Extensão das APIs do MPEG-J para o Suporte Multiusuário no MPEG-4 através da Ferramenta SVRT

Todesco, G. [1 2] Rückert, U. [1] Araujo, R. B. [1]
{glauco, uli, regina}@dc.ufscar.br

[1] Universidade Federal de São Carlos
Departamento de Computação
São Carlos - SP - Brasil

[2] Faculdade Prudente de Moraes
Departamento de Informática
Itú - SP - Brasil

Resumo

O MPEG-4 pode ser definido, genericamente, como um padrão para comunicações multimídia. Porém ele ainda esta em fase de desenvolvimento e várias questões estão em aberto, entre elas o suporte multiusuário, que poderá ser feito através do MPEG-J. O MPEG-J é um conjunto de APIs que permitem o acesso externo a um *player* MPEG-4 através de uma aplicação Java.

O SVRT - *Shared Virtual Reality Tool*, é uma ferramenta implementada em JAVA e originalmente desenvolvida para aplicações VRML que permite o compartilhamento de ambientes virtuais por múltiplos usuários e que suporta aplicações genéricas de Realidade Virtual como, por exemplo: educação a distância, jogos distribuídos, feiras, treinamento militares, etc.

Este artigo tem como objetivo propor a extensão das APIs do MPEG-J para permitir o suporte multusuário dentro do MPEG-4 através da ferramenta SVRT.

Palavras Chaves: MPEG-4, MPEG-J, Multimídia e Realidade Virtual.

1. Introdução

A multimídia pode ser caracterizada pela combinação de diferentes tipos de mídias em uma mesma apresentação, em que pelo menos uma das mídias seja contínua, isto é, dependente do tempo. Já os ambientes virtuais distribuídos são caracterizados pelo compartilhamento simultâneo de um mesmo ambiente tridimensional, sintetizado pelo computador, por múltiplos usuários. Estes usuários interagem com o ambiente e com os outros usuários participantes em tempo real. Com o surgimento e a integração das linguagens VRML [*VRML*] e Java, a disponibilização desses ambientes virtuais distribuídos na WWW é uma realidade, porém com várias limitações (suporte a multimídia, tempo de resposta, qualidade, quantidade de usuários suportados, etc.) devido principalmente à complexidade de projeto da linguagem VRML.

O MPEG-4 é um padrão ISO/IEC (ISO/IEC JTC1/SC29/WG11) desenvolvido pelo MPEG (*Motion Picture Experts Group*), responsável também pelo desenvolvimento dos padrões MPEG-1 e MPEG-2, e vem se tornando uma alternativa interessante para o desenvolvimento de aplicações multimídia na WWW, permitindo também a criação de ambientes virtuais integrados à multimídia.

A integração da multimídia à ambientes virtuais é uma aliança poderosa que visa aumentar o realismo e a sensação de imersão do usuário em aplicações que suportem ambientes virtuais do tipo telecolaboração, aplicações educacionais ou de treinamento. A demanda por estes dois tipos de aplicações vem se tornando, cada vez mais, uma necessidade crescente na WWW.

Porém o suporte multiusuário esta em fase de padronização e que poderá ser feito via MPEG-J. O MPEG-J é um conjunto de APIs desenvolvidas em Java que permitem que aplicações e *applets* Java, possam controlar os componentes de um *player* MPEG-4. Isto será feito através da ferramenta SVRT-

Shared Virtual Reality Tool, que originalmente foi desenvolvida para aplicações VRML e foi escrita em JAVA. Ela permite, entre outras coisas, o compartilhamento de ambientes virtuais.

O foco deste artigo é propor a extensão do MPEG-J [*M5660*], para padronizar a interação multiusário dentro do MPEG-4 através da ferramenta SVRT.

2. MPEG-4

O grupo MPEG vem trabalhando no padrão MPEG-4 desde 1993. O MPEG-4 [*N2201*] deverá prover serviços de televisão digital, aplicações gráficas interativas e multimídia interativa (em especial na WWW), satisfazendo as necessidades de autores, provedores de serviços e usuários finais.

As cenas audiovisuais são compostas por objetos de mídia, organizados de forma hierárquica, em uma estrutura de árvore. Os objetos de mídias (imagens, vídeo, áudio, etc.) são representados nas folhas da árvore. A árvore não é necessariamente estática - nós podem ser adicionados, substituídos ou removidos. As propriedades dos nós podem ser alteradas, como por exemplo, os parâmetros de posicionamento.

O MPEG-4 implementa a descrição da cena no formato *BIFS* (*BInary Format for Scenes*). As funcionalidades que podem ser realizadas dentro de uma cena incluem:

- Posicionamento dos objetos de mídia em qualquer espaço dentro do sistema de coordenadas;
- Aplicação de transformações para modificação de atributos dos objetos de mídia;
- Agrupamento dos objetos de mídia, que possibilita a formação de objetos compostos;
- Aplicação de fluxo de dados nos objetos de mídia;
- Alteração do ponto de vista do usuário;
- Disparo de eventos quando da seleção de um objeto especifico, por exemplo, a iniciação ou interrupção de um vídeo. Tipos complexos de comportamento podem ser disparados também, como, por exemplo, uma cena onde um telefone virtual toca, o usuário atende e um *link* de comunicação é estabelecido.

A descrição de uma cena em MPEG-4 estende os conceitos da linguagem VRML para permitir as funcionalidades acima, e desta forma, também permite uma maior facilidade para que implementadores VRML possam migrar para o MPEG-4, devido a grande semelhança existente em ambos os casos para descrever uma cena.

Objetos de mídia e *BIFs* são todos convertidos em segmentos de dados (*streams*) que são transportados em um ou mais segmentos elementares (ES - *Elementary Stream*). Todos os segmentos associados a um objeto de mídia são identificados por um descritor de objeto, o que permite manipular os dados de forma hierárquica, bem como associar meta informações sobre o conteúdo dos dados e direitos autorais.

Cada segmento de dados é caracterizado por um grupo de descritores que trazem informações de configuração, como por exemplo: determinação dos decodificadores necessários e o tempo de decodificação. Os descritores podem trazer também, indicações de *QoS* necessárias para a transmissão da informação multimídia, como prioridade, taxa máxima de velocidade, etc.

A sincronização dos segmentos elementares – SE, é realizada através de *time stamping* de unidades de acesso individuais dentro dos próprios SE. A identificação de tais unidades de acesso e de *time stampings* são executados na camada de sincronização. Independente do tipo de mídia, esta camada permite a identificação das unidades de acesso (como por exemplo, quadros de vídeo ou áudio, comandos de descrição da cena) em segmentos elementares, recriando os tempos base dos objetos de mídia ou a descrição da cena, permitindo assim a sincronização entre eles.

Através do uso de um *player*, um usuário pode navegar e interagir no ambiente virtual da mesma maneira que acontece com um *browser* VRML para a visualização de ambientes virtuais, sendo que as duas interfaces, ou seja, a maneira na qual o usuário vai interagir com o *player* ou *browser* são bastante parecidas. A *Figura 1* mostra um exemplo de um *player* MPEG-4 que além de compor e sintetizar o ambiente virtual, como acontece em um *browser* VRML, também é responsável pela descompressão, sincronização e recepção dos dados.

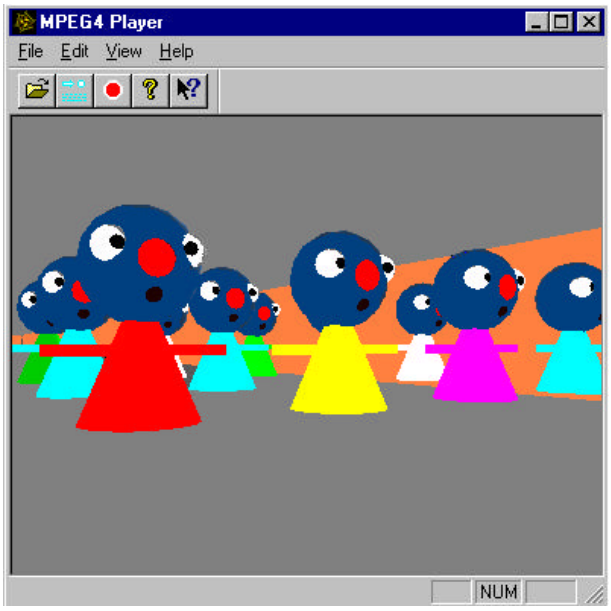

*Figura 1 – Player MPEG-4 (*ISO/IEC JTC1/SC29/WG11)*

Como o MPEG-4 é um padrão em desenvolvimento, existem vários pontos em aberto que são essenciais para a criação de ambientes virtuais distribuídos, tal como o suporte multiusuário.

3. SVRT (*Shared Virtual Reality Tool*) - Uma Ferramenta de Suporte às Aplicações de Sistemas Distribuídos de Realidade Virtual

O SVRT - *Shared Virtual Reality Tool* (*Figura 2*), é uma ferramenta implementada em JAVA e originalmente desenvolvida para aplicações VRML [*Oliv98*] que permite o compartilhamento de ambientes virtuais por múltiplos usuários e que suporta aplicações genéricas de Realidade Virtual, como exemplo: educação a distância, jogos distribuídos, feiras em ambientes virtuais, treinamento de forças policiais ou militares, cidades virtuais para fins educacionais ou de entretenimento, e outros.

Em uma versão preliminar, ela foi estendida para suportar aplicações multiusuário MPEG-4 através de comandos BIFS Remoto [*Todes00*]. O SVRT implementa uma estrutura completa de comunicação para a troca de mensagens entre os diversos usuários do AV – Ambiente Virtual, através de comunicação multiponto, por meio de um protocolo definido pela própria aplicação.

3.1 Arquitetura do Sistema

A arquitetura global proposta para o SVRT (*Figura 3*) inclui um servidor WWW responsável por armazenar os mundos virtuais MPEG-4. Este servidor pode ser replicado para evitar a formação de gargalos. O *player* MPEG-4 é então utilizado pelos usuários do sistema para interagir com o ambiente virtual. O SVRT proporciona a interface com o *player*, que será implementada através do MPEG-J. As interações com os outros usuários do sistema são realizada via um módulo de compartilhamento.

O módulo de compartilhamento é responsável por refletir as ações do participante local, a partir da interação do usuário com o *player*, para os outros participantes do sistema através de comunicação multiponto.

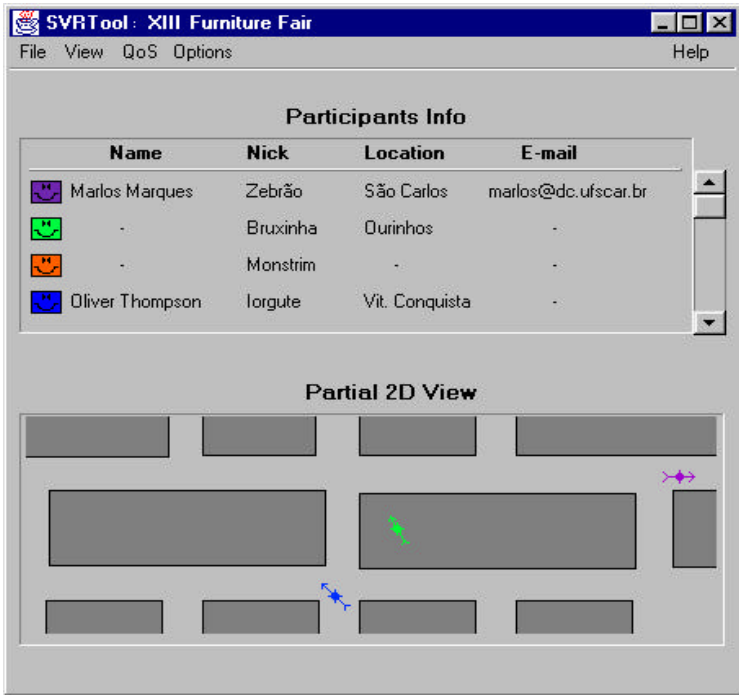

Figura 2 – Tela principal do SVRT

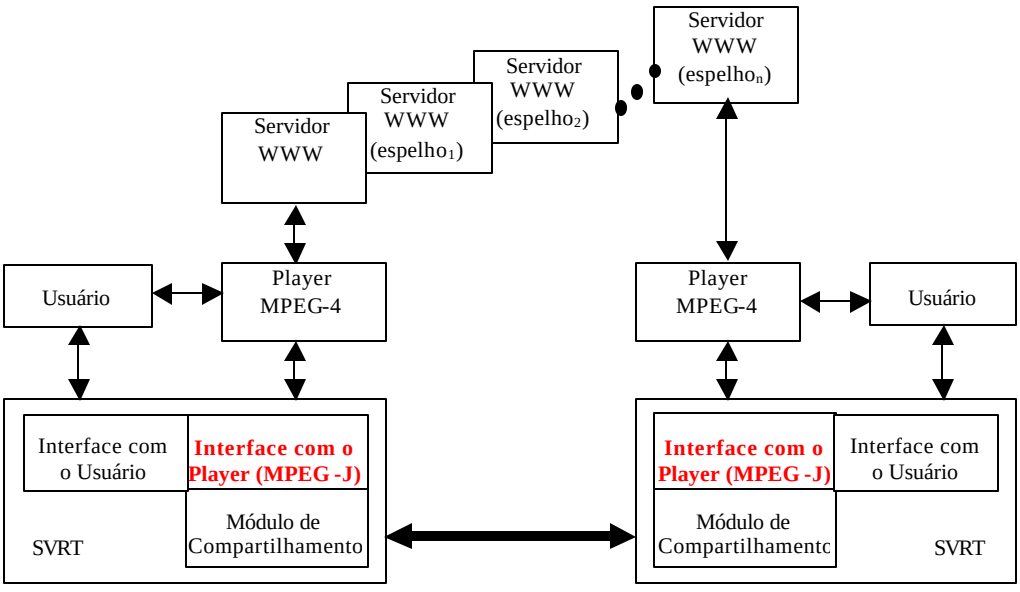

Figura 3 - Arquitetura global do sistema (a seta mais escura denota interação multiponto e a seta mais clara denota interação ponto-a-ponto)

4. MPEG-J

O MPEG-J [*M4646*] foi introduzido na versão 2 do MPEG-4. Ele contém um conjunto de APIs (*Application Programming Interface*) que possibilitam que aplicações e *applets* Java possam controlar um *player* MPEG-4, permitindo respostas apropriadas para os eventos na rede, servidor ou entradas do usuário. As aplicações Java são distribuídas da mesma maneira que os demais fluxos (vídeo, áudio, descrição da cena – BIFs, etc.) através de um Fluxo Elementar - ES, ou seja, uma aplicação Java possui um descritor de objeto, e ela é multiplexada e distribuída dentro de um arquivo no formato MPEG-4.

A arquitetura do MPEG-J, incorporado em um sistema MPEG-4 é mostrada na *Figura 4*. As aplicações Java podem acessar todos os componentes do *player* MPEG-4 através das APIs. Tais componentes incluem os recursos de apresentação e execução, decodificadores, recursos de rede e a cena gráfica.

4.1 Iniciando uma aplicação MPEG-J

Quando uma aplicação MPEG-J é recebida juntamente com as demais *streams*, ela não é iniciada enquanto o *player* MPEG-4 não receber um descritor de objeto MPEG-J. Após o recebimento, o *player* segue os seguintes passos: Abre um novo canal para o fluxo elementar da aplicação MPEG-J. O fluxo é tratado da mesma maneira que os demais fluxos (vídeo, áudio, etc.). O fluxo elementar é dividido em Unidades de Acesso. As Unidades de Acesso MPEG-J são enviadas para o *Class Loader*, no qual realiza a carga de todas as classes da aplicação. Um "decodificador" MPEG-J é responsável em receber as Unidades de Acesso e tomar as decisões para executar a aplicação. Podem existir um ou mais pontos de entrada dentro de um fluxo MPEG-J. A cada ponto de entrada, é disparado uma nova *thread* para a sua execução.

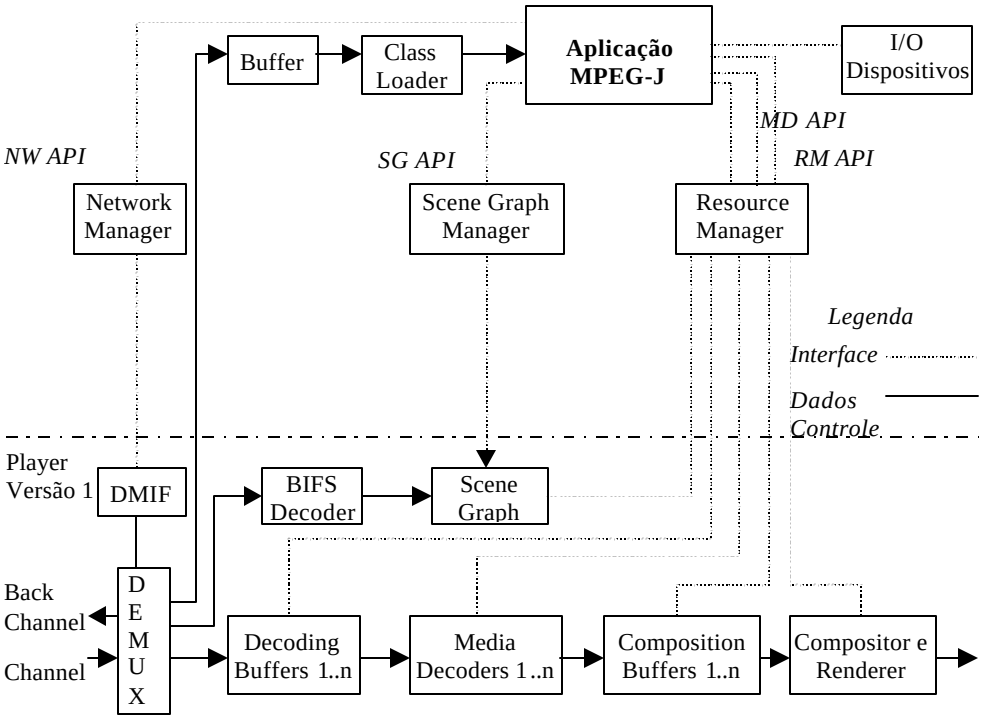

*Figura 4 – Interação do Componentes MPEG-4 com o MPEG-J - *ISO/IEC JTC1/SC29/WG11*

Existem várias questões que envolvem a distribuição das aplicações MPEG-J que estão fora do escopo deste artigo como segurança, perda de pacotes, dependência de classes, compressão, etc.

4.2 MPEG-J APIs

Os pacotes que compõem atualmente as APIs do MPEG-J são mostrados abaixo:

- *org.iso.mpeg.mpegj.scene*: Acesso à cena gráfica, permitindo a criação e modificação dos nós da cena;
- *org.iso.mpeg.mpegj.resourceManager*: Gerenciamento dos recursos do sistema e acesso estatístico e as características dinâmicas do *player*;
- *org.iso.mpeg.mpegj.network*: Acesso e controle dos decodificadores usados;
- *org.iso.mpeg.mpegj.decoder*: Acesso e controle dos componentes de rede de um *player* MPEG-4.

Para que uma aplicação MPEG-J possa acessar os gerenciadores implementados em cada terminal, ela deve fazer uso da classe *MPEG-J Terminal*, que possui os métodos específicos para o acesso a cada gerenciador (*getNetworkManager()*, *getResourceManager()* e *getrSceneManager ())*.

4.3 Extensões para o suporte Multiusuário

O maior parte dos requisitos [N3205] necessários para a interação multiusuário é conseguido através da extensão do MPEG-J. A *Tabela 1* mostra a proposta de extensões, onde três novas APIs são propostas e uma estendida.

API	Tipo	Descrição
Scene Graph Manager	Version 2 + Sugestão	Acesso à cena gráfica. Permitindo a criação e modificação de nós e DEF IDs.
Message Manager	Sugestão	Transmissão das Mensagens entre terminais/servers.
User Interface Manager	Sugestão	Acesso aos dispositivos de entrada do usuário.
Shared State Manager	Sugestão	Gerenciamento dos estados compartilhados entre terminais/servidores

Tabela 1 – APIs para o suporte multiusuário

4.3.1 *Scene Graph Manager*

Scene Graph Manager (*Figura 5*) é a parte mais importante do MPEG-J para a implementação de sistemas multisuário. Através do acesso à cena gráfica, uma aplicação pode modificar e controlar o modo em que a cena é apresentada. Na atual versão do MPEG-J, somente os nós que foram referenciados por um identificador DEF poderão ser acessados. A única maneira de obter acesso à cena, é implementado a interface *SceneListener*, que permite monitorar as alterações ocorridas na cena através do método *notify()*. A interface *Scene* permite o acesso aos nós da cena que foram identificados pelo DEF. A interface *Node* representa um nó na cena gráfica. Esta interface contém os métodos para acessar e modificar os atributos de cada nó.

Os eventos que podem ocorrer na cena são identificados por duas interfaces: *EventIn* e *EventOut*. A aplicação pode ser notificada da ocorrência de qualquer evento através do uso da interface *EventOutLinstener*.

A API *Scene Graph* possui métodos que podem retornar um valor de um campo. Dois tipos de campos são suportados. *SFField*, para campos com um único valor e *MFField*, para campos com múltiplos valores.

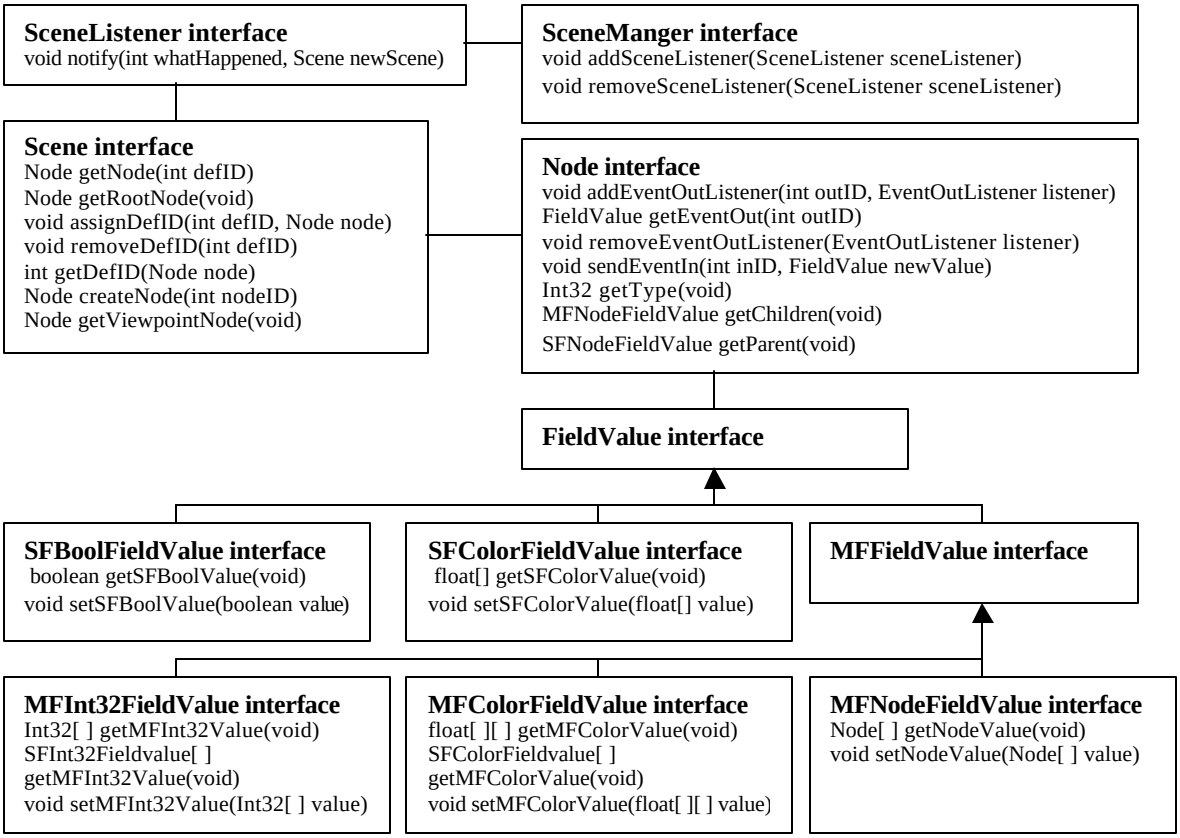

*Figura 5 – Scene Graph Manager – *Diagrama de Classe (UML)*

As seguintes alterações foram realizadas:

- Para facilitar a navegação na árvore da cena gráfica, o método *getRoot()* foi acrescentado, que possibilita o acesso à raiz na arvore. Os métodos *getChildren()* e *getParent()* foram incluídos para facilitar a navegação. Atualmente, o acesso aos nós da cena só é possível através do uso de um identificador DEF;
- O método *getViewpointNode()* facilita o acesso ao ponto de vista do usuário dentro da cena;
- A interface *Scene* contém três novos métodos para assinalar, remover e acessar os identificadores DEF aos nós. Isto facilita a referência a partes especiais dentro da cena como o controle de uma avatar;
- O método *createNode()* foi adicionado para permitir a criação de novos nós, como objetos ou avatares;
- Subclasses da classe *FieldClass* subclasses foram adicionadas. Todos os *FieldClass* subclasses possuem métodos que além de retornar o valor, também alteram o valor dos campos;

4.3.2 *User Interface Manager*

A interface *User Manager* é responsável em acessar os dispositivos de entrada para permitir o tratamento desses eventos dentro da aplicação. Enquanto as características básicas estão disponíveis

através dos vários nós, tal como o nó *ProximitySensor* (Sensor de Proximidade), outras atualmente não são tratadas. Por exemplo: entrada de dados pelo teclado para iniciar ações especiais; tratamento movimento do mouse para mover objetos dentro da cena; entrada de vídeo ou áudio utilizando uma câmera ou um microfone; aparência do cursor do mouse, modificação da barra de estados; etc. A *Figura 6* mostra o desenho desta API.

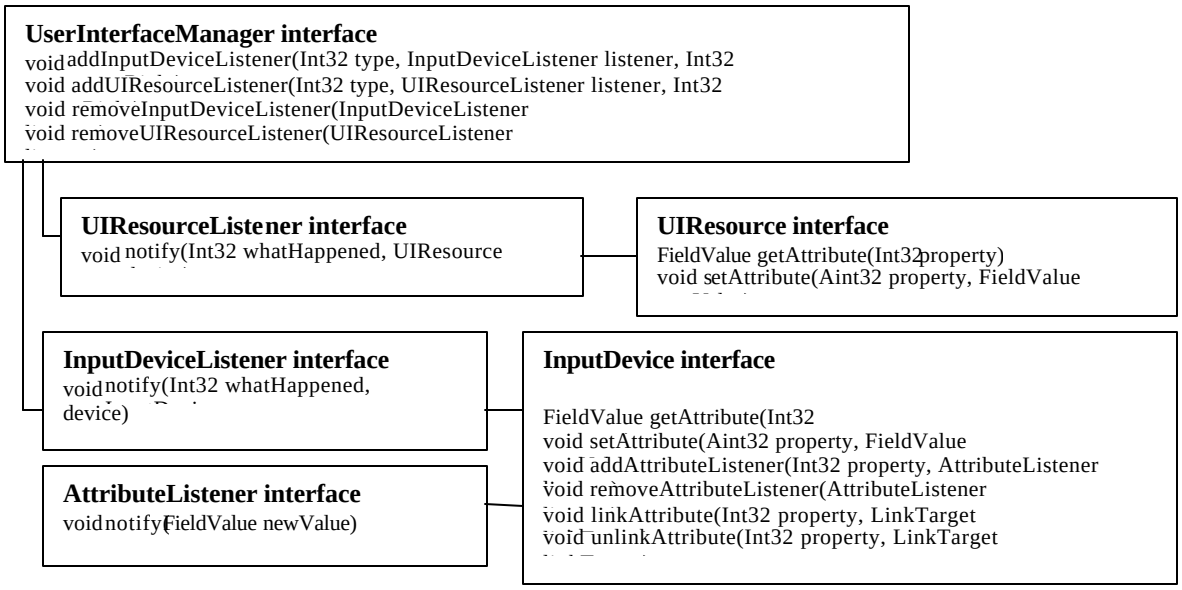

*Figura 6 – User Interface Manager - *Diagrama de Classe (UML)*

A API foi projetada como uma interface de baixo-nível para os componentes interativos de um *player* MPEG-4. A parte principal desta API é a interface *UserInterfaceManager*. Uma aplicação MPEG-J pode registrar um *Listener* nos dispositivos de entrada ou nos recursos de interface, através desta interface. Estes *Listeners* retornam uma mensagem de estado e uma referência para um dispositivo de entrada (*InputDevice*) ou um recurso de interface (*UIResource*). Desde que dispositivos de entrada e recursos de interface do usuário são recursos limitados em um *player* MPEG-4, a aplicação MPEG-J tem que especificar o tipo de acesso (compartilhado ou exclusivo). Em ambos os casos, se a alocação do recurso falhar, ela deve indicar uma exceção apropriada. Se a alocação foi bem sucedida, o método *notify()* receber um *InputDevice* ou um *UIResource* como parâmetro. A interface *InputDevice* oferece métodos para a leitura ou gravação de valores de certos atributos. O número e tipo de atributo dependem do tipo de dispositivo. Por exemplo, um *mouse* deve suportar dois ou mais valores booleanos como atributos. Para melhorar a facilidade de uso e reduzir redundâncias, os valores são encapsulados em subinterfaces da interface *FieldValue* da API *SceneGraph Manager*. Desde que aplicações MPEG-J também necessitam manipular eventos, a interface *InputDevice* pode registrar um *AttributeListener*. O método *notify()* desta interface é chamado, toda vez que um valor do atributo for trocado.

Para permitir a manipulação de dispositivos que requerem grande largura de banda, como microfones ou câmeras de vídeo, a interface *InputDevice* também possui dois métodos que ligam um fluxo de entrada com outros componentes do MPEG-4. A interface *UIResource* foi projetada como a interface *InputDevice*. Entretanto ela não tem a habilidade de registrar *Listiners* ou fazer a ligação de fluxo de dados com os componentes do MPEG-4. Tipicamente recursos de interfaces do usuário são a aparência do cursor do mouse e barra de estados.

4.3.3 *Message Manager*

A principal motivação desta API é de transmitir eventos que ocorrem durante a execução de uma aplicação entre os terminais MPEG-4. Exemplos desses eventos são quando o usuário entra do ambientes (*join*), mensagens de reposicionamento dos avateres, envio e recepção de texto entre dois avatares (*chat*), etc.

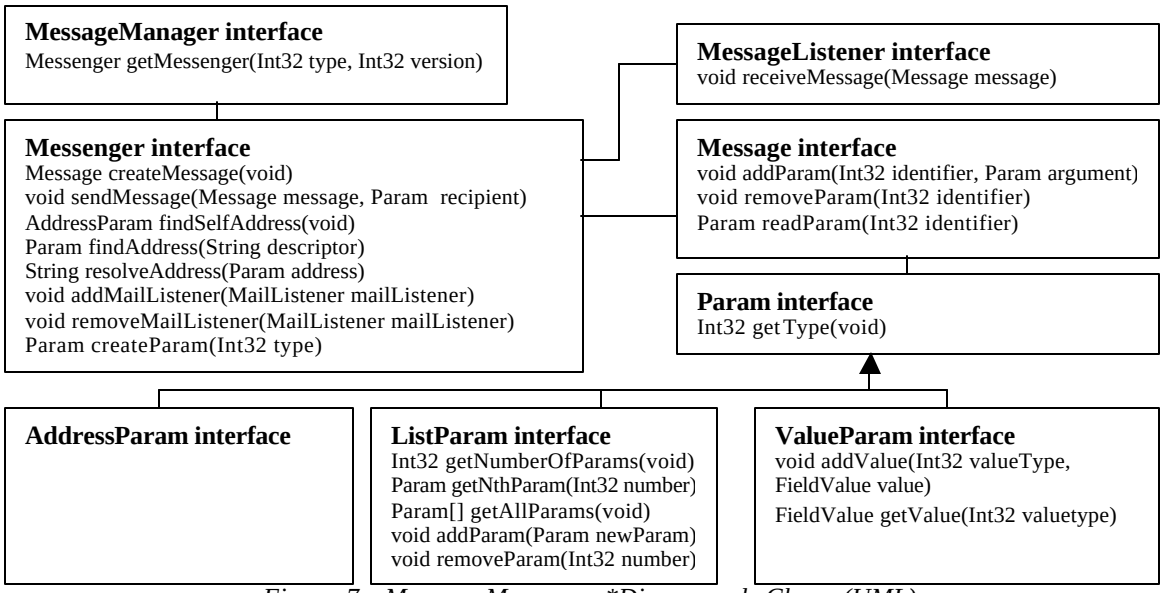

*Figura 7 – Message Manager - *Diagrama de Classe (UML)*

A principal interface desta API e a *MessageManager*. O método *getMessenger()* permite o acesso a um dos serviços de mensagens instalado, representado pela interface *Messenger*. Um terminal pode suportar diferentes tipos de mensagens. Os detalhes de implementação, como o protocolo de rede, são deixados como detalhes de implementação. Entretanto, a padronização de um especial serviço de mensagem, no qual é presente em todos os *players* conformantes com o MPEG-4, deve ser adotado.

A interface *Messeger* possui métodos para criar novas mensagens, enviar as mensagens para um endereço (representado pela interface *AddressParam*) ou para toda a lista de endereços (representado pela interface *ParamList*, no qual contem um conjunto de *AddressParams)*. Endereços de outros terminais são obtidos através do método *findAddress()*. O recebimento da mensagem é manipulado pela interface *MenssageListener*.

Uma mensagem é representada pela interface *Message*. Ela contém métodos que permitem a adição, remoção e avaliação dos parâmetros das mensagens. Cada parâmetro tem um identificador (*ID*) único para distinguir um parâmetro do outro. Enquanto alguns identificadores de parâmetros já estão determinados, cada aplicação MPEG-J pode adicionar outros parâmetros nas mensagens. Cada mensagem deve ter sempre um parâmetro do tipo *messageID* (identificador da mensagem). Como também alguns identificadores de mensagem já estão definidos (*avartarJoin*, *avatarTranslation*, *avatarRotation*, *avatarLeave*, etc.), novos tipos de mensagens podem ser criadas. Quando uma mensagem é enviada, o método *sendMessage()* automaticamente adiciona o endereço origem como um parâmetro do tipo *senderAddress* a mensagem. Um parâmetro é representado por uma das três subinterfaces: um *AddresParam* contem o endereço destino ou origem. Um parâmetro *AddressParam* pode ser unicamente obtido através do método *findAddress()*. O único parâmetro deste método é uma

string descrevendo um ou mais endereços dos terminais, neste caso, o objeto retornado será do tipo *ListParam*, que conterá mais que um endereço. A *Figura 7* mostra o projeto desta API.

4.3.4 *Shared State Manager*

Aplicações multiusuário, muitas vezes, necessitam controlar os estados dos objetos dentro da cena: uma porta pode estar aberta ou fechada, um objeto pode estar sendo usado ou não por uma avatar, um vídeo pode ser adiantado, atrasado ou congelado, etc. Todos estes estados devem estar acessíveis para todos os participantes do ambiente. Se existir um grande número de estados e um grande número de terminais que compartilham estes estados, este pode ser um ponto de gargalo. Uma solução para este problema é minimizar a troca de estados entre os terminais, nos quais não são diretamente influenciados por estes estados. Apesar disso, uma eficiente implementação dos estados compartilhados pode ser um desafio. O propósito desta API (*Shared State Manager)* é prover um flexível modo na gerência destes estados em um sistema multiusuário. Do mesmo modo que existem uma grande quantidade de mensagens, existem também várias formas de gerenciamento dos estados compartilhados. Desta forma, não é interessante escolher um tipo e adotá-lo como padrão. Em vez disto, é implementado unicamente a interface da API, no qual os detalhes de implementação são deixados para os implementadores.

A troca de um estado compartilhado pode comunicado aos demais terminais através do envio de uma simples mensagem. Um pode sugerir que a API *Message Manager* possui todas as necessidades para controlar os estados compartilhados, tornando desnecessário o uso da API *Shared State Manager*. Isto pode ser verdadeiro para estados que podem ser compartilhados através de troca de mensagens, porém existem outros requisitos que esta API não cobre, como por exemplo: a sincronização do acesso a um estado compartilhado, *deadlocks* ou manipulação do tráfico da rede causado pela quantidade de estados compartilhados.

A API *Shared State Manager* contém a classe *SharedStateManager* que fornece o acesso à funcionalidade principal. A implementação do modo em que os estados compartilhados serão gerenciados, não é especificada pelo padrão. A aplicação deve usar o método *getProvider()* para obter acesso a interface *SharedStateProvider*, no qual representa o gerenciador de estados instalado no *player* MPEG-4. A interface *SharedStateProvider* oferece métodos para a criação, registro e subscrição dos domínios dos estados compartilhados. Um domínio contém um grupo de estados compartilhados, organizados em estrutura de arvore. Cada terminal pode criar novos domínios e registra-los em um *broker* central. Após o registro, outros terminais podem realizar chamadas aos serviços do *broker* por domínios disponíveis e receber uma descrição dos domínios que ele deseja acessar. Esta descrição é então usada para subscrever ao domínio usando o método *subscribeDomain()*. Os serviços do *broker* não são especificados aqui, porque cada aplicação pode ter diferentes tipos de requisitos. O dono do domínio pode criar novos estados e adicioná-los ao domínio. Ele pode também estabelecer os direitos de acesso para os demais terminais através do método *setAccessPermission()*. Um domínio pode conter um ou mais estados compartilhados, que são representados pela interface *State*. A interface *ValueState* contém um estado compartilhado e o seu valor atual. O valor pode ser alterado ou obtido com os seus respectivos métodos. Ao contrário dos possíveis tipos de parâmetros que uma mensagem pode ter, o valor de um estado só possui um tipo de parâmetro. A interface *ListState* contém um grupo de estados. Ela contém métodos para adicionar, remover e acessar o conjunto de estados. Cada domínio contém uma raiz do tipo *ListState*, no qual pode ser obtida usando o método *getRootState()*. O método *getState()* possibilita o acesso rápido os estados na árvore. Por exemplo: *getState("Sala.Porta.locked")* recupera o *ValueState "locked"* contido no *ListState "Porta"* , no qual faz parte do *ListState "Sala"*.

Para prevenir condições de *deadlock*s um terminal pode travar um ou mais estados. Os estado são protegidos de gravação até que a trava seja liberada. E finalmente, a interface *StateListener* pode ser utilizada para notificar trocar de estados para a aplicação. A *Figura 8* mostra esta API.

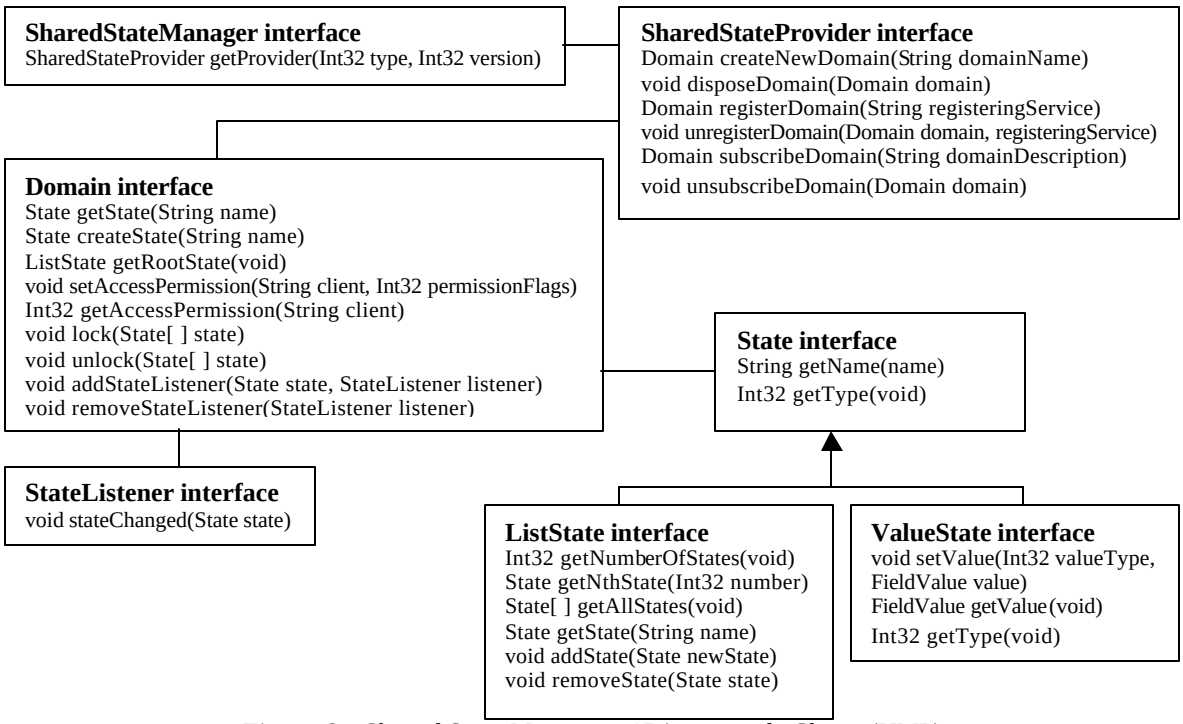

*Figura 8 – Shared State Manager - *Diagrama de Classe (UML)*

4.4 Outras Extensões

As extensões acima são o primeiro passo para a padronização de sistemas multiusuário no MPEG-4. Outras modificações são necessárias, mas isto esta fora do escopo deste artigo. Algumas destas modificações são:

- Criação de novos nós, tal como *SharedNodes* proposto pelo LivingGroup [*LIVW*];
- Uma Server API, muitas aplicações multiusuário que são tipicamente do tipo cliente/servidor;
- Serviços de alto nível: a proposta apresenta neste artigo especifica um conjunto de APIs de baixo nível, por motivo de flexibilidade. Porém um conjunto de APIs de alto nível pode também ser especificado.

4.5 Exemplo de Uso: Enviando uma Mensagem de Entrada de um Avatar.

O código abaixo mostra o envio de uma mensagem de entrada de um avatar (*AvatarJoin*) em um ambiente virtual para os demais participantes do ambiente. A mensagem contém o seu identificador (*messageID*), o nome do avatar e a sua posição inicial.

```
void sendAvatarJoinMessage(String recipientAddress, String avatarName, SFVec3fValue position)
{
        MessageManager messageManager = gTerminal.getMessageManager();
        Messenger messenger = messageManager.getMessenger(MessengerType.UDPMessenger, 1);
        Message message = messenger.createMessage();
```

```
        // Adiciona o identificador da mensagem.
        SFInt32FieldValue intFieldValue = new SFInt32FieldValue;
        intFieldValue.setSFInt32Value(MessageID.AvatarJoin);
        ValueParam messageIDParam = (ValueParam)(messenger.createParam(ParamType.ValueParam));
        messageIDParam.addValue(ValueType.SFInt32Value, intFieldValue);
        message.addParam(ParamType.MessageID, messageIDParam);

        // Adiciona o nome do avatar
        SFStringFieldValue stringFieldValue = new SFStringFieldValue;
        stringFieldValue.setSFStringValue(avatarName);
        messageIDParam = (ValueParam)(messenger.createParam(ParamType.ValueParam));
        messageIDParam.addValue(ValueType.SFStringValue, stringFieldValue);
        message.addParam(ParamType.AvatarName, messageIDParam);

        // Adiciona a posição atual
        messageIDParam = (ValueParam)(messenger.createParam(ParamType.ValueParam));
        messageIDParam.addValue(ValueType.SFVec3fValue, position);
        message.addParam(ParamType.AvatarPosition, messageIDParam);

        // Envia a mensagem
        Param recipient = messenger.findAddress(recipientAddress);
        messenger.sendMessage(message, recipient);
}
```

5. Conclusões

Este artigo estendeu o conjunto de APIs do MPEG-J para possibilitar o suporte multiusuário dentro do MPEG-4 através da ferramenta SVRT. Alguns detalhes desta especificação podem sofrer alterações, pelo fato de não serem ainda testadas, entretanto esta arquitetura deve servir como ponto de partida para o suporte multiusuário dentro do padrão MPEG-4.

6. Agradecimentos

A CAPES e a FAPESP.

7. Referências Bibliográficas

[LIVW]	*http://www.vrml.org/WorkingGroups/living-worlds/*
[M4646]	*Alex MacAulay, Julien Signes: Review of the MPEG-J specification, May 1999, contribution to the ISO/IEC JTC1/SC29/WG11 standardization process.*
[M5660]	*Glauco Todesco, Uli Rucket, Regina B. Araujo: MPEG-J Multi-User Proposal, Jan-2000, proposal to the ISO/IEC JTC1/SC29/WG11 standardization process.*
[N2201]	*Alexandrros Eleftheriadis et al.: Text for ISO/IEC FCD 14496-1 Systems, 15th May 1998, text for the ISO/IEC JTC1/SC29/WG11 standardization process*
[N3205]	Multi-users technology (Requirements and Applications) - ISO/IEC JTC1/SC29/WG11 standardization process. December 1999.
[Oliv98]	Oliveira, M.A.M., Araujo, R.B., Bila, W.S. - SVRT – Uma Ferramenta de Suporte às Aplicações de Sistemas Distribuídos de Realidade Virtual através da Integração de Mecanismos e Ferramentas das Redes MBONE e WWW – XXIV Latinamerican Conference of Informatics (CLEI'1998) Quito - 1998.
[Todes00]	MPEG-4 Support to Multiuser Virtual Environments - Todesco, G., Araujo, R.B. ICDCS'2000 – IEEE – Mar/2000.
[VRML]	The Virtual Reality Modeling Language Specification International Standard ISO/IEC 14772-:1997, eletronic document in http://www.vrml.org/Specifications/VRML97/index.html, 1998 (12 Oct.).