3
DWeb3D

Supondo que as limitações apresentadas anteriormente sejam o motivo principal para a baixa adoção do X3D, propõe-se desenvolver um conjunto de ferramentas que reduza a curva de aprendizado e o tempo necessário para o desenvolvimento de uma aplicação X3D, ao mesmo tempo que promova a reutilização racional de código.

Seguindo esta linha de raciocínio foi desenvolvido um toolkit para auxiliar o desenvolvimento de aplicações colaborativas utilizando o X3D e uma aplicação que faça uso deste toolkit como prova de conceito. Entende-se como um toolkit um conjunto de bibliotecas de desenvolvimento que lidem com diversos dos desafios encontrados no desenvolvimento de uma aplicação com determinadas características. O objetivo principal é retirar o fardo de lidar com tarefas repetitivas e complexas, dando assim ao desenvolvedor a possibilidade de focar-se no problema que ele deseja resolver, tornando o processo mais criativo. (Gre07).

3.1
Metodologia

Tanto para o estudo como para a aplicação de demonstração foram usadas metodologias bem definidas.

3.1.1
Metodologia para o estudo

Primeiro foram analisados quais são os problemas e como abordá-los. Analisou-se o porquê da baixa adoção do X3D e foram propostos métodos de trabalhos e ferramentas que possam ajudar a mudar esse quadro e tentar medir de alguma forma esta proposta.

Para poder medir o nível de aceitação do padrão foi utilizada uma análise das ocorrências informadas por uma ferramenta de busca. Mesmo não sendo um método muito preciso, ele resulta em valores interessantes, pois a ferramenta indexa tudo que encontra na internet e não é afetada pelo tipo de arquivo.

Assim o sendo, é razoável supor que sua base esteja distribuída igualmente entre todos os formatos.

Destes resultados partiu-se para análise das capacidades do X3D, através de pesquisa na documentação do mesmo e análise dos produtos existentes que o utilizam. O objetivo desta analise foi detectar onde o X3D está sendo utilizado, onde não está e o porquê disto. O resultado foi a lista de limitações detectadas.

Finalmente, com as limitações foi gerada uma proposta que tenta melhorar parte dos problemas. Esta proposta concretizou-se no tookit desenvolvido, que serviu como método de analisar se a proposta pode ou não ajudar no processo de desenvolvimento de aplicações X3D.

3.1.2 Metodologia para o desenvolvimento da aplicação

Como existia somente uma pessoa trabalhando no desenvolvimento da tecnologia, métodos de desenvolvimento mais tradicionais seriam dispendiosos e pouco práticos, portanto foi adotado uma metodologia um pouco diferente que se mostrou eficaz para o caso de uma pessoa como desenvolvedor.

Inicialmente as tecnologias disponíveis foram avaliadas e uma foi selecionada. Os critérios para a avaliação foram:

- Familiaridade: se o desenvolvedor já a conhece, tanto melhor pois não exige uma curva de aprendizado
- Popularidade: é uma tecnologia popular que vários outros desenvolvedores conhecem e utilizariam?
- Robustez: é algo robusto e que permitiria uma modularização razoável?
- Aplicabilidade: a tecnologia possui características que a permitam ser utilizada no ambiente escolhido?

Para o desenvolvimento foi utilizado um método que é uma adaptação das metodologias ágeis. Para isso alterou-se ligeiramente a ideia de ciclos e gestão de tarefas pequenas do scrum (Sch01, Bat08). Os ciclos e as revisões com os usuários foram trocados por metas. Assim sendo, cada meta seria um ciclo para o desenvolvimento incremental. Antes de cada ciclo se determinava uma lista de tarefas necessárias para alcançá-lo e a cada dia uma tarefa era selecionada. Esta lista poderia ser modificada na medida da necessidade. A prática de se definir um construto finalizado ao fim de cada ciclo foi trocada pela construção de um teste da funcionalidade desejada (Figura 3.1).

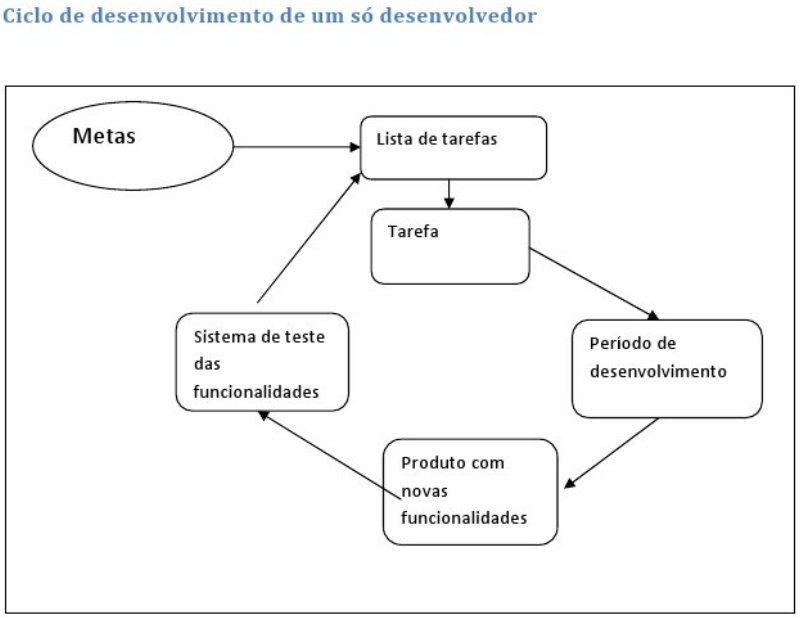

Figura 3.1: Ciclo de desenvolvimento para um só desenvolvedor

3.2 Tecnologias

Para o desenvolvimento do toolkit foi eleita a linguagem C# dentro da plataforma .NET. A escolha foi feita seguindo os critérios de escolha anteriormente listados, sendo:

- Popularidade: é amplamente utilizada e mantida por uma empresa muito grande, além de possuir uma comunidade de desenvolvedores crescente e bastante robusta.
- Robustez: existe há vários anos no mercado e é utilizada nos mais diversos meios, desde de aplicações corporativas a jogos.
- Aplicabilidade: o .NET possui um framework para trabalhar com aplicações web e é bastante utilizado neste meio, além de possuir capacidade de criação de bibliotecas compiladas e pacotes para redistribuição de código.

Como controle de versão foi utilizado o Mercurial[1]. As qualidades que o tornam interessante para um projeto de um só desenvolvedor são a possibilidade de funcionamento sem um servidor centralizado e a possibilidade de sincronização de diversos repositórios, desta forma possibilitando o desenvolvimento em máquinas diferentes e facilitando as cópias de segurança.

[1] http://mercurial.selenic.com/wiki/

Finalmente, para o caso de estudo foi decidido desenvolver um plugin de renderização para a Unity3D[2]. A Unity3D é um motor gráfico desenvolvido por um grupo americano, que se propõe a ser modularizável e extensível. Ela suporta uma grande gama de formatos para importação, possui um IDE[3] bem completo além de outras características interessantes. A escolha foi feita porque a Unity3D possui uma linguagem de scripts baseada em C#, facilitando a integração com o código já existente, possui um plugin para utilização na web e possibilitaria uma interface mais rica do que utilizar um browser X3D existente, uma vez que com estes a interface teria que ser feita em ECMAScript e seria mais complexa.

O X3D possui uma API em ECMAScript já definida chamada SAI[4] que se propõe a disponibilizar o acesso de objetos da cena dentro do browser. O problema de sua utilização é que sua programação fica restrita ao ambiente do cliente de visualização e portanto não é própria para uma interação mais complexa.

3.3 Estrutura do DWeb3D

O toolkit se divide em módulos organizados por função, cada módulo representa uma dll[5]. Isto possibilita um uso mais racional, uma vez que se não forem necessárias todas as funções, não é necessário adicionar à aplicação todas as dlls. O Apêndice A contém alguns gráficos UML representando a estrutura do DWeb3D.

A divisão das dlls é a seguinte (Figura 3.2):

- Model: dll onde está o grafo. É a base de todo o toolkit e necessária para se utilizar as outras partes.
- ObjetSync: dll responsável para sincronização de características dos objetos em diversas instâncias através da rede.
- DWebServer: executável que demonstra como um servidor pode funcionar.
- Unity Render: plugin para possibilitar o uso do X3D com a Unity.
- Util: conjunto de ferramentas utilizado pelo Unity render.
- Unity WebPlayer: plugin para browser que permite a visualização do projeto na Unity.

[2]http://unity3d.com/
[3]Integrated Development Enviroment
[4]Scene Access Interface
[5]Biblioteca dinâmica. Do inglês Dinamic Load Library.

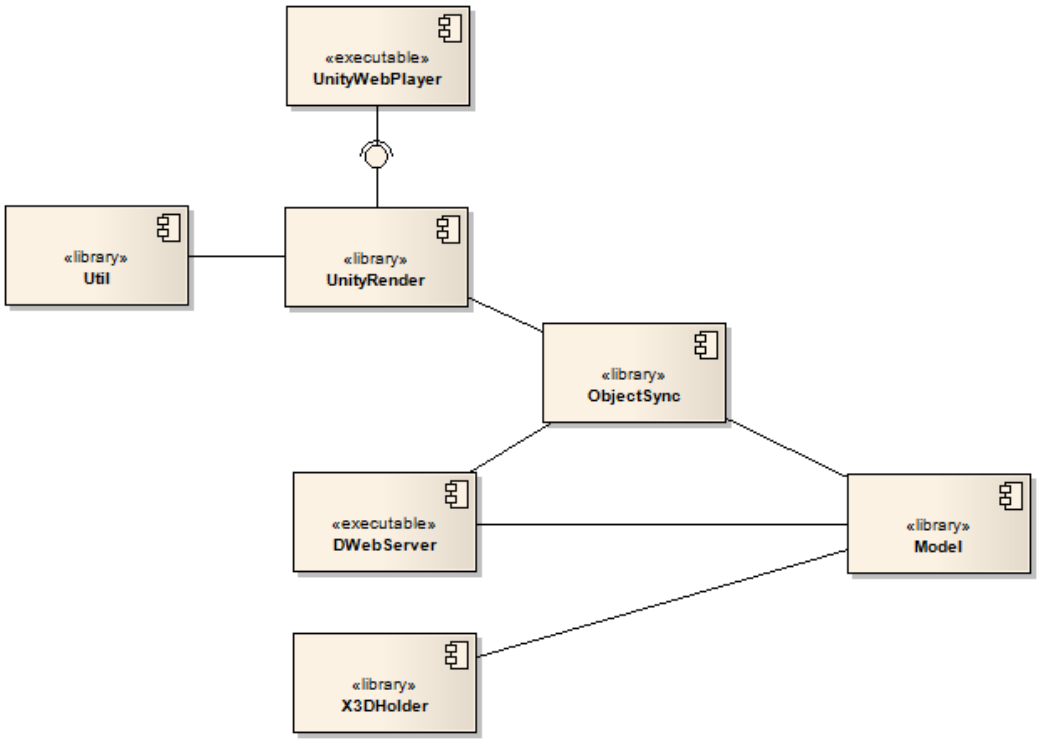

Figura 3.2: Modelo de componentes do DWeb3D.

3.3.1 Model

Este construto é a base do funcionamento do toolkit. Dentro dele está o grafo e todas as suas classes que representam a estrutura do X3D. Também nele estão as classes principais de leitura e escrita de arquivos X3D, como a classe X3DRepresentation ou o X3DHolder.

O X3DRepresentation é a principal classe de manipulação do grafo. Ele é o responsável pela leitura dos arquivos X3D e a escrita do grafo em formato XML. Já o X3DHolder é a classe responsável pela utilização da estrutura num ambiente asp.net.

O grafo em si é uma estrutura mais complexa e por isso será discutido mais a frente.

3.3.2 ObjectSync

Este é o pacote responsável pela sincronização de vários objetos do grafo através da rede. O modelo de classes do ObjectSync é apresentado na Figura 3.3. Para que o ObjectSync possa ser utilizado basta que o objeto alvo seja do tipo ObjectSync ou herde dele e que ele seja registrado no gerente de sincronismo. A interface define quais propriedades serão sincronizadas e

permite disparar o sincronismo a partir de uma única chamada de função.

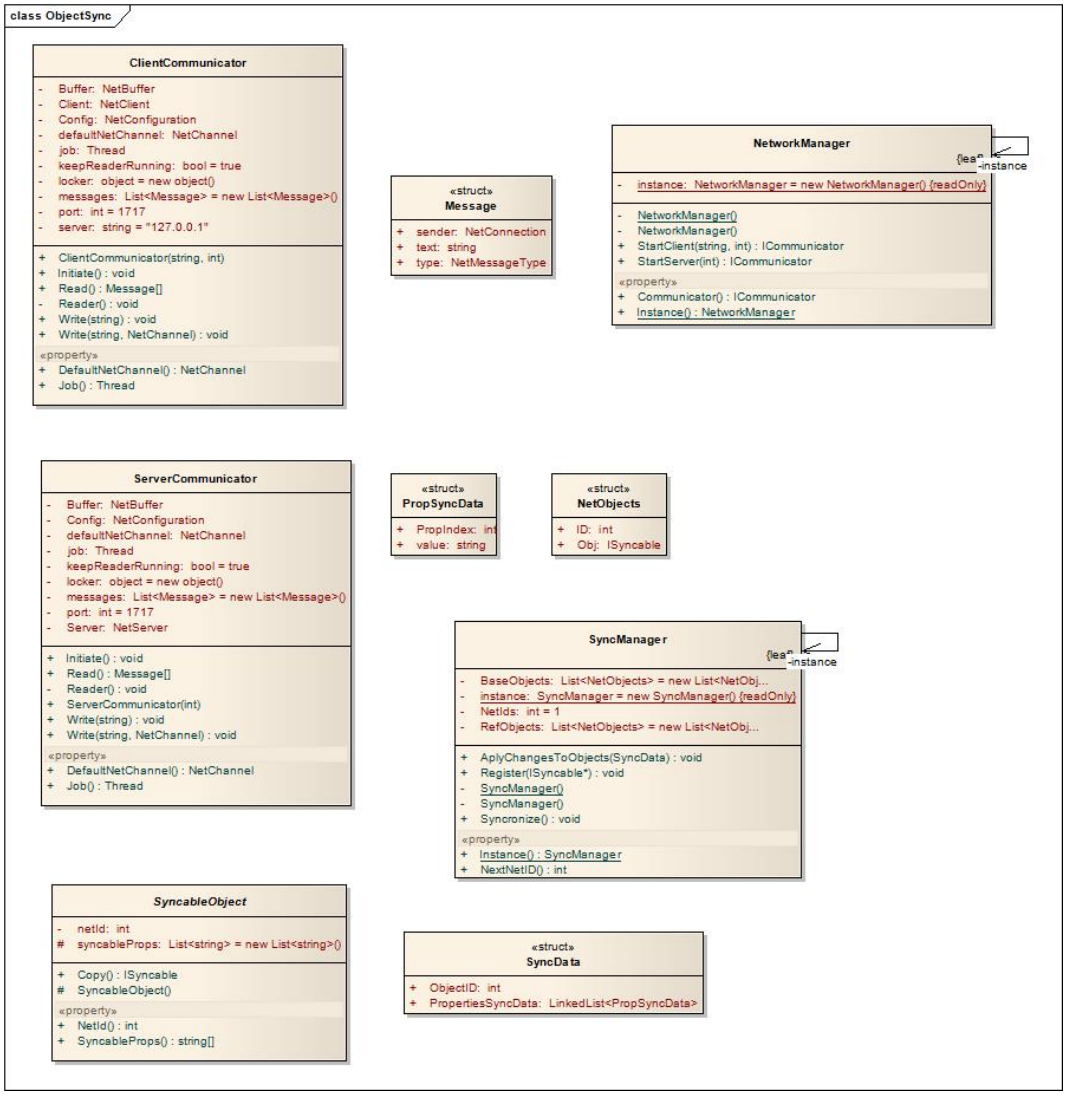

Figura 3.3: Modelo das classes do ObjectSync.

O importante desta biblioteca é que ela lida com toda a camada de rede e torna bastante simples a tarefa de fazer a sincronização.

3.3.3 DWebServer

Dll responsável pela criação de um servidor. Ela somente recebe as sincronizações e repassa para os outros clientes. A principal característica é que ele faz uso das classes contidas no ObjectSync e que por sua vez possuem capacidades de transmitir as sincronizações. Ele mantém uma cópia própria do grafo. Vale lembrar que a sincronização sempre é feita do cliente para o servidor.

3.3.4 Unity Render

É um plugin de redenderização (na forma de uma dll) que permite utilizar o grafo X3D dentro do ambiente da Unity3D. O que ele faz é criar objetos Unity através das respresentações do X3D e mantê-los sincronizados com os objetos do grafo.

3.3.5 Util

Não é utilizado diretamente por desenvolvedores externos. Na verdade este pacote contém código de auxílio interno do toolkit, como conversões e outros códigos reutilizados dentro do toolkit.

3.3.6 Unity WebPlayer

Este é um executável com os scripts desenvolvidos para o DWeb3D já colocados, de forma a servir de demonstração da interação do plugin Unity com o motor gráfico e de como colocar o plugin para funcionar. O WebPlayer poderá ser utilizado como um substituto de um browser X3D por aqueles que desejarem utilizar o plugin.

3.4 O Grafo X3D

Sendo o grafo a parte mais complexa do toolkit, ele merece uma seção para explorá-lo em separado.

O grafo segue uma estrutura hierárquica com um modelo de árvore (Figuras 3.4 e 3.5), sendo que sua raiz é sempre a classe X3DRepresentation. Esta classe contém os métodos de carga e renderização, além de uma lista de nós filhos. Os nós são objetos que herdam do tipo Node. A X3DRepresentation separa os nós do cabeçalho dos nós das cena por suas diferenças funcionais. Todos os nós de cena herdam do objeto SceneNode que possui suas caraterísticas básicas.

Algumas classes são chaves no funcionamento do grafo e merecem uma explicação especial.

3.4.1 SceneNode

Classe base das classes de cena, ela não é utilizada diretamente quando se for manipular o grafo, mas é ela que contém muitas das operações comuns aos

Simple Part 1.jpg

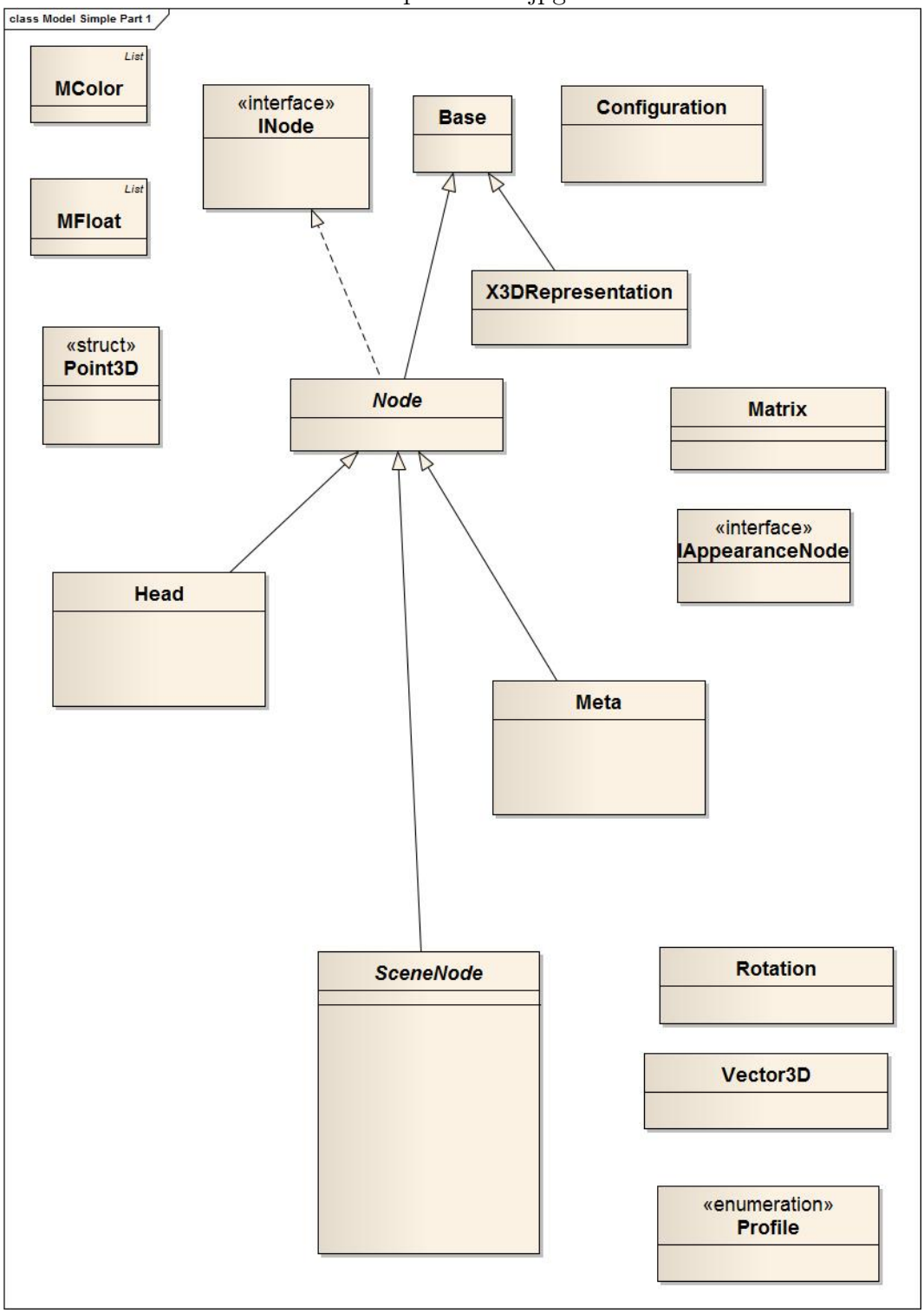

Figura 3.4: Modelo de domínio do Grafo X3D (I).

Simple Part 2.jpg

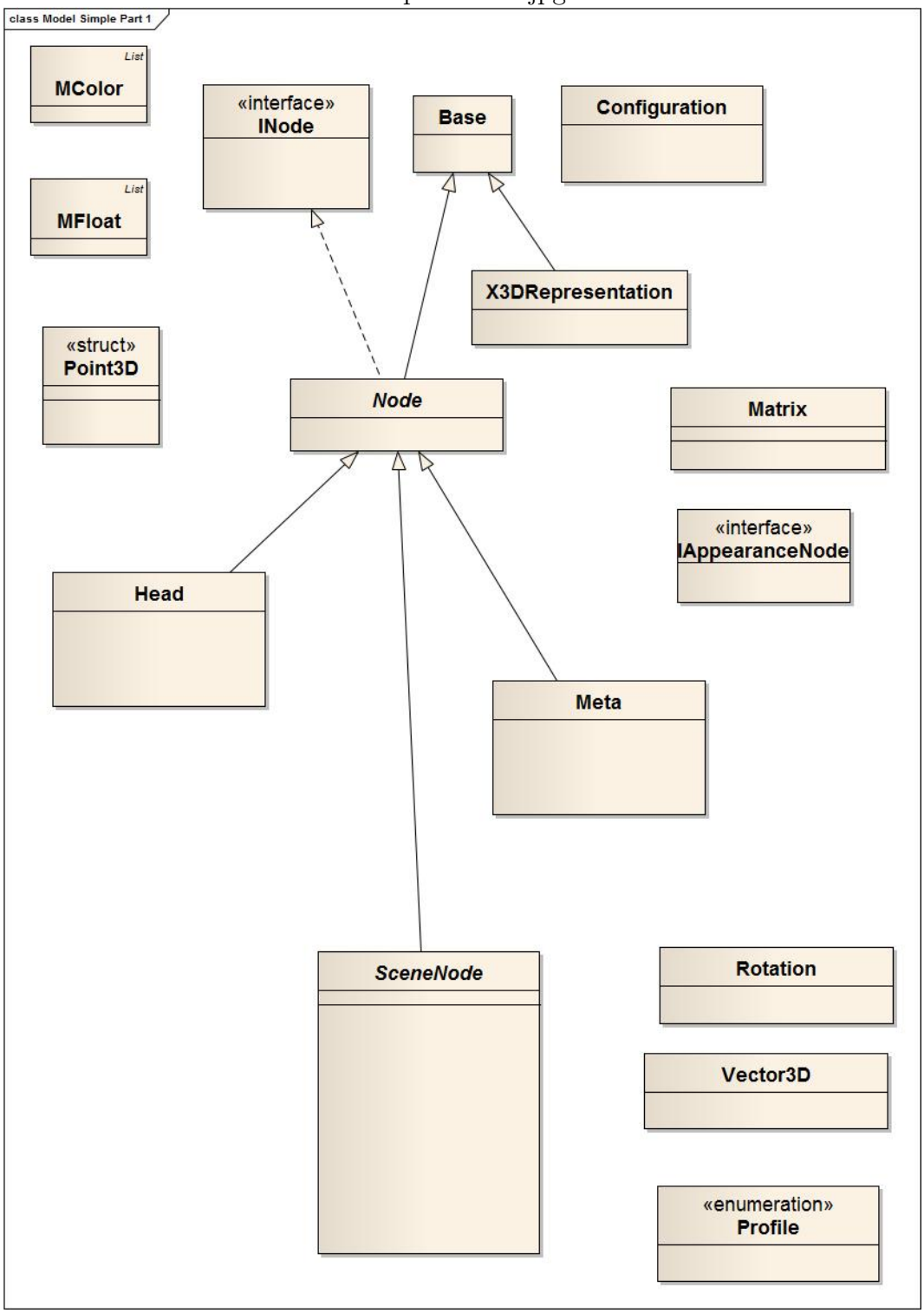

Figura 3.5: Modelo de domínio do Grafo X3D (II).

nós de cena. Por exemplo, buscar filhos, que é implementada como uma função recursiva que busca em profundidade na árvore de filhos. É nela também que está implementado o padrão delegate, ou seja, através dela é delegada a cada classe a responsabilidade por fazer as buscas em seus filhos e por fazer a sua renderização.

A utilização desta estrutura hierárquica possibilitou uma economia considerável de código e por conseguinte fica mais fácil mantê-lo ou estendê-lo.

3.4.2 Group

Esta classe serve para agrupar outros objetos; agrupar e nomear de forma que seja possível encontrar um objeto buscando pelo grupo ao qual ele pertence. A capacidade de nomear é bastante importante porque é assim que se permite uma melhor organização no X3D.

3.4.3 Transform

Trata-se de um grupo com capacidades especiais. Esta classe lida com toda e qualquer transformação necessária no mundo X3D. Por isso ela possui a capacidade de armazenar quaternions para rotações, vetores de translação e vetores de escala. O Util possui métodos para lidar com conversões de matrizes para vetores, de forma que é possível trabalhar também com matrizes. É importante ressaltar que todo objeto vai necessitar estar dentro de uma translação sozinho ou em grupo com outros, pois é a única forma de posicioná-lo na cena.

3.4.4 Appearance

Um appearance pode possuir vários shapes dentro dele. Isto pode parecer sintaticamente estranho, mas é a forma que o X3D organiza as coisas, portanto o Appearance agrupa formas, materiais e texturas, possibilitando definir uma aparência.

3.4.5 ViewPoint

Um ViewPoint determina uma posição estática de uma câmera. Uma cena pode ter vários viewpoints, mas precisa de pelo menos um para definir a posição inicial da câmera. No caso de vários existirem, o primeiro é tido como a viewpoint padrão.

3.4.6 Background

Define as configurações de fundo da cena assim como a iluminação básica desta.

3.5 Implementação das metas no DWeb3D

De forma a ilustrar a adequação do toolkit aos objetivos propostos, deve-se demonstrar como ele pode resolver os problemas apresentados e como sua estrutura foi pensada para estes casos.

3.5.1 Colaboração

Para se alcançar a meta de possibilitar a colaboração alguns fatores são importantes:

- Facilitar a criação de um servidor.
- Possibilitar a sincronização das posições dos objetos entre as visualizações de diversos clientes.
- Possibilitar a criação de novos objetos via interface do cliente.

Para se criar um servidor básico, o código necessário se resume a:

```
public void StartDefaultServer()
{
   // Criando o servidor na porta 1717
    SceneServer s = new SceneServer(1717);
    // Definindo e criando a cena básica que este servidor
    // vai trabalhar
    s.SceneGraph = new X3DRepresentation();
    s.SceneGraph.CreateBasicScene();
    // Definindo que este é o servidor padrão
    // (ele prevê a existência de outros)
    s.IsDefault = true;
    // Adicionando o servidor ao gestor de cenas.
    DefaultServer = s;
    SceneServers.Add(s);
}
```

Pode-se notar que se tratam de quatro operações. Cria-se o objeto, diz-se qual a cena padrão dele (por ser uma cena vazia), define-se que ele é o servidor padrão (podemos ter vários na mesma máquina) e registra-se o servidor no gerente que o vai organizar.

As diversas funções de baixo nível como criar socket, tratar de threads, e criar loops de leitura necessários para um servidor estão englobadas no DWeb3D e, portanto, o desenvolvedor não precisa lidar com elas. Desta forma o tamanho e a complexidade do código produzido são em muito reduzidos.

Para se lidar com a sincronização foi criado o módulo ObjetSync. Para utilizá-los basta que os objetos a serem sincronizados implementem uma interface específica. Esta interface permite definir quais campos queremos que sejam sincronizados no objeto e com isso podemos promover o sincronismo através de uma simples chamada de função como exemplificado na Figura 3.5.1.

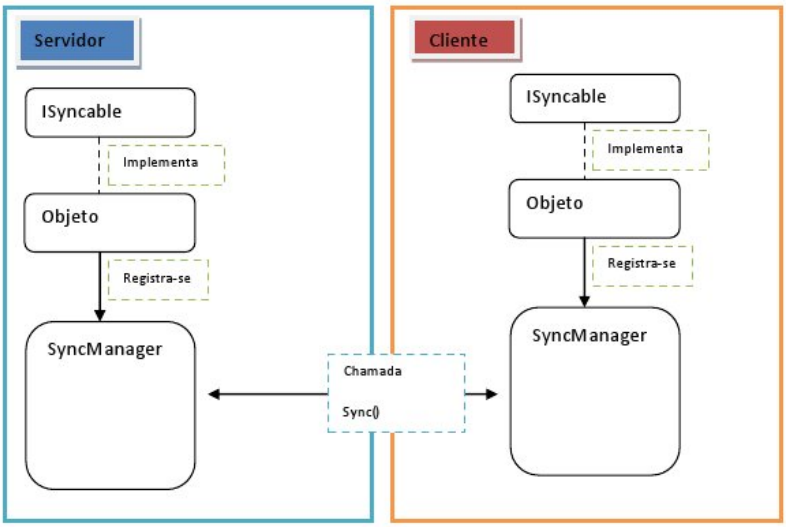

Figura 3.6: Figura exemplificando o processo de sincronização

Um pequeno trecho de código mostrando isto pode ser visto a seguir:

```
// Criação de um objeto transformável
var t = new Transform();
t.LoadFromReader(xmlReader);
```

```
// Definindo o objeto como um ISyncable (O transform é)
ISyncable tmp = t;
// Definindo um netIDválido
tmp.NetId = SyncManager.Instance.NextNetID;
// Registrando o objeto no gerente
SyncManager.Instance.Register(ref tmp );
```

Com pouco código obtém-se o registro do objeto. Porém para o caso dos objetos provenientes do grafo esse passo não é necessário pois eles já se registram na criação. Para sincronizar basta a seguinte linha:

```
SyncManager.Instance.Syncronize()
```

Se o código for proveniente do grafo o registro não é necessário. Mas isto pode ser utilizado para outras classes como por exemplo para disponibilizar um chat.

Apesar de o ObjectSync não lidar com a criação de novos objetos via interface do cliente, é factível criar um objeto que gerencie a criação de novos objetos através de uma propriedade sincronizável.

3.5.2 Persistência e carga

A persistência é feita também através dos mecanismos que permitem a colaboração, porém com um novo mecanismo, o X3DHolder. Este mecanismo utiliza um "truque"; como ele é o responsável pela exposição do objeto em um ambiente web, ele registra um cookie que identifica quem é o usuário. Ele também lê o cookie do usuário caso ele exista e, junto à classe UserManager, detecta quem é o usuário no servidor e permite que este recupere as características de sua última visita.

3.5.3 Interação com outras aplicações

Para que a aplicação X3D possa interagir com outras aplicações é necessário exisitr um mecanismo que permita expor os eventos ocorridos dentro da cena a outras aplicações. Para isto dois caminhos foram identificados:

1- Programar utilizando ECMAScript chamadas a mecanismos externos como webservices e programar nesses webservices a lógica desejada.

2- Programar um visualizador X3D que permita a adição de scripts a eventos do grafo (acesso a um nó, modificação de alguma propriedade, etc) ou até programação de gatilhos como clicar em um certo nó.

O DWeb3D decidiu seguir pelo segundo caminho e para isso foi desenvolvido o UnityRender. O que este módulo faz é converter o grafo X3D num grafo da Unity 3D de forma que seja possível publicar na web um visualizador (a Unity 3D possui essa ferramenta). Este visualizador pode receber mais plugins em formato de scripts em C# que podem vir ou do grafo ou de adições no projeto do visualizador. A Figura 3.7 ilustra o seu funcionamento.

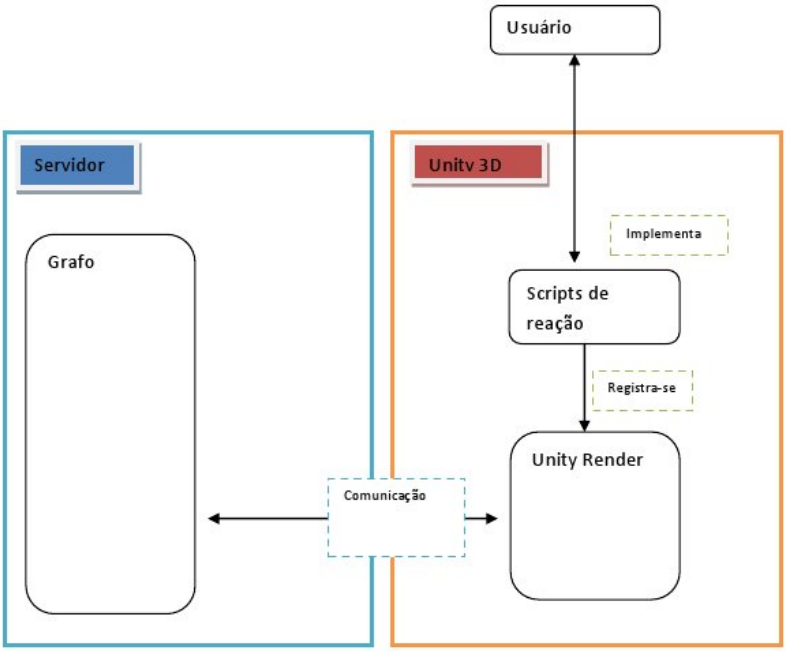

Figura 3.7: Esquema ilustrando o funcionamento do renderizador.

Com esse plugin a interação é alcançada, pois os scripts podem alterar o grafo, que por sua vez se sincroniza com o objetSync e altera o grafo (e outros objetos) no servidor e este por sua vez pode possuir código programado para interagir com ainda mais outras aplicações.

A Figura 3.8 ilustra o esquema de sincronismo proposto.

3.5.4 Interação com a GUI web

Apesar de também ser uma forma de interação com outras aplicações, o GUI web tem um papel especial por ser o ambiente onde a cena X3D se hospeda. Por isso ele possui uma forma especial de interação, que mesmo sendo parecida com a forma mencionada anteriormente utiliza, na parte web, um mecanismo especial e merece ser destrinchado a seguir.

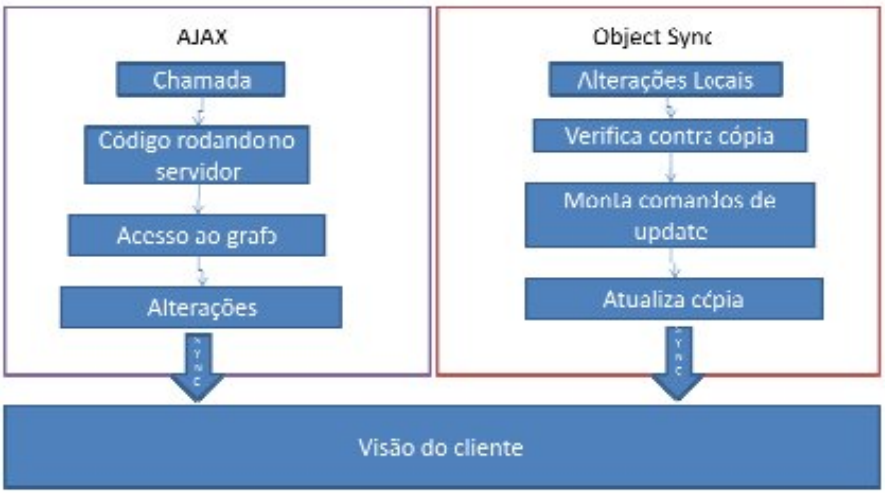

Figura 3.8: Esquema de sincronismo.

Atualmente um dos campos em maior expansão na web são as aplicações RIA[6] (Rou04, Par08), que são aplicações que possibilitam uma interface mais rica com o usuário. O progresso dos browsers e dos computadores pessoais tem permitido que muitas das aplicações antes feitas para rodar em desktop migrem para a web, e com essas aplicações os usuários passaram a demandar funcionalidades que são comuns a elas, como arrastar e soltar, seleção com o mouse, etc. As aplicações muito comumente alcançam esta meta através de aplicações de técnicas Ajax[7] (Rio08, Pud07, Dah08). Assim sendo, também o DWeb3D vai fazer uso de técnicas Ajax para alcançar uma interação rica com a aplicação 3D.

Ajax é o uso metodológico de tecnologias como JavaScript e XML, providas por navegadores, para tornar páginas mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. Ajax não é somente um novo modelo, é também uma iniciativa na construção de aplicações web mais dinâmicas e criativas. Ajax não é uma única tecnologia, são realmente várias tecnologias conhecidas trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. Ajax incorpora em seu modelo:

- Apresentação baseada em padrões, usando XHTML e CSS.
- Exposição e interação dinâmica usando o DOM.
- Intercâmbio e manipulação de dados usando XML e XSLT.

[6] Rich Internet Application
[7] Asynchronous Javascript And XML

– Recuperação assíncrona de dados usando o objeto XMLHttpRequest.

– JavaScript unindo todas elas em conjunto.

Ajax3D trata de se aplicar a mesma ideia do Ajax tradicional para objetos X3D. Uma vez que o X3D pode receber scripts em ECMAScript e expõe suas características para o domínio do Browser, é possível através do ECMAScript e de técnicas de chamadas assíncronas interagir com o X3D. Com essa ideia foi desenvolvido o projeto Ajax3D que disponibiliza uma pequena biblioteca com as funções básicas para esta interação.

O funcionamento da interação Ajax / cena 3D pode ser observado na Figura 3.9. O importante é notar que sempre o servidor é atualizado, garantindo assim não só a interação com este usuário, mas também a alteração do grafo, que por sua vez possibilita a alteração da cena.

Figura 3.9: Ciclo de interação Ajax / cena 3d.

O papel do Ajax no DWeb3D é permitir que os componentes do GUI web se comuniquem de forma transparente com a cena. Se fosse necessário forçar um *refresh* a cena seria totalmente recarregada. Este tipo de comportamento tornaria impraticável a integração pois a cena deve ter um funcionamento contínuo e uniforme.

3.6 Testes executados durante o desenvolvimento

Durante o processo de desenvolvimento diversos testes foram escritos para verificar o funcionamento correto dos algoritmos propostos. Eles visavam verificar a conectividade, interação web, sincronismo e a geração e leitura do grafo. Os testes foram:

- Geração do grafo: Foi testada a criação de um grafo através da criação manual de classes seguindo a hierarquia de um arquivo XML e foi pedida a renderização do xml em seguida. Se o resultado fosse condizente com o arquivo de origem o teste estava ok.
- Leitura de um grafo: Foi pedido que a classe geradora lesse um arquivo XML e gerasse um grafo. Depois foi pedido a esse grafo que gerasse o XML. Se o resultado fosse condizente com o arquivo original o teste estaria ok.
- Conectividade: Foi gerado um servidor de exemplo e uma classe cliente. Em seguida de máquinas diferentes foram executadas várias instâncias do cliente verificando se o servidor acusava conexão.
- Sincronismo: Foi gerado um grafo padrão tanto num servidor quanto num cliente. O grafo foi registrado como sincronizável e em seguida várias características no grafo do cliente foram alteradas. O servidor ficava imprimindo em tela as alterações e os outros clientes conectados faziam o mesmo.
- Interação web: Foi gerado um cliente que mostrava um grafo com um cubo. Foi programada uma página .Net que alterava através de chamada ajax o tamanho do cubo. Ao clicar num botão da página o cubo deveria mudar de tamanho.