4
Abordagens Tecnológicas para o Desenvolvimento de Ambientes Virtuais Distribuídos

Depois de elaborarmos um metamodelo para auxiliar a configurar a arquitetura do espaço de trabalho virtual colaborativo, neste capítulo nos concentramos em rever e analisar os DVEs (*Distributed Virtual Environments* – Ambientes Virtuais Distribuídos) existentes de modo a selecionar quais deles possuem as características mais apropriadas para servir como base para apoiar a implementação que será desenvolvida para validar nosso metamodelo.

Tratamos de duas das abordagens tecnológicas atuais para criar e construir sistemas distribuídos, isto é, *Middleware* e Ambientes Virtuais Distribuídos Puros, e a seguir descrevemos brevemente os sistemas que foram avaliados.

4.1.
Considerações sobre o Desenvolvimento de Ambientes Colaborativos

Para ser eficiente, um CVE (*Collaborative Virtual Environment* – Ambiente Virtual Colaborativo), que é outro nome para DVE, deve garantir suporte à distribuição, uma rede eficiente com grande largura de banda, comunicação adequada e modelos de armazenamento de dados. Além disso, esse ambiente distribuído deve propiciar interfaces colaborativas, operações e metáforas apropriadas para permitir que os usuários trabalhem efetivamente.

Para um ambiente virtual ser colaborativo, ele precisa ser distribuído entre os participantes que desejem compartilhá-lo. A escolha da arquitetura de comunicação é parametrizada pelo grau em que as estruturas de dados que representam o ambiente virtual são replicadas ou postas em cache nos nós computacionais e na tecnologia base de transporte (West & Hubbold, 2001).

Entretanto, qualquer que seja a tecnologia, as latências de comunicação são em última análise insuperáveis. A interação compartilhada em tempo real é particularmente sensível ao atraso, especialmente no caso de interfaces imersivas.

A sensibilidade da aplicação a tal atraso depende do grau desejado de interação genuinamente compartilhada. Por exemplo, em um passeio passivo através de um modelo, pequenas discrepâncias temporais entre as experiências do mundo pelos participantes podem passar despercebidas. No extremo oposto, duas pessoas operando uma vareta através dos limites de um equipamento de petróleo irão constatar que a coordenação é difícil ou impossível com qualquer atraso apreciável.

Se não é possível atingir a sincronia adequada, uma solução é ao menos focar os recursos naquelas atividades que são mais sensíveis ao atraso, i.e., naquelas que produzem as descontinuidades mais pronunciadas de experiência perceptual quando o atraso está presente. Isto envolve determinar adequadamente uma métrica de significância e urgência de priorização dentro da infra-estrutura de comunicação (West & Hubbold, 2001). Em última análise, como os componentes de um sistema CVE interagem de maneiras complexas, o projetista precisa considerar a aplicação como um sistema unificado. Invariavelmente, tentar otimizar um elemento de um CVE pode ter impacto adverso no comportamento de outros componentes. De fato, o desenvolvimento de um CVE é um ato de difícil balanceamento entre requisitos conflitantes de engenharia (Singhal & Zyda, 1999).

É impossível prever os requisitos de rede dos CVEs de modo isolado; preferivelmente, necessitamos de um modelo de operação de CVE que englobe a aplicação, o usuário e assuntos de software e hardware. Greenhalgh (2001) propõe um modelo que tem seis camadas:

1. Requisitos de tarefa/aplicação/colaboração: o que as pessoas querem ou tentam fazer.
2. Comportamento do usuário: que ações particulares as pessoas executam e quando. É importante considerar o comportamento do usuário quando se está tentando entender os requisitos de rede dos CVEs porque quase todos esses requisitos derivam do que os usuários escolhem fazer e quando escolhem fazê-lo. Por exemplo, se os usuários falam apenas raramente, e nunca ao mesmo tempo, então o requisito de rede para áudio pode ser muito limitado. Por outro lado, para alguns cenários de uso, é necessário haver uma largura de

banda suficiente para cada usuário poder falar ao mesmo tempo. Este poderia ser o caso, por exemplo, de sistemas de emergência.

3. Comportamento do processo: como a aplicação reage. Todo sistema CVE tem seu próprio conjunto de capacidades e suas próprias maneiras de representar usuários e mundos virtuais. Por exemplo, cada sistema suporta um diferente subconjunto de ações possíveis do usuário. Novamente, o cenário da emergência pode ser um bom exemplo: enquanto as pessoas responsáveis por toda a operação podem executar qualquer comando, outros especialistas podem executar apenas as tarefas para as quais foram designados.
4. Arquitetura de distribuição: o que se comunica com o quê. A escolha da arquitetura de distribuição determina que informação precisa ser comunicada a quais partes do sistema. A arquitetura de distribuição irá portanto determinar como os mundos virtuais são organizados e divididos, quais usuários podem se comunicar com quais outros e se existem ou não processos ou servidores de coordenação central. As arquiteturas de distribuição podem basicamente ser divididas em três modelos: cliente/servidor, *peer-to-peer* e híbrido.
5. Protocolos de comunicação: como a informação é trocada. Os protocolos podem ser *unicast* ou *multicast*. O desenvolvedor tem que escolher a melhor combinação arquitetura de distribuição/protocolo de comunicação que se adeqüe à aplicação particular que está sendo elaborada.
6. Comunicação na rede: o que realmente acontece na rede. Existem alguns requisitos de rede que dependem do lugar exato na rede em que a aplicação está sendo executada e como a rede está conectada (p.ex. LAN – *Local Area Network* ou WAN – *Wide Area Network*), já que a largura de banda real em diferentes partes da rede irá variar. Isto também inclui sistemas móveis.

A escolha de uma arquitetura de distribuição é o principal fator determinante de grande parte do "sentimento" de multi-usuário de um CVE. Por exemplo, a arquitetura de distribuição determinará tipicamente:

- A estrutura possível de um mundo virtual, isto é se ele lida bem com grandes espaços externos ou com interiores de construções complexos. Na indústria de óleo e gás, os usuários têm que lidar com ambas as situações, por exemplo ao construirem uma estrutura *offshore* complexa (interior complexo) ou monitorarem oleodutos dispostos sobre uma montanha (grande espaço externo). O desafio se torna ainda mais complexo em alguns cenários de emergência, em que os especialistas possivelmente têm que examinar vários detalhes da estrutura usando um ponto-de-vista egocêntrico (espaço interno), observando ao mesmo tempo o efeito da simulação de possíveis operações de emergência usando um ponto-de-vista exocêntrico (espaço externo). Nesses casos, o sistema tem que tratar simultaneamente dos requisitos dos espaços interno e externo.
- Quantos usuários podem compartilhar um único mundo virtual. Hoje em dia, as aplicações de óleo e gás típicas são executadas em não mais do que meia dúzia de salas de visualização simultaneamente, com não mais do que 20 especialistas em cada uma. Em alguns casos, especialistas espalhados pelo espaço externo podem ter a necessidade de entrar no mundo virtual usando um sistema móvel.
- A dinâmica do mundo virtual, incluindo as maneiras como os comportamentos podem ser executados e que coisas podem ou não ser mudadas. Por exemplo, é extremamente difícil mudar dinamicamente geometrias básicas em alguns sistemas, enquanto isto é relativamente fácil em outros. Esta é uma questão muito importante das aplicações de óleo e gás, tendo em vista a complexidade dos dados envolvidos. O que usualmente é feito é replicar o modelo de dados nos locais antes de a sessão colaborativa começar. Para evitar sobrecarga na rede, normalmente durante uma sessão é permitido aos usuários fazer apenas atualizações incrementais, transmitindo apenas pequenos dados e/ou mensagens de ações.

Até este momento, é razoável dizer que não existe uma escolha universal de arquitetura de distribuição ou comunicação, mas mais propriamente uma faixa de contrabalanços em questões de performance e organização.

Outro requisito importante em ambientes distribuídos é a necessidade de se ter um modelo de armazenamento de dados para suportar as atividades colaborativas. De acordo com Macedonia e Zyda (1997), os modelos de armazenamento de dados podem ser classificados em: *(i)* bancos de dados centralizados e compartilhados; *(ii)* bancos de dados replicados de mundos homogêneos; *(iii)* bancos de dados distribuídos e compartilhados com modificações *peer-to-peer*; e *(iv)* bancos de dados cliente/servidor distribuídos e compartilhados.

Os bancos de dados centralizados fornecem um modelo de programação fácil. O desenvolvedor pode acessar informações em qualquer tempo, ignorando a existência da rede. A única indicação de se um dado está em cache ou está remoto será no tempo de acesso. Adicionalmente, a sincronização está assegurada visto que (ignorando a operação de colocação em memória cache) existe apenas uma representação de cada objeto no banco de dados. A principal desvantagem dos bancos de dados centralizados é uma perda de performance em potencial, associada com o acesso dos dados através da rede.

Os bancos de dados replicados são populares em muitos sistemas de realidade virtual. As representações locais dos dados significam que as atualizações locais são rápidas.

Devido ao alto valor dos seus dados, as aplicações de óleo e gás possuem requisitos rigorosos de consistência e segurança de bancos de dados, sugerindo um modelo de bancos de dados centralizado e compartilhado. O conceito de grade computacional parece atender a esses requisitos, já que ele é conceitualmente centralizado com dados reais espalhados em vários lugares de modo transparente para a aplicação. Todavia, devido a requisitos de performance e como a grade computacional ainda é um conceito novo, pode ser observado que quase todas as aplicações práticas no momento usam o modelo replicado, com algumas permitindo aos usuários fazer o que chamamos de atualizações incrementais.

4.2. Abordagens de Arquiteturas

Depois de discutirmos sobre considerações gerais acerca do desenvolvimento de ambientes colaborativos, iremos agora nos concentrar em duas abordagens tecnológicas atuais para criar e construir sistemas distribuídos, isto é, *Middleware* e Ambientes Virtuais Distribuídos Puros, descrevendo brevemente os sistemas que foram avaliados.

4.2.1. *Middleware*

Os sistemas baseados na abordagem *Middleware* têm o objetivo de criar uma interface que estabeleça um padrão prático e portável para a comunicação entre processos. Nas subseções seguintes, descrevemos brevemente cinco desses sistemas.

4.2.1.1. *Message Passing Interface* (MPI)

O objetivo da MPI (*Message Passing Interface* – Interface de Passagem de Mensagens) dito de forma simples é desenvolver um padrão amplamente utilizado para escrever programas de passagem de mensagens (Dongarra et al., 1993). As principais vantagens de se estabelecer um padrão de passagem de mensagens é a portabilidade e a facilidade de uso. Em um ambiente de comunicação de memória distribuída no qual as rotinas e/ou abstrações de nível mais alto são construídas sobre rotinas de passagem de mensagens de nível mais baixo, os benefícios da padronização ficam particularmente evidentes.

A passagem de mensagens é um paradigma amplamente utilizado em certas classes de máquinas paralelas, especialmente aquelas com memória distribuída. Embora existam diferentes variações, o conceito básico de comunicação entre processos por meio de mensagens é bem entendido. Nos últimos dez anos, foi feito um progresso substancial na modelagem de aplicações importantes por meio deste paradigma. Cada fabricante implementou sua própria variação.

A MPI-1 (Dongarra et al., 1993) foi completada no início de 1993 e enfocava principalmente a comunicação ponto-a-ponto. Ela não incluía nenhuma rotina de comunicação coletiva e não era segura com relação a *threads*.

A MPI-2 (2006) adicionou algumas extensões ao padrão básico. A MPI/RT é a última versão. Ela incorpora inúmeras facilidades, tais como uma API (*Application Programmer's Interface*), suporte para ambientes heterogêneos, *bindings* C e Fortran, canais ponto-a-ponto e coletivos, uma interface de comunicação confiável e segurança com relação a *threads*.

4.2.1.2. CORBA e TAO

CORBA – *Common Object Request Broker Architecture* (OMG, 1995) é um padrão aberto para comunicação entre processos locais e remotos. Sobre o pacote de transmissão, CORBA oferece diversos serviços padrão que automatizam várias tarefas comuns de programação de rede, como registro, localização e ativação de objetos; operação de demultiplex de solicitação e despacho de operação. No nível base, a comunicação é definida por RPCs (*Remote Procedure Calls* – Chamadas de Procedimento Remoto) entre processos. O desenvolvedor define a interface para os processos usando a IDL (*Interface Description Language* – Linguagem de Descrição de Interface). A IDL é compilada em uma interface que existe entre o cliente e o servidor. Depois disso, os detalhes da comunicação são manipulados por CORBA. Os serviços mencionados acima são então dispostos em camadas sobre este *framework* de comunicação.

As implementações iniciais de CORBA não lidavam com questões de processamento em tempo real abertamente (embora isto não tenha impedido o desenvolvimento de um pequeno número de aplicações de realidade virtual baseadas em CORBA, p.ex., COVRA-CAD (Junghyun et al., 1998)).

Doug Schmidt e colegas da Universidade do Estado de Washington desenvolveram *The ACE ORB* (TAO) (Schmidt, 2006), um *framework middleware* compatível com CORBA que trata de alguns desafios de tempo real do processamento distribuído. Schmidt e colegas identificam um conjunto de padrões e componentes de *framework* que podem ser aplicados sistematicamente

para eliminar vários aspectos tediosos, sujeitos a erros e não portáveis do desenvolvimento e da manutenção de aplicações distribuídas.

4.2.1.3. Grade Computacional e Globus

A grade computacional pode ser definida pela seguinte lista de verificação imaginária:

"(Uma Grade Computacional…)

- Coordena recursos que não estão sujeitos a um controle centralizado – (Uma Grade Computacional integra e coordena recursos e usuários que experimentam diferentes domínios de controle – por exemplo, o *desktop* do usuário vs. computação central; diferentes unidades administrativas da mesma companhia; ou diferentes companhias – e trata das questões de segurança, política, pagamento, associação e assim por diante, que surgem nessas configurações.)
- Utilizando protocolos e interfaces padrões, abertos e de propósito geral – (Uma Grade Computacional é construída a partir de protocolos e interfaces multipropósito que tratam de questões fundamentais tais como autenticação, autorização, descoberta de recursos e acesso a recursos...)
- Para oferecer qualidades não triviais de serviço – (Uma Grade Computacional permite que seus recursos constituintes sejam usados de uma maneira coordenada para oferecer várias qualidades de serviço, relacionadas por exemplo com o tempo de resposta, *throughput*, disponibilidade e segurança, e/ou co-alocação de múltiplos tipos de recursos para atender a demandas complexas do usuário, de modo que a utilidade do sistema combinado seja significativamente maior que a soma de suas partes.)" (Foster & Kesselman, 1998).

Existem várias infra-estruturas de software para Grade Computacional (p.ex. Globus (Foster & Kesselman, 1997), Legion (Grimshaw & Wulf, 1997) e SNIPE

(Fagg et al., 1997)). Serviços como autenticação, acionamento de programas e mecanismos de transferência de dados estão incluídos nas infra-estruturas.

Globus é indiscutivelmente a mais popular dessas infra-estruturas e foi implementada como o mecanismo de distribuição em vários *frameworks* de realidade virtual (incluindo CAVERNSoft (Park et al., 2000)).

Globus é projetado para oferecer facilidades tais como acesso uniforme a recursos distribuídos com diversos mecanismos de agendamento, serviço de informação para publicação, descoberta e seleção de recursos, e performance melhorada por meio de múltiplos protocolos de comunicação (Foster & Kesselman, 1997).

4.2.1.4. *Common Component Architecture* (CCA)

A CCA – *Common Component Architecture* (Common Component Architecture Forum, 2006) define um *framework* de comunicação baseado em componentes que conceitualmente se situa sobre o *middleware*, tal como CORBA ou Globus, ou IP de nível mais baixo.

Existem inúmeras vantagens em se usar programação por componentes ao se implementar ferramentas de visualização de alto desempenho. Isto é verificado quando se considera que o desenvolvimento de aplicações de realidade virtual é cada vez mais um empreendimento multidisciplinar. Ao prover estruturas de núcleo flexíveis, o esforço de desenvolvimento mais amplo pode incorporar equipes de pesquisa com uma gama significativa de habilidades. A programação por componentes suporta as diferentes abordagens e requisitos inevitáveis em um esforço de desenvolvimento multidisciplinar. Entretanto, possivelmente a faceta mais interessante da programação por componentes com relação ao presente trabalho é que os componentes podem ser configurados para serem executados localmente ou em lugares remotos.

CCA apresenta um meio transparente de definir e gerenciar componentes de software. Dessa forma, ela trata da interoperabilidade no nível do componente, isto é, da conexão flexível a uma arquitetura por um componente compatível por meio de uma interface bem definida. Existem diversos *frameworks* compatíveis com CCA em uso atualmente, incluindo CCAFFEINE (Armstrong et al., 1999) e

XCAT (2006). O modelo de fluxo de dados no SCIRUN (Parker, 1999) também incorpora um *framework* compatível com CCA.

4.2.1.5. InfoGrid

InfoGrid, previamente CSGrid (Lima et. al, 2005), é um sistema cliente/servidor para ambientes de grade computacional que, além do suporte ao uso e gerenciamento de recursos computacionais distribuídos, oferece facilidades para integrar aplicações e gerenciar dados e usuários (Figura 11). O InfoGrid apresenta a seus usuários, por meio de um navegador Web, um espaço de trabalho com todas as aplicações disponíveis e com os arquivos de dados de cada usuário organizados por projeto. Um usuário pode estender o sistema, adicionando novas aplicações. O InfoGrid também oferece a seus usuários algumas facilidades de trabalho colaborativo.

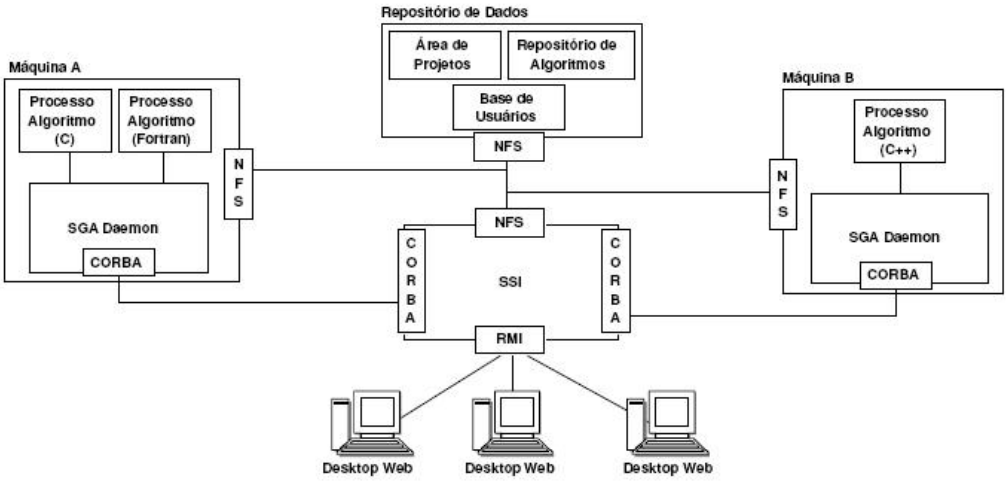

Figura 11 - Arquitetura do InfoGrid

As aplicações que são executadas no cliente utilizam os serviços disponíveis no servidor para ter acesso e gerenciar os recursos computacionais distribuídos. Um desses serviços é o de execução remota de algoritmos em máquinas que estão ligadas ao InfoGrid. Algoritmos, no contexto do InfoGrid, são programas executáveis implementados em qualquer linguagem que admitem parâmetros de entrada, geram uma saída e não possuem nenhum tipo de interação com o usuário durante sua execução. Diversas máquinas podem ser incorporadas ao ambiente de grade computacional para servir como plataforma de execução de algoritmos.

Novos algoritmos são facilmente tornados disponíveis no ambiente e o processo para executá-los passa a ser uma tarefa transparente para o usuário.

Uma característica importante na arquitetura do InfoGrid é a interoperabilidade entre o servidor principal do sistema e os servidores que são executados nas diversas máquinas de execução de algoritmos. Independente da linguagem de implementação dos algoritmos e das máquinas onde estes estão sendo executados, o InfoGrid oferece uma interface padrão para a execução remota e a monitoração das máquinas do ambiente.

O InfoGrid possui um *desktop* Web onde os usuários utilizam os recursos computacionais distribuídos como se estivessem trabalhando localmente. Por meio deste cliente, os usuários podem criar suas próprias áreas de trabalho, monitorar as máquinas da grade computacional, instalar algoritmos, executar aplicações instaladas no *desktop*, comandar a execução remota de algoritmos e monitorar os processos que estão executando esses algoritmos nas máquinas do ambiente.

O InfoGrid foi implementado como uma extensão do CSBase (2006), um *framework* para o gerenciamento de recursos e execução de algoritmos em um ambiente computacional distribuído e heterogêneo. O CSBase é o resultado de diferentes projetos desenvolvidos pela parceria entre o Tecgraf/PUC-Rio e a Petrobras. O primeiro desses projetos, WebSintesi, implementou e tornou disponível um ambiente integrado para análise e síntese de dados geofísicos. No WebSintesi, máquinas multiprocessamento e aglomerados de computadores são ligados a um servidor principal que realiza, de modo transparente para os usuários, a execução remota de programas que necessitam de processamento de alto desempenho e manipulam dados sísmicos de centenas de gigabytes.

Após o primeiro ano de desenvolvimento do WebSintesi, o projeto InfoPAE Dinâmico foi iniciado para atender a usuários da Petrobras que utilizam aplicações de gestão de emergências, tais como programas de análise de estabilidade de unidades *offshore*. Para permitir o re-uso de serviços já implementados para o WebSintesi e para facilitar o desenvolvimento de novos serviços e aplicações particulares para os projetos WebSintesi e InfoPAE Dinâmico é que foi criado o *framework* CSBase.

Em função de todos esses desenvolvimentos, decidimos tomar o InfoGrid como um exemplo desta primeira abordagem (*Middleware*) para delinear um protótipo (descrito no Capítulo 5) que será uma das provas do nosso metamodelo.

4.2.2. Ambientes Virtuais Distribuídos Puros

Nesta subseção, investigamos a segunda abordagem para o desenvolvimento de ambientes virtuais distribuídos, isto é, Ambientes Virtuais Distribuídos Puros, na qual ambientes distribuídos específicos são desenvolvidos para atender aos requisitos. Esta abordagem oferece um nível de abstração mais alto quando comparada com *Middleware*, no sentido de que seus protocolos, arquiteturas e sistemas oferecem aos desenvolvedores APIs que são mais próximas do nível da aplicação do que do nível da implementação.

4.2.2.1. SIMNET, DIS e HLA

Distributed Interactive Simulation (DIS) e seu predecessor SIMNET são padrões para simulações interativas distribuídas (Locke, 1993; Youngman, 2006). DIS retém um banco de dados replicado e usa *dead reckoning* para manter o acompanhamento de outros. *Dead reckoning* é uma estratégia para lidar com problemas de latência de rede que envolve o empacotamento do local de uma entidade, de uma marca de tempo e de um vetor de velocidade. O computador central da aplicação que recebe os dados pode então predizer onde o objeto deve estar. Cada entidade executa uma simulação local junto com o modelo de *dead reckoning*, e quando os dois divergem por mais do que uma certa tolerância uma mensagem é enviada a todos os participantes para reconciliar a discrepância.

DIS traz em conjunto várias idéias interessantes para a realidade virtual distribuída:

- Formatos padrões de mensagem.
- Não uso de servidor central.
- *Dead reckoning*.
- AOIM (*Area of Interest Management* – Gestão de Área de Interesse).
- Uso potencial de *multicasting*.

Mensagens *multicast* podem ser enviadas para um grupo específico de máquinas. *Multicasting* é popular em sistemas de realidade virtual distribuída, particularmente os que suportam um grande número de participantes. Macedonia et al. (1995) afirmam que foram mostradas algumas simulações que reduziram o tráfego de rede de 90% ao empregarem *multicasting*.

Algumas aplicações implementam os protocolos DIS, tais como *VR-Link* (Morrison, 1995), *Close Combat Tactical Trainer* (CCTT) (Mastaglio & Callahan, 1995), PARADISE (Singhal, 1996) e NPS *Networked Vehicle Simulator* (NPSNET) (Macedonia et al., 1995). NPSNET-V (Capps et al., 2000) é desenvolvida pela *Naval Postgraduate School*. Ela é implementada em Java e incorpora o conceito de uma entidade para representar um objeto do mundo virtual, definindo entidades de objetos que contêm informações sobre artefatos no ambiente virtual (p.ex., tanques ou pessoas). As entidades definem informações de estado, tais como local e velocidade. A partir de uma lista de protocolos definidos, a cada entidade pode ser atribuído um conjunto de comportamentos.

O desenvolvimento da *High Level Architecture* (HLA) (Defense, 2006; Dahmann et al., 1999) tem por objetivo facilitar a interoperabilidade e a composição da maior gama possível de simulações baseadas em componentes. A HLA não se originou como um padrão aberto, mas foi mais tarde reconhecida e adotada pelo *Object Management Group* (OMG, 2006) e pelo *Institute of Electrical and Electronics Engineers* (IEEE, 2006). O desenvolvimento da HLA ocorreu de forma semelhante ao padrão DIS – isto é, o projeto inicial foi produzido dentro do Departamento de Defesa dos EUA e então entregue a um organismo externo (IEEE e/ou OMG) para padronização (Singhal & Zyda, 1999).

A arquitetura HLA pode ser entendida como um projeto de ambiente virtual de rede orientado a objetos. Cada simulador, conhecido como um *federado*, é um componente que representa uma coleção de objetos, cada um destes tendo um conjunto de atributos e sendo capaz de iniciar e receber eventos. O federado registra cada um de seus objetos com um pedaço de *middleware* chamado *Run-Time Infrastructure* (RTI). A RTI colabora com instâncias RTI em outros computadores para aprender sobre participantes (objetos) e fornecer informações sobre esses participantes para o federado local. O federado local, por sua vez, tipicamente instancia objetos locais representando esses participantes remotos. Atualizações de atributos e eventos são também trocados através da RTI, que é

responsável por manipular a gestão de área de interesse, a sincronização do tempo e outros serviços de ambiente virtual de rede de nível baixo, segundo o interesse da aplicação. A coleção de federados, juntamente com as suas instâncias associadas de RTI, é chamada de *federação*. Os simuladores na federação enviam e recebem informações de estado por meio de chamadas para a e da RTI (Singhal & Zyda, 1999). A HLA como um padrão IEEE será melhor discutida mais adiante neste capítulo.

4.2.2.2. DIVE

DIVE (*Distributed Interactive Virtual Environment*) (Swedish, 2006) é desenvolvido pelo Instituto Sueco de Ciência da Computação. O esforço de desenvolvimento se concentrou principalmente no componente de distribuição do ambiente virtual. Certamente, em seus primeiros dias DIVE emergiu como um exemplo importante de como desenvolver realidade virtual distribuída.

O elemento de distribuição é conceitualmente muito simples. A memória compartilhada é tornada disponível na rede e o sistema controla o envio de sinais para os processos. DIVE implementa um banco de dados distribuído, completamente replicado e dinâmico. Ele tem a capacidade de adicionar novos objetos e modificar bancos de dados existentes de um modo confiável e consistente. Protocolos *multicast* confiáveis e controle de concorrência são empregados por meio de um mecanismo de bloqueio distribuído para facilitar a atualização desses bancos de dados.

4.2.2.3. MASSIVE

MASSIVE (Greenhalgh et al., 2000) é um sistema orientado a objetos desenvolvido na Universidade de Nottingham para apoiar ambientes virtuais colaborativos multi-usuário, baseado no “modelo espacial” de interação (Benford et al., 1993). Ele é desenvolvido principalmente como uma ferramenta de pesquisa acadêmica, para pesquisadores que investigam a colaboração on-line em ambientes virtuais. O foco do trabalho é promover a percepção (*awareness*) de outros no ambiente virtual.

O sistema suporta interação com os ambientes virtuais via texto, interfaces gráficas 2D e 3D, e possui suporte para áudio em tempo real, permitindo que os usuários se comuniquem entre si usando a fala. A ênfase é mais na interação usuário-a-usuário do que na interação de um usuário com objetos no ambiente. MASSIVE usa uma combinação de processos *peer-to-peer* e cliente/servidor, e emprega *multicasting* como a linha básica de colaboração.

4.2.2.4. Avango

Avango é desenvolvido pelo Centro de Pesquisas Nacional da Alemanha para a Tecnologia de Informação (GMD) (Bierbaum & Just, 1998; Tramberend, 1999). Ele é ostensivamente um *framework* orientado a objetos desenvolvido sobre o Performer para a construção de ambientes virtuais em rede.

Os principais subsistemas (interação, gráfico, operação em rede) em uma aplicação de realidade virtual são integrados em um. Ele provê um grafo de cena replicado através da rede. Os objetos podem ser instanciados ou como locais ou como distribuídos (públicos ou privados). São definidas duas categorias de objetos:

1. Nós – definem elementos de grafo de cena para *rendering*.
2. Sensores – importam dados de dispositivos externos para a aplicação.

Avango é desenvolvido sobre um modelo de fluxo de dados usando campos dentro dos objetos para suportar uma interface genérica de fluxo. O grafo de fluxo de dados define o comportamento dos objetos no mundo.

Ele possui uma API C++ e um *binding* para uma linguagem interpretada, Scheme (Dybvig, 1996). Tipicamente, funcionalidades complexas e críticas em termos de performance são implementadas em C++. A partir deste ponto, a aplicação é implementada usando *scripts* da Scheme.

Avango usa um modelo de grupos de processos. Assegura-se que cada membro do grupo recebe mensagens enviadas através da rede exatamente na mesma ordem. Além disso, quando um novo membro se junta ao grupo, toda a

comunicação é suspensa até que o estado do novo membro seja atualizado, o que garante consistência de estado.

4.2.2.5. DEVA/MAVERIK

DEVA (Pettifer et al., 2000) é um núcleo de realidade virtual distribuída desenvolvido pelo *Advanced Interfaces Group*, da Universidade de Manchester.

DEVA suporta bancos de dados que incluem objetos virtuais, e separa a representação de um objeto virtual em objeto do lado do cliente e objeto do lado do servidor. A parte do lado do servidor representa o objeto virtual e define o seu comportamento, uma parcela do qual pode ser genérica e outra pode ser específica de uma aplicação. A parte do lado do cliente contém interpretações do seu comportamento e só responde a instruções vindas do servidor. As mensagens do servidor para o cliente são semelhantes a um vocabulário de alto nível usado para descrever o efeito do comportamento. DEVA estabelece um local de controle que gerencia estados em um lugar. Isto significa que o estado de uma entidade pode ser gerenciado localmente em vez de no servidor, se isto for apropriado.

Em vez de usar *dead reckoning* para combater o atraso e a tremulação, DEVA emprega "*twines*" (Marsh et al., 1999), que suaviza atualizações irregulares.

DEVA está fortemente acoplado a um núcleo de realidade virtual (MAVERIK (Hubbold et al., 2001)), que provê capacidades de *rendering*, entrada, saída e gerenciamento espacial. A principal vantagem é que dados da aplicação não precisam ser duplicados e armazenados em uma estrutura de dados específica como o grafo de cena. E, em lugar disso, MAVERIK promove o uso das estruturas de dados próprias da aplicação, o que evita a necessidade de se ter que estar de acordo com um estrutura de dados rígida e potencialmente inapropriada.

4.2.2.6. Outros DVEs

O DIVERSE *Toolkit* (DTK), da Virginia Tech (Arsenault et al., 2001), é projetado para auxiliar a implementação de aplicações de realidade virtual distribuída e está disponível tanto sob a *Gnu Public License* (GPL) quanto sob a

LGPL. Este conjunto de ferramentas é implementado como uma API C++ para um servidor e clientes, e oferece uma biblioteca extensiva mas de relativo baixo nível para distribuição, em vez de um *framework* de realidade virtual.

COVISE (*Collaborative Visualisation and Simulation Environment*) (Rantzau et al., 1998) explora infra-estruturas de rede de alta velocidade em computação distribuída e engenharia colaborativa. Ele se concentra predominantemente em visualizar problemas de supercomputação *high end*. COVER (2006), baseado no IRIS Performer, provê suporte à visualização. COVISE media uma sessão distribuída por meio de um sistema de tomada de vez. O banco de dados é completamente replicado através dos nós ao se fazer a conexão. Toda a entrada do usuário, saída do sistema e administração do sistema é manipulada em um único ponto, reduzindo a complexidade de se manter todos os usuários sincronizados, mas com o custo de atrasos de ida-e-volta e o risco potencial de o ponto central de controle tornar-se um gargalo à medida que mais usuários são adicionados.

CAVERNsoft (Park et al., 2000) combina uma biblioteca de operação de rede e um banco de dados, facilitando o desenvolvimento de ambientes virtuais interativos colaborativos. CAVERNsoft é baseado em um modelo cliente-servidor. As aplicações usam o *Information Request Broker* (IRB) para mediar a comunicação entre as instâncias de rede. Os usuários podem solicitar o tipo de conexão de rede que desejam usar. CAVERNsoft fornece a opção de TCP/IP, UDP ou IP/*Multicast*. O usuário define a largura de banda desejada e atributos de rede aceitáveis (latência, tremulação), e o IRB remoto tenta satisfazê-los. Desse modo, o usuário especifica a qualidade desejada de serviço logo ao entrar.

Existem também *frameworks* de realidade virtual mais genéricos que oferecem suporte para implementar DVEs. Por exemplo, VR Juggler (Bierbaum et al., 2001) traz suporte a operações de rede fornecido por meio de um gerente abstrato de rede. Outro exemplo é Bamboo (Watsen & Zyda, 1998), desenvolvido sobre o ADAPTIVE *Communication Environment* (ACE) (Schmidt, 1994), que provê o sistema com operações de rede, concorrência (*threading*) e funções de sincronização.

4.2.2.7. HLA

Simulação distribuída é uma aplicação da tecnologia de sistemas distribuídos que habilita modelos a serem ligados através de redes como a Internet, de modo a trabalharem de forma conjunta (ou interoperarem) durante a execução de uma simulação. A HLA (*High Level Architecture*) é um padrão que define a tecnologia de sistema distribuído a tornar possível esta interoperabilidade. Mais do que um protocolo de rede (padrão para fio), como o DIS, a HLA define uma arquitetura com um conjunto de padrões de API. As aplicações de simulação (conhecidas como federados na HLA) se comunicam fazendo chamadas às APIs HLA. Um pedaço de software conhecido como RTI (*Run-time Infrastructure*) implementa a API HLA e é responsável por transportar dados de um federado para o outro. Assim como no DIS, padrões HLA são propriedade do IEEE. Existem três documentos que compreendem o padrão HLA, todos disponíveis a partir do IEEE (2006):

- IEEE 1516-2000 – IEEE *Standard for Modeling and Simulation* (M&S) *High Level Architecture* (HLA) – *Framework and Rules*: provê as regras e definições para implementar e usar HLA. Seu código de produto IEEE é SH94882;
- IEEE 1516.1-2000 – IEEE *Standard for Modeling and Simulation* (M&S) *High Level Architecture* (HLA) – *Federate Interface Specification*: define os vários serviços fornecidos por uma RTI de HLA (veja Figura 12) e contém as APIs. Seu código de produto IEEE é SH94883;
- IEEE 1516.2-2000 – IEEE *Standard for Modeling and Simulation* (M&S) *High Level Architecture* (HLA) – *Object Model Template* (OMT) *Specification*: define o formato usado para descrever modelos de objetos em HLA. Um modelo de objeto dita que tipos de dados um conjunto particular de federados HLA trocará. Seu código de produto IEEE é SH94884.

Existe um quarto documento que não é tecnicamente parte da definição da HLA, mas que define algumas das práticas recomendadas para usar HLA. Ele é

chamado de IEEE 1516.3-2003 – IEEE *Recommended Practice for High Level Architecture* (HLA) *Federation Development and Execution Process* (FEDEP), e seu código de produto IEEE é SH95088.

Enquanto a série IEEE 1516 de padrões representa a versão "corrente" de HLA, várias simulações HLA ainda estão usando uma versão mais antiga, conhecida como HLA 1.3. Esta versão era mantida pelo DMSO (*Defense Modeling and Simulation Office*) e pelo Departamento de Defesa dos EUA antes da padronização do IEEE.

A HLA foi inicialmente desenvolvida pelo Departamento de Defesa dos EUA para a simulação de cenários de campos de batalha. O padrão emergente HLA do IEEE tem as seguintes características:

- Suporta simulações compostas de diferentes componentes de simulação.
- Suporta ambiente de tempo real e problemas de simulação grandes e complexos.
- Suporta re-usabilidade: modelos de simulação por componentes podem ser re-usados em diferentes cenários e aplicações de simulação.
- Suporta interoperabilidade: simulações por componentes re-usáveis podem ser combinadas com outros componentes sem a necessidade de recodificação.

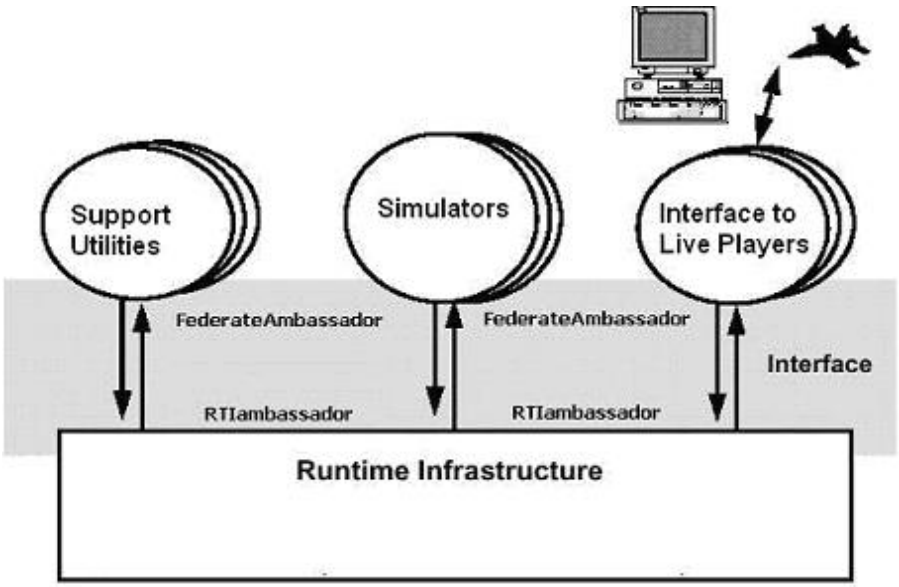

Figura 12 - HLA e RTI (Dahmann et al., 1999)

Podemos então concluir que a HLA preenche nosso requisito de suporte à colaboração em tempo real, assim como parece ser uma arquitetura baseada em componentes flexível, de acordo com todos os princípios que estivemos buscando até agora. A HLA também parece satisfazer vários aspectos de um espaço de trabalho colaborativo para a gestão de desastres na área de óleo e gás, exceto pelo fato de que ela é desenvolvida principalmente para ter vários nós com atualização precisa em tempo real, enquanto que na área de óleo e gás existem menos nós mas com maiores demandas em termos de visualização.

O uso do padrão HLA não é restrito a ambientes virtuais distribuídos, o que significa que, à medida que a demanda pela integração de processos, sistemas e bancos de dados dentro das companhias continua a crescer, esta é uma das abordagens que devem ser consideradas ao se definir soluções de integração.

Os conceitos fundamentais na HLA são:

- Federado: quando uma simulação é implementada como parte de uma simulação compatível com HLA, ela é denominada *federado*.
- Federação: é uma coleção de federados trabalhando juntos para resolver um problema específico.

Note que as federações podem incluir mais do que simulações. Elas também podem incluir interfaces para operadores humanos/jogadores, para hardware real e para software geral executando funções tais como coleta de dados, análise de dados e exibição de dados.

Os três principais componentes na HLA são:

- Regras HLA:
  - Asseguram interação apropriada de federados em uma federação.
  - Descrevem as responsabilidades dos federados e federações.
- Especificação da Interface:
  - Define os serviços e interfaces da RTI (*Run-time Infrastructure*).
  - Identifica funções "*callback*" que cada federado precisa prover.
- *Object Model Template* (OMT):
  - Prescreve o formato e a sintaxe para a gravação de informações.
  - Estabelece o formato de modelos chave:
    - *Federation Object Model* (FOM).

- *Simulation Object Model* (SOM).
- *Management Object Model* (MOM).

A Figura 13 mostra uma visão lógica de alto nível de uma Federação HLA em execução. Todos os componentes mostrados na figura são parte de uma única federação com exceção do RtiExec. Uma breve descrição de cada um desses componentes é fornecida abaixo:

- Federado: programa de simulação por componentes compatível com HLA, mais um SOM.
- Federação: simulação composta de um conjunto de federados interagindo via os serviços da RTI, mais um FOM.
- FedExec: gerencia a federação. Ele permite que os federados se associem e se desliguem da federação, e facilita a troca de dados entre federados participantes.
- Arquivo FDD: arquivo *FOM Document Data*, contém informações derivadas do FOM e usadas pela RTI em tempo de execução.
- RtiExec: processo global que gerencia a criação e a destruição de FedExec's.
- Arquivo RID: *RTI Initialization Data*. Informação específica do fabricante da RTI necessária para se executar uma RTI.

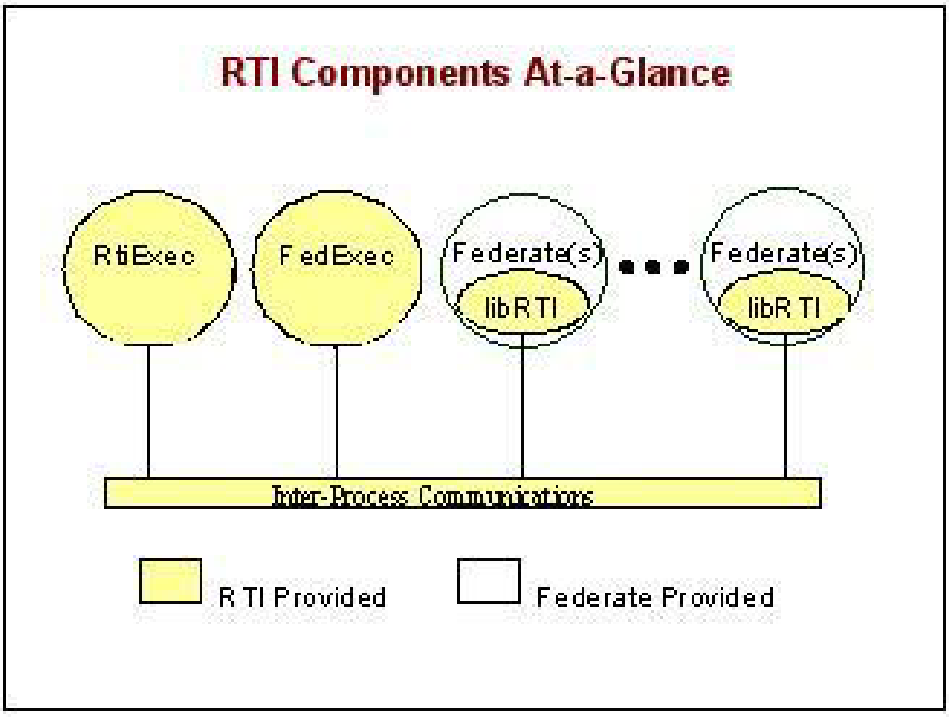

Figura 13 - Visão Lógica de uma Federação HLA (McLeod, 2006)

Consideremos agora um sistema usando HLA com múltiplas federações em execução. O sistema está executando duas federações: Federação 1 e Federação 2. Embora um sistema possa executar duas federações, as Regras da HLA especificam que elas são independentes uma da outra e não podem trocar nenhuma informação. Este sistema está ilustrado na Figura 14 e possui os seguintes componentes:

- Existe um único RtiExec e um único arquivo RTI.RID, compartilhado por ambas as federações.
- Cada federação contém seu próprio FedExec e arquivo FDD. FedExec1 controla a execução da Federação 1 e FedExec2 controla a execução da Federação 2.
- A Federação 1 contém dois federados: o Federado Branco (*White Federate*) e o Federado Verde (*Green Federate*).
- A Federação 2 contém três federados: os Federados Púrpura (*Purple*), Laranja (*Orange*) e Azul (*Blue*).

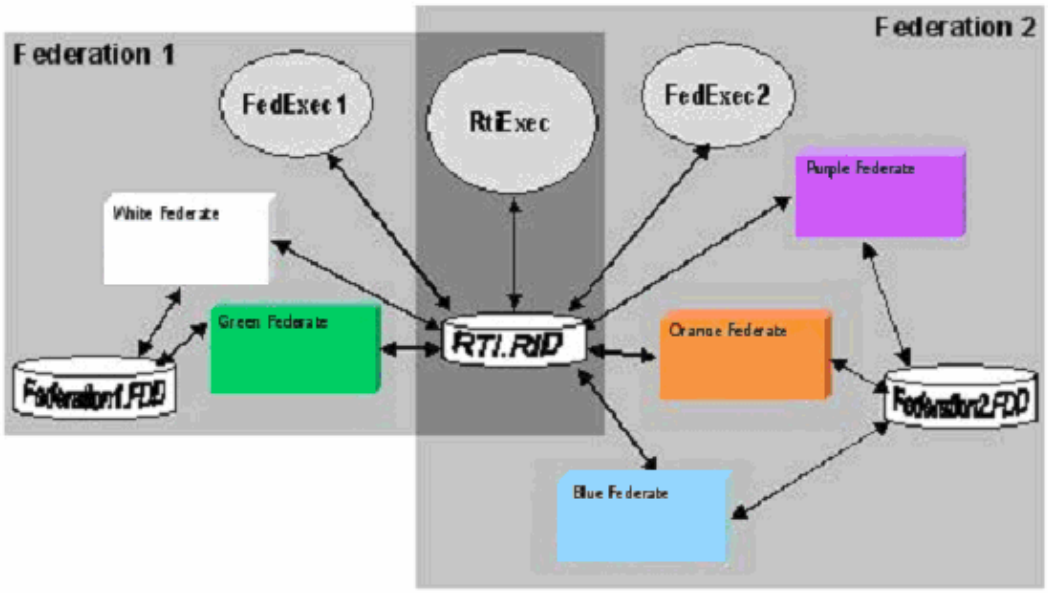

Figura 14 - O Esquema Geral – Federações em Execução (McLeod, 2006)

Na Figura 15, mostramos os passos no processo de iniciar uma execução de federação:

1. Quando uma federação é executada, o RtiExec é disparado primeiro.

2. Então um federado, agindo como um gerente, cria uma execução de federação invocando o método RTI *createFederationExecution*.
3. Então um nome é reservado com o RtiExec, um processo FedExec é gerado, e este FedExec registra seu endereço de comunicação com o RtiExec. A execução da federação está em andamento.
4. Uma vez que uma execução de federação existe, outros federados podem se ligar a ela. O RtiExec é consultado para obter o endereço do FedExec, e *joinFederationExecution()* é invocado no FedExec. Federados adicionais podem se associar por meio do mesmo processo.

Em face de todas essas características, escolhemos HLA como um exemplo dessa segunda abordagem (DVEs Puros) para delinear um protótipo (Capítulo 5) que será uma das provas do nosso metamodelo, do mesmo modo que o InfoGrid foi escolhido como uma plataforma da primeira abordagem (*Middleware*).

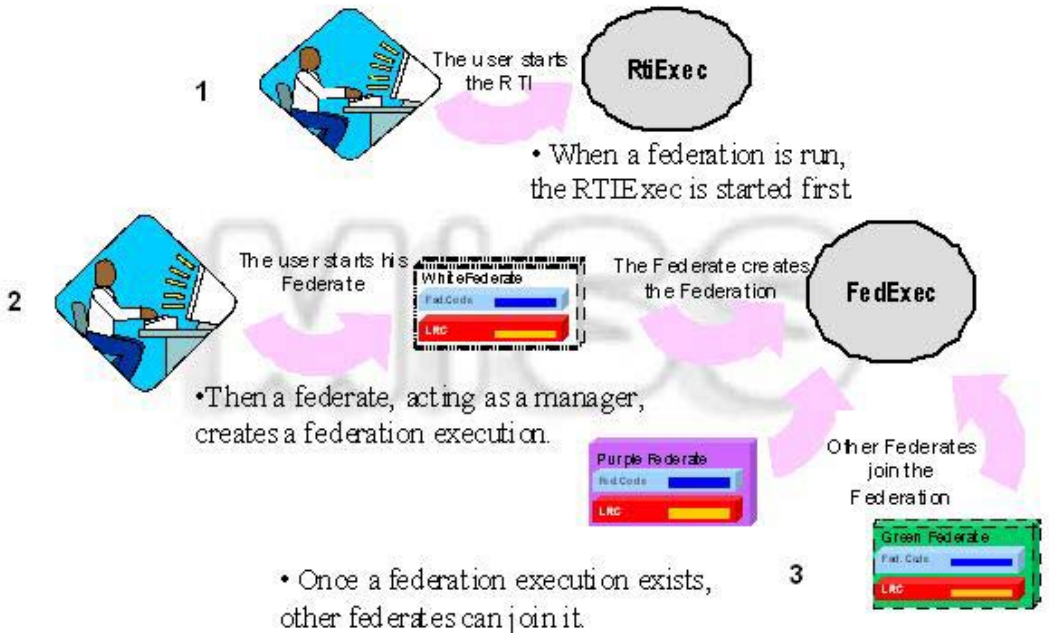

Figura 15 - Passos no Processo de Execução de uma Federação (McLeod, 2006)

Ainda tínhamos que escolher a RTI (*Run-time Infrastructure*) da HLA a ser usada com o nosso protótipo. Nossos requisitos eram:

- Ter código fonte aberto.

- Ser distribuível gratuitamente.

Com base nesses requisitos e no fato de que ela estava bem documentada em uma dissertação de mestrado, escolhemos a XRTI – *The Extensible Run-Time Infrastructure* – de Kapolka (2003). Suas características básicas são:

- Tem plena compatibilidade com o padrão HLA.
- Especifica um protocolo de mensagens padronizável.
- Suporta extensão e composição de modelos de objetos dinâmicos.
- Está escrita em Java e usa modelos de objetos XML.
- Usa uma topologia cliente-servidor pura na qual os federados só se comunicam com outros através do XRTI Executive, uma aplicação de servidor.
- Os federados mantêm dois canais com o Executive: um canal TCP (*Transmission Control Protocol*) para comunicação confiável e um canal UDP (*User Datagram Protocol*) para mensagens não confiáveis.