Configuração e Reflexão Computacional em Aplicações Multimídia Distribuídas

Alexandre Sztajnberg[1]
(alexszt@ime.uerj.br)
IME/UERJ e GTA/COPPE/UFRJ

Orlando Loques[2]
(loques@caa.uff.br)
CAA/UFF

Resumo

Neste trabalho são avaliados dois paradigmas que tem se apresentado como solução viável para a programação modular e flexível de sistemas: configuração e reflexão computacional. Os conceitos associados a cada modelo são discutidos e as situações onde existe melhor adequação de cada um deles são levantadas. Os dois paradigmas têm em comum a proposta de oferecer mecanismos para o desenvolvimento de sistemas com um número arbitrário de requisitos funcionais e não-funcionais. A adequação destes paradigmas à construção de grandes sistemas, com ênfase em sistemas multimídia distribuídos, é discutida. Com base neste estudo, as fronteiras da vocação das técnicas relativas à configuração e reflexão são delineadas, permitindo que se conclua que estas podem ser usadas de forma complementar e aplicadas em um nível de granularidade diferente.

Abstract

This paper presents and evaluates two paradigms that have been proposed as solutions to provide mechanisms for modular and flexible distributed system programming: configuration and computational reflection. The concepts are discussed, and the situations were the use of each of them is best suitable are pointed out. Both paradigms, lead to mechanisms that intend to facilitate the accomplishment of functional and non-functional application requirements. The fitness of these paradigms for the construction of large systems, with focus on multimedia distributed applications, is discussed. Based on this study, some comparisons between the configuration and reflection techniques is performed. The main conclusion is that these techniques can be used in a complementary way and be applied at different granularity levels.

1. Introdução

A implementação de aplicações distribuídas ainda é bastante dependente do ambiente computacional disponível. No caso ideal este ambiente deveria oferecer serviços que facilitem a programação de grandes sistemas distribuídos. Uma das dificuldades básicas para a concepção de sistemas distribuídos está na ampla faixa de requisitos a serem atendidos, principalmente no que se refere a requisitos funcionais e não-funcionais de cada módulo componente destes sistemas. A situação é agravada se consideradas as possibilidades de combinações destes módulos (cada qual com seus requisitos locais) e o estilo desejado para a interconexão dos mesmos.

Particularmente, com relação a serviços de suporte básico os mecanismos de RPC - *Remote Procedure Call* são unanimemente considerados como necessários em qualquer plataforma para desenvolvimento de aplicações distribuídas, segundo o modelo cliente-servidor. Entretanto, a realização de sistemas de grande porte passa por outras fases de concepção em que a programação de módulos individuais deveria ser flexibilizada. O uso de abstrações que tornem a programação mais segura e rápida, como por exemplo a programação orientada a objetos ou ambientes que permitam a composição de módulos através de configuração, é bastante adequado.

A categoria de sistemas multimídia possui uma ampla faixa de requisitos funcionais e operacionais a serem contemplados, como por exemplo, transporte de mídias com características isócronas, sincronização temporal, gerenciamento de múltiplos fluxos que compõem as apresentações, compressão e descompressão dos fluxos de mídias, especificação das apresentações, etc. Faz-se, portanto, necessário o suporte de mecanismos para facilitar a tarefa de produção da parte funcional destes sistemas, bem como para expressar e manter a qualidade de todos os requisitos operacionais demandados (chamados genericamente de QoS - *Quality of Service*).

Neste trabalho são avaliadas duas metodologias para a construção de aplicações distribuídas compostas por um número arbitrário de componentes: configuração e reflexão computacional. O objetivo comum da aplicação destas metodologias em sistemas distribuídos é oferecer um grau de abstração e flexibilidade ao projetista de forma que

---

[1] Departamento de Informática e Ciências da Computação / Instituto de Matemática e Estatística / *Universidade do Estado do Rio de Janeiro*
Grupo de Teleinformática e Automação / Coord. dos Programas de Pós-Graduação em Engenharia *Universidade do Federal do Rio de Janeiro*

[2] Pós-Graduação em Computação Aplicada e Automação / *Universidade Federal Fluminense*

a preocupação central durante o projeto esteja restrita à aplicação em si, liberando o projetista de problemas com a interligação dos módulos componentes da aplicação ou com a adaptação de módulos semelhantes a situações operacionais específicas. Tanto a configuração quanto a reflexão serão investigadas em sua adequação para aplicações multimídia distribuídas.

Na seqüência deste artigo serão introduzidos os fundamentos sobre configuração, composição de módulos e reflexão computacional. Especial atenção será dada à reflexão computacional e a adequação deste conceito a linguagens orientadas a objetos. Em seguida, as duas técnicas consideradas são avaliadas, tendo-se como base uma aplicação multimídia. Como resultado da utilização destas duas técnicas algumas conclusões são evidenciadas.

2. Paradigma de Configuração

O conceito genérico de configuração pressupõe que existem componentes de software dos quais se pode dispor para construir um sistema ou aplicação com características particulares através da interconexão destes componentes. Ao se estabelecer uma configuração é possível a seleção de atributos específicos, dentre um conjunto de opções disponíveis, para cada componente [Sha 95 e Mag 96].

Ambientes distribuídos com suporte à configuração facilitam o projeto de aplicações distribuídas complexas. O projetista pode atender mais facilmente requisitos funcionais da aplicação utilizando módulos básicos já desenvolvidos e depurados, e respeitar requisitos operacionais selecionando modos específicos de operação e interligação para cada um deles, incluindo o protocolo de comunicação. O resultado é esquematizado na figura 2.1. Dentre os requisitos operacionais que podem ser eventualmente necessários destacam-se a tolerância a falhas, que pode ser configurada seletivamente para cada módulo [Loq 97] e aplicações com módulos sincronizados no tempo e com fortes exigências na qualidade dos serviços (QoS) contratados com os subsistemas de suporte e comunicação, tipicamente aplicações multimídias [Aky 96, Bla 96, Geo 96 e Flo 96].

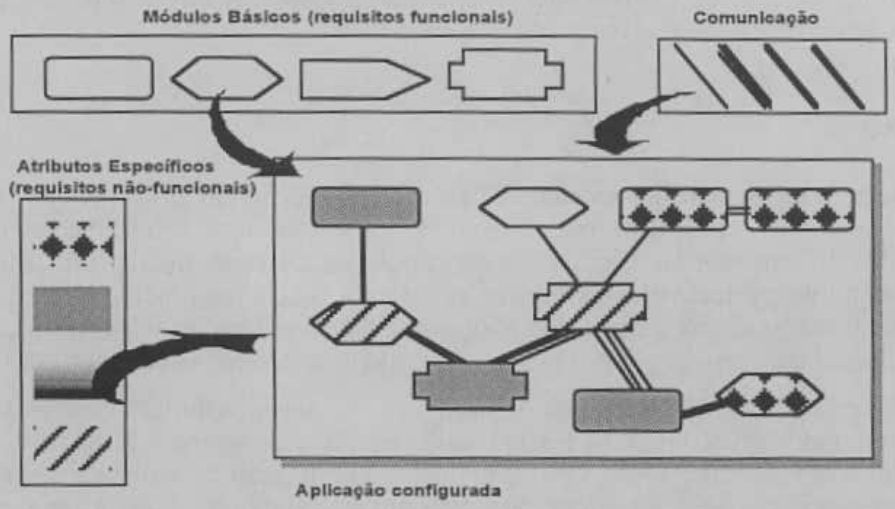

Figura 2.1 - suporte para configuração de aplicações distribuídas

Um mecanismo utilizado para permitir a ligação de módulos é genericamente chamado de **conector** [Bis 96]. O conector é uma abstração utilizada no nível de configuração que oferece para o programador os mecanismos de comunicação nativos dos sistemas operacionais de uma maneira simplificada e padronizada, sem a necessidade de cuidados com detalhes específicos. Os módulos são projetados de forma que as interfaces pelas quais eles podem se comunicar com outros módulos sejam completamente definida pelos conectores. Através de um comando de configuração, conectores de um ou mais módulos podem ser interligados. A figura 2.2 ilustra este mecanismo.

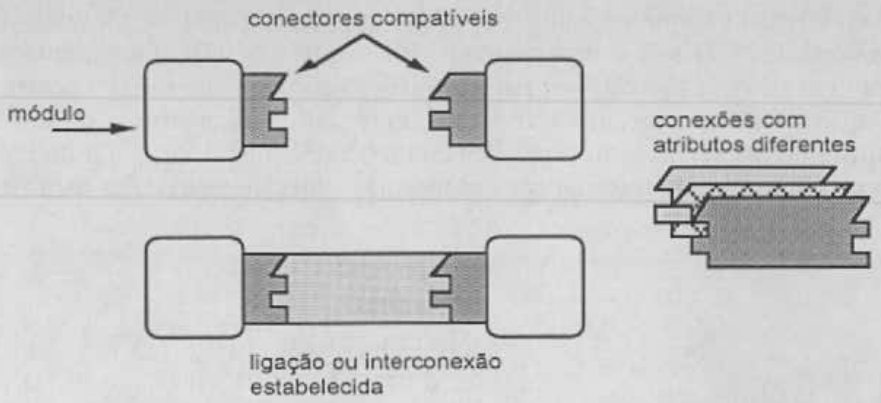

Figura 2.2 - módulos independentes e módulos interconectados

Uma das vantagens de se explicitar a conexão entre dois módulos é a clareza na programação dos requisitos funcionais e não-funcionais, feitos separadamente [Bis 96 e Sha 95]. Na figura 2.2, isto está esquematizado pela seleção de um dos atributos disponíveis para a conexão. Além disso várias outras características podem ser associadas à ação de se interligar um par de conectores:

- os tipos e estruturas de dados que são suportados pelos conectores podem ser verificados durante a conexão, evitando-se que conectores de tipos diferentes e portanto que transportam dados (ou mensagens) diferentes possam ser interconectados. Além disto, módulos com conectores de mesmo tipo podem ser interligados automaticamente;
- a semântica e o protocolo utilizados na comunicação através desta conexão pode ser administrada de forma mais flexível se esta comunicação é facilmente identificada. Por exemplo, no caso de uma conexão de módulos residentes em um mesmo nó, a comunicação pode, em um nível maior de detalhe, ser feita através de *sockets*, *pipes*, *streams* ou memória compartilhada. No caso da comunicação de módulos distribuídos através da Internet, pode-se optar por um protocolo como o TCP, UDP ou RTP. O ideal é que o ambiente permitia a seleção do mecanismo a ser utilizado e também possa sugerir a melhor opção.

Para aplicações multimídia distribuídas a possibilidade de se contratar os parâmetros de QoS na conexão entre os diversos módulos, necessários para o funcionamento adequado do sistema, é especialmente interessante. Em [Flo 96] é proposto um conjunto de abstrações para especificar, monitorar e garantir os parâmetros do QoS. Estes parâmetros são especificados através de uma linguagem chamada QuAL e monitorados externamente ao programa que implementa os módulos. Neste caso, o ambiente que oferece o suporte aos conectores também se encarrega de gerenciar a qualidade dos parâmetros.

Ambientes de configuração para composição de módulos são suportados por conceitos como **linguagens de configuração** (CPL - *Configuration Programming Language*) e **linguagens de descrição de arquiteturas** (ADL - *Architectural Description Language*) [Bis 96] que traduzem roteiros de configuração (*scripts*) em comandos primitivos do ambiente alvo. A utilização de ambientes gráficos [Wer 95] como interface para as linguagens de configuração permite que os sistemas sejam descritos de forma intuitiva e visual. As linguagens de configuração possuem características interessantes tais como:

- possibilitar a construção hierárquica de componentes (composição);
- facilitar reutilização de módulos e sistemas que já foram compostos em algum momento;
- facilitar a verificação de propriedades formais como a ausência de bloqueio, terminação e corretude (a exemplo de [Mag 96] que mapeia π-*calculus* para conectores) e propriedades operacionais (como por exemplo a coerência e viabilidade dos parâmetros de QoS solicitados por cada módulo de um sistema multimídia).

Em contraponto às vantagens do conceito de configuração, um dos desafios conhecidos para realização de um ambiente com esquema de conectores é a dificuldade em implementá-lo com um contorno e estruturas bem definidas dentro do código de um módulo [Bis 96]. A implementação tradicional de conectores dentro dos módulos, que em princípio pode ser feita em qualquer linguagem de programação, apresenta-se de forma difusa no código. Este problema tem sido contornado através da geração de esqueletos de código pelas próprias linguagens de configuração.

3. Conceitos Básicos sobre Reflexão

A idéia central do paradigma de reflexão não é exatamente nova. Entretanto, mecanismos atuais de alto nível tornam o esquema de reflexão um aliado útil na adição de características operacionais ou não-funcionais 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 95 e Lis 95]. Um módulo que utiliza reflexão é composto pelo **componente *base*** e um **componente *reflexo***. O componente reflexo possui uma interface para o componente básico (que pode ter a mesma assinatura), entretanto não possui o mesmo comportamento. Ele funciona como um filtro, interceptando as informações ou invocações que deveriam ir diretamente ao componente básico, podendo alterar ou ajustar as características das mesmas, antes de repassá-las para este (figura 3.1).

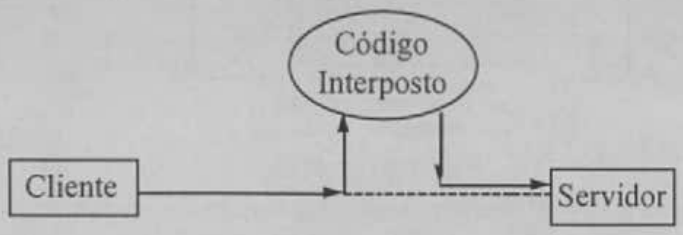

Figura 3.1 - Reflexão como interposição de código

Em situações mais complexas 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 devem compreender o protocolo de *redirecionamento*. Uma aplicação que solicite uma leitura ou escrita em um arquivo remoto, o faz como se o arquivo fosse local. O próprio sistema operacional intercepta e redireciona o pedido para o sistema 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 figura 3.2. Dentre exemplos de aplicações que podem se beneficiar do modelo de reflexão e que estão sendo bastante considerados em alguns trabalhos elencam-se: tolerância a falhas, sistemas de tempo-real, sincronização de sistemas multimídia e esquemas para tratamento de exceções.

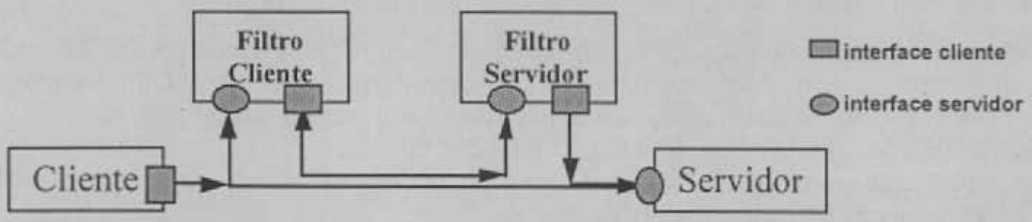

Figura 3.2 - Interfaces necessárias em um esquema reflexivo

Alguns sistemas operacionais modernos oferecem a reflexão como característica própria 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. Além da facilidade com que se pode combinar as interfaces dos componentes originais e dos componentes refletidos nestes sistemas, alguns mecanismos, como *named-pipes* e *streams*, são oferecidos para permitir a implementação eficiente de módulos de interposição e a construção de filtros a nível de núcleo, principalmente no caso de protocolos de comunicação.

Uma outra opção para possibilitar a utilização dos conceitos de reflexão computacional é a inclusão destes no nível de linguagem de programação. Algumas implementações têm se concentrado em extensões especiais de compiladores e linguagens orientadas a objetos, como o C++ e Smaltalk. Esta abordagem será explorada na próxima seção.

3.1 Reflexão Computacional e 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 de herança 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 dos métodos necessários para dotar o objeto-base com as novas funcionalidades. Deve ficar claro, entretanto, que o objeto reflexo não depende apenas de herança para ser implementado: são necessários mecanismos que permitam a interceptação das mensagens, estabelecendo uma conexão entre a instância do objeto-base e seu reflexo, e que tornem disponíveis para o objeto-reflexo as

informações relevantes do objeto-base. Neste contexto a reflexão é dita computacional ou comportamental [Lis 95] e os objetos refletidos são chamados de **meta-objetos**. A instanciação de um objeto-base é acompanhada da instanciação do meta-objeto correspondente (figura 3.3).

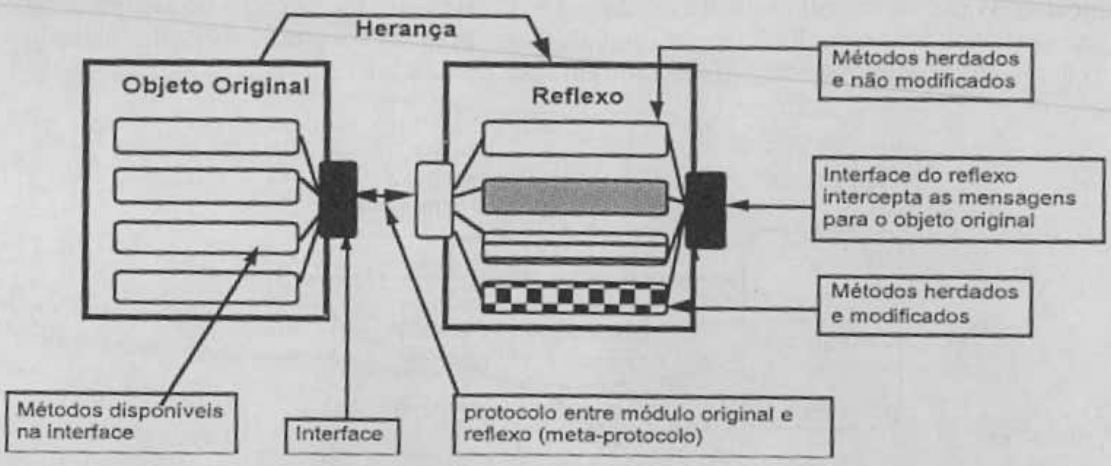

Figura 3.3 - Objeto original e o objeto refletido

Todos os objetos-base são tratados em um **nível base** que cuida da computação externa da aplicação [Lis 95] e os meta-objetos são tratados em um **meta-nível** que opera a computação interna da aplicação. Similarmente ao descrito na seção anterior, os meta-objetos são responsáveis por modificar, ajustar ou filtrar a reação do objeto-base em relação às mensagens que chegam, 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 3.4).

Figura 3.4 - Reflexão utilizada para adicionar características não-funcionais

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 oferece características reflexivas [Chi 95]. Advoga-se que os conceitos de reflexão são empregados apenas para a geração de 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 suporte em tempo de execução para o meta-nível.

Entretanto, se considerarmos que em sistemas distribuídos existe uma forte necessidade de flexibilidade na operação das aplicações, a solução de um 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 96].

Uma outra proposta de implementação utiliza a orientação de objetos e reflexão com o suporte de um executivo que gerencia a instanciação de objetos-base, meta-objetos e administra a coerência dos sistemas gerados a partir destes elementos [Cam 96, Zim 96].

Comentário

Deve existir um compromisso entre eficiência e flexibilidade. Se uma linguagem orientada a objetos for utilizada, perde-se, em princípio, a modularidade porque o objeto-base e o meta-objeto associado vão estar compilados como um só objeto. Seria desejável 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 interface padrão associada ao objeto-base permitiria que os meta-objetos fossem selecionados e instanciados apenas no momento da execução, como ilustra a figura 3.5. Existem alguns esforços no sentido de disponibilizar este tipo de característica nos compiladores de linguagens de programação reflexiva, mas estas ainda não são realidade, pois demandariam uma abordagem de configuração associada à reflexão.

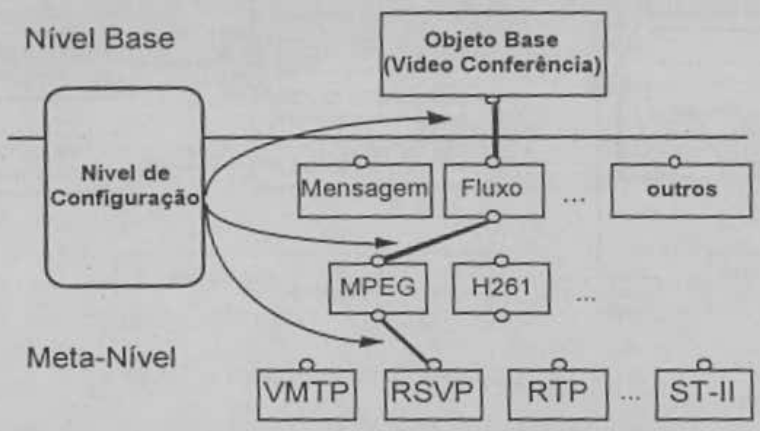

Figura 3.5 - Ambiente multiprotocolo configurável baseado em reflexão

Um exemplo de utilização de reflexão por configuração de meta-objetos seria o caso de uma aplicação multimídia, como a vídeo conferência, onde uma série de parâmetros indicando os requisitos de QoS determina a seleção adequada dos meta-objetos a serem interpostos em uma hierarquia de meta-objetos. Na figura anterior o objeto-base, a aplicação de vídeo-conferência utiliza um meta-objeto que implementa o mecanismo de fluxo contínuo compactado para realizar a comunicação, que por sua vez utiliza o protocolo RSVP para o transporte. O conhecimento do objeto-base, entretanto, se limita a uma simples comunicação genérica.

Para o programador, as vantagens da reflexão computacional estão na transparência das novas funcionalidades agregadas por meta-objetos e no conforto sintático. Observe no trecho de código seguinte [Chi 95] que na programação sem reflexão, o objeto *Node* deve herdar as características da classe *PersistentObjetct* para se tornar persistente, enquanto que na programação com reflexão a persistência é implementada como extensão da linguagem C++.

```
class Node: public PersistentObject
{
   public:
      Node *next;
      doublevalue;
};
```

```
persistent class Node
{
   public:
      Node *next;
      doublevalue;
};
```

4. Aplicações Multimídia Distribuídas

O modelo de referência clássico [Bla 96] para aplicações multimídia contempla propostas para solução dos diversos problemas de aplicações multimídia. Este modelo, dividido em quatro níveis (Especificação, Objeto, Fluxo e Mídia) aborda pontos como a descrição de apresentações, serviços de sincronização intermídia e intramídia, garantias da qualidade esperada destes serviços (QoS), etc. Naturalmente, não fazem parte do modelo soluções específicas para a implementação de serviços básicos e intermediários para o mesmo. Assim sendo, variações de mecanismos de suporte em torno deste modelo são freqüentemente adotadas [Gib 91, Bla 96, Geo 96 e Rot 96], visando principalmente adaptar parte do modelo a algum requisito específico, como por exemplo a orquestração de apresentações. Desta forma, tanto o modelo, mesmo sendo bastante abrangente, como os mecanismos de implementação para sistemas multimídia precisam ser flexíveis para acompanhar a constante evolução tecnológica e permitir o atendimento de novos requisitos.

Mecanismos baseados em configuração como propostos em [Gib 91, Rot 96, e Szt 96] procuram oferecer uma parte dos serviços necessários para a implementação de aplicações multimídia distribuídas através de composição de módulos. Vários outros projetos tem como objetivo encontrar soluções para oferecer serviços, ferramentas ou abstrações que facilitem a especificação de cenários de sincronização em apresentações mutimídia. Além disso,

como abordado na seção 2, existem várias propostas para a especificação dos parâmetros de QoS exigidos pela aplicação que têm influência em vários outros subsistemas, como por exemplo na comunicação.

Nas próximas seções são avaliados esquemas de implementação, baseados em configuração e reflexão computacional, que permitem a construção uniforme dos serviços, desde os mais básicos, até os mais abstratos, necessários em sistemas multimídia. Atenção especial será dada na concepção de um ambiente que favoreça a construção de aplicações multimídia pré-orquestradas.

5. Aplicação Exemplo

Para fundamentar a utilização dos conceitos apresentados, na área de multimídia distribuída, será apresentado um exemplo de uma aplicação com a exibição de fontes de dados armazenadas, pré-orquestradas, com *cenários temporais determinísticos* onde as relações e interações temporais são completamente e antecipadamente definidas, como descrito em [Lim 96]. Os módulos de serviço e apresentação farão uso de sincronismo intermídias e será empregando o serviço implementado em [Lim 96], que utiliza o método baseado nas especificações do modelo XOCPN (*Extended Object Composition Petri Net*). Neste método as restrições temporais entre os fluxos de mídia são descritas através de uma Rede de Petri do tipo OCPN [Lit 90]. A partir do OCPN, são derivadas duas redes XOCPN que carregam as informações de sincronismo e orquestração para as fontes de mídia e para o destino destas mídias, onde deverá ocorrer a apresentação.

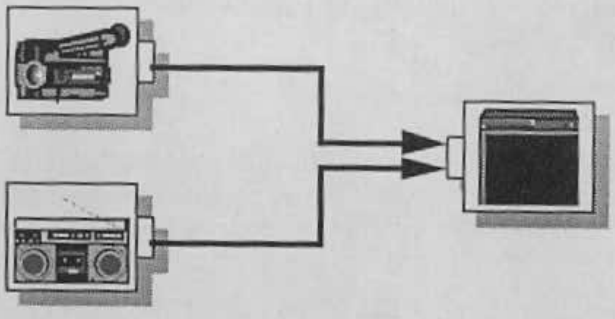

Figura 5.1 - Aplicação com uma fonte de vídeo e áudio independentes

A aplicação alvo é composta de duas fontes de mídias, vídeo e áudio, localizadas em estações distintas e pré-armazenadas em algum meio físico. A apresentação destas mídias, segundo um planejamento prévio, será executada em uma terceira estação. Nas próximas subseções dois experimentos diferentes, um explorando a abordagem da configuração e outro da reflexão, implementarão esta mesma aplicação.

5.1 Construindo a aplicação por Configuração

Para este exemplo, será considerado um ambiente distribuído de configuração e execução, descrito em detalhes em [Szt 95], e que os módulos básicos já foram programados. Os mecanismos disponíveis neste ambiente serão utilizados de forma apropriada para distribuir as redes de sincronização XOCPN. Ao ser entregue para cada estação envolvida, o *texto* da rede de sincronização é traduzido em comandos de configuração nativos do ambiente.

A configuração da aplicação deve ser especificada segundo a linguagem de descrição de arquitetura do ambiente de execução. No exemplo seguinte um trecho do código em Babel [Mal 93] é apresentado. Como a maioria das ADLs, Babel é essencialmente declarativa. Esta especificação também pode ser feita graficamente facilitando a tarefa de concepção do sistema. Inicialmente, os componentes como conectores, módulos que serão utilizados, e a própria aplicação devem ser declarados. Em seguida a interconexão dos módulos deve ser feita. Na última fase os módulos compostos devem ser conectados e a aplicação instanciada. Estes passos são ilustrados a seguir.

```
// declara as estações físicas

    STATION audio_station parati.gta.ufrj.br;
    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 entrada para
      pin VIDEO in video;          // o fluxo já está planejado
      pin AUDIO in audio;  };

connector Fonte_Video_Type {       // conector de saida de video
      pin VIDEO out video; };
...

class Planned_Video_Source {     // módulo composto fonte de vídeo

      conector Planned_Video_Type video_out;

      // módulo com fonte de vídeo sem compressão
      class Video_Source_Class {
        connector Fonte_Video_Type vs;
        extern code C "video_source";
      } VS;

      // módulo que comprime o vídeo
      class Video_Coder_Class {
        connector Compress_Video_Type vcod;
        extern code C "video_coder";
} VCOD;

...
      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 se apresentarão com visibilidade externa são exportadas.

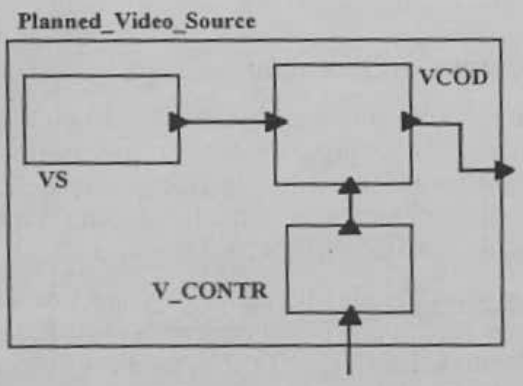

Figura 5.2 - Módulo composto fonte de vídeo com planejamento XOCPN

A seguir, os módulos compostos são instanciados e 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";  };
```

Finalmente, a aplicação é instanciada e executada:

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

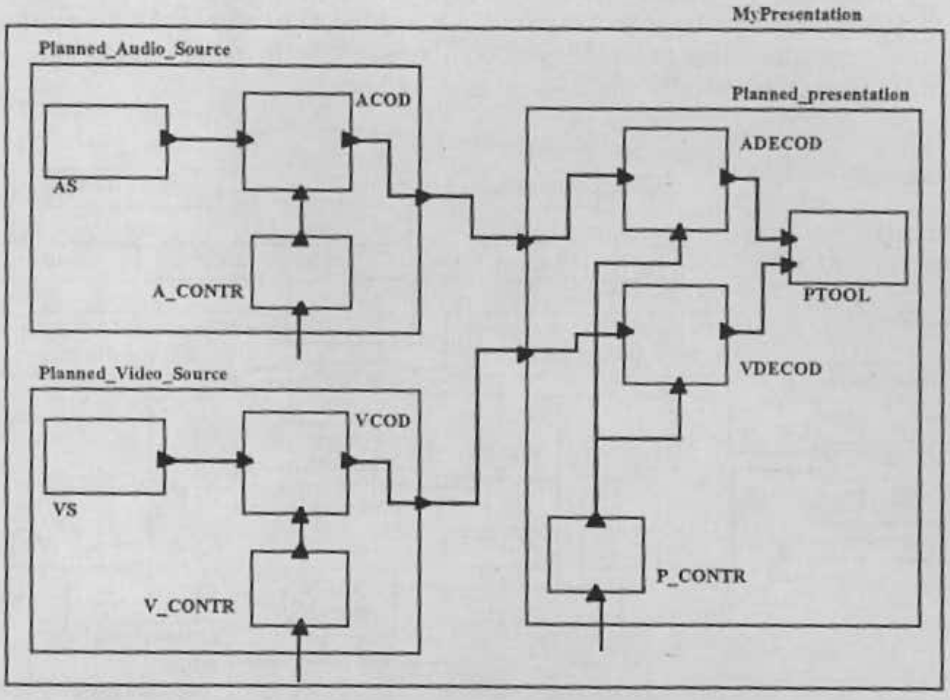

Figura 5.3 - Aplicação configurada pela conexão dos 3 módulos compostos

Uma das vantagens neste estilo de projeto é a possibilidade de selecionar os módulos que irão compor a aplicação, intercambiá-los e reutilizá-los em outras aplicações. Além disso cada conexão pode utilizar características especiais sempre que a aplicação é reconfigurada. A utilização de um outro módulo de compressão/descompressão com características mais atraentes seria facilmente arranjada. Da mesma forma, a adoção de outras técnicas de sincronização seria facilitada através do uso de comandos de configuração.

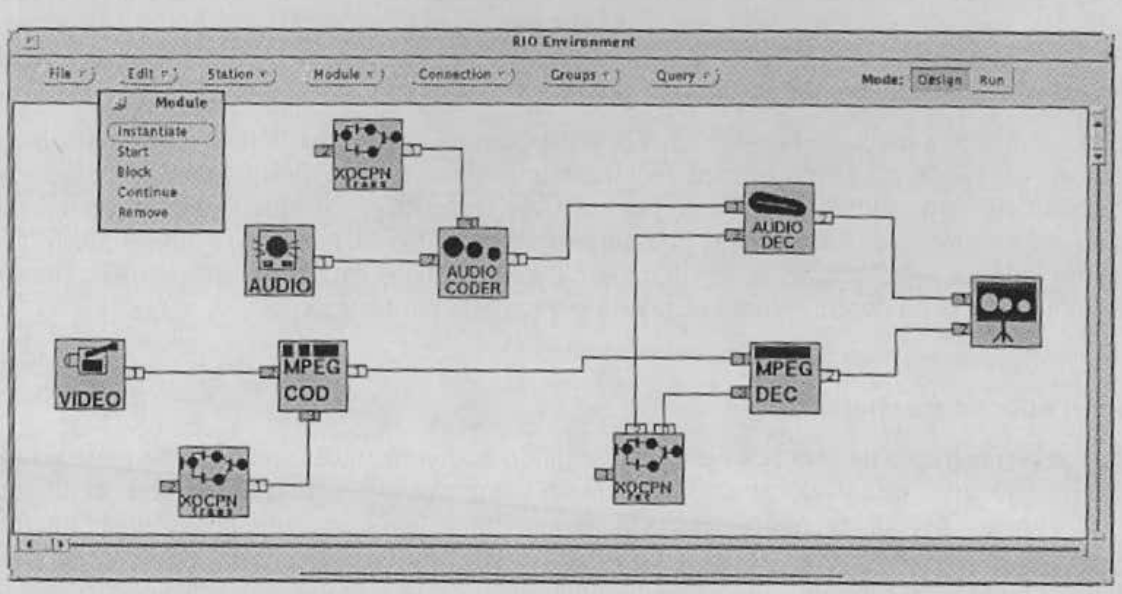

Figura 5.4 - O exemplo descrito através de uma interface gráfica

Como observação, destaca-se que as ADLs são úteis para fins de armazenagem da arquitetura descrita para uso posterior e verificação de propriedades. Entretanto, a configuração através de interfaces gráficas permite que arquiteturas de grande complexidade sejam elaboradas e inspecionadas de forma mais conveniente. Além disso, a conversão de uma configuração em ADL para uma representação gráfica desta configuração (e vice-versa) não apresenta grandes dificuldades. A figura 5.4 apresenta o exemplo configurado graficamente.

5.2 Construindo a Aplicação por Reflexão

Na técnica de reflexão, os mesmos procedimentos para a interconexão dos módulos da aplicação por configuração devem ser previstos no nível da programação. No caso de um sistema programado em uma linguagem orientada a objetos com caraterísticas reflexivas, todos os objetos-base e meta-objetos são instanciados automaticamente.

A programação da aplicação-exemplo possui 4 objetos-base: **fonte-de-vídeo** e **fonte-de-áudio**, que são descendentes da classe fonte de mídia, e **apresentação-de-vídeo** e **apresentação-de-áudio**, que vão compor o objeto apresentação. As características específicas de compressão dos fluxos de mídias, sincronização intermídias e a orquestração da apresentação são adicionadas através de reflexão.

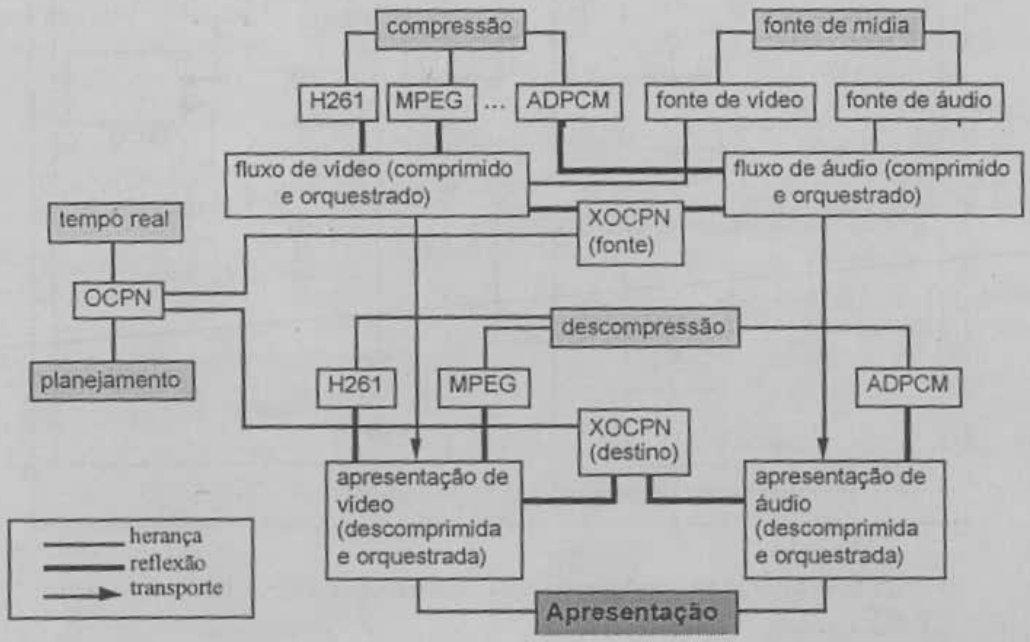

Figura 5.5 - Hierarquia de objetos e meta-objetos da aplicação de apresentação multimídia

Os objetos da aplicação exemplo representam as fontes de som e vídeo, e são pré-orquestrados por XOCPN. Os meta-objetos implementam e interpretam as redes de XOCPN contendo o planejamento da apresentação e provêem a compressão e descompressão das mídias. As questões de distribuição dos objetos da aplicação não foram detalhadas neste exemplo, mas está implícito que existe um transporte dos fluxos de mídia através de uma rede. Além disso, a orquestração descrita em OCPN é expandida em duas rede XOCPN, para a fonte e destino e também devem chegar de alguma forma aos seus destinos.

A figura 5.5 exemplifica o relacionamento dos objetos-base e seus respectivos meta-objetos. Tomando como exemplo a *fonte-de-vídeo*, ela herda as características de *fonte-de-mídia*. Por sua vez, o objeto *fluxo-de-vídeo* herda as características de *fonte-de-mídia* e, por reflexão, este fluxo de vídeo passa a ter características de compressão (utilizando um dos algoritmos de compressão de vídeo disponíveis) e pré-orquestração. De forma semelhante, no lado da apresentação, o objeto *apresentação* herda as características do objeto *apresentação-de-vídeo* (e de áudio) que já foi descomprimida e já está orquestrada por reflexão.

Observação sobre os experimentos

Comparando-se a construção da aplicação-exemplo segundo as duas técnicas apresentadas pode ser verificado que o esforço na programação é equivalente. Isto também foi comprovado em [Loq 97] na área de tolerância a falhas. Os módulos ou objetos devem ser programados em alguma linguagem e em seguida devem ser interconectados. O que difere as duas técnicas, é que, no caso da reflexão, a estrutura da aplicação é descrita através de uma linguagem de programação especializada, enquanto que esta mesma estrutura é descrita externamente, através de uma ADL, independente da linguagem de programação, no caso da configuração.

Deve ser observado que nas linguagens de programação com características reflexivas não existem mecanismos simples que permitam a seleção dinâmica do reflexo a ser usado. Portanto, no exemplo da figura 5.5, deveria ser selecionado em tempo de programação qual o algorítmo de compressão a ser utilizado. Observa-se que a reflexão pode ser utilizada adotando-se, na descrição dos sistemas, a técnica da configuração.

6. Configuração e reflexão em CORBA

O **CORBA** - *Common Object Request Brocker Architecture* , uma arquitetura, baseada no modelo de objetos OMA - *Object Management Architecture* da OMG - *Object Management Group*, vem se firmado como padrão para ambientes e programação de objetos distribuídos. CORBA define um conjunto de componentes e mecanismos para a interação (chamada de métodos) entre objetos distribuídos [OMG 91]. Dentre as características desta arquitetura destacam-se a transparência de localização (realizada através de serviços de diretórios de objetos e serviços) e transparência quanto à linguagem de programação (obtida utilizando-se uma linguagem padronizada para descrição de interfaces). Estas características oferecem a interoperabilidade e portabilidade para objetos distribuídos.

A IDL - *Interface Definiton Language*, um dos componentes importantes no CORBA, é a linguagem de descrição de interfaces (serviços ou métodos oferecidos pelos objetos) padronizada pela OMG. O compilador para IDL permite a geração automática de código a ser ligado ao programa-cliente, chamado de *stub*, e à implementação do objeto, chamada de *skeleton*. Este código contém as funções necessárias para a interação cliente/servidor através do ORB. A geração deste código pode ser mapeada para praticamente qualquer linguagem de programação. Além disto, a partir da descrição das interfaces dos objetos, a IDL pode gerar uma representação da interface para ser consultada dinamicamente em tempo de execução (DII - *Dinamic Invocation Interface*) e armazenar a interface descrita no repositório de interfaces (IR - *Interface Repository*) para consultas de diretório.

A implementação dos conceitos de configuração e reflexão sobre o CORBA é factível [Fra 96, Mag 97 e Loq 97]. Não é difícil admitir, por exemplo, que, da mesma forma que a comunicação através de conectores ou entre meta-objetos poderia ser feita através de RPC, esta mesma comunicação poderia ser especificada de uma forma abstrata através de simples ativação de um método de um objeto remoto, sendo que a própria infra-estrutura do CORBA seria encarregada de efetivar a comunicação necessária.

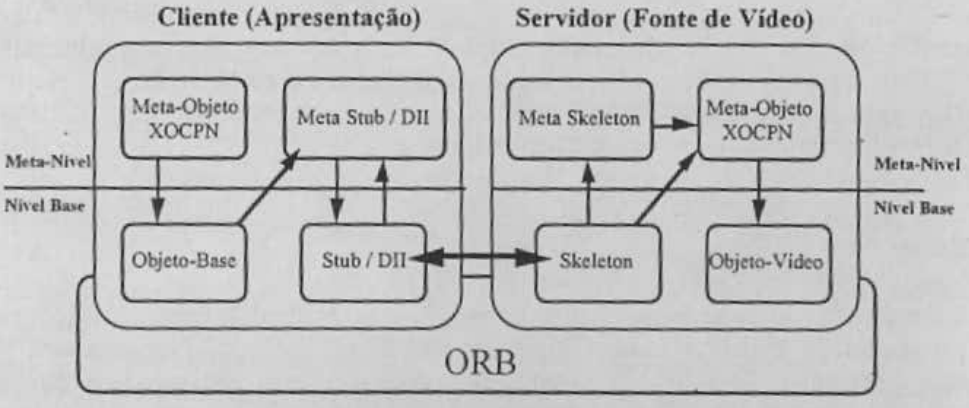

Figura 6.1 - Arquitetura CORBA com meta-nível.

Os mecanismos apresentados até aqui se enquadram 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* - implementação mínima do adaptador de objetos) foram concebidos para aceitar ampliação de suas funções, como a adição de novos protocolos e semânticas de comunicação [Maf 96]. 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 técnicas mais utilizadas em sistemas multimídia. Por exemplo, como proposto em [Fra 96], os meta-objetos que controlam o QoS em determinado serviço de mídia poderiam utilizar meta-objetos para se ajustar em tempo de execução aos atributos enviados como parte da mensagem de ativação de método. A figura 6.1 ilustra o caso do exemplo-base neste contexto.

A especificação das interfaces com a IDL, 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 [Fra 96]. Assim, na arquitetura CORBA passa a existir um meta-nível que intermedia as requisições e respostas dos objetos-base.

Como solução para a construção de aplicações distribuídas por configuração, a arquitetura CORBA tem se mostrado também bastante atraente. Em [Mag 97] é apresentado um esquema que traduz descrições de composição de sistemas em Darwin, uma ADL, em IDL para CORBA. Em [Loq 97] são sugeridas metodologias que permitem a configuração de aplicações tendo como módulos básicos, objetos distribuídos. Neste sentido, foram desenvolvidos objetos para o próprio ambiente CORBA que implementam as abstrações necessárias para sistemas configuráveis (gerenciadores de configuração, gerenciadores de instâncias de objetos, etc.).

Para se obter sobre o CORBA um ambiente que suporte configuração é necessário estender também a funcionalidade dos módulos básicos. Além do BOA, um novo objeto o Gerente de Configuração deve prover as funções de administrar as interações entre objetos remotos em tempo de execução. Métodos específicos para o suporte de configuração também devem ser adicionados às classes clientes. Isto pode ser feito automaticamente usando a propriedade de herança de interfaces da IDL.

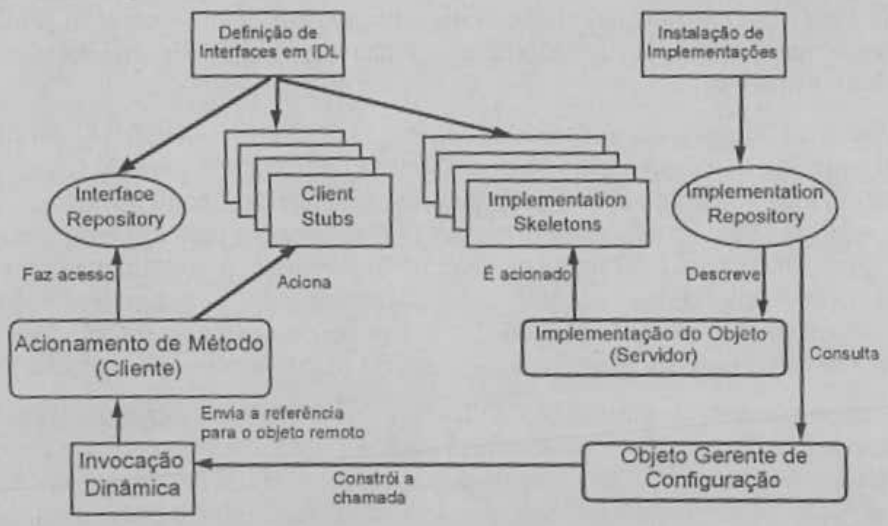

Figura 6.2 - Esquema para geração de uma aplicação CORBA configurável

A utilização de CORBA como plataforma para implementação de sistemas multimídias distribuídos tem sido sugerida como em [Col 96, Kra 96 e Her 96]. Estas propostas estão concentradas em problemas como o transporte por ATM, estabelecimento de conexão em fluxo contínuo de dados e a gerência de QoS nesta plataforma. Não é central nestes trabalhos, entretanto, a preocupação com a facilidade de reutilização de objetos básicos ou adaptação de CORBA à programação de sistemas complexos.

8. Considerações Finais

Duas técnicas para a concepção de sistemas distribuídos foram apresentadas: configuração e reflexão baseada em linguagem de programação especializada. Como observado no exemplo, tanto a configuração, no nível arquitetural, como a reflexão, no nível de linguagem, podem ser utilizadas para a construção de sistemas complexos. A reflexão oferece flexibilidade na programação dos componentes básicos, mas não permite a interferência externa para ajuste do comportamento da aplicação. Este ajuste pode ser previsto, no entanto, na programação da aplicação. A solução por reflexão é pouco dinâmica em relação aos requisitos operacionais. A configuração, por outro lado, obriga que os módulos sejam programados utilizando-se conectores para a troca de mensagens, mas permite que os módulos sejam interconectados sob demanda e que a comunicação seja ajustada antes e durante a execução da aplicação. O atendimento à requisitos operacionais no caso da configuração é bastante simples e dinâmico.

A reflexão computacional, se utilizada em um ambiente baseado em objetos possui um papel semelhante à configuração, porém de forma mais estática, além de ser dependente de linguagem de programação. A interligação de um objeto-base e seu meta-objeto, com as ferramentas atualmente disponíveis já está definida em tempo de compilação. A flexibilidade desejada se dá principalmente durante a programação dos objetos. Além disso, estas linguagens de programação não suportam nativamente programação distribuída. Algumas propostas têm sido apresentadas no sentido de permitir e ligação entre objeto-base e meta-objeto em tempo de execução (*late binding*). Ainda assim, a possibilidade da intervenção de um operador externo na seleção do meta-objeto a ser utilizado ainda não esta clara neste ambiente.

A utilização de reflexão em combinação com linguagens orientadas a objeto mostrou-se promissora na construção de módulos completos de fontes de mídia com requisitos de sincronização temporal, enquanto que as vantagens

da utilização desta técnica para a geração de grandes aplicações multimídia, como o vídeo sob-demanda ou vídeo conferência, se comparadas com as vantagens obtidas com a técnica de configuração, não são evidentes.

Como conclusão preliminar, parece atrativo utilizar a reflexão computacional para a programação de componentes básicos onde um ajuste fino de características não-funcionais é necessário, em um nível de detalhe bastante particular. A técnica de configuração seria melhor indicada na composição de módulos complexos e na interligação de componentes para formação de grandes sistemas. Para dar continuidade a estas investigações estamos analisando a possibilidade de implementação destes conceitos em uma plataforma que reúne CORBA, WWW e a linguagem Java. Embora a linguagem Java não apresente fortes características para comunicação, ela é portável e possui algumas características reflexivas [Way 96 e Wu 97], o que pode ser útil para a programação de objetos básicos. Por outro lado, várias ferramentas para uso de WWW estão incorporando interpretadores Java e o protocolo IIOP - *Internet Inter-ORB Protocol*, um gateway entre ORBs através da Internet, que permite o uso de CORBA e Java de forma integrada. Além disso já existem algumas implementações de CORBA com mapeamento direto de IDL para Java, tornando este ambiente atraente para a implementação de sistemas distribuídos.

9. 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.

[Bla 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 Third International Conference on Configurable Distributed Systems, Annapolis, Maryland, maio de 1996.

[Cam 96] Campo, M., Price, N.H., "Meta-Object Manager: A framework for Customizable Meta-Objetc Support for Smaltalk 80", I Simpósio Brasileiro de Linguagens de Programação, Belo Horizonte, MG, setembro de 1996.

[Chi 95] Chiba, S., "A Metaobject Protocol for C++", OOPSLA 95, 10th Conference on Objetc-Oriented Programming Systems, Languages and Applications, Austin, Texas, EUA, 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 SBRC, Fortaleza, maio de 1996.

[Fab 95] Fabre, J.C.; Perennou, T. e Blain, L., "Meta-Object 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, PE, agosto de 1996.

[Fra 97] Fraga, J., Maziero, C., Lung, L.C., e Loques, O. G., "Implementing replicated services in open systems using a reflective approach", Third International Symposium on Autonomous Descentralized Systems, Berlin, Germany, abril de 1997.

[Flo 96] Florissi, P. GS., e Yemini, Y. "Management of Application Quality of Service", Distributed Comuting and Communications Lab, Computer Science Dep. Columbia University, NY, 1996.

[Geo 96] Georganas, N. D., Steinmetz, R., Nakagawa, T., "Synchronization Issues in Multimedia Communcations", Guest Editorial, IEEE JSAC, Vol 14, N° 1, pág. 1-4, janeiro de 1996.

[Gib 91] Gibbs S., "Composite Multimedia and Active Objects", OOPSLA'91, SIGPLAN Notices, Vol. 26, N° 11, pág. 97-112, novembro de 1991.

[Her 96] Herbert, A., "CORBA Extensions for Real-Time and Multimedia", APM.1311.02, ANSA Phase III, 12 páginas, 1996.

[Jon 93] Jones, M. B., "Interposition Agents: Transparently Interposing User Code at the System Interface", ACM 14th ACM Simposium on Operating Systems Principles, Asheville, EUA, 1993.

[Kra 96] Kramer, A. e Crawford, B., "The Amber project", APM.1686.00.1, ANSA Phase III (draft) Request for Comments, 1996.

[Lim 96] Lima, R. M. e Duarte, O. C. M. B, "Comunicações Sincronizada de Dados Multimídias Pré-Orquestrados", XX Seminário Integrado de Software e Hardware, Recife, PE, agosto de 1996.

[Lit 90] Little, T. D. C., e Ghafoor, A. "synchronization and storage models for multimedia objects", IEEE Selected Areas Communications, Vol. 8, No. 3, pp. 413-427, abril de 1990.

[Lis 95] Lisbôa, M. L., "Reflexão Computacional no Modelo de Orientação a Objetos", 25º Jornadas Argentinas de Informatica y Investigacion Operativa, Buenos Aires, Argentina, setembro de 1995.

[Loq 96] Loques, O., Leite, J., Botafogo, R. e Lobosco, M., "Configuração, Reflexão e CORBA: Algumas Considerações", III Workshop do Projeto ASAP, UFCE, Fortaleza, Ceará, 1996.

[Loq 97] Loques, O., Botafogo, R., Leite, J. "A Configuration Approach for Distributed Object-Oriented System Customization", IEEE Third International Workshop on Object-oriented Real-time Dependable Systems, Newport Beach, California, fevereiro de 1997.

[Maf 95] Maffeis, S. "Adding group communication and fault-tolerance to CORBA", Proceedings of the 1995 USENIX Conference on Object Oriented Technologies, Monterey, CA, EUA, junho de 1995.

[Maf 96] Maffeis, S., "Electra 2.0b Programer's Manual", Dept. of Comuter Science, Cornell University, February, 1996.

[Mag 96] Magee, J., Kramer, J., "Dynamic Structure in Software Architectures", Fourth Symposium on the Foundations of Software Engineering, San-Francisco, California, EUA, outubro de 1996.

[Mag 97] Magee, J., Kramer, J., "Composing Distributed Objects in CORBA", Third International Symposium on Autonomous Descentralized Systems, Berlin, Germany, abril de 1997.

[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.

[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 Computadores, Fortaleza, maio de 1996.

[Sén 96] Sénac, Patric., Diaz, M., Léger, A. e Saqui-Sannes, P. "Modeling Logical and Temporal synchronization in Hypermedia Systems", IEEE Journal on Selected Areas in Communications, Vol 14, No. 1, janeiro de 1996.

[Sha 95] Shaw, M., DeLine, R., Klein, D. e Zelesnik, G., "Abstractions for software architecture and tools to support them", IEEE Transactions on Software Engineering, Vol 21, No. 4, abril de 1995.

[Str 93] Strouds, R., "Transparency and Reflection in Distributed Systems", ACM Operating Systems Review, Vol 22, Num 2, abril de 1993.

[Szt 95] Sztajnberg, A e Loques, O. G., "Ambiente RIO: Estado do Projeto", XXV Congresso da SBC, Canela, RS, agosto de 1995.

[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.

[Tsa 96] Tsang, R. P., Du, D. H.C., Pavan, A., "Experiments with video transmission over an ATM network", Multimedia Systemas, ACM Press, Vol. 4, No. 4, pp 157-171, agosto de 1996.

[Tur 96] Turletti, T. e Huitema, C., "Videoconferencing on the Internet", IEEE/ACM

Transactions on Networking, Vol. 4, Num 3, pp. 640-350, junho de 1996.

[Way 96] Wayner, P. "Better Java Programming", Revista Byte, pp 63-64, setembro de 1996.

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

[Wu 97] Wu, Z., Schwiderski, S., "Design of Reflective Java", APM.1911.01, ANSA Phase III (approved) Technical Report, 30 páginas, 1997.

[Zim 96] Zimmermann, C., Cahill, V. "It's Your Choice - On the Design and Implementation of a Flexible Metalevel Architecture", IEEE, Proceedings of The Third International Conference on Configurable Distributed Systems, Maryland, EUA, maio de 1996.