to Simulation. In: 2007 WINTER SIMULATION CONFERENCE. 2007. Washington, DC, USA. **Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come**. Washington, DC, USA: IEEE Press, 2007. 26-37.

GREENHALGH, Chris; BENFORD, Steve. A Multicast Network Architecture for Large Scale Collaborative Virtual Environments. In: FDIDA, Serge; MORGANTI, Michel. **Multimedia**

Applications, Services and Techniques — ECMAST '97: Second European Conference Milan, Italy, May 21–23, 1997 Proceedings. Berlin/Heidelberg: Springer, 1997. 113-128.

GREENHALGH, C. e BENFORD, S. MASSIVE: A Virtual Reality System for Tele-Conferencing. **ACM Transactions on Computer Human Interfaces**, v. 2, n. 3, p. 239-261, 1995.

GREENHALGH, C., PURBRICK, J. e SNOWDON, D. **Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring**, Proceedings of the Third International Conference on Collaborative Virtual Environments, p. 119-127, 2000.

GUPTA, A. K.; SIVAKUMAR, A. I.; SARAWGI S. Shop Floor Scheduling with Simulation Based Proactive Decision Support. 2002 WINTER SIMULATION CONFERENCE. 2002. San Diego, California, USA. Proceedings of the 34th Winter Simulation Conference: Exploring New Frontiers. San Diego, California, USA: ACM Press, 2002. 1897-1902.

HAND, Chris. Other Faces of Virtual Reality. In: BRUSILOVSKY, Peter; KOMMERS Piet A. M.; STREITZ Norbert A. **Multimedia, Hypermedia, and Virtual Reality Models, Systems, and Applications:** First International Conference, MHVR'94 Moscow, Russia, September 14–16, 1994 Selected Papers. Berlin/Heidelberg: Springer, 1994. 107-116.

HANISCH, A., TOLUJEW, J., RICHTER, K. e SCHULZE, T. Online Simulation of Pedestrian Flow in Public Buildings. In: 2003 WINTER SIMULATION CONFERENCE. 2003. New Orleans, Louisiana, USA. **Proceedings of the 35th Winter Simulation Conference: Driving Innovation.** New Orleans, Louisiana, USA: ACM Press, 2003. 1635-1641.

HARMONOSKY, C. M. Simulation-Based Real-time Scheduling: Review of Recent Developments. In: 1995 WINTER SIMULATION CONFERENCE. 1995. Arlington, VA, USA. **Proceedings of the 1995 Winter Simulation Conference.** Arlington, VA, USA: ACM Press, 1995. 220-225.

HARRELL, C. Process Simulation vs. System Simulation. In: IEEE INFORMATION TECHNOLOGY CONFERENCE. 1998. IEEE **Information Technology Conference Proceedings**. 1998. 41-44.

HECKBERT, Paul. Ten Unsolved Problems in Rendering. In: WORKSHOP ON RENDERING ALGORITHMS AND SYSTEMS, GRAPHICS INTERFACE '87. 1987. Toronto. **Workshop On Rendering Algorithms And Systems, Graphics Interface '87 Proceedings**. 1987.

HIBBARD, Bill. Top Ten Visualization Problems. **ACM SIGGRAPH Computer Graphics**, v. 33, n. 2, p. 21-22, 1999.

HOFER, R. C. e LOPER, M. L. DIS Today. **Proceedings of the IEEE**, v. 83, n. 8, p. 1124-1137, 1995.

HSIA, Wen-yang. **An HLA-based Simulation Environment for Virtual Reality via Java3D**. Kaohsiung, Taiwan: National Sun Yat-sen University, 2000.

IJSSELSTEIJN, Wijnand. A.; RIDDER, Huib; FREEMAN, Jonathan; AVONS, Steve E. **Presence: Concept, Determinants and Measurement**. In: SPIE-International Society for Optical Engineering: Human Vision and Electronic Imaging V. 2000. San Jose, CA, USA. Human Vision and Electronic Imaging V (Proceedings Volume). 2000.

JOHNSON, C. R. e SANDERSON, A. R. **A Next Step: Visualizing Errors and Uncertainty.** IEEE Computer Graphics and Applications, v. 23, n. 5, p. 6-10, 2003.

JOHNSON, C. R. Top Scientific Visualization Research Problems. **IEEE Computer Graphics and Applications**, v. 24, n. 4 p. 13-17, 2004.

JOHNSON, C.; PARKER, S. G.; HANSEN, C.; KINDLMANN, G. L.; LIVNAT, Y. Interactive Simulation and Visualization. **Computer**, v. 32, n. 12, p. 59-65, 1999.

KATZ, D.; MANNIVANNAN, S. Exception Management on Shop Floor Using Online Simulation. In: 1993 WINTER SIMULATION CONFERENCE. 1993. Los Angeles, California, USA. **Proceedings of the 25th Winter Simulation Conference**. Los Angeles, California, USA: ACM Press, 1993. 888-896.

KULJIS, Jasna e PAUL, Ray J. A Review of Web-Based Simulation: Whither We Wander? In: 2000 WINTER SIMULATION CONFERENCE. 2000. Orlando, FL, USA. **Proceedings of the 2000 Winter Simulation Conference**. Orlando, FL, USA: ACM Press, 2000. 1872-1881.

KULJIS, Jasna; PAUL, Ray J. Web-Based Discrete Event Simulation Models: Current States and Possible Futures. **Simulation & Gaming**, v. 34, n. 1, p. 39-53, 2003.

MACKINLAY, Jock D. Opportunities for Information Visualization. IE**EE Computer Graphics and Applications**, v. 20, n. 1 p. 22-23, 2004.

MARSHALL, Robert; KEMPF, Jill; DYER, Scott; YEN, Chieh-Cheng. Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie. In: 1990 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS. 1990. Snowbird, Utah, United States. **Proceedings of the 1990 Symposium on Interactive 3D Graphics**. New York, NY, USA: ACM Press, 1990. 89-97.

MAZUR, F., CHROBOK, R., HAFSTEIN, S.F., POTTMEIER A. e SCHRECKENBERG, M. Future of Traffic Information - Online-Simulation of a Large Scale Freeway Network. IADIS In: INTERNATIONAL CONFERENCE WWW/INTERNET 2004. 2004. Madrid, Spain. **Proceedings of IADIS International Conference WWW/Internet 2004**. 2004. 665-672.

MCCORMICK, B. H. Visualization in Scientific Computing. **ACM SIGBIO Newsletter**, v. 10, n. 1, p. 15-21, 1988.

MCLELLAN, D. **Very Dynamic Web Interfaces**. Disponível em: http://www.xml.com/pub/a/2005/02/09/xml-http-request.html. Acesso em: abril de 2008.

MÖLLER, Björn; DAHLIN, Clarence. A First Look at the HLA Evolved Web Service API. In: 2006 EUROPEAN SIMULATION INTEROPERABILITY WORKSHOP. 2006. Stockholm, Sweden. **Proceedings of the 2006 European Simulation Interoperability Workshop.** 2006.

MÖLLER, Björn; LÖF, Staffan. A Management Overview of the HLA Evolved Web Service API. In: 2006 FALL SIMULATION INTEROPERABILITY WORKSHOP. 2006. Orlando, FL. **Proceedings of 2006 Fall Simulation Interoperability Workshop**. 2006.

MUNZNER, T. Guest Editor's Introduction: Information Visualization. **IEEE Computer Graphics and Applications**, v. 22, n. 1, p. 20-21, 2002.

NG, Beatrice; LI, Frederick W. B.; LAU, Rynson W. H.; SI, Antonio; SIU, Angus. A Performance Study on Multi-Server DVE Systems. **Information Sciences – Informatics and Computer Science: An International Journal**, v. 154, n. 2, p. 85-93, 2003.

O'REILLY, T. **What is Web 2.0: design patterns and business models for the next generation of software.** Disponível em: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Acesso em: abril de 2008.

PAGE, E. H. The Rise of Web-Based Simulation: Implications for the High Level Architecture. In: 1998 WINTER SIMULATION CONFERENCE. 1998. Washington DC, USA. **Proceedings of the 30th Conference on Winter Simulation**. Washington DC, USA: ACM Press, 1998. 1663-1668.

PARISI, T. **Ajax3D: The Open Platform for Rich 3D Web Applications.** Disponível em http://www.ajax3d.org/whitepaper/. Acesso em: abril de 2008.

PULLEN, J. Mark; BRUNTON, Ryan; BRUTZMAN, Don; DRAKE, David; HIEB, Michael; MORSE, Katherine L.; TOLK, Andreas. Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment. **Future Generation Computer Systems**, v. 21, n. 1, p. 97-106, 2005.

RAUCH, Travis M. **Savage Modeling Analysis Language (SMAL): Metadata for Tactical Simulations and X3D Visualizations**. Monterey, Califórnia: Naval Postgraduate School, 2006.

RHYNE, Theresa-Marie. Does the Difference between Information Visualization and Scientific Visualization Really Matter? **IEEE Computer Graphics and Applications**, v. 23, n. 3, p. 6-8, 2003.

SALISBURY, Chad F.; FARR, Steven D.; MOORE, Jason A. Web-based simulation visualization using Java3D. In: 1999 WINTER SIMULATION CONFERENCE. 1999. Phoenix, Arizona, USA. **Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 2**. New York, NY, USA: ACM Press, 1999. 1425-1429.

SCHOLLMEIER, R. A Definition of Peer-To-Peer Networking for the Classification of Peer-To-Peer Architectures and Applications. In: 2001 INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING. 2001. Sweden. **First International Conference on Peer-to-Peer Computing Proceedings**. 2001. 101-102.

SCHUBERT, Thomas; FRIEDMANN, Frank; REGENBRECHT, Holger. Embodied Presence in Virtual Environments. In: PATON, Ray; NEILSON, Irene. **Visual Representations And Interpretations**. Springer, 1999. 269-277.

SCHUG, Klaus; GUPTA, Sandeep. K. S.; JAYASUMANA, Anura; SRIMANI Pradip K. CORBA Based HLA/RTI Design Approach. In: 1997 SUMMER COMPUTER SIMULATION CONFERENCE. 1997. Arlington, Virginia. Summer Computer Simulation Conference Proceedings. 1997.

SIVAKUMAR, A. I. Optimization of Cycle Time and Utilization in Semiconductor Test Manufacturing Using Simulation-Based, On-Line, Near-Real-Time Scheduling System. In: 199 WINTER SIMULATION CONFERENCE. 1993. Phoenix, Arizona, USA. **Proceedings of the 1999 Winter Simulation Conference**. 1999. 727-735.

STEUER, J. Defining Virtual Reality: Dimensions Determining Presence. **Journal of Communication,** v. 42, n. 4, p. 73-93, 1992.

VAN WIJK, J. J. Views on Visualization. **IEEE Transactions on Visualization and Computer Graphics**, v. 12, n. 4, p. 421-432, 2004.

WATERS, R. C.; ANDERSON, D. B.; BARRUS, J. W.; BROGAN, D. C.; CASEY, M. A.; MCKEOWN, S. G.; NITTA, T.; STERNS, I. B.; YERAZUNIS, W. S. Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability. **Presence: Teleoperators and Virtual Environments**, v. 6, n. 4, p. 461-480, 1997.

YANG, B.; GARCIA-MOLINA, H. Comparing Hybrid Peer-to-Peer Systems**.** In: 27TH INTERNATION CONFERENCE ON VERY LARGE DATA BASES. 2001. Roma, Italy. **Proceedings of the 27th International Conference on Very Large Data Bases**. 2001. 561-570.

APÊNDICE A – A Framework to Generate HLA compliant Visualization Federates through Web Services and X3D (Symposium on Virtual and Augmented Reality 2008)

A Framework to Generate HLA compliant Visualization Federates through Web Services and X3D

Regina B. Araujo[1] Azzedine Boukerche[2] Ednaldo Pizzolato[1] Fabio M. Iwasaki[1]
[1]*CS, Federal University of S. Carlos, SP, Brazil*
[2]*SITE, University of Ottawa, Ottawa, Canada*
{regina, ednaldo, fmiwasaki} @ dc.ufscar.br, boukerch @ site.uottawa.ca

Abstract
This paper presents an open standard and web-based framework that generates HLA compliant simulation visualization interfaces with support to control and management functionalities. Differently from other existing approaches, our framework can provide greater flexibility for customization and deployment of HLA-based simulations in different hardware and software platforms.

1. Introduction

The modeling and visualization of simulations, particularly those aimed at training situations, such as emergency actions by Special Forces (e.g., police and fire fighters) are not trivial to build, manage and control. By having control over a simulation, in real-time, a specialist user, such as a fire fighter commander, can add and/or remove existing objects in the simulation, as well as change existing object properties, making the application a flexible environment to train human capabilities on different equipments and situations. Management, on the other hand, can provide important statistics on the performance of humans and equipments in different simulation runs. This paper presents an open standard, web3D-based framework, which can generate visualization interfaces to HLA (High Level Architecture) compliant simulations, with support to control and management functionalities.

Similar approaches to 3D visualization of HLA-based simulations were proposed using Java3D, most of them with their own protocol for communication. However, since the time they were presented, Web Services technology became a better alternative. Web Services use XML formatted messages to exchange data using primarily HTTP and HTTPS and are less prone to security-related problems. Moreover the solutions mentioned above do not deal with simulation control and management but only visualization.

For visualization, instead of using Java3D and having the 3D content being compiled along with the Java application or applet, X3D allows the content to be written in a text file that can be changed without the need to recompile the application. Web Services and X3D standards can provide greater flexibility for the customization of the functionalities of an HLA-based visualization application. Moreover, they can provide simulation deployment in different hardware and software platforms.

The paper is organized as follows: Section 2 gives a brief review on HLA and web services showing how they relate to our solution. X3D is briefly reviewed on section 3. Our framework for simulation visualization, control and management is described in Section 4, followed by Conclusions and References.

2. High Level Architecture and Web Services

The framework presented in this paper is based on the IEEE standard High Level Architecture (HLA) for distributed simulations [1]. HLA-based simulations can interoperate and have its components reused. An HLA compliant simulation is referred to as a **federate**, and a **federation** is defined as a set of federates working together. The HLA has three main components: the HLA rules, the HLA federate interface specification and the HLA object model template (OMT). The HLA rules define the responsibilities federates and federations must hold. The interface specification defines the standard services and interfaces of an underlying software architecture, the **Runtime Infrastructure (RTI),** which supports the data exchange among federates. These interfaces are arranged in 7 service groups: federation management, declaration management, object management, ownership management, time management, data distribution management and support services.

Web Services can be used to implement an architecture that meet the requirements of Service Oriented Architecture, such as interoperability between different systems, clear and unambiguous description language and retrieval of services. Web Services and HLA can be used to integrate heterogeneous simulations and international efforts have been made towards the

provision of standardized Web Service interfaces independent of the RTI implementation (HLA WSDL API) [2][3][4]. This approach eliminates the need for customized software to route the calls from Web Services and a vendor specific RTI implementation. Even though the underlying work is still the same, it is transparent to federates and reusable by other simulations. Next section describes our solution for the creation of HLA-based simulation visualization for control and management that uses web services and X3D for greater flexibility.

3. HLA Compliant Simulation Visualization

There are three techniques for simulation visualization (with varying degree of integration between the simulation control and its visualization): *post-processing*, *tracking* and *steering* [5]. In *post-processing* technique, the resulting images from the visualization are generated after the simulation ends. In *tracking*, it is possible to view the current internal state of an executing simulation but not to change it. In the *steering* technique, the user has control over the simulation parameters and can change them at run time – this is the technique supported by our framework. In a training simulation, the user's inputs are continuously changing his/her corresponding virtual environment states. To interact with a simulation, it is necessary to define a strategy to further refine its visualization [6]. The visualization of a training simulation can be more or less effective depending on its specification and the user's perception [7]. The user can have a richer experience if the visualization specification can be changed during the simulation execution or customized for the user's level of expertise. For this reason, this framework uses the X3D standard [8], VRML successor, to render the simulated environment and user interface. Next section presents how X3D is used in our framework.

Using Extensible 3D for Visualization

Extensible 3D (X3D) is an ISO standard for real-time 3D graphics that defines a file format using XML and a run-time environment to be embedded in applications. The use of X3D features in our framework allows a wide variety of customization of the visualization client interface. X3D base components are sets of predefined nodes with some functionality. Those nodes are used to describe 3D objects and their relationships in a scene. Unlike Java3D or vendor-specific visualization software for HLA that must be compiled with the application, X3D is interpreted dynamically from human-readable text files. Also, many X3D content authoring software are being released with the support of well known modeling tools like Blender, for example. In order to create an X3D scene from a HLA simulation, the following base nodes are used to generate a basic X3D file:

- Prototype node: a new node type defined in terms of built-in or other prototype nodes by using the PROTO statement.
- Script node: a node used to program a behavior in the scene.
- Sensor node: a node that detects user inputs.
- Transform node: a node to position an object in the space.
- Inline node: a node that embeds a scene from another X3D file.

A mapping between the HLA object and interaction classes to X3D prototype nodes is performed and described in [9]. The prototype is named after the object or interaction class from the HLA/Federation Object Model - FOM (whose main function is to specify, in a common standardized format, the nature of the data exchanged among federates). Its attributes or parameters are linked from the prototype interface fields into a script node fields. Within the script node, function stubs are generated for the attributes or parameters converted to fields. Each prototype can link to other X3D files localized through URL using the inline node. By linking other files, the scene graph they define can be reused, such as the geometric models, animation scripts, user interface etc. Thus, for the same simulation, the visualization federate can have multiple interfaces with varying graphics level of details and specific interface for different users. The description of the framework is presented in the next section.

4. A Framework to Generate Vizualization Federates

Our framework generates visualization federates from a FOM and establishes communication with the RTI through Web Services. The federate is a web application and is ran within the context of a web browser. The application architecture is shown in Figure 1, and is divided into three layers:

- The **communication layer** uses the HLA WSDL API to exchange data with a federation through the RTI services;
- The **application layer** is the link between the communication and the presentation layers (Figure 2) and is generated by the framework based on a FOM;

- The **presentation layer** embeds the 3D browser into the federate and presents the data from the federation.

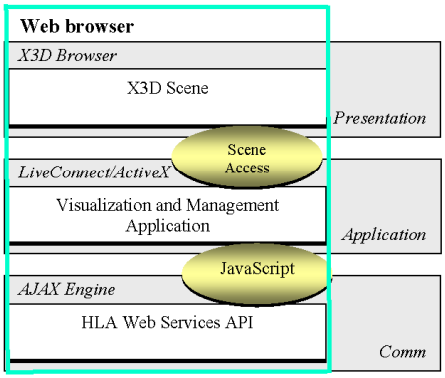


Figure 1. Application Architecture Overview

This layered architecture supports different X3D browsers (such as Flux Player [10] or the Xj3D toolkit [11]) and different HLA APIs by defining an implementation-specific middleware for web-based or standalone federate deployment. By embedding the visualization application into a web page, the simulation data can be presented in two ways: In the X3D scene; and using HTML document elements such as tables.

Also, when the application is embedded in an HTML document, the AJAX [12] techniques are used to make Web Service calls and exchange data within the application layers. The application written in Java is an applet that exchanges function calls with JavaScript using the LiveConnect or ActiveX technologies.

Application Layer Functionalities Description

The framework defines functions stubs that must be implemented to attend to federation-specific requirements a federate must meet, such as subscription and publication of data, synchronization points and time management policies, for example. Some functionality for object discovery and update are default for all classes but this can be changed: when a new object instance is discovered from the federation and notified to the Federate Ambassador (responsible for handling all outgoing information passed from the user simulation to the RTI), the corresponding prototype node is instanced into the scene and its handle and reference are kept in the application layer. The handle and reference are used to find and update the object attributes either in the federation or in the scene.

Interactions can be sent by triggering sensor nodes and object attributes can be changed through script nodes. Listener for field change events from the script nodes are generated in the application layer. Those listeners are callback functions that will request the desired service to the RTI Ambassador to send interactions or update object attributes.

A performance evaluation of the prototype was carried out in two different scenarios (Core 2 Duo 2.4GHz processor with 4GB RAM with a GeForce 7950 GT KO 512MB, Pentium 4 HT 3.2GHz with 2GB RAM with a GeForce 6800 XT 256MB). The simulation was hosted locally for both machines, receiving constant updates on position and other attributes of around 50 simulation-driven objects. In both platforms, the frame rate was kept over 15 FPS with low polygon count models. Further data analysis is needed though the update rates of data sent by each federate varies depending on the simulation model.

Performance will depend mainly on browser implementation, and to some extent, it will depend also on the number of scripts being performed. In case performance falls under acceptable response time, the 3D models being used can be replaced by less complex models. More complex simulations are under implementation to be tested in a LAN environment.

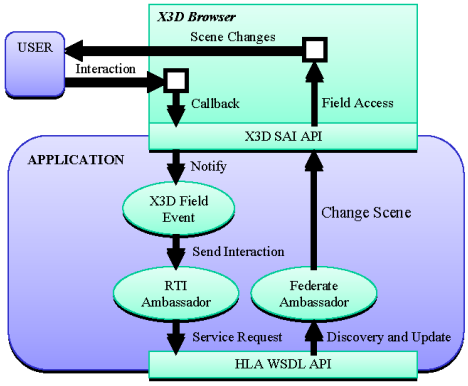


Figure 2. A Close Look into the Application Layer

5. Conclusions

This paper presents an HLA-based framework that generates HLA-based visualization interfaces with support to simulation control and management. Differently from other existing approaches, our framework can provide greater flexibility for customization and deployment of HLA-based

simulations in different hardware and software platforms, by making use of web services and X3D technologies. A performance evaluation of the prototype was made in two different platforms, achieving a 15 fps rate in a scenario with low complexity geometric models at constant update. The tests show that our solution can be a powerful tool for the generation of visualization federates, in which control and management can be done at simulation run time. A graphic tool is being built to make it easier to use our framework to build visualization federates.

6. Acknowledgements

This work was supported by Sao Paulo State Research Support Foundation, FAPESP, under Process 2006/00741-7 and CAPES.

References

[1] IEEE: "IEEE 1516, High Level Architecture (HLA)", www.ieee.org, March 2001.

[2] Doug Barry: "Web Services and Service Oriented Architectures", www.service-architecture.com

[3] J.M. Pullen et al., "Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment," Proc. Workshop HLA-Based Distributed Simulation on the Grid, LNCS 3038, Springer-Verlag, 2004, pp. 835-847.

[4] Björn Möller, Clarence Dahlin: "A first look at the HLA Evolved Web Service API", Proceedings of the 2006 European Simulation Interoperability Workshop", June 2006.

[5] Marshall, R., "Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie", Proceedings of the 1990 symposium on Interactive 3D graphics, pp.89-97, 1990.

[6] Johnson, C, Parker, S. G., Hansen, C., Kindlmann, G. L., Livnat, Y., "Interactive Simulation and Visualization", Computer, vol.32, no.12, pp.59-65, December 1999.

[7] van Wijk, J.J., Eindhoven, T.U., "The Value of Visualization", vis, p. 11, 16th IEEE Visualization 2005 (VIS 2005), 2005.

[8] Web 3D Consortium: "ISO/IEC 19775:2004, X3D Abstract", www.web3d.org, November 2005.

[9] Araujo, R.B., Iwasaki, F.M., Pizzolato, E., Boukerche, A. Creating HLA Compliant Simulations for Visualization in Heterogeneous Devices. Submitted to the 13th Intl. Symposium on 3D Web Technology, LA, USA.

[10] Media Machines, www.mediamachines.com, last access in September 2007.

[11] The Xj3D Project, www.xj3d.org, last access in September 2007.

[12] Garret, J.J., "Ajax: A New Approach to Web Applications", www.adaptivepath.com/publications/essays/archives/000385.php, February 2005.

APÊNDICE B – WebBased Visualization and Control of Distributed Simulations using HLA and Web Services (The 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications)

Web-Based Distributed Simulations Visualization and Control with HLA and Web Services

Boukerche, A.
SITE, University of Ottawa
Ottawa, Canada
boukerch @ site.uottawa.ca

Iwasaki, F. M. Araujo, R. B. Pizzolato, E. B.
Federal University of S. Carlos
SP, Brazil
{fabio_iwasaki, regina, ednaldo} @ dc.ufscar.br

Abstract

Dynamic Data Driven Application Systems (DDDAS), systems with online simulations models fed with sensorial data from the environment, are increasingly being used for emergency management strategic planning. This paper describes an approach to establish the communication and exchange information among the many subsystems composing a DDDAS using the High Level Architecture (HLA) infrastructure´s services. We present the LAViE-3D, a framework that generates Web-based simulation front-ends using 3D graphics to manage and control HLA-compliant distributed simulations. The LAViE-3D framework uses open-source software and is designed to make it easier to implement and deploy visualization applications on different platforms with customizable interface for simulations runtime control and management.

1. Introduction

Systems with online simulations models [3] fed with sensorial data from the environment can be used by emergency services (e.g., police, fire and medical services) to assist them in strategic planning during crisis management. These systems, also called Dynamic Data Drive Application Systems (DDDAS) [4], need to deliver quick and accurate results to provide decision support using available data from sensors, blueprints, databases, simulations models and other systems. DDDAS simulation back-end can be created ad-hoc [5], being designed bottom-up with available data and services for the emergency situation. To establish the communication between the many subsystems composing such DDDAS, the services from the High Level Architecture (HLA) infrastructure for distributed simulation can be used. The HLA Runtime Infrastructure (RTI) with Web Services technology makes possible the reuse of simulation models and interoperation between different systems.

This paper presents the LAViE-3D, a framework created as part of a larger project on emergency management, a collaborative effort between Networked Virtual Reality Lab (Federal University of S. Carlos, Brazil) and Paradise lab (SITE, University of Ottawa), as shown in Figure 1. LAViE-3D is used to generate Web-based simulation front-ends using 3D graphics to remotely control and manage HLA-compliant distributed simulations.

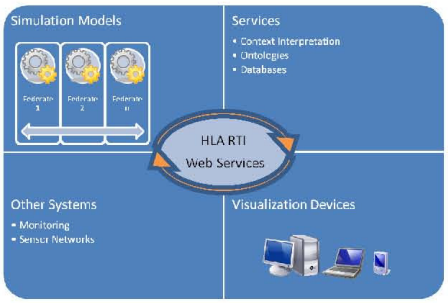


Figure 1. DDDAS for emergency management.

3D graphics and animation can provide more intuitive information on objects through natural human perception than their 2D representations, which might require some expertise to interpret. Although 3D processing requires more computational resources, now even handheld devices have 3D hardware acceleration. Using the Extensible 3D (X3D) standard for interactive 3D graphics and other Web-based technologies and open-standards, LAViE-3D deploys a cross-platform

application for simulation visualization and control. With the HLA and X3D standards, our framework can also be used to create an application for training simulations in collaborative virtual environments. Distributed simulations can be created for training scenarios that might increase the chances of survival for both rescuers and victims in the field during an emergency response.

Modeling and implementing a simulation from the ground up is time consuming, but for most problem domains, a specific simulation tool might be available through commercial off-the-shelf (COTS) software. The main drawbacks of using such tools are the time needed to learn how to use them and the risk of vendor lock-in, not to mention the price. While a specific simulation package can offer a complete solution for the selected domain, it can get quickly outdated and might not be reusable or might not communicate with other systems, wasting the time and effort (and also money) put in the implementation.

This paper is organized as follows: Section 2 outlines HLA and Web Services' features and introduces simulation visualization with X3D as used in our framework. The implementation of the LAViE-3D framework and the process to generate Web-based visualization and control federates are described in Section 4. Section 5 presents a use case and examines performance analysis results. In Section 6, related works are presented followed by conclusions and bibliographic references.

2. Distributed simulation and simulation visualization: technologies and standards

The LAViE-3D framework uses two well-known industry standards: The High Level Architecture standard for distributed simulation and the Extensible 3D for real-time 3D graphics. In the following subsections the main features of those standards and how they are used in the LAViE-3D framework are described.

2.1. High Level Architecture and Web Services

The LAViE-3D framework is based on the IEEE standard High Level Architecture (HLA) [6] to create discrete-event, process-oriented training simulations. The main concerns addressed by HLA are interoperability and reusability of simulation models. The HLA has three main components: Framework and Rules, Object Model Template Specification and Federate Interface Specification.

The Framework and Rules describe the responsibilities federates (simulations, supporting utilities or interfaces to live systems) and federations (set of federates working together) must hold for implementation.

The Object Model Template (OMT) Specification defines the Federation Object Model (FOM) and the Management Object Model (MOM). The FOM is where all the data exchanged among the federates are described in a common standardized format, which can be: a FED (Federation Execution Details) file, a lisp-like format for the HLA version 1.3 or; a XML document, also called FDD (Federation Document Data), for the HLA version 1516. The MOM is a set of management objects and interactions classes that can be included in a FOM and is used to extract information on the executing federation and its participating federates.

The Federate Interface Specification defines the standard services and interfaces for the software architecture called Runtime Infrastructure (RTI) to support the data exchange among the federates. These interfaces are arranged in seven service groups: federation management, declaration management, object management, ownership management, time management, data distribution management and support services. HLA is language independent but defines Ada 95, Java and C++ APIs. Currently, a new API based on Web Services is being added.

Web Services can be used to implement an architecture that meet the requirements of Service Oriented Architecture [7], such as interoperability between different systems, clear and unambiguous description language and retrieval of services. With the standardization of the HLA WSDL API [8], any system or device can connect directly to the RTI eliminating the need for customized software or proxy federate to route the calls from Web Services and a vendor specific implementation RTI.

The task of deploying Web Services and implementing a Web Services client can be facilitated by the currently available frameworks for a programming language of choice that generates implementation code from the WSDL. For the Java programming language, as example, the services and clients can be generated and implemented using the Java API for XML Web Services (JAX-WS) or Plain Old Java Objects (POJO) using the Apache CXF framework, the Apache Axis2 engine or an IDE like the NetBeans IDE or the Eclipse Web Tools Platform.

For this work, we used the NetBeans IDE, to create the Web Services implementation in Java from an early draft of the WSDL file describing the HLA services.

Our Web Services Provider (Figure 2), deployed in a Web container, such as Apache, Tomcat or Glassfish application servers, acts as a proxy to exchange messages between the RTI and the Consumer (federate). Because the current HLA WSDL API supports only request-response message exchange pattern between the Consumer and the Provider, a specific service (namely the *EvokeMultipleCallbacks* service) needs to be called to get callback messages from the RTI stored in the session managed by the Provider.

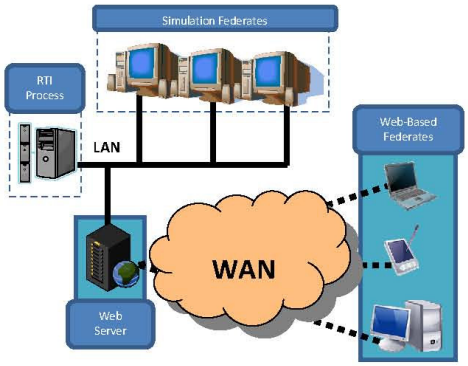


Figure 2. Web services RTI access for Federates from different platforms.

For faster interplay between federates and federation, the choice is still the use of the C++ or Java API. By using Web Services, a wider range of applications and platforms can share the same federation execution context and benefit from the RTI services.

2.2. Simulation visualization with Extensible 3D

There are three techniques for simulation visualization (with varying degree of integration between the simulation control and its visualization): post processing, tracking and steering [9]. In post processing technique, the resulting images from the visualization are generated after the simulation ends. In tracking, it is possible to view the current internal state of an executing simulation but not to change it. In the steering technique, the user has control over the simulation parameters and can change them at run time.

In a training simulation, the user's inputs are continuously changing its corresponding virtual environment state. To interact with a simulation (steering), it is necessary to define a strategy to further refine its visualization [10]. The visualization of a training simulation can be more or less efficient and effective depending on its specification and on the user's perception [11]. The user can have a richer experience if the visualization specification can be changed during the simulation execution or customized for the user's level of expertise. For this reason, this framework uses the X3D standard, the VRML successor, to allow real-time interaction with the simulated environment and dynamic changes in the user interface.

Extensible 3D (X3D) is an ISO standard for real-time 3D graphics that defines a file format using XML and a run-time environment to be embedded in applications. Its base components are sets of predefined nodes with some functionality. Those nodes are grouped in a scene graph and used to describe 3D objects and their relationships in a 3D scene. The use of X3D modular architecture and features in this framework allows the reuse of existing X3D worlds as well as the contents from many authoring software and 3D modeling tools like Blender, for example.

An X3D scene or world is presented through an X3D browser implementation. The same scene is expected to have the same behavior in different browser implementations, from handheld devices to immersive virtual reality systems. The X3D standard defines navigational paradigms and interaction mechanisms without specifying inputs or outputs devices. To make X3D worlds a multi-user environment, there are at least three approaches to include networking capabilities in the content: external scripting, internal scripting or direct networking.

External scripting through Web browser supported JavaScript makes use of AJAX [12] techniques to interact with the X3D scene graph using the Scene Access Interface (SAI) API and performing network communication with the *XMLHttpRequest* API. This approach is also known as Ajax3D [13].

Internal script node networking uses Java or C++ classes in Script nodes to communicate with the network. However, the content is not portable if the X3D browser does not support Java or C++.

Direct networking is a proposal from the X3D Networking Working Group to use a node interface in the scene to connect to the network.

We chose to use external scripting for the LAViE-3D framework, but to avoid defining our own communication protocols or mechanisms to transfer

data over the network we use the Web Services and the services defined in the HLA standard.

An HLA-compliant simulation is expected to have a FOM document. In our framework we developed a mapping from the HLA objects and interaction classes described in the FOM to X3D Prototype nodes. Figure 3 shows an example: to create an X3D scene from a HLA simulation five types of nodes are used: Prototype, Transform, Inline, Sensor and Script nodes.

```
<objects>
  <objectClass name="HLAobjectRoot">
    <attribute
name="HLAprivilegeToDeleteObject"
      dataType="NA"/>
    <objectClass name="ObjectX"/>
      <attribute name="AttributeA"
        dataType="HLAboolean"/>
      <objectClass name="ObjectY">
        <attribute name="AttributeB"
          dataType="HLAunicodeString"/>
      </objectClass>
    </objectClass>
  </objectClass>
</objects>
```

```
<ProtoDeclare name="ObjectY">
  <ProtoInterface>
    <field name="AttributeA"
      type="SFBool"
      accessType='inputOutput"/>
    <field name="AttributeB"
      type="SFString"
      accessType="inputOutput"/>
  </ProtoInterface>
  <ProtoBody>
    <Transform DEF="OBJECTY-Transform">
    </Transform>
    <Script DEF="OBJECTY-Script">
      <field name="AttributeA"
        type="SFBool"
        accessType="inputOutput"/>
      <field name="AttributeB"
        type="SFString"
        accessType="inputOutput"/>
      <IS>
        <connect nodeField="AttributeA"
          protoField="AttributeA"/>
        <connect nodeField="AttributeB"
          protoField="AttributeB"/>
      </IS>
      <![CDATA[ecmascript:
        function initialize() {}
        function set_AttribueA(value, timestamp){}
        function set_AttribueB(value, timestamp){}
      ]]>
    </Script>
  </ProtoBody>
</ProtoDeclare>
```

Figure 3. An example of FDD mapping to X3D.

Each Prototype can link to other X3D files localized through a URL using the Inline node. By linking other files, the scene graph they define can be reused, such as the geometric models, animation scripts, user interface etc. Therefore, for the same simulation, the visualization specification of the federate can use multiple geometric models with varying level of details and specific graphical interfaces for each different user.

3. Framework implementation

The LAViE-3D framework generates a Web application along with an X3D file for each HLA FOM (Section 2.2). This application is a federate run with the context of a Web browser and will establish communication with the RTI to join a specific federation using Web Services (Section 2.1). The components of the generated code of the visualization federate are shown in Figure 4, and they basically divide the application in three layers: Communication, Federate Application and Presentation layers.

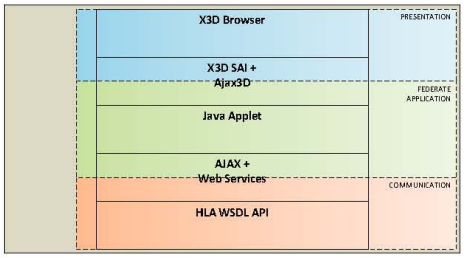


Figure 4. Three-layer Web-based federate.

The Communication layer is a Web Services client generated using the HLA WSDL API and is responsible for data exchanges with a federation by requesting the RTI services. AJAX techniques can be used to make Web Services calls and exchange data within the application layers. The federate application written in Java is an applet that exchanges function calls with JavaScript using the LiveConnect or ActiveX technologies. The Web Services client can also be implemented within the applet code.

The Federate Application layer handles the data coming from the HLA federation and the user interactions with the X3D scene. Based on the objects and interactions classes described in the FOM, functions stubs are defined but they must be implemented to attend to federation-specific requirements the federate must meet such as subscription and publication of data, synchronization points and time management policies, for example.

Some functionality for discovery and update of objects are generated by default for all classes: when a

new object instance is discovered from the federation and notified to the federate, the corresponding prototype node is instanced into the scene and its handle and reference are kept in the application layer. The handle and reference are used to find and update the object attributes either in federation or in the scene.

Interactions can be sent by triggering sensor nodes and object attributes can be changed through script nodes. Listener for field change events from the script nodes are generated in the federate application layer. Those listeners are callback functions that will request the desired service to the RTI or Provider to send interactions or update object attributes. Figure 5 shows an overview of the HLA-X3D Communication.

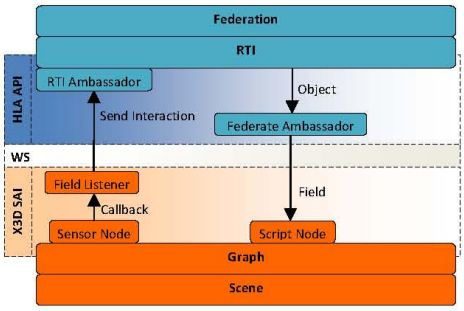


Figure 5. HLA-X3D communication.

The presentation layer embeds an X3D browser into the federate and presents the data from the federation. Different X3D browsers with the Scene Access Interface implementations, such as Flux Player (http://www.mediamachines.com/) or the Xj3D toolkit (http://www.xj3d.org/), are available for Web-based or standalone federate deployment. By embedding the visualization application into a Web page, the simulation data can be presented in the X3D scene and also use HTML document elements (e.g., tables).

4. Use case and performance analysis

To validate our framework and to find better practices for virtual environment modeling, we implemented a multi-user interactive simulation of cars with rigid body physics. In our tests we used for each federate a desktop computer with a Core 2 Duo 2.4GHz processor with 2GB RAM and a GeForce 7600 GS (512MB RAM) video board. The RTI process was executing with the Web services (deployed in the Apache Tomcat server) in the same machine (Core 2 Duo 2.4GHz processor with 4GB RAM).

Each car in the federation is described with the attributes: position, orientation, speed, steering and handbrake and brake indicator and the X3D model used for visualization from the SAVAGE library is composed of 1298 triangles and 795 vertices. In the visualization federate code we included a physics simulation component using the Open Dynamics Engine library (http://www.odejava.org/). Each federate (now a simulation federate) executes a simultaneous simulation using the speed and steering values from each car instanced in the federation. Since each federate simulates its own world instance, a federation wide synchronization is needed at regular interval of times with the position and orientation of every car. For this test, each federate registered ten car instances and published and subscribed all their attributes. No optimization was made in the simulation code and we also forced each federate to send all data of their respective car instances every simulation step before advancing the federation logical time to stress the publish/subscribe operations with a large number of messages going through the RTI.

The chart in Figure 6 shows the frame rates obtained every time after ten cars were added to the scene using the Xj3D browser embedded in a Java applet. When a new car is added to the scene, a noticeable frame rate drop occurs while the geometries and textures load. The maximum frame rate was obtained while moving around the space. The minimum frame rate was obtained while all the objects were visible to the display. If most objects are hidden from the viewpoint, the frame rate could reach 60 FPS (the maximum). When the maximum FPS dropped to below 20 after 100 cars, the car textures didn't load anymore because the default memory limit was used for the Java applet (around 90MB). The Xj3D browser we used kept loading the same texture (a JPEG file) for every car inserted in the scene instead of creating a single instance to be referenced.

In this test, the open-source RTI implementation in Java from the Portico Project (http://porticoproject.org/) was used with a federate proxy acting as the Web Service provider. Figure 7 shows a comparison between Java API and WSDL API. The Java implementation uses JSOP (Java Serialized Object Protocol) for communication. Analyzing the network traffic it was shown that the messages between the federates and RTI were broken in small packets segments by JSOP (up to 14 Bytes each) while the SOAP messages (up to 3 KB) were segmented in 3 or 4 parts. When the server reached a limit of processing

around 150.000 packets per second, the response time for the Java implementation was longer than the Web Services implementation for the same federation.

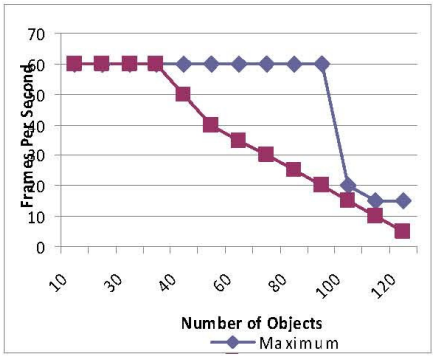


Figure 6. Frame rates using Xj3D browser.

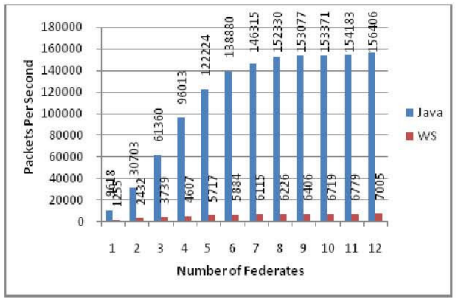


Figure 7. Java API x WSDL API Number of packets sent.

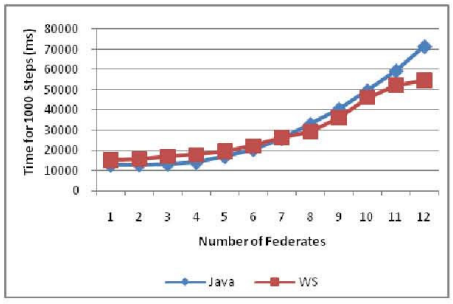


Figure 8. Java API x WSDL API Communication delay.

Figure 8 shows a comparison between Java API and WSDL API latency. While it is expected increased overhead and network latency when using Web Services, the message segmentation detected when using JSOP actually dropped performance of the RTI Java implementation below the Web Service´s as more federates exchange messages between them.

5. Related work

Unlike other approaches that use Java3D for HLA-compliant simulations visualization [14, 15], with X3D the user interface can be customized for different software and hardware platforms and its contents can be reused by other federations. COTS tools using X3D available from Blaxxun Platform (http://www.blaxxun.com/), Bitmanagement Software (http://www.bitmanagement.de) and Octaga (http://www.octaga.com) are attractive as they seem to be more reliable and easy to use, but their reusability can be limited because they focus on particular applications. Also, if their features are not based on open standards, integrating them with other systems will not be possible.

Recent works using X3D include collaborative virtual environments and e-learning platforms like EVE [16]. A similar use of HLA and X3D is described for a Virtual Hospital [17]. The differential of our framework is the flexibility for creating visualization applications on different platforms with customizable interface for simulations runtime control and management.

6. Conclusions

This paper presented LAViE-3D, a HLA-based framework that generates visualization interfaces using X3D with support to simulation control and management as part of a DDDAS project for emergency management. Moreover, our framework can be used to generate distributed simulations for training in virtual environments. With further optimization and data filtering strategies, the tests show that our solution can be a useful tool for the generation of visualization federates on different platforms, in which control and management can be done at simulation runtime.

7. Acknowledgments

This work was supported by Sao Paulo State Research Support Foundation, FAPESP, under Process 2006/00741-7 and CAPES. In Canada, this work was partially supported by the Canada Research Chair Program, NSERC, and Early Researcher Award.

8. References

[1] Madey, G., Barabasi, A., Chawla, N., Gonzalez, M., Hachen, D., Lantz, B., Pawling, A., Schoenharl, T., Szabo, G., Wang, P. and Yang, P. "Enhanced Situational Awareness: Application of DDDAS Concepts to Emergency and Disaster Management", *in Proceedings of the 7th International Conference on Computational Science*, v. 4487, Jul. 2007, pp. 1090-1097.

[2] Becerra-Fernandez, I., Madey, G., Prietula, M., Rodriguez, D., Valerdi, R. and Wright, T. "Design and Development of a Virtual Emergency Operations Center for Disaster Management Research, Training, and Discovery", *in Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, Jan. 2008, pp. 27-27.

[3] Davis, W. "On-Line Simulation: Need and Evoling Research Requirements", *in Handbook of Simulation*. Edited by Jerry Banks. John Wiley & Sons, Inc., 1998, pp. 465-516.

[4] Darema, F. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements", *in Proceedings of the 4th International Conference on Computational Science*, v. 3038, May 2004, pp. 662-669.

[5] Fujimoto, R., Hunter, M., Sirichoke, J., Palekar, M., Kim, H. and Suh, W, "Ad Hoc Distributed Simulations", *in Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, Jun. 2007, pp. 15-24.

[6] IEEE, "Std. 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules", http://www.ieee.org/, 2001.

[7] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S. "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", *IEEE Internet Computing*, v. 6, n. 2, Mar./Apr. 2002, pp. 86-93.

[8] Möller, B. and Dahlin, C. "A First Look at the HLA Evolved Web Service API", *in Proceedings of 2006 Euro Simulation Interoperability Workshop*, Simulation Interoperability Standards Organization, 06E-SIW-061, 2006.

[9] Marshall, R., Kempf, J., Dyer, S. and Yen, C. "Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie", *in Proceedings of the 1990 Symposium on Interactive 3D Graphics*, 1990, pp. 89-97.

[10] Johnson, C., Parker, S., Hansen, C., Kindlmann, G. and Livnat, Y. "Interactive Simulation and Visualization", *Computer*, v. 32, n. 12, Dec. 1999, pp. 59-65.

[11] Wijk, J. and Eindhoven, T. "The Value of Visualization", *16th IEEE Visualization 2005*, Oct. 2005, pp. 79-86.

[12] Garret, J. "Ajax: A New Approach to Web Applications", http://www.adaptivepath.com/publications/essays/archives/000385.php, 2005.

[13] Parisi, T. "Ajax3D White Paper", http://www.ajax3d.org/whitepaper/, 2006.

[14] Salisbury, C., Farr, S. and Moore, J. "Web-Based Simulation Visualization Using Java3D", *in 1999 Winter Simulation Conference Proceedings*, Volume 2, 1999, pp. 1425-1429.

[15] Hsia, W. 2000. "An HLA-Based Simulation Environment for Virtual Reality via Java3D", Master Thesis, National Sun Yat-Sen University Kaohsiung.

[16] Bouras, C., Tegos, C., Triglianos, V. and Tsiatsos, T. "X3D Multi-user Virtual Environment Platform for Collaborative Spatial Design", *in Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*, Jun. 2007, pp. 40-40.

[17] Ahmad, L., Boukerche, A., Al Hamidi, A., Shadid, A. and Pazzi, R. "Web-Based e-Learning in 3D Large Scale Distributed Interactive Simulations Using HLA/RTI", *IEEE International Symposium on Parallel and Distributed Processing, 2008*, Apr. 2008, pp. 1-4.

APÊNDICE C – carfederation-fom.fed (MOM omitido)

```
(FED
  (Federation CarFederation)
  (FEDversion v1.3)
  (spaces
  )
  (objects
    (class ObjectRoot
      (attribute privilegeToDelete reliable timestamp)
      (class RTIprivate)
      (class Car
        (attribute positionX reliable timestamp)
        (attribute positionY reliable timestamp)
        (attribute positionZ reliable timestamp)
        (attribute orientationX reliable timestamp)
        (attribute orientationY reliable timestamp)
        (attribute orientationZ reliable timestamp)
        (attribute orientationA reliable timestamp)
        (attribute id reliable timestamp)
        (attribute handbrake reliable timestamp)
        (attribute speed reliable timestamp)
        (attribute steering reliable timestamp)
        (attribute name reliable timestamp)
        (class RigidBody
          (attribute chassisPositionX reliable timestamp)
          (attribute chassisPositionY reliable timestamp)
          (attribute chassisPositionZ reliable timestamp)
          (attribute frontLeftWheelPositionX reliable timestamp)
          (attribute frontLeftWheelPositionY reliable timestamp)
          (attribute frontLeftWheelPositionZ reliable timestamp)
          (attribute frontRightWheelPositionX reliable timestamp)
          (attribute frontRightWheelPositionY reliable timestamp)
          (attribute frontRightWheelPositionZ reliable timestamp)
          (attribute rearLeftWheelPositionX reliable timestamp)
          (attribute rearLeftWheelPositionY reliable timestamp)
```

```
          (attribute rearLeftWheelPositionZ reliable timestamp)
          (attribute rearRightWheelPositionX reliable timestamp)
          (attribute rearRightWheelPositionY reliable timestamp)
          (attribute rearRightWheelPositionZ reliable timestamp)
          )
        )
      )
    )
  )
  (interactions
    (class InteractionRoot reliable timestamp
      (class RTIprivate reliable timestamp)
      (class Control reliable timestamp
        (parameter id)
        (class Accelerate reliable timestamp
          (parameter force)
        )
        (class Steer reliable timestamp
          (parameter steer)
        )
      )
    )
  )
)
```

APÊNDICE D – carfederation.x3dv

```
#X3D V3.0 utf8
PROFILE Immersive

PROTO Car [
  # CAR_Transform fields
  inputOutput SFRotation rotation 0 0 1 0
  inputOutput SFVec3f scale 1 1 1
  inputOutput SFVec3f translation 0 0 0
  # CAR_Script fields
  inputOutput SFString name ""
  inputOutput SFInt32 id -1
  inputOutput SFBool handbrake FALSE
  inputOutput SFFloat speed 0
  inputOutput SFFloat steering 0
] {
  DEF CAR_Transform Transform {
    children [
      Transform {
        children [
          DEF CHEVY_Inline Inline {
            url ["Chevy/ChevyTahoe.x3d"]
          }
        ]
        rotation 0 1 0 1.57
        translation 0 -1.3 -0.1
      }
      Transform {
        children [
          Billboard {
            #axisOfRotation 0 0 0
            children [
              Shape {
              geometry DEF CAR_Text Text {
                fontStyle FontStyle {
```

```
                justify "MIDDLE"
              }
              string [""]
            }
            }
          ]
        }
      ]
      translation 0 3 0
    }
  ]
  rotation IS rotation
  scale IS scale
  translation IS translation
}
DEF CAR_Script Script {
  inputOutput SFString name IS name
  inputOutput SFInt32 id IS id
  inputOutput SFBool handbrake IS handbrake
  inputOutput SFFloat speed IS speed
  inputOutput SFFloat steering IS steering
  outputOnly MFString string_changed
  url ["ecmascript:
    var _name;
    var _id;
    var _handbrake;
    var _speed;
    var _steering;
    function initialize()
    {
     _name = name;
     _id = id;
     _handbrake = handbrake;
     _speed = speed;
     _steering = steering;
    }
    function set_name(value) {
```

```
        var temp = new MFString(value);
        string_changed = temp;
        temp = null;
       }
       function set_id(value) {}
       function set_handbrake(value) {}
       function set_speed(value) {}
       function set_steering(value) {}
    "]
  }
  ROUTE CAR_Script.string_changed TO CAR_Text.set_string
}

PROTO Accelerate [
  inputOutput SFInt32 id -1
  inputOutput SFFloat force 0
] {
  DEF ACCELERATE_Script Script {
    inputOutput SFInt32 id IS id
    inputOutput SFFloat force IS force
    url ["ecmascript:
      function initialize() {}
      function set_id(value) {}
      function set_force(value) {}
    "]
  }
}

PROTO Steer [
  inputOutput SFInt32 id -1
  inputOutput SFFloat steer 0
] {
  DEF STEER_Script Script {
    inputOutput SFInt32 id IS id
    inputOutput SFFloat steer IS steer
    url ["ecmascript:
      function initialize() {}
```

```
      function set_id(value) {}
      function set_steer(value) {}
    "]
  }
}

DEF ACCELERATE Accelerate {}

DEF STEER Steer {}
```

APÊNDICE E – carfederation.x3d

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile='Immersive' >
<head>
</head>
<Scene>
<ProtoDeclare name='Car'>
 <ProtoInterface>
  <field name='rotation' accessType='inputOutput' type='SFRotation' value='0 0
1 0'/>
  <field name='scale' accessType='inputOutput' type='SFVec3f' value='1 1 1'/>
  <field name='translation' accessType='inputOutput' type='SFVec3f' value='0 0
0'/>
  <field name='name' accessType='inputOutput' type='SFString' value=''/>
  <field name='id' accessType='inputOutput' type='SFInt32' value='-1'/>
  <field name='handbrake' accessType='inputOutput' type='SFBool'
value='FALSE'/>
  <field name='speed' accessType='inputOutput' type='SFFloat' value='0'/>
  <field name='steering' accessType='inputOutput' type='SFFloat' value='0'/>
 </ProtoInterface>
 <ProtoBody>
  <Transform DEF='CAR_Transform'>
   <Transform DEF='dad_CHEVY_Inline'
    containerField='children'
    translation='0 -1.3 -.1'
    rotation='0 1 0 1.57'>
    <Inline DEF='CHEVY_Inline'
     containerField='children'
     url='"Chevy/ChevyTahoe.x3d"'/>
   </Transform>
   <Transform
    containerField='children'
    translation='0 3 0'>
    <Billboard
     containerField='children'
```

```
   axisOfRotation='0 0 0'>
   <Shape
    containerField='children'>
    <Appearance
     containerField='appearance'/>
    <Text DEF='CAR_Text'
     containerField='geometry'
     string='""'
     maxExtent='0.000'>
     <FontStyle
      containerField='fontStyle'
      family='SERIF'
      style='PLAIN'
      justify='"MIDDLE"
       "BEGIN"'
      size='1.000'
      spacing='1.000'/>
    </Text>
   </Shape>
  </Billboard>
 </Transform>
 <IS>
  <connect nodeField='rotation' protoField='rotation'/>
  <connect nodeField='scale' protoField='scale'/>
  <connect nodeField='translation' protoField='translation'/>
 </IS>
</Transform>
<Script DEF='CAR_Script'
 directOutput='FALSE'
 mustEvaluate='FALSE'>
 <field name='name' accessType='inputOutput' type='SFString' value=''/>
 <field name='id' accessType='inputOutput' type='SFInt32' value=''/>
 <field name='handbrake' accessType='inputOutput' type='SFBool' value=''/>
 <field name='speed' accessType='inputOutput' type='SFFloat' value=''/>
 <field name='steering' accessType='inputOutput' type='SFFloat' value=''/>
 <field name='string_changed' accessType='outputOnly' type='MFString'/>
 <![CDATA[ecmascript:
```

```
      var _name;
      var _id;
      var _handbrake;
      var _speed;
      var _steering;
      function initialize()
      {
       _name = name;
       _id = id;
       _handbrake = handbrake;
       _speed = speed;
       _steering = steering;
      }
      function set_name(value) {
       var temp = new MFString(value);
       string_changed = temp;
       temp = null;
      }
      function set_id(value) {}
      function set_handbrake(value) {}
      function set_speed(value) {}
      function set_steering(value) {}


   ]]>
   <IS>
    <connect nodeField='name' protoField='name'/>
    <connect nodeField='id' protoField='id'/>
    <connect nodeField='handbrake' protoField='handbrake'/>
    <connect nodeField='speed' protoField='speed'/>
    <connect nodeField='steering' protoField='steering'/>
   </IS>
  </Script>
  <ROUTE fromNode='CAR_Script' fromField='string_changed' toNode='CAR_Text'
toField='set_string'/>
 </ProtoBody>
</ProtoDeclare>
<ProtoDeclare
```

```
 name='Accelerate'>
 <ProtoInterface>
  <field name='id' accessType='inputOutput' type='SFInt32' value='-1'/>
  <field name='force' accessType='inputOutput' type='SFFloat' value='0'/>
 </ProtoInterface>
 <ProtoBody>
  <Script DEF='ACCELERATE_Script'
   directOutput='FALSE'
   mustEvaluate='FALSE'>
   <field name='id' accessType='inputOutput' type='SFInt32' value=''/>
   <field name='force' accessType='inputOutput' type='SFFloat' value=''/>
   <![CDATA[ecmascript:
      function initialize() {}
      function set_id(value) {}
      function set_force(value) {}

   ]]>
   <IS>
    <connect nodeField='id' protoField='id'/>
    <connect nodeField='force' protoField='force'/>
   </IS>
  </Script>
 </ProtoBody>
</ProtoDeclare>
<ProtoDeclare
 name='Steer'>
 <ProtoInterface>
  <field name='id' accessType='inputOutput' type='SFInt32' value='-1'/>
  <field name='steer' accessType='inputOutput' type='SFFloat' value='0'/>
 </ProtoInterface>
 <ProtoBody>
  <Script DEF='STEER_Script'
   directOutput='FALSE'
   mustEvaluate='FALSE'>
   <field name='id' accessType='inputOutput' type='SFInt32' value=''/>
   <field name='steer' accessType='inputOutput' type='SFFloat' value=''/>
   <![CDATA[ecmascript:
```

```
      function initialize() {}
      function set_id(value) {}
      function set_steer(value) {}

   ]]>
   <IS>
    <connect nodeField='id' protoField='id'/>
    <connect nodeField='steer' protoField='steer'/>
   </IS>
  </Script>
 </ProtoBody>
</ProtoDeclare>
<ProtoInstance DEF='ACCELERATE' name='Accelerate'/>
<ProtoInstance DEF='STEER' name='Steer'/>
</Scene>
</X3D>
```