Implementação de um Ambiente para Processamento Distribuído Baseado em TINA

Alexandre S. Pinto*
DCA - FEEC - UNICAMP
Universidade Estadual de Campinas
C.P. 6101, Campinas, SP, Brasil

Luis F. Faina
DEINF - CETEC - UFU
Universidade Federal de Uberlândia
C.P. 593, Uberlândia, MG, Brasil

Eleri Cardozo
DCA - FEEC - UNICAMP
Universidade Estadual de Campinas
C.P. 6101, Campinas, SP, Brasil

Sumário

A arquitetura TINA (*Telecommunications Information Networking Architecture*) provê conceitos, modelos e mecanismos para o projeto, implementação e gerência de serviços de telecomunicações. Este artigo descreve o projeto e a implementação de um componente central da arquitetura TINA, o DPE (*Distributed Processing Environment*).

O DPE-TINA é uma infra-estrutura de suporte à distribuição dos componentes das aplicações de telecomunicações. Nossa implementação oferece duas funcionalidades básicas às aplicações TINA: serviços de ciclo de vida e de *stream*. O serviço de ciclo de vida permite a distribuição e a gerência de objetos em um sistema heterogêneo, enquanto o serviço de *stream* oferece suporte a fluxos de mídia contínua (e.g.: áudio e vídeo).

Palavras-chave: TINA-C, DPE, CORBA, Sistemas de Computação Móveis, Sistemas Multimídia

Abstract

TINA (*Telecommunications Information Networking Architecture*) provides concepts, models and mechanisms for the design, implementation and management of telecommunications services. This paper describes the design and implementation of a fundamental component of TINA, the DPE (*Distributed Processing Environment*).

TINA DPE is an infrastructure necessary to distribute the telecommunications service components. Our implementation offers two basic facilities to the TINA applications: life-cycle and stream services. Life-cycle service allows distributed objects to be deployed and managed in a heterogeneous system, while stream service provides support to continuous multimedia flows (e.g.: audio and video).

Keywords: TINA-C, DPE, CORBA, Mobile Computing Systems, Multimedia Systems

1 Introdução

Nas duas últimas décadas, o mercado de telecomunicações tornou-se um dos mais cobiçados negócios do planeta devido ao imenso aumento da demanda por este tipo de

*{souza,lffaina,eleri}@dca.fee.unicamp.br

serviço. Para muitos, este crescimento deve persistir por um bom tempo já que o fenômeno de globalização, cada vez mais presente na sociedade, acaba instituindo exigências por novas classes de serviços [1]. A demanda reprimida por serviços de telecomunicações e a integração cada vez maior de voz, dados e vídeo [2, 3] proporcionam um horizonte de oportunidades quase ilimitado para operadoras, provedores de serviços e fabricantes que atuam neste mercado.

Entretanto, este mercado tão promissor pode ser vítima do seu próprio sucesso. Muitas oportunidades estão se tornando, na verdade, ameaças. A indústria de telecomunicações freqüentemente não consegue oferecer serviços que atendam as necessidades cada vez maiores do consumidor. A introdução de novos serviços e funcionalidades em grande escala converteu-se em um processo complexo, caro e lento pois, em geral, exige profundas mudanças na infra-estrutura computacional existente. Além de não conseguir acompanhar o ritmo imposto pelas necessidades do consumidor, esta situação acaba desencorajando novos empreendedores já que o custo de entrada neste mercado é cada vez maior.

Com este cenário em vista, em 1992, cerca de 40 organizações de vários países, incluindo operadoras de telecomunicações, fabricantes de equipamentos de telecomunicações e de computação formaram o consórcio TINA-C [4] (*Telecommunications Information Networking Architecture Consortium*).

O objetivo do consórcio é apresentar uma arquitetura de *software* aberta, denominada arquitetura TINA, com um conjunto completo de especificações para o desenvolvimento e gerência de serviços de telecomunicações, com qualquer grau de complexidade, em escala global.

Neste trabalho, abordamos o projeto e a implementação do ambiente para processamento distribuído (DPE) da arquitetura TINA. Nossa implementação baseia-se em padrões CORBA (*Common Object Request Broker Architecture*) [5] e RM-ODP (*Reference Model for Open Distributed Processing*) [6, 7] e tem como destaques o suporte à migração de objetos e à comunicação via fluxos multimídia.

O artigo está organizado da seguinte forma. A seção 2 apresenta uma visão geral da arquitetura TINA. Na seção 3, são abordados os requisitos de uma arquitetura DPE-TINA. A seção 4 trata dos detalhes de projeto e implementação de um DPE CORBA-*compliant*. A seção 5 descreve uma aplicação multimídia móvel que utiliza as funcionalidades do DPE implementado. Finalmente, a seção 6 apresenta as conclusões do trabalho e suas possíveis extensões.

2 Visão Geral da Arquitetura TINA

TINA [8] é uma arquitetura de *software* para sistemas de telecomunicações baseada em dois conceitos fundamentais: computação distribuída e orientação a objeto, sendo dividida em subconjuntos de escopo bem definido. As quatro principais divisões da arquitetura são:

- **Arquitetura de serviço**: fornece conceitos e princípios para o oferecimento, acesso e utilização de serviços de telecomunicações.
- **Arquitetura de rede**: define um modelo para os recursos da rede (*switches*, enlaces, etc.) em diferentes níveis de detalhe.
- **Arquitetura de computação**: fornece um conjunto de conceitos e princípios para o projeto e implementação de *software* e também para o projeto e implementação

do ambiente de suporte às aplicações distribuídas.

- **Arquitetura de gerência**: provê especificações genéricas para o projeto e implementação de sistemas de *software* utilizados para gerência de serviços, de outros *softwares* e de recursos da rede de transporte.

A estrutura básica de um ambiente TINA é apresentada na figura 1. A camada superior é composta de objetos que proporcionam as funcionalidades das aplicações de telecomunicações. O ambiente para processamento distribuído (*Distributed Processing Environment*, DPE) oferece o suporte a estas aplicações. As funções básicas do DPE são proporcionar uma visão independente de tecnologia das camadas inferiores (potencialmente heterogêneas) e esconder das aplicações a natureza distribuída do sistema, além de fornecer suporte à localização e interação remota dos objetos de aplicação.

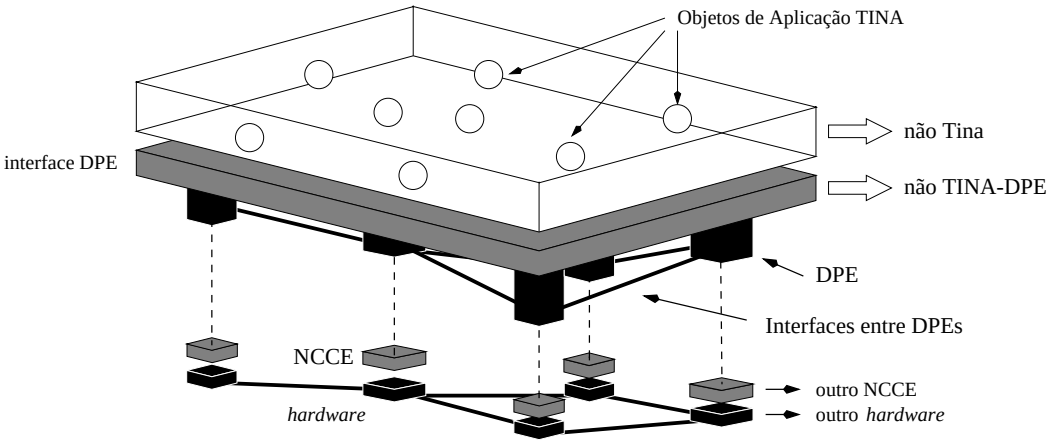

Figura 1: Ambiente TINA [8]

Abaixo das aplicações de telecomunicações e do DPE existem mais duas camadas. A camada mais baixa engloba os recursos de *hardware* (processadores, memória e dispositivos de comunicação). Acima desta, existe uma camada que contém o sistema operacional, protocolos de comunicação e outros *softwares* de suporte a um sistema computacional (compiladores, editores). Tal camada é denominada ambiente de comunicação e computação nativa (*Native Computing and Communications Environment*, NCCE). O projeto e implementação dos NCCEs e dos recursos de *hardware* está fora do escopo da arquitetura TINA. Observe, ainda na figura 1, que um sistema TINA pode interconectar-se, em qualquer nível, com sistemas não-TINA.

É possível a existência de NCCEs com variados graus de complexidade e de diferentes implementações do DPE. A diversidade na camada DPE pode ser devida a questões técnicas ou de mercado. Entretanto, todas as implementações do DPE devem apresentar à camada de aplicação as mesmas funcionalidades (denominada interface do DPE). Além disso, os DPEs devem ter uma interface padrão para comunicação inter-DPE a fim de permitir que as diferentes implementações trabalhem em conjunto dando a impressão de uma infra-estrutura homogênea.

3 Arquitetura do DPE

Os conceitos e princípios da subarquitetura de computação TINA são fortemente influenciados pelo modelo de referência RM-ODP da ISO (*International Organization for Standardization*). Como no padrão ODP, a subarquitetura de computação define cinco pontos de vista (modelos) que devem ser considerados na especificação de um sistema de *software* de telecomunicações. São eles: empreendimento, informação, computacional, engenharia e tecnologia.

A arquitetura DPE, definida no modelo de engenharia [9], é especificada na forma de conceitos, modelos e mecanismos que tratam de aspectos como: suporte à visão computacional e às unidades de distribuição, mecanismos que garantem transparências de distribuição e suporte a *bindings* entre objetos.

3.1 Suporte ao Modelo Computational

O modelo computacional TINA [10] descreve as aplicações de software do sistema em termos de entidades chamadas de objetos computacionais. Estes objetos interagem entre si para proporcionar as funcionalidades da aplicação.

Objetos computacionais comunicam-se exclusivamente por meio de envio e recebimento de informação através de suas interfaces. As interfaces oferecidas pelos objetos podem ser classificadas em dois tipos: interface operacional e interface *stream*. A interação que ocorre na interface operacional é estruturada em termos de invocações de uma ou mais operações e possíveis respostas a estas invocações. Em uma interface *stream*, a interação ocorre via fluxos de mídia unidirecionais (e.g.: áudio e vídeo). Um objeto pode oferecer várias interfaces e de tipos distintos.

O modelo também descreve um conjunto de funções genéricas relacionadas com a gerência do ciclo de vida dos objetos de aplicação. As principais funções são:

- **Criação e remoção de objetos e interfaces:** a criação de um objeto é realizada por meio de uma instanciação de um *template* de objeto. O instanciador do objeto pode ser um construtor de uma linguagem de programação ou um objeto fábrica. Quando um objeto é removido todas as suas interfaces também são eliminadas. Interfaces podem ser dinamicamente adicionadas e removidas de um objeto.
- **Ativação e desativação de interfaces e objetos:** quando uma interface está desativada, nenhuma interação é possível via esta interface. A desativação de um objeto implica na desativação de todas as suas interfaces. Analogamente, o processo de ativação de um objeto ativa todas as suas interfaces.
- **Migração:** os objetos podem ser movidos de localização durante o seu ciclo de vida. O modelo não especifica a maneira como deve ser implementada a migração.

3.2 Suporte às Unidades de Distribuição

Na visão de engenharia, um objeto computacional é representado como um objeto computacional de engenharia (*engineering Computational Object*, eCO). As interfaces de um eCO representam as mesmas interfaces do objeto computacional correspondente. Estas interfaces podem ser ligeiramente modificadas por meio de conversões de tipos e/ou adições de operações necessárias para garantir a interação com outros objetos de engenharia. Além disto, interfaces de gerência podem ser adicionadas ao eCO. O comportamento

que foi definido para o objeto computacional não é alterado durante este processo de conversão.

Os objetos computacionais de engenharia são agrupados em unidades de recurso e de distribuição. As unidades de recurso representam uma coleção de recursos computacionais e de políticas de reserva destes recursos. No modelo de engenharia, duas unidades de recurso foram definidas: nó e cápsula.

Existem várias classificações distintas para o conceito de nó. Um nó de rede é uma unidade que representa qualquer terminal da rede (e.g.: telefone, computador e roteador). Um nó de rede que possui capacidade de processamento é chamado de nó computacional. Se um nó computacional possui uma plataforma DPE é também classificado como um nó DPE. Um nó DPE, a partir daqui denominado simplesmente nó, consiste de recursos de processamento (processos, *threads*, escalonador), de memória (RAM, discos) e de comunicação (pilhas de protocolo, mecanismos de comunicação inter-processos). O nó administra estes recursos de forma autônoma.

A cápsula é uma unidade para fins de alocação dos recursos computacionais de um nó. Os objetos localizados em uma mesma cápsula compartilham as mesmas políticas de reserva de recursos realizada pelo núcleo do nó (e.g.: processo UNIX).

O modelo de engenharia definiu, até o momento, apenas uma unidade de distribuição: o *cluster*. O *cluster* é um conjunto de objetos (eCOs) que possui as propriedades de uma unidade de localização, ativação e migração [9]. Os *clusters* são criados dentro de uma cápsula e nenhum eCO pode pertencer a dois *clusters* distintos. Como o *cluster* não possui propriedades de uma unidade de instanciação [9], é possível adicionar e remover eCOs durante o ciclo de vida do *cluster*.

Os conceitos de nó, cápsula e *cluster* do modelo de engenharia do consórcio TINA são similares aos do modelo de referência ODP.

3.3 Suporte às Transparências de Distribuição

Transparências de distribuição escondem certos aspectos relacionados com a interação entre objetos distribuídos em um sistema heterogêneo. A especificação TINA descreve as seguintes transparências relacionadas com a distribuição:

- **Acesso:** oculta diferenças relacionadas com a representação dos dados e com os mecanismos de invocação existentes em sistemas de computação heterogêneos.
- **Localização:** esconde a localização do objeto de outros componentes da aplicação que interagem com ele.
- **Migração:** permite que o componente que está migrando preserve seu estado durante a mudança de local sem prejudicar a sua interação com outros objetos.
- **Federação:** oculta diferenças existentes entre domínios administrativos distintos.
- **Falha:** esconde dos demais objetos as falhas e ações para recuperar um objeto defeituoso.
- **Recurso:** oculta de um componente da aplicação os processos de ativação e desativação de outros componentes.
- **Concorrência:** possibilita o acesso concorrente de um objeto por vários outros componentes.

No DPE, o suporte às transparências de acesso e localização é considerado essencial enquanto o suporte às demais transparências é opcional.

3.4 Suporte ao *Binding* de Objetos

Uma interação envolvendo múltiplos objetos é chamada de *binding*. Os *bindings* são classificados em dois tipos: implícitos e explícitos. O *binding* implícito é aquele que é estabelecido pela infra-estrutura de distribuição (DPE) sem a participação direta da aplicação. O *binding* explícito é estabelecido por meio de uma requisição explícita feita por algum objeto. Tal *binding* é modelado como um objeto computacional comum (*binding object*) que encapsula os mecanismos de *binding* e oferece operações de controle sobre a interação (adição e remoção de participantes, modificação da qualidade de serviço).

A comunicação entre objetos que estão no mesmo *cluster* está fora do escopo do modelo de engenharia. Este tipo de interação pode ser realizada via memória compartilhada, chamadas locais de procedimento ou comunicação inter-processos.

Para a comunicação entre objetos pertencentes a *clusters* distintos, mecanismos devem proporcionar suporte às transparências de distribuição. Estes mecanismos são oferecidos como parte de um canal (figura 2), como na definição RM-ODP. O conceito de canal vale tanto para *bindings* explícitos quanto implícitos. No caso de *bindings* explícitos, o canal oferece uma interface de controle sobre o *binding* (figura 2). Ou seja, na visão de engenharia, a interface do *binding object* é representada pela interface de controle do canal.

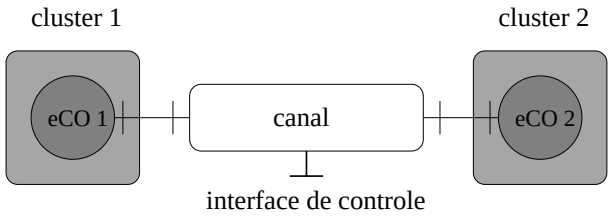

Figura 2: Canal ligando eCOs pertencentes a *clusters* distintos

3.5 Suporte aos Serviços Comuns

Os serviços comuns, também conhecidos como serviços DPE, são aquelas funcionalidades genéricas o suficiente para serem utilizadas por um amplo espectro de aplicações. Exemplos destes serviços são: *trading*, notificação, segurança, transação e propriedade. Os serviços comuns são descritos na forma de objetos computacionais de tal forma que os objetos de aplicação utilizem estes serviços como se estes fossem uma aplicação TINA comum. A implementação dos serviços comuns não é considerada como obrigatória em uma plataforma DPE-TINA.

3.6 Suporte aos Requisitos Não-Funcionais

A última categoria de requisitos é chamada de não-funcional pois não pode ser expressa como um conjunto de funcionalidades ou mecanismos oferecidos pela infra-estrutura (e.g.: desempenho, tolerância a falhas, disponibilidade e escalabilidade).

4 Implementação de um DPE utilizando CORBA

A arquitetura do DPE implementado é apresentada na figura 3. O DPE foi construído sobre a plataforma CORBA [11] do OMG (*Object Management Group*). Uma plataforma CORBA permite interações entre objetos em um sistema distribuído heterogêneo. Instâncias do DPE em cada nó do sistema são conectadas por meio do protocolo IIOP (*Internet Inter-Orb Protocol*), um padrão do OMG [5]. Nesta seção abordamos aspectos de projeto e de implementação das soluções encontradas para atender os requisitos de um DPE-TINA (seção 3).

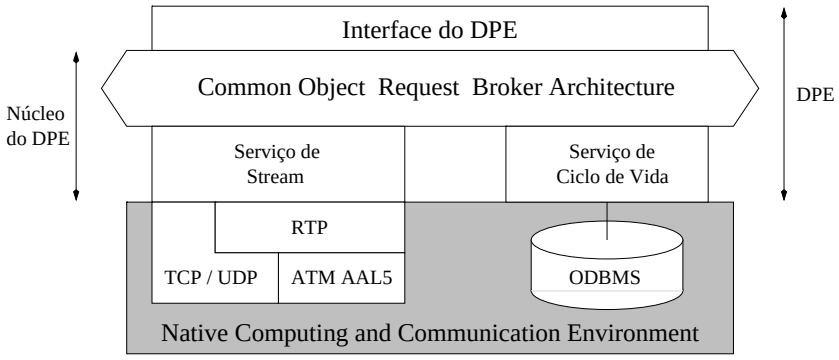

Figura 3: Arquitetura do DPE implementado

4.1 Serviço de Ciclo de Vida

O serviço de ciclo de vida fornece suporte aos conceitos do modelo computacional e às unidades de distribuição TINA. Para implementar este serviço é necessário tratar as diferenças entre os modelos de objetos de uma especificação derivada do RM-ODP (TINA) e do OMG.

De acordo com o modelo computacional TINA, objetos podem ter múltiplas interfaces que são adicionadas e removidas dinamicamente. Tais interfaces são especificadas em ODL (*Object Definition Language*), um padrão ISO [12]. Esta linguagem permite a definição de um objeto com várias interfaces operacionais e *stream*. Em contrapartida, objetos CORBA oferecem apenas uma única interface (operacional) descrita em IDL (*Interface Definition Language*)[1].

Assim, para manter a compatibilidade com a especificação TINA, um objeto computacional de engenharia (eCO) com múltiplas interfaces é representado por vários objetos CORBA, onde cada objeto CORBA oferece os serviços de uma das interfaces do eCO. Em outras palavras, um objeto TINA é especificado por meio de várias interfaces IDL, sendo que cada interface é implementada como um objeto distinto.

Além das interfaces de aplicação, cada eCO possui uma interface adicional para fins de gerência (interface `eCOManager`). Os objetos que implementam a interface `eCOManager` são chamados de gerentes de eCO. A figura 4 mostra como um objeto TINA com três interfaces de aplicação é implementado em CORBA. Os quatro objetos CORBA são administrados como uma entidade única.

[1] OMG está considerando o conceito de múltiplas interfaces em seu modelo de objetos [13].

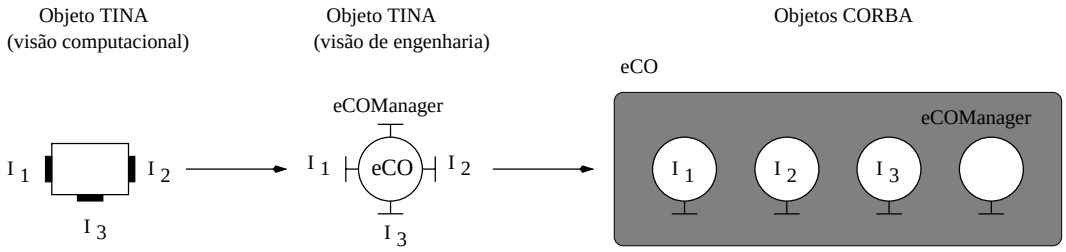

Figura 4: Objeto TINA e sua implementação CORBA correspondente

Interfaces de gerência também são propostas para *clusters*, cápsulas e nós. Os objetos que implementam estas interfaces são chamados de gerentes de *cluster*, gerentes de cápsula e gerentes de nó, respectivamente[2]. Estas interfaces são descritas a seguir.

4.1.1 Gerência de Objetos

Como descrito anteriormente, objetos TINA são representados por meio de um ou mais objetos CORBA, um deles atuando como gerente e os demais proporcionando funções específicas da aplicação (figura 4). Além disso, os objetos de aplicação devem implementar algumas funções de gerência. Tais operações são invocadas pelo gerente de eCO[3] e possuem a seguinte descrição IDL:

- `long checkpoint(in string template)`: armazena o estado do objeto CORBA em um *template*;
- `long recover(in string template)`: recupera o estado do objeto CORBA a partir de um *template*;
- `long Delete()`: destrutor do objeto CORBA, libera todos os recursos reservados ao objeto.

Normalmente, os *templates* correspondem a entradas em um banco de dados que guarda o estado dos objetos. Na nossa implementação, esta armazenagem é feita por um sistema de banco de dados orientado a objetos (ODBMS, figura 3).

A interface `eCOManager` descreve as funções de gerência sobre um objeto TINA:

```
interface eCOManager {
  long addInterface(in Object interface_ref);
  long removeInterface(in Object interface_ref);
  long checkpoint(in string template);
  long recover(in string template);
  long Delete();
  long deactivate(in string template);
};
```

[2]O consórcio OMG possui um serviço de ciclo de vida padronizado [14]. Entretanto, devido à ausência de certas operações de gerência (ativação e desativação, por exemplo) e às diferenças entre os modelos de objetos, decidimos projetar um novo serviço que atendesse satisfatoriamente os requisitos das especificações do consórcio TINA.

[3]O gerente de eCO invoca as funções dos objetos de aplicação via interface de invocação dinâmica (DII) [11]

Interfaces de aplicação são adicionadas e removidas de um eCO via métodos `addInterface()` e `removeInterface()`, respectivamente. Estas operações tem como parâmetro de entrada a referência da interface (ou seja, a referência do objeto CORBA que disponibiliza a interface). Os métodos `checkpoint()`, `recover()` e `Delete()`, simplesmente invocam os métodos correspondentes nos objetos administrados. O método `deactivate()` equivale a uma operação de `checkpoint()` seguida de `Delete()`.

O projetista da aplicação é o responsável pela implementação de uma fábrica para instanciação dos objetos de aplicação a partir de *templates* (interface `AppFactory`). Desta forma, permite-se que a aplicação implemente qualquer esquema de persistência para seus objetos. Em contrapartida, o projetista acaba sendo obrigado a preocupar-se com aspectos não diretamente relacionados com sua aplicação. Como opção, adaptadores de objetos CORBA que suportam persistência [15] e linguagens de programação com serialização nativa (e.g.: Java) permitiriam que esta tarefa fosse realizada exclusivamente pelo DPE.

4.1.2 Gerência de *Clusters*

A gerência de *clusters* é proporcionada por gerentes de *cluster* que implementam a seguinte interface IDL:

```
interface ClusterManager {
  eCOManager makeObject(in string object_name);
  long checkpoint(in string template);
  long recover(in string template);
  long Delete();
  long deactivate(in string template);
};
```

A operação `makeObject()` cria um novo eCO dentro de um *cluster* e retorna a referência de seu gerente. O parâmetro de entrada é o nome pelo qual o eCO é identificado no *cluster*. Este novo eCO é criado sem nenhuma interface de aplicação. Os métodos restantes invocam operações correspondentes em cada gerente de eCO pertecente ao *cluster*.

4.1.3 Gerência de Cápsulas

Funções de gerência de cápsula ainda não foram especificadas pelo consórcio TINA. Assim, definimos um conjunto mínimo de funcionalidades consideradas importantes para o DPE:

```
interface CapsuleManager {
  ClusterManager makeCluster(in string cluster_name);
  long reactivate(in string cluster_name, in string template);
  long migrate(in string cluster_name, in string capsule_name,
               in string node_name);
};
```

A operação `makeCluster()` cria um *cluster* vazio dentro da cápsula e retorna a referência de seu gerente. O parâmetro de entrada é o nome pelo qual o *cluster* é identificado dentro da cápsula. O método `reactivate()` cria um novo *cluster* a partir de um *template* que foi gerado durante um *checkpoint* do *cluster*. A figura 5 mostra a recuperação de

um *cluster* a partir de um *template*. `AppFactory` é um objeto que implementa a fábrica (*pattern Abstract Factory* [16]) responsável pela instanciação dos objetos de aplicação. A operação `createApp()` instancia na cápsula um objeto de um dado tipo e a partir de um determinado *template*. O método `migrate()` realiza a migração de um *cluster* da sua cápsula atual para uma nova cápsula. Esta migração é feita por meio de uma desativação seguida de uma reativação do *cluster* (figura 6).

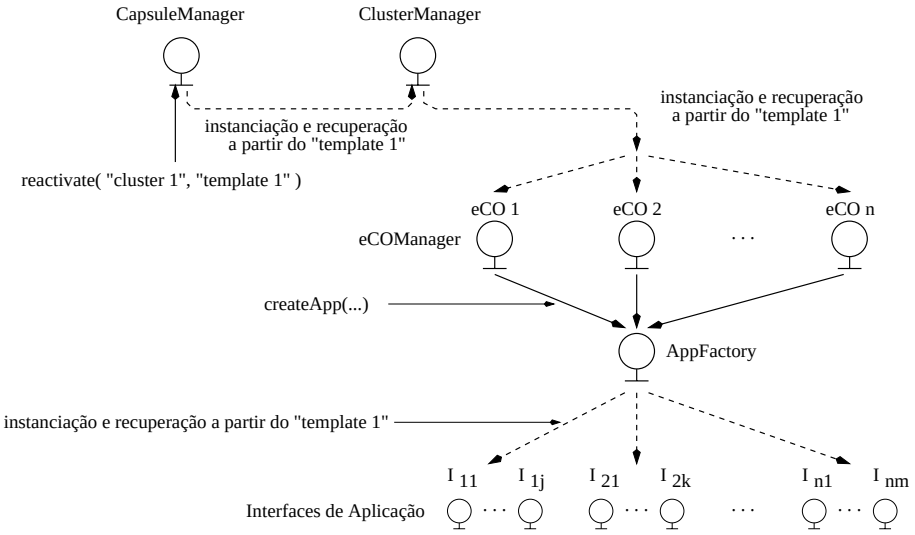

Figura 5: Reativação de um *cluster*

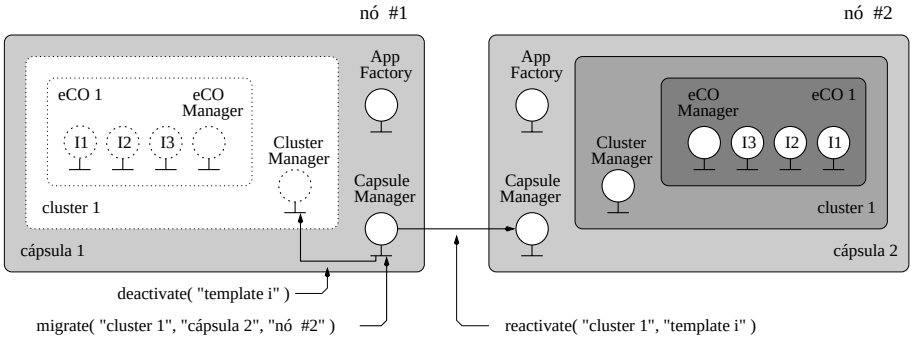

Figura 6: Migração de um *cluster*

4.1.4 Gerência de Nó

O consório TINA também não definiu funções para gerência de nó, mas um pequeno conjunto foi estabelecido para garantir a funcionalidade DPE:

```
interface NodeManager {
  sequence<CapsuleManager> getCapsules();
};
```

Esta interface define apenas uma operação de gerência de ciclo de vida, `getCapsules()`, que retorna todas as cápsulas criadas naquele nó.

4.2 Suporte às Transparências de Distribuição

Das transparências de distribuição citadas na seção 3.3, a implementação realizada suporta três: transparência de acesso, de localização e de recursos.

A plataforma CORBA proporciona total transparência de acesso: mecanismos como *stubs* garantem a transparência na comunicação entre objetos desenvolvidos em linguagens diferentes e/ou que executam em infra-estruturas de *hardware/software* distintas. Por ser aderente à arquitetura CORBA, nosso DPE "herda" o suporte à transparência de acesso.

A transparência de localização é garantida por mecanismos como os serviços de localização do CORBA. Nossa implementação utiliza um *locator* proprietário [17], embora um serviço de localização padronizado (e.g.: serviço de nomes [14]) fosse o recomendado.

O suporte a transparência de recursos é garantido pois o estado de um *cluster* sempre é armazenado antes de sua desativação. Desta forma, é possível ativar e desativar um *cluster* sem que os clientes percebam qualquer efeito deste processo.

Apesar de proporcionar migração, nossa implementação não suporta transparência de migração. Isto ocorre pois as referências dos objetos CORBA tornam-se inválidas após a mudança de localização dos objetos. Novos *frameworks* baseados em Java [18] são capazes de suportar transparência de migração preservando a integridade da referência do objeto CORBA. Tais facilidades poderiam ser incluídas em um DPE baseado na linguagem Java.

4.3 Serviço de *Stream*

O suporte ao *binding* implícito é proporcionado diretamente pela plataforma CORBA utilizada na implementação do DPE. Já o suporte ao *binding* explícito é oferecido pelo serviço de *stream*. Este serviço é uma extensão de uma especificação (*Control and Management of A/V Streams*) do OMG [19].

As interfaces descritas em [19] relevantes para as aplicações TINA são: dispositivo multimídia (interface `MMDevice`) e controlador de *stream* (interface `StreamCtrl`).

Dispositivos multimídia representam dispositivos lógicos ou físicos que geram ou consomem fluxos de mídia. Estes dispositivos comunicam-se via *streams* que podem conter um ou mais fluxos unidirecionais. *Streams* são controlados por um objeto chamado controlador de *stream*. Este objeto oferece operações para o estabelecimento e controle (iniciar, parar) dos fluxos entre os dispositivos multimídia. A figura 7 mostra um *stream* com um fluxo de áudio conectando microfone e alto-falante.

O serviço de *stream* implementado permite a transmisão de fluxos de dados sobre os protocolos UDP, TCP e RTP/UDP.

Para suportar o *binding* explícito na plataforma DPE foram feitas algumas extensões ao serviço de *A/V Streams* do OMG. A interface do controlador do canal (`ChannelCtrl`) herda todas as funcionalidades da interface `StreamCtrl` e adiciona algumas operações:

```
interface ChannelCtrl : AVStreams::StreamCtrl {
  long configureChannel(in AVStreams::streamQos the_qos,
                        in AVStreams::flowSpec the_spec);
  long configureEndPoints(in eCOManager a_eco, in AppMMDevice a_party,
                          in eCOManager b_eco, in AppMMDevice b_party);
  long changeEndPoint(in eCOManager eco, AppMMDevice party);
  long deactivateChannel();
  long reactivateChannel();
};
```

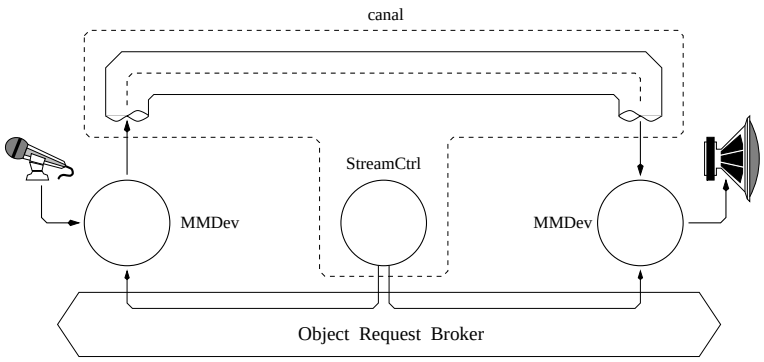

Figura 7: Uma configuração básica de *stream* de áudio

O método `configureChannel()` configura o canal com as informações referentes a qualidade de serviço e aos fluxos (como definido em [19]). O método `configureEndPoint()` indica quais objetos (e respectivos gerentes de eCO) participam da interação. Os demais métodos proporcionam o suporte a migração e são descritos posteriormente. A interface do controlador de canal pode ser especializada para cada tipo de mídia (`AudioChannelCtrl`, `VideoChannelCtrl`, etc.)

Os dispositivos multimídia (interface `MMDevice`) também são estendidos para que possam ser administrados como uma interface de aplicação de um eCO. A interface `AppMMDevice` proporciona métodos de armazenagem/recuperação do estado do objeto e de auto-destruição.

```
interface AppMMDevice : AVStreams::MMDevice {
  long checkpoint(in string templ);
  long recover(in string templ);
  long Delete();
};
```

O gerente de nó é responsável pela criação e destruição do canal. Para isto, foram definidos dois métodos adicionais para a interface `NodeManager`:

```
interface NodeManager {
  long createChannel(in eCOManager a_eco, in AppMMDevice a_party,
                     in eCOManager b_eco, in AppMMDevice b_party,
                     in AVStreams::streamQos the_qos,
                     in AVStreams::flowSpec the_spec,
                     in ChannelCtrl the_ctrl);
  long destroyChannel(in ChannelCtrl the_ctrl);
};
```

Para criar um canal entre dois `AppMMDevice` localizados em *clusters* distintos, o cliente deve obter as referências dos dispositivos multimídia, dos gerentes de eCO correspondentes, do controlador de canal, definir a qualidade de serviço (`the_qos`) e a especificação do fluxo (`the_spec`) e invocar `createChannel()`. O gerente de nó se encarrega de invocar as funções de configuração do canal (`configureChannel()` e `configureEndPoints()`). Após criado o *binding*, o cliente controla o fluxo (`start()`, `stop()`) via controlador de canal.

4.4 Mobilidade de *Streams* de Áudio e Vídeo

Nossa implementação do DPE permite não somente a migração dos objetos do *cluster* como também a migração dos canais associados. Para isto, é necessária a integração do serviço de *stream* com o serviço de ciclo de vida.

O processo de migração do *binding* explícito é mostrado no diagrama de sequência da figura 8. Cada gerente de eCO mantém uma lista com a referência de todos os controladores de canal pertencentes ao eCO. No processo de desativação do eCO, o gerente invoca o método `deactivateChannel()` dos controladores a fim de liberar os recursos de comunicação alocados para cada fluxo. Na cápsula destino, no processo de reativação do eCO, o gerente informa sua nova localização (método `changeEndPoint()`), e reativa o canal (`reactivateChannel()`).

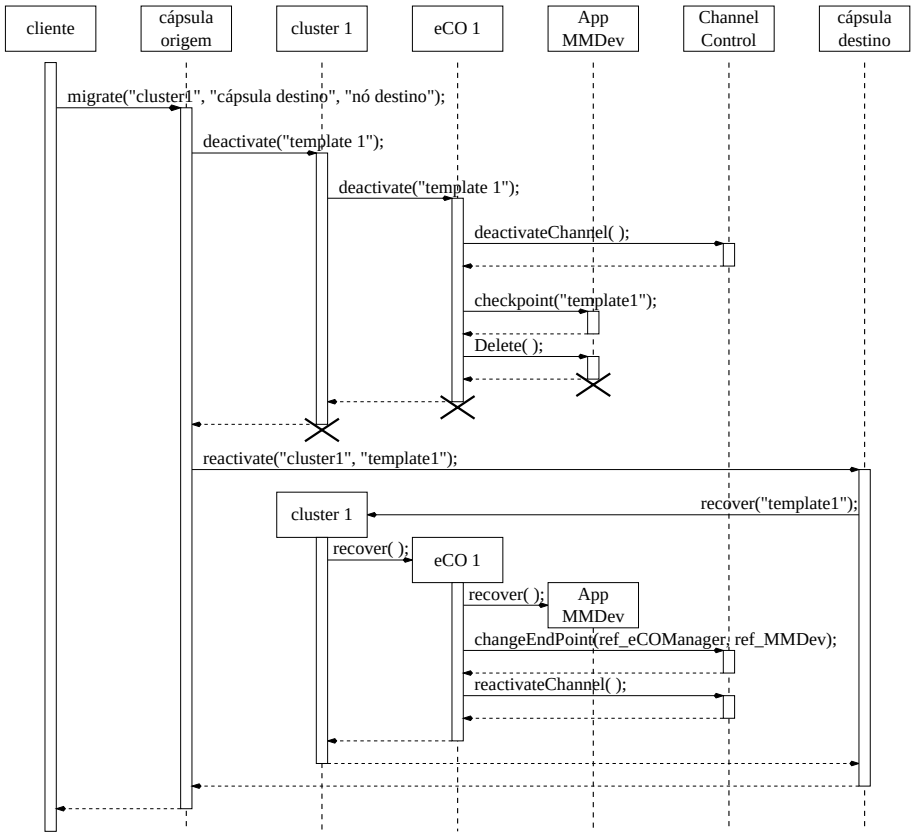

Figura 8: Migração de um canal

Note que controlador de canal não migra e sua localização não interfere na migração do *cluster*. O controlador pode ou não estar em uma das cápsulas que contém os *clusters* comunicantes.

4.5 Suporte aos Serviços Comuns

A especificação TINA ainda não definiu os requisitos de cada um dos serviços DPE. Enquanto isto não é feito, a solução mais natural é utilizar implementações disponíveis dos serviços comuns (eventos, *trading*, nomes, segurança) da arquitetura do consórcio OMG [14].

Nossa implementação atual do DPE não oferece nenhum serviço comum às aplicações TINA.

4.6 Requisitos Não-Funcionais

O modelo de engenharia que descreve as propriedades e qualidades de serviço de uma plataforma DPE também é bastante geral. Assim, não é possível fazer uma análise mais rigorosa das características não funcionais de uma implementação DPE.

De qualquer maneira, um aspecto interessante a ser citado é a característica *multithreaded* da plataforma implementada. Como a especificação CORBA 2.0 não prescreve um modelo de concorrência, utilizou-se uma abordagem proprietária [17]. Este mecanismo, conhecido como filtro (*pattern Chain of Responsability* [16]), permite a definição de vários modelos de concorrência (*pool* de *threads*, *thread* por objeto, *thread* por operação).

Na nossa implementação, o modelo de concorrência escolhido para os objetos gerentes (eCO, *cluster* e cápsula) é o de *thread* por objeto. Já os objetos de aplicação podem ou não possuir uma *thread* de execução exclusiva.

5 Aplicação Multimídia Móvel

Para o desenvolvimento da plataforma DPE foi utilizada a ferramenta Orbix 2.3 [17] da IONA. Orbix é uma implementação da especificação CORBA 2.0 e suporta o mapeamento padrão IDL/C++. O estado dos objetos de aplicação e dos gerentes de *cluster* e de eCO é armazenado em uma base de dados orientada a objetos (ObjectStore 5.0 [20], da Object Design). A plataforma de *hardware/software* utilizada é composta de estações de trabalho Sparc 5 e Ultra 1 da SUN com sistema operacional Solaris 2.6 e interface de rede ATM (155 Mbits/s) conectadas por meio de um *switch* Xylan. O DPE foi implementado na forma de bibliotecas que devem ser ligadas ao código dos objetos de aplicação (clientes e servidores).

Temos então o seguinte cenário de implementação: uma ou mais interfaces de um eCO correspondem a uma *thread* de execução de um processo Unix; um *cluster* representa um conjunto de eCOs que devem estar em uma mesma cápsula; uma cápsula corresponde a um processo UNIX e um nó representa um conjunto de cápsulas (mais núcleo do sistema operacional) em um *host* administradas por um processo servidor Orbix que contém o gerente de nó.

A aplicação protótipo desenvolvida para avaliar os serviços do DPE funciona da seguinte maneira (figura 9). Inicialmente, são instanciadas três cápsulas em três nós distintos. Em duas delas (cápsula 1 e 2) são criados *clusters* de "áudio", contendo um eCO com duas interfaces. Cada uma das interfaces representa um dispositivo de áudio, `Mic` e `Spk`. Estas interfaces são do tipo `AudioDevice` (derivado de `AppMMDevice`). A terceira cápsula (cápsula 3) é criada vazia. Para estabelecer o canal entre os *clusters* de áudio, o cliente invoca o método `createChannel()` em um dos gerentes de nó (os gerentes de nó não são mostrados na figura 9). O fluxo então passa a ser controlado via `AudioChannelCtrl` (derivado da interface `ChannelCtrl`). A migração do *cluster* da cápsula 1 para a cápsula 3 é feita via invocação do método `migrate()` da cápsula 1. Durante o processo de reativação, na cápsula 3, o DPE restabelece o fluxo entre os dois *clusters* de áudio.

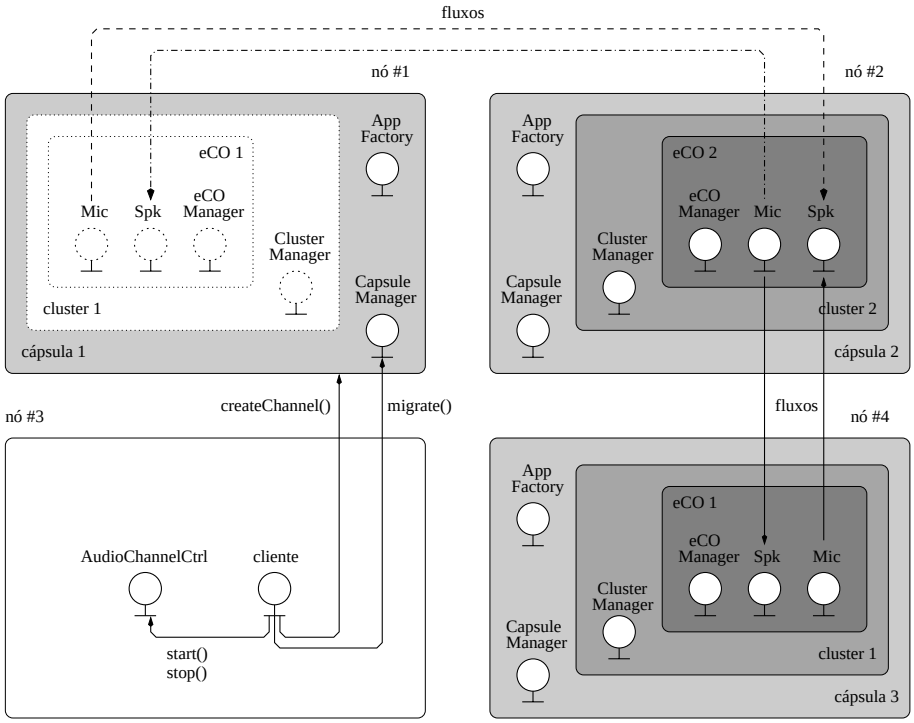

Figura 9: Aplicação multimídia móvel

6 Conclusão

Neste artigo tratamos da implementação de uma infra-estrutura DPE-TINA baseada em CORBA. Em especial, abordamos o suporte da infra-estrura à mobilidade de objetos que comunicam-se via fluxos multimídia. Entre as futuras extensões ao DPE implementado, podemos citar:

- **Canal ponto-multiponto:** A implementação atual suporta apenas a criação de canais ponto-ponto. Novas extensões devem ser feitas à implementação da interface do controlador de canal para permitir este tipo de *binding*.
- **Migração:** A forma de migração implementada (desativação/reativação a partir de um *template*) possui algumas desvantagens. A necessidade de que a cápsula destino possua acesso aos repositórios onde estão armazenados o estado de todos os objetos que migram pode causar problemas de segurança e escalabilidade. Outras formas de migração, via serialização, estão sendo investigadas.
- **Segurança:** implementação de mecanismos de autenticação e autorização que garantam a segurança tanto dos objetos que migram como a da cápsula destino.

Além das extensões ao DPE, estão sendo implementadas aplicações de telecomunicações, seguindo a arquitetura de serviço TINA, sobre a infra-estrutura atual [21].

Agradecimentos

Esta pesquisa é suportada pelas seguintes agências: CAPES (que provê bolsas de mestrado e doutorado para o 1º e 2º autor, respectivamente), CNPQ (300723/93-8) e FINEP (1588/96).

Referências

[1] R. W. Lucky. "New Communications Services - What Does Society Want?". *Proceedings of the IEEE*, 85(10), Outubro 1997.

[2] D. Minoli and E. Minoli. *"Delivering Voice Over IP Networks"*. John Wiley, 1998.

[3] M. R. Macedonia e D. P Brutzman. "Mbone Provides Audio and Video Over the Internet". *IEEE Computing*, Abril 1994.

[4] F. Dupuy e G. Nilsson e Y. Inoue. "Tina Consortium: Toward Networking Telecommunications Information Services". *IEEE Communications Magazine*, 33(11):78–83, Novembro 1995.

[5] Object Management Group. *"The Common Object Request Broker: Architecture and Specification"*. `http://www.omg.org`, Março 1998. Versão 2.2.

[6] ISO/IEC 10746-2 / ITU-T Draft Recommendation X.902. *"ODP Reference Model Part 2, Foundations"*, Junho 1995.

[7] ISO/IEC 10746-3 / ITU-T Draft Recommendation X.903. *"ODP Reference Model Part 3, Architecture"*, Junho 1995.

[8] M. Chapman e S. Montesi. *"Overall Concepts and Principles of TINA"*. `http://www.tinac.com/`, Fevereiro 1995. Versão 1.0.

[9] P. Graubmann et al. *"Engineering Modelling Concepts (DPE Architecture)"*. `http://www.tinac.com/`, Dezembro 1994. Versão 2.0.

[10] TINA-C Core Team. *"Computational Modelling Concepts"*. `http://www.tinac.com/`, Maio 1996. Versão 3.2.

[11] S. Vinoski. "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments". *IEEE Communications Magazine*, 14, Fevereiro 1997.

[12] A Parhar. *"TINA Object Definiton Language Manual"*. `http://www.tinac.com/`, Julho 1996. Versão 2.3.

[13] Object Management Group. "Multiple Interfaces and Composition Request for Proposals". Technical Report ORB/96-01-04, Object Management Group, `http://www.omg.org/`, Agosto 1996.

[14] Object Management Group. *"CORBAservices: Common Object Services Specification"*. `http://www.omg.org`, November 1997. formal/98-07-05.

[15] IONA Technologies Ltd. "Orbix + ObjectStore Adapter - White Paper". `http://www.iona.com/`, 1997.

[16] E. Gamma and et al. *"Design Patterns : Elements of Reusable Object-Oriented Software"*. Addison Wesley, 1995.

[17] IONA Technologies Ltd. *"Orbix 2.3c MT - Programmer's Guide"*, Novembro 1997.

[18] Ad Astra Engineering Inc. "Jumping Beans: Mobile Application Framework". `http://www.jumpingbeans.com/`, 1998.

[19] Object Management Group. *"CORBAtelecoms: Telecommunications Domain Specification"*. `http://www.omg.org`, June 1998. formal/98-07-12.

[20] Object Design. *"ObjectStore - C++ API User Guide - Release 4"*, Junho 1995.

[21] A. S. Pinto et al. "TINA-based Environment for Mobile Multimedia Services". In *TINA'99 Conference*, Havaí, EUA, Abril 1999.