Um Modelo para Implementação de Aplicações Multimídia com Sincronização Pré-Orquestrada utilizando Reflexão Computacional

Alexandre Sztajnberg (alexszt@gta.ufrj.br)

GTA / COPPE / UFRJ

Resumo

Na primeira etapa deste trabalho foi proposto um modelo de serviço de sincronização temporal de módulos para aplicações multimídia. No serviço apresentado, uma aplicação poderia ser composta de vários módulos distribuídos, que contariam por sua vez com o suporte de uma ambiente de execução configurável. Este serviço deve contar com mecanismos de controle temporal de eventos e atividades, e um escalonador de eventos que imponha a ordenação temporal desejada pela aplicação. Neste trabalho será proposta uma arquitetura de implementação baseada em um modelo de reflexão que procura tornar mais simples a programação dos módulos que farão parte da aplicação multimídia em relação aos requisitos e mecanismos de sincronização a serem utilizados. Será proposta complementarmente uma hierarquia de objetos e meta-objetos para a composição de apresentações multimídia pré-orquestradas e como esta arquitetura poderá ser implantada sobre um ambiente baseado em ORB[1]s.

Abstract

In the first part of the this work it was proposed a temporal sincronization service model for mutimedia applications. In the presented service, an aplication could be composed by several distributed modules, that would be suported, on their turn, by a configurable distributed environment. This service is based on event and activity temporal control mecanisms, and an event scheduler that can enforce the desired temporal order for the aplication. In the present text an implementation architecture based on a reflective model will be proposed, such that the programming of the modules that composes a multimidia aplication, in relation to the sincronization mechanisms to be used, is simplified. Complementarly it will be proposed an objetc and meta-objetc hierarchy to compose pre-orchestred multimidia aplications and how this architecture can be installed over an ORB based environment.

1. Introdução

A implementação de aplicações multimídia distribuídas é bastante depende do ambiente computacional disponível. No caso ideal este ambiente deveria suportar e oferecer serviços de sincronização intermídia e intramídia de maneira que uma aplicação pudesse selecionar o serviço a ser utilizado e a qualidade esperada com que este serviço será oferecido (QoS). Um ambiente ainda mais favorável ao projeto e execução de aplicações multimídia contemplaria os vários níveis de abstração proposto pelo modelo de referência para multimídia em [Bal 96].

Uma das dificuldades básicas para a concepção de sistemas multimídia distribuídos está na ampla faixa de requisitos a serem atendidos, como por exemplo, transporte de mídias com características isócronas, sincronização temporal, gerenciamento de múltiplos fluxos que compõe as apresentações, compressão e descompressão das mídias, especificação das apresentações, etc. Embora o modelo de

---

[1] ORB - *Objetc Request Broker*, é o elemento principal que implementa os mecanismos necessários para a comunicação transperente entre objetos em um ambiente baseado em CORBA (*Common Object Request Broker Architecture*) [OMG 91].

referência proporcione uma forma organizada de *atacar* estes problemas (figura 1.1), ele não consegue atender a todas as situações e, em conseqüência, algumas variações em torno deste modelo são freqüentemente adotadas [Bal 96], [Geo 96] e [Gib 91], principalmente para se adaptar a alguma implementação que já esteja operacional.

Particularmente, com relação a implementação de serviços básicos e intermediários dentro deste modelo não existe um consenso em relação a melhor forma de realizá-la. Mecanismos baseados em módulos configuráveis como proposto na primeira etapa deste trabalho [Szt 96] e em [Rot 96] procuram oferecer uma parte dos serviços necessários para a implementação de aplicações multimídia distribuídas. Vários outros projetos tem como objetivo de encontrar soluções para oferecer serviços, ferramentas ou mesmo abstrações que facilitem a especificação de cenários de sincronização em apresentações mutimídia.

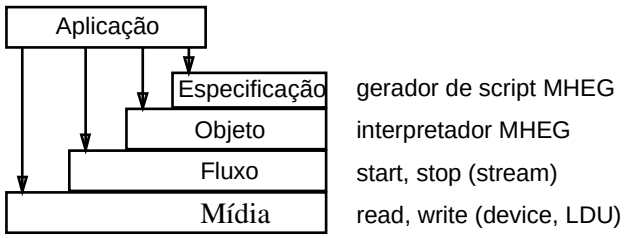

Figura 1.1 - Modelo de referência para sistemas Multimídia

Neste trabalho é proposto um esquema de implementação, que permite a construção uniforme dos serviços, desde os mais básicos, até os mais abstratos, necessários em sistemas multimídia. Este esquema, baseado em **reflexão** de módulos, pode ser utilizado em sistemas distribuídos configuráveis e, em outra vertente, pode ser utilizado em linguagens orientadas a objeto que implementem o conceito de reflexão através de **meta-objetos**. Atenção especial será dada na concepção de um ambiente que favoreça a construção de aplicações multimídia pré-orquestradas, estendendo e refinando a proposta do mecanismo apresentado em [Szt 96].

No restante do trabalho serão introduzidos os conceitos básicos de reflexão de módulos, meta-objetos e meta-protocolos. Em seguida será apresentado um esquema utilizando estes conceitos para a implementação de um conjunto de serviços para aplicações mutimídia. Baseado neste esquema será apresentado um exemplo que utiliza módulos programados com reflexão e interconectados por configuração.

2. Conceitos Básicos sobre Reflexão

A idéia central do paradigma de reflexão não é nova. Entretanto os mecanismos e a forma com que estes conceitos são tratados, torna o esquema de reflexão um aliado bastante útil para adicionar características operacionais a módulos primitivos. A reflexão computacional permite que um sistema execute processamento sobre si mesmo, para o ajuste ou a modificação de seu comportamento [Fab 93] e [Jon 93]. Um componente que utiliza reflexão é composto pelo **componente básico** e um **componente *reflexo***. O componente reflexo possui a mesma interface que o componente básico, entretanto não possui o mesmo comportamento. Ele funciona como um filtro, realizando operações no fluxo de informações que deveria ir diretamente ao componente básico, alterando ou ajustando as características operacionais deste módulo.

Por exemplo, a sincronização intermídia e intramídia pode ser adicionada a módulos de apresentação multimídia interpondo-se os mecanismos necessários através de reflexão. A idéia central da reflexão, de interposição de código, é bastante utilizada (sem o refinamento da reflexão) há bastante tempo. Por exemplo, o *hábito* de interceptar interrupções no sistema operacional DOS para oferecer alguns serviços adicionais, estendendo ou modificando o sistema básico. Nas situações mais simples, basta a

interposição de código junto ao servidor (o programa *doskey* do DOS, que *melhora* as características do teclado, por exemplo) para uma filtragem e tratamento prévio das requisições, adicionando novas funções ao serviço básico.

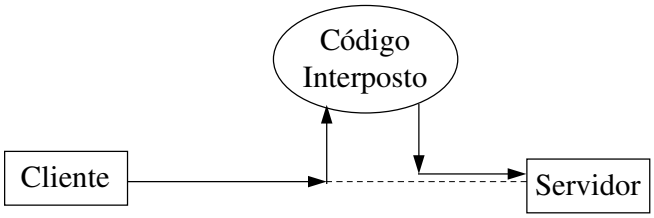

Figura 2.1 - Reflexão como interposição de código

Em uma situação mais complexa pode ser necessário que tanto o módulo cliente como o módulo que oferece o serviço sejam refletidos. Por exemplo em sistemas distribuídos de arquivos tanto a parte do cliente local como o serviço remoto de arquivos deve compreender o protocolo de **redirecionamento** (que por sinal é um termo com a semântica bastante similar a *interposição* ou *filtragem*). Uma aplicação que solicite uma leitura ou escrita em um arquivo remoto, o faz como se o arquivo fosse local. O próprio sistema operacional redireciona o pedido a um sistema de remoto de arquivos. Este por sua vez recebe o pedido e reencaminha o mesmo para o seu próprio sistema local de arquivos. O esquema de interposição de código para esta situação é ilustrada na próxima figura.

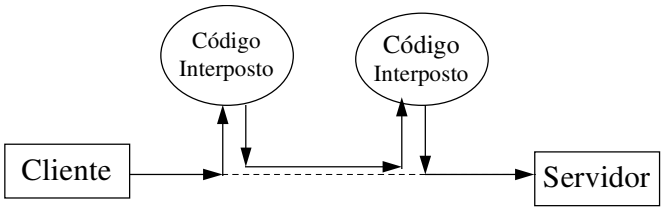

Figura 2.2 - A reflexão utilizada nos dois componentes de um sistema cliente servidor

Observe-se que a semântica local do serviço ou função que ser quer estender ou ajustar precisa ser flexível o suficiente para que não seja necessário reprojetar totalmente o serviço ou utilizar recursos que não ofereçam a transparência desejada. Ainda em relação ao exemplo do serviço remoto de arquivos, se a semântica de endereçamento de uma unidade de disco ou diretório não permitisse que o redirecionamento fosse feito de forma natural (utilizando o nome dos drivers f:, g:, etc., no DOS ou o esquema de montagem de diretórios no UNIX, por exemplo) seria necessário estender o sistema operacional nativo e as aplicações para adotar a possível solução.

Os sistemas operacionais mais modernos já oferecem a reflexão como um atributo do próprio sistema operacional para permitir a ampliação e ajuste de suas funções. Alguns sistemas baseados em micronúcleo (*microkernel*) [Str 93], orientam o projeto de serviços básicos para a execução a nível de usuário e para isso oferecem mecanismos reflexivos eficientes. Um aspecto importante a ser considerado é a facilidade com que se pode combinar as interfaces dos componentes originais e dos componentes refletidos. A figura 2.3 indica de forma genérica a configuração básica de um serviço cliente-servidor salientando a combinação necessária de interfaces.

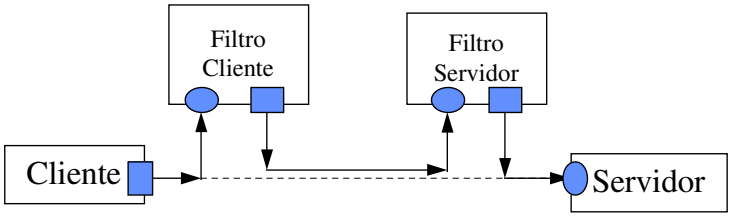

Figura 2.3 - Interfaces necessárias em um esquema reflexivo

Uma outra configuração possível para o ajuste da comunicação de serviços é apresentada na figura 2.4. O mais importante nas duas possibilidades é a transparência da interposição. O objetivo é controlar a comunicação entre cliente e servidor, adicionando qualidades de serviço a esta comunicação, que não são especificamente relacionados com a natureza da comunicação, de forma transparente aos componentes envolvidos.

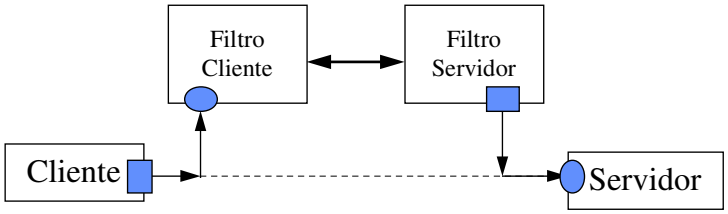

Figura 2.4 - A comunicação entre os filtros cliente e servidor podem ser especializadas

Dentre exemplos de sistemas que podem se beneficiar do modelo de reflexão e que estão sendo bastante considerados em outros trabalhos elencam-se:

- tolerância a falhas baseada em replicação de componentes, técnicas de controle de replicação e segurança;
- sistemas de tempo-real baseado em objetos, cujos componentes nativos podem ou não necessitar de controle temporal;
- sincronização de sistemas multimídia onde, através da reflexão é possível a interposição transparente das diversas propostas e técnicas de sincronização em módulos fonte e destino de mídias;
- tratamento de exceções que podem ser implementados externamente aos componentes básicos filtrando-se a interação do mesmo com o ambiente de execução externa, permitindo a seleção da qualidade deste tratamento.

Dentre as facilidades oferecidas por alguns sistemas operacionais para permitir a implementação eficiente de módulos de interposição, além dos mecanismos existentes em sistemas baseados em micronúcleo, algumas tem merecido destaque em diversas pesquisas:

- abstrações e esquemas de comunicação eficientes como ***named-pipes*** e ***streams*** que permitem a interposição de código e a construção de filtros a nível de núcleo, principalmente no caso de protocolos de comunicação;
- compiladores modernos que já oferecem facilidades para a codificação de módulos de interposição. Embora este esquema esteja sendo bastante pesquisado e desenvolvido, a associação entre o módulo básico e o módulo interposto se dá em tempo de compilação.

O projeto de sistemas reflexivos baseado em compiladores tem se apresentado como uma tendência na construção de sistemas complexos com o projeto orientado a objetos. Esta abordagem será apresentada

na próxima seção. Podemos antecipar, entretanto, que um compromisso entre sistemas configuráveis em que se pudesse selecionar o código a ser interposto e a eficiência de um código compilado como um todo seria bastante atraente.

3. Reflexão em Orientação a Objetos

O paradigma da reflexão ganhou aspectos particulares em sistemas construídos com linguagens orientadas a objetos devido a aptidão natural destas linguagens em facilitar a especificação de objetos com características de outros objetos, mais primitivos, através da *herança*. Com a utilização das propriedades de ***herança*** e ***polimorfismo***, um objeto primitivo (ou **objeto base**) pode ser utilizado para a criação de seu reflexo, que herda, por exemplo a interface do objeto base e seus métodos básicos, e altera os procedimentos de atendimento destes métodos. Neste contexto os objetos refletidos são chamados de **meta-objetos**. A instanciação de um objeto base é acompanhada da instanciação do meta-objeto correspondente.

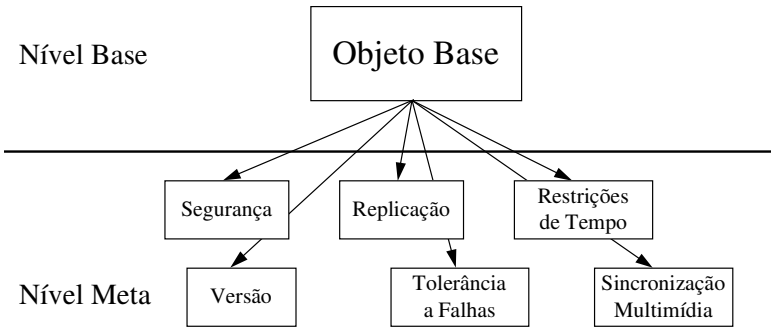

Figura 3.1 - Reflexão utilizando o paradigma de objetos baseado em nível meta e meta-objetos

Todos os objetos base são tratados em um **nível base** e os meta-objetos são tratados em um **nível meta**. Portanto, similarmente ao que foi conceituado na seção anterior, os meta-objetos são responsáveis por modificar, ajustar ou filtrar as mensagens que chegam aos objetos base, ou que são enviadas por estes em resposta a uma solicitação externa. Um objeto base que ofereça serviços genéricos, pode ter estes serviços especializados ou diferenciados por características de segurança, tempo-real ou tolerância a falhas através de meta-objetos adequados (figura 3.1).

Atualmente as vantagens potenciais do uso de reflexão através de meta-objetos são limitadas pela tecnologia dos compiladores. A implementação dos conceitos de meta-objetos em linguagens orientadas a objetos como o C++ não parece ser trivial [Chi 95]. Um exemplo deste tipo de linguagem é o Open-C++, um pré-processador para C++ que já oferece algumas características reflexivas [Chi 95]. Advoga-se que os conceitos de reflexão podem ser empregados apenas para gerar código mais facilmente ou modularmente, antes da compilação. Desta forma, o objeto-base e o meta-objeto seriam ligados em um mesmo módulo, não havendo necessidade de qualquer suporte em tempo de execução para o nível meta.

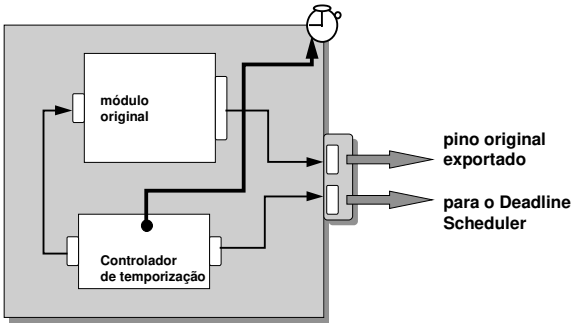

Figura 3.2 - Módulo composto: original + Controlador de execução temporal

Por exemplo, o módulo proposto na primeira fase do trabalho, que incluiria características temporais para funções de planejamento de apresentação podem ser implementados de forma eficiente utilizando uma linguagem de programação que suporte reflexão, como o Open C++. O módulo composto herdaria os métodos de tempo-real e planejamento de apresentação baseados em XOCPN, além dos métodos do modulo base, tornando-se um módulo composto com as características desejadas. Apenas os métodos necessários para a entrada ou saída dos fluxos de mídia teriam visibilidade externa. Assim sendo, quando determinada apresentação estiver sendo configurada ou programada, basta que os objetos fundamentais da apresentação herdem as características dos meta-objetos de controle de sincronização necessários.

O produto obtido desta forma, a princípio, teria características eficientes por ser um componente único, sem a necessidade de troca de mensagens entre objetos. Entretanto, se considerarmos que em sistemas distribuídos existe uma forte necessidade de flexibilidade na operação das aplicações, a solução de uma sistema compilado como um só bloco pode não atender a maioria das eventuais situações. Mesmo assim poderíamos conceber um repositório de meta-objetos com as implementações mais populares de extensões ou críticas em desempenho [Loq 95].

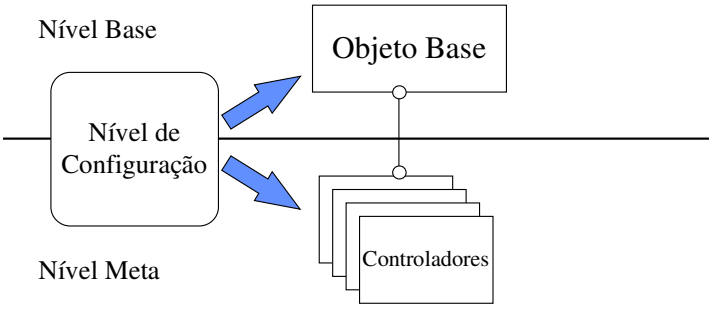

Figura 3.3 - Selecionando meta-objeto adequado através de configuração

Mais uma vez deve existir um compromisso entre eficiência e flexibilidade. Se o modelo de objetos for utilizado, perde-se, em princípio, a modularidade porque o objeto base e o meta-objeto associado vão estar compilados como um só objeto. Seria desejável, entretanto, ter à disposição um repositório de objetos com seus respectivos meta-objetos e selecioná-los quando necessário. A técnica de **ligação dinâmica** também poderia ser usada neste esquema. Uma outra possibilidade é ter uma interface padrão associada ao objeto base e permitir que os meta-objetos sejam instanciados apenas no momento da configuração, como ilustra a figura (3.3). Existem alguns esforços no sentido de disponibilizar este tipo de característica nos compiladores de linguagens de programação reflexiva, mas estas ainda não são realidade.

A troca de mensagens entre o objeto base e o meta-objeto; e destes com outros meta-objetos para realizar as funções esperadas é chamado de **meta-protocolo** (**MOP - *Meta Objetct Protocol***). Em

alguns trabalhos como [Chi 95], [Fab 95] e [Jon 93] este termo também é usado para referência à própria estrutura do compilador. Um MOP não é um protocolo no sentido mais usado do termo, mas está relacionado a uma convenção arbitrária para a interação entre objetos.

Uma técnica adequada para a construção de aplicações utilizando reflexão é a dosagem adequada entre implementar objetos únicos compostos por um objeto base e meta-objetos e a implementação de módulos simples que podem ser interconectados por configuração. Neste esquema, a ligação de objetos base com seus meta-objetos é feita dinamicamente, permitindo a flexibilidade de se ter um conjunto de meta-objetos que passam a atuar na medida em que são necessários.

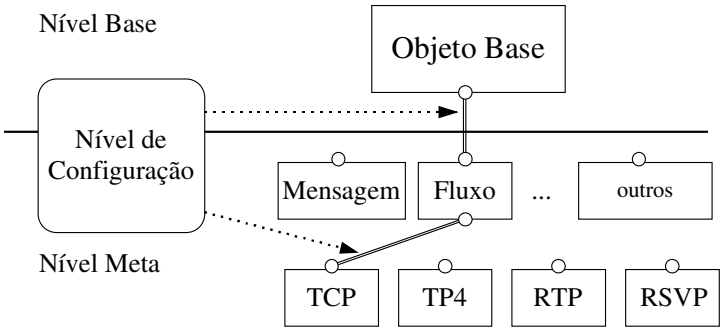

Figura 3.4 - Ambiente multiprotocolo configurável baseado em reflexão

Um exemplo de utilização de reflexão e configuração seria o caso em que determinado objeto base desejando enviar uma mensagem para outro objeto poderia fazê-lo da forma mais adequada selecionando-se os meta-objetos a serem interpostos em uma hierarquia de meta-objetos. Na figura anterior o objeto-base se comunica através de um meta-objeto que implementa o mecanismo de um fluxo contínuo, que por sua vez utiliza o protocolo TCP para o transporte. O conhecimento do objeto-base, entretanto, se limita a uma simples comunicação genérica.

4. Configuração de uma aplicação

Para ilustrar a utilização da metodologia proposta será apresentado o exemplo de uma apresentação multimídia com a exibição de fontes de dados armazenadas, pré-orquestradas como descrito em [Lim 96]. Os módulos de serviço e apresentação farão uso de sincronismo intermídias (não serão considerados os aspectos de sincronização intramídia neste exemplo) empregando o serviço implementado em [Lim 96], que utiliza o método baseado nas especificações do modelo XOCPN (Extended Object Composition Petri Net).

O ambiente distribuído de execução possui um módulo especial, chamado **Configurador Global (CG)**, que tem a função de coordenar as atividades dos módulos envolvidos na aplicação. Mais especificamente, o módulo **CG**, distribui as atividades de coordenação entre os módulos de **Configuração Local (CL),** que tem a propriedade de instanciar, bloquear, destruir e sincronizar módulos na sua estação física de atuação. Este ambiente genérico já foi descrito em [Szt 96].

Os mecanismos disponíveis neste ambiente serão utilizados de forma bastante apropriada para distribuir as **redes de sincronização**. O conjunto das redes de sincronização a serem distribuídas pelas fontes de mídias e o destino é o resultado do planejamento especificado em OCPN (que é expandido em XOCPN) para determinada apresentação. Ao ser entregue para o CL de cada estação envolvida, o *texto* da rede de sincronização é traduzido em comandos de configuração nativos do ambiente. A aplicação alvo é composta de duas fontes de mídias localizadas em estações distintas (vídeo e áudio armazenados). A apresentação destas mídias, segundo um planejamento prévio, será executada em uma terceira estação.

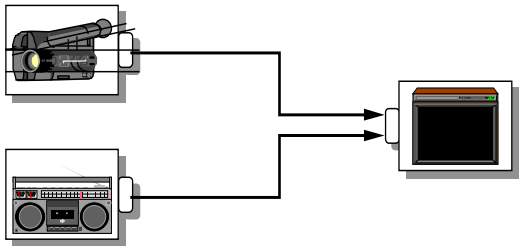

Figura 4.1 - Aplicação com uma fonte de vídeo e áudio independentes

Inicialmente a aplicação deve ser configurada. O código a seguir exemplifica esta configuração, que poderia ser também ser descrita graficamente [Mal 93] e [Wer 95]. Na configuração, os atributos ou parâmetros passados para cada instância de módulo farão com que os objetos e os meta-objetos necessários sejam instanciados de forma conjugada e atômica, para atender os requisitos operacionais (QoS, sincronização temporal, etc.). Se os módulos de uma aplicação representam fontes de som e vídeo, e são pré-orquestrados por OCPN, estes serão automaticamente instanciados em conjunto com os módulos refletidos que implementam e interpretam as redes de XOCPN contendo o planejamento da apresentação.

Primeiramente as estações envolvidas e as interfaces dos módulos são descritas. Observe que toda a aplicação é considerada como uma classe e, portanto, será instanciada como um objeto:

```
STATION audio_station parati.gta.ufrj.br;         // declara as estações físicas
STATION video_station angra.gta.ufrj.br;
STATION presentation_station buzios.gta.ufrj.br;

class Apresentacao_Planejada_Class {

   // Inicialmente são descritas as interfaces (conectores e pinos)

   connector Presentation_IN_Type {   // conector com pinos de entada para
        pin VIDEO in video;            // o fluxo nos pinos já está planejado
        pin AUDIO in audio;  };
   connector Fonte_Video_Type {       // conector com pinos de saida de video
        pin VIDEO out video; };
   connector Fonte_Audio_Type {       // conector com pinos de saida de audio
        pin AUDIO out audio; };
   connector Compress_Video_Type {    // conector com 1 pino de entrada para video
        pin VIDEO in video;            // e 1 pino de saída para vídeo comprimido
        pin COMP_VIDEO out c_video;
        pin XOCPN in plano;  };
   connector Decompress_Video_Type {  // conector com 1 pino de entrada para vídeo
        pin COMP_VIDEO in c_video;     // comprimido e 1 pino de saída para vídeo
        pin VIDEO out video;
        pin XOCPN in plano;  };
   connector Compress_Audio_Type {    // conector com 1 pino de entrada para áudio
        pin AUDIO in audio;            // e 1 pino de saída para audio comprimido
        pin COMP_AUDIO out c_audio;
        pin XOCPN in plano;  };
   connector Decompress_Audio_Type {  // conector com 1 pino de entrada para áudio
        pin COMP_AUDIO in c_audio;     // comprimido e 1 pino de saída para áudio
        pin AUDIO out audio;
        pin XOCPN in plano;  };
   connector XOCPN_Type {              // conector com 1 pino de entrada para
        pin OCPN in espec;             // espeficação OCPN e um pino de saída
```

```
    pin XOCPN out plano;  };        // com o planejamento em XOCPN
conector Planed_Video_Type {
    pin COMP_VIDEO out c_video;};
conector Planed_Audio_Type {
    pin COMP_AUDIO out c_audio;   };
conector Planed_Audio_Video {         // conector que recebe os fluxos planejados
    PAIR connector COMP_VIDEO c_video;
    PAIR connector COMP_AUDIO c_audio;  };
```

Em seguida, cada módulo básico e meta-módulos são configurados de forma composta:

```
class Planned_Video_Source { // módulo composto fonte de vídeo
    conector Planned_Video_Type video_out;
    class Video_Source_Class {          // módulo com fonte de vídeo sem compressão
       connector Fonte_Video_Type vs;
       extern code "video_source";
    } VS;
    class Video_Coder_Class {           // módulo que comprime o vídeo
       connector Compress_Video_Type vcod;
       extern code C "video_coder";
    } VCOD;
    class Controler_Class {             // módulo de controle de planejamento XOCPN
       connector XOCPN_Type ctl;
       extern code C "xocpn_controler";
    } V_CONTR;

    instantiate VS; instantiate VCOD; instantiate V_CONTR;

    link VS.vs.video        VCOD.vcod.video  ;
    link V_CONTR.clt.plano  VCOD.vcod.plano  ;

    // Exporta as portas que terão visibilidade externa

    bind video_out.c_video  VCOD.vcod.c_video;
} planned_video;
```

Para cada módulo, depois de descritos os componentes, as conexões internas são estabelecidas e as interfaces que terão visibilidade externa são exportadas.

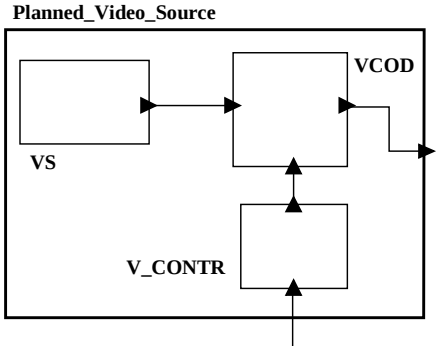

Figura 4.2 - Módulo composto fonte de vídeo com planejamento XOCPN

```
class Planned_Audio_Source {
     conector Planned_Audio_Type audio_out;

     class Audio_Source_Class {          // módulo fonte de áudio sem compressão
         connector Fonte_Audio_Type as;
         extern code "audio_source";
     } AS;
     class Audio_Coder_Class {           // módulo de codificação de áudio
         connector Compress_Audio_Type acod;
         extern code C "audio_coder";
     } ACOD;
     class Controler_Class {             // módulo de controle de planejamento XOCPN
         connector XOCPN_Type ctl;
         extern code C "xocpn_controler";
     } A_CONTR;

     instantiate VS; instantiate VCOD;   instantiate A_CONTR;

     link AS.as.audio        ACOD.acod.audio  ;
     link A_CONTR.ctl.plano  ACOD.acod.plano  ;

     // Exporta as portas que terão visibilidade externa

     bind audio_out.c_audio  ACOD.acod.c_audio;
} planned_audio;
```

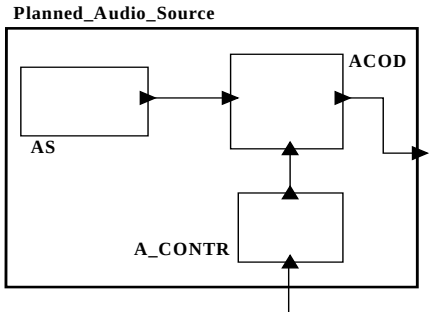

Figura 4.3 - Módulo composto fonte de áudio com planejamento XOCPN

```
class Planned_Presentation {
     connector Planned_Audio_Video audio_video_input;

     class Presentation_Tool {           // módulo de apresentação de áudio e vídeo
        connector Presentation_IN_Type p_input;
        extern code C “presentation_tool”;
     } PTOOL;
     class Video_Decoder_Class {      // módulo decodificador de vídeo
        connector Decompress_Video_Type vdecod;
        extern code C "video_decoder";
     } VDECOD;
     class Audio_Decoder_Class {      // módulo decodificador de áudio
        connector Decompress_Audio_Type adecod;
        extern code C "audio_decoder";
```

```
    } ADECOD;
    class Controler_Class {             // módulo controlador de planejamento XOCPN
       connector XOCPN_Type ctl;
       extern code C "xocpn_controler";
    } P_CONTR;

    instantiate VDECOD; instantiate ADECOD; instantiate P_CONTR; instantiate PTOOL;

    link P_CONTR.clt.plano   ADECOD.adecod.plano  ;
    link P_CONTR.clt.plano   VDECOD.vdecod.plano  ;

    link ADECOD.adecod.audio PTOOL.p_input.audio  ;
    link VDECOD.vdecod.video PTOOL.p_input.video  ;

    // Exporta as portas que terão vizibilidade externa
    bind audio_video_input.c_video VDECOD.video_in.c_video;
    bind audio_video_inout.c_audio ADECOD.audio_in.c_audio;
} planned_presentation;
```

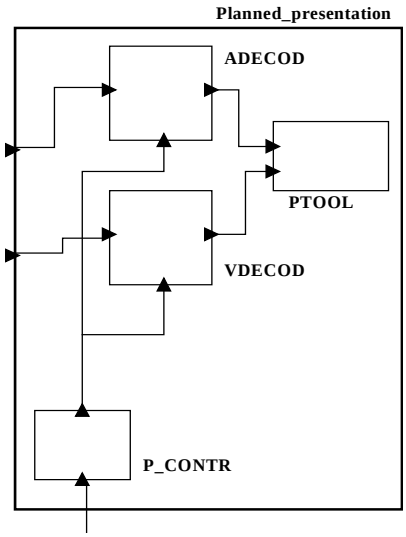

Figura 4.4 - Módulo composto apreentação com planejamento XOCPN

Finalmente os módulos compostos são interconectados...

```
// Definidas as classes, agora devem ser instanciados os módulos
   instantiate planned_audio at audio_station;
   instantiate planned_video at video_station;
   instantiate planned_presentation at presentation_station;

// ... e as portas devem ser conectadas
   link planned_audio.audio_out.c_audio planned_presentation.audio_in.c_audio;
   link planned_video.video_out.c_video planned_presentation.video_in.c_video;

   echo "Fim da Classe Apresentação_Planejada_Class";   };
```

... e a aplicação é instanciada e executada.

```
class Apresentacao_Planejada_Class MyPresentation;
instantiate MyPresentation;
```

Desta forma, podemos utilizar linguagens que suportam reflexão através de meta-objetos para construir os módulos da aplicação e ainda ter a flexibilidade de executar uma aplicação construída por configuração, selecionando os mecanismos mais apropriados para cada apresentação.

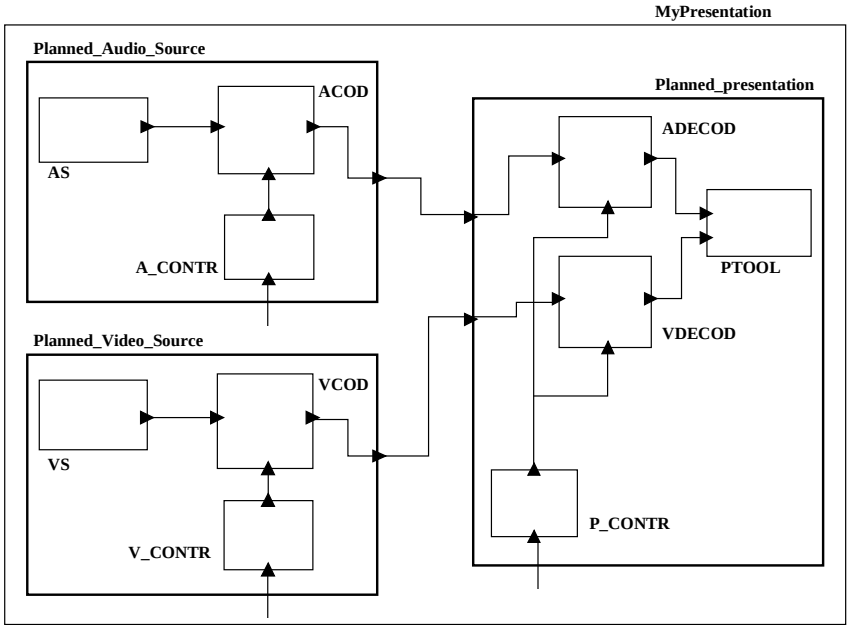

Figura 4.6 - Aplicação configurada pela conexão dos 3 módulos compostos

5. Implantando Reflexão em um ambiente baseado em CORBA

A implantação dos conceitos de objetos com reflexão em um ambiente baseado em CORBA (*Common Object Resource Brocker Architecture*) é bastante direto. Não é difícil admitir que, da mesma forma que a comunicação entre meta-objetos poderia ser feita de forma tradicional através de RPC (Remote Procedure Call), esta mesma comunicação poderia ser especificada de uma forma abstrata através de simples ativação de um método, sendo que a própria infra-estrutura do CORBA seria encarregada de efetivar a comunicação necessária. De uma forma simplista podemos considerar o ambiente CORBA como um suporte de comunicação.

Os mecanismos apresentados até aqui se enquadram ajustadamente na implementação de extensões de objetos de usuário, objetos básicos e da própria funcionalidade da arquitetura CORBA. Tanto os ORBs como os BOAs (Basic Object Adapter), componentes principais da arquitetura CORBA[2], foram concebidos para aceitar ampliação de suas funções, como a adição de novos protocolos e semânticas de comunicação. Com a técnica de reflexão seria oportuna a implementação de meta-objetos que poderiam residir nos BOAs e se encarregariam da sincronização de fluxos de mídia segundo as propostas mais utilizadas nestes sistemas. Por exemplo, como proposto em [Fra 96], os meta-objetos

[2] Não é objetivo deste trabalho apresentar ou detalhar os conceitos sobre CORBA. Admite-se que o assunto pode ser revisto em maior profundidade nas referências bibliorágicas.

que controlam o QoS em determinado serviço de mídia poderia utilizar polimorfismo para se ajustar em tempo de execução aos atributos enviados como parte da mensagem de ativação de método.

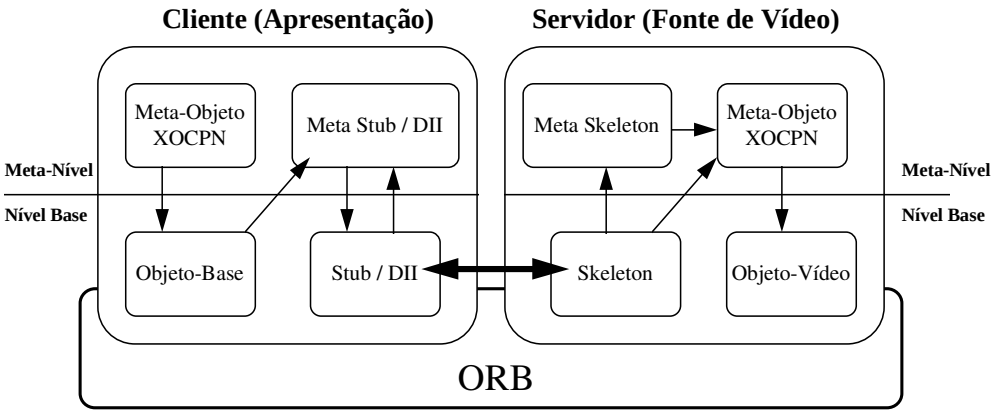

Figura 5.1 - Arquitetura CORBA com nível meta.

A especificação das interfaces com o IDL (*Interface Definition Language*), em um ambiente reflexivo, deve ser estendida para oferecer a possibilidade de se definir as interfaces dos meta-objetos, chamados de ***meta-stubs*** e ***meta-skeletons*** para a implementação e manipulação das aplicações de usuário e objetos de serviço respectivamente [Col 96]. Assim, na arquitetura CORBA passa a existir um nível meta que intermedia as requisições e respostas dos objetos base.

6. Conclusões

Foi apresentado um mecanismo para o projeto e implementação de aplicações mutimídia baseado em reflexão. A utilização deste mecanismo permite a concepção uniforme e reutilizável de módulos necessários para esta categoria de aplicação desde os níveis mais básicos, como a sincronização de mídias, até o nível mais próximo do usuário.

Várias propostas para a utilização de reflexão em ambientes orientados a objetos e mais especificamente baseados em CORBA tem sido apresentadas e com algum êxito a exemplo de [Fab 95], [Rot 96] e [Str 93], principalmente para solucionar o problema de sistemas de tempo-real e tolerância a falhas baseados em replicação de componentes. O controle de tolerância a falhas, por exemplo, é todo construído no nível meta de forma transparente à programação de módulos.

A utilização de CORBA como plataforma de suporte para a construção de aplicações mutimídia foi proposta em [Col 96] onde a interposição de código foi utilizada para adicionar qualidade de serviço através da abstração chamada *MediaPipes*, mas o paradigma de reflexão não foi abordado. A utilização desta técnica poderia facilitar futuras implementações.

7. Referências Bibliografia

[Aky 96] Akyildiz, I. F, Yen Wei, “Multimedia Group Snchronization Protocols for Integrated Services Networks”, IEEE JSAC, Vol 14, No 1, pp. 162-173, janeiro de 1996.

[Amo 92] Amorim, R.E.H, "Compartilhamento de Dados em Sistemas de Controle Tempo-Real Distribuído", dissertação de mestrado, DEE / PUC-RJ, agosto de 1992.

[Bal 96] Blakowski, G. e Steinmetz, R., “A Media Synchronization Survey: Reference Model, Specification, and Case Studies, IEEE JSAC, Vol 14, No 1, pág. 5-35, janeiro de 1996.

[Bis 96] Bishop, J. e Faria, R, “Connectors in Configuration Programming Languages: are They Necessary?”, IEEE, 1996.

[Bri 95] Brito, O. F. G. e Loques, O. G, "Técnica de Replicação de Módulos no Ambiente RIO", VI Simpósio em

Computadores Tolerantes a Falhas, Canela, RS, julho de 1995.

[Chi 95] Chiba, S. “A Metaobject Protocol for C++”, OOPSLA 95, 10th Conference on Objetc-Oriented Programming Systems, Languages and Applications, Austin, Texas, outubro de 1995.

[Col 96] Colcher, S. e Soares, L.F.G, “Modelo de Objetos Baseado no CORBA para Sistemas Hipermídia Abertos com Garantias de Sincronização”, XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza, maio de 1996

[Cri 90] Cristian, F., "Synchronous Atomic Broadcast for Redundant Broadcast Cannels", The Journal of Real-Time Systems, Vol. 2, pág. 195-212, 1990.

[Fab 95] Fabre, J.C.; Perennou, T. e Blain, L., “Meta-Objetc Protocols for Implementing Reliable and Secure Distributed Applications”, LAAS REPORT 95037, CNRS, fevereiro de 1995.

[Fra 96] Fraga, J., Farines, J., Furtado, O., Siqueira, F., “Programação de Aplicações Distribuídas Tempo-Real em Sistemas Abertos, XXIII Seminário Integrado de Software e Hardware, Recife , agosto de 1995.

[Geo 96] Georganas, N. D., Steinmetz, R., Nakagawa, T., “Synchronization Issues in Multimedia Communcations”, Guest Editorial, IEEE JSAC, Vol 14, Nº 1, pág. 1-4, janeiro de 1996.

[Gib 91] Gibbs S., “Composite Multimedia and Active Objects”, OOPSLA’91, SIGPLAN Notices, Vol. 26, Nº 11, pág. 97-112, novembro de 1991.

[Jon 93] Jones, M. B., “Interposition Agents: Transparently Interposing User Code at the System Interface”, ACM SIGOPS, 1993.

[Lam 78] Lamport, L, "Time, Clocks, ans the Ordering of Events in a Distributed System", Communication of the ACM, Vol. 21, Num. 7, julho de 1978.

[Lim 96] Lima, R. M. e Duarte, O. C. M. B, “Comunicações Sincronizada de Dados Multimídias Pré-Orquestrados”, XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza, maio de 1996.

[Loq 95] Loques, O., Leite, J.C.B., Bortafogo, R. e Lobosco, M., “Configuração, Reflexão e CORBA: Algumas Considerações”, LASJ, 1995.

[Mal 93] Malucelli, V. V, "RIO - Linguagem de Configuração - Tolerância à Falhas", Nota Interna, DEE / PUC-RJ, fevereiro de 1993.

[OMG 91] _____, "The Common Object Request Brocker: Architecture and Specification", OMG - Document Number 91.12.1, Revision 1.1, dezembro de 1991.

[Per 96] Pérez-Luque, M. J e Little, T. D. C., “A Temporal Reference Framework for Multimedia Synchronization”, IEEE JSAC, Vol 14, No 1, pp. 36-51, janeiro de 1996.

[Rot 96] Rothermmel, K. e Helbig, T., “Clock Hierarchies: An Abstraction for Gouping and Controlling Media Streams”, IEEE JSAC, Vol 14, No 1, pp. 174-184, janeiro de 1996.

[Rub 96] Rubinstein, M. G. e Duarte, O. C. M. B., “Definição de um Serviço de Sincronização Contínua para Aplicações Multimídias”, XIV Simpósio Brasileiro de Redes de Comput. Fortaleza, maio de 1996.

[Str 93] Strouds, R., “Transparency and Reflection in Distributed Systems”, Operationg Systems Review, Vol 27, Num 2, abril de 1996.

[Szt 93] Sztajnberg, A. e Loques, O. G., "O Sistema de Comunicação Multi-Protocolo do Ambiente RIO", V Simpósio de Computadores Tolerante a Falhas, São José dos Campos, outubro de 1993.

[Szt 94] Sztajnberg, A, Loques, O. G. e Leite, J., "Implementação e Teste do Sistema de Comunicação do Ambiente RIO", XII Simpósio Brasileiro de Redes de Computadores, Curitiba , maio de 1994.

[Szt 96] Sztajnberg, A., “Proposta de Extensão de Ambientes Distribuídos Configuráveis para Aplicações com Múltiplos Fluxos Sincronizados”, relatório de curso, COPPE/UFRJ, maio de 1996.

[Wer 95] Werner, J.A.V., "Metodologia e Suporte para Sistemas Distribuídos Configuráveis", dissertação de mestrado, DEE / PUC-RJ, março de 1995.