2
Trabalhos Relacionados

Esse capítulo apresenta uma visão geral de trabalhos que versam sobre temas relacionados à implementação de CFDs em um middleware de componentes distribuídos.

Atualmente existem diversos trabalhos sobre Conectores de Fluxos de Dados e abstrações para lidar com eles. No entanto, até o momento da escrita deste documento, apenas um trabalho foi encontrado que se relaciona diretamente com o objetivo de fornecer Conectores de Fluxos de Dados para um modelo de componentes distribuídos. O trabalho em questão é a especificação em andamento de *Streams* para CCM da OMG mencionada anteriormente.

Deixando o contexto de programação baseada em componentes de lado, existem trabalhos que tratam de problemas similares ao deste texto. Os temas variam de soluções específicas para bancos de dados a soluções para telefonia. Devido à extensa quantidade de trabalhos nessa área e dado o objetivo em mãos, um critério de escolha precisou ser estabelecido. Apenas os trabalhos que utilizam sistemas de computação com objetos distribuídos e conectores de fluxo de dados foram escolhidos.

Além da especificação da OMG para o CCM, os trabalhos eleitos pelo critério de escolha estabelecido foram: OMG *Audio / Video Stream Specification* e *"Self-organizing Distribution Structures for Streaming Media"*. Esses trabalhos implementam abstrações de objetos remotos utilizando conectores *RPC* e fornecem abstrações para lidar com fluxos de dados.

As três primeiras seções deste capítulo descrevem, de uma forma geral, os aspectos importantes desses três trabalhos selecionados. A última seção conclui com comentários sobre os requisitos, funcionalidades e limitações desses trabalhos.

2.1
Streams for CCM

Desenvolvido por um grupo especializado da OMG, o trabalho descreve uma extensão do *CORBA Component Model* que especifica o suporte a Conectores de Fluxos de Dados [12]. Essa especificação tenta abranger uma grande

parte dos casos de uso desses conectores, prevendo diferentes mecanismos de transporte e diferentes tipos de dados. Oferece, por exemplo, suporte a transportes como TCP, UDP e RTP e à transmissão de dados simples, de dados estruturados e de áudio/vídeo. Apesar da classificação como trabalho em andamento pela OMG, existem três artigos sobre o modelo e sua utilização publicados por Stoinski [18, 19, 20].

Essa especificação é a que mais se assemelha ao objetivo deste trabalho, pois provê suporte a fluxo de dados em um modelo de componentes distribuídos. O CCM, como o SCS, é baseado em CORBA e fornece um modelo de componentes genérico projetado para atender às necessidades de diversos domínios de aplicações distribuídas. Com essa extensão, o CCM poderá atender às necessidades de aplicações que lidam com fluxos de dados.

Um dos objetivos do trabalho da OMG é facilitar o desenvolvimento de aplicações distribuídas, portanto, boa parte do código não relacionado com a lógica do negócio é implementado pelo contêiner do componente. Com isso, pretende-se obter um componente com uma implementação final mais simples.

Na abordagem proposta pela especificação, podemos notar três características importantes relevantes aos conectores e sua utilização: 1 - os conectores possuem tipos definidos em IDL; 2 - a conectividade das portas de cada componente é realizada através de uma API programática; 3 - são suportados diferentes modelos de execução. Essas características influenciam a arquitetura final de uma solução, e por isso serão abordadas de forma mais detalhada no próximo capítulo.

A arquitetura global da extensão é detalhada na especificação através de interfaces internas (contêiner) e externas (componente) descritas em IDL. São descritos novos tipos de portas e definidos perfis de transporte e de fluxo de dados. A sintaxe da IDL é levemente alterada para suportar novas declarações e, como consequência direta dessa decisão, existe a necessidade de alterar a implementação dos geradores de código que processam essa linguagem. Outra parte interessante encontrada no texto são as regras de gerenciamento de *buffers*, definidas para minimizar o número de cópias dos dados, possibilitando um melhor desempenho.

O meta-modelo estendido do componente (*StreamComponent*) provê uma nova porta (conector) chamada de *StreamPort* (Figura 2.1), que fornece ao componente a habilidade de se comunicar com outros componentes através de fluxos de dados. Uma *StreamPort* suporta apenas um *StreamType*, e pode ser definida como um provedor (*source*) ou consumidor (*sink*) de dados. Assim como as facetas e os receptáculos, portas de fluxos de dados possuem um nome único no escopo do componente.

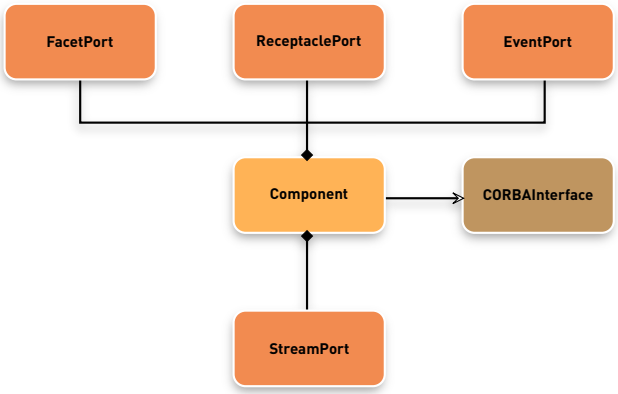

Figura 2.1: O meta-modelo de um CCM StreamComponent

De forma análoga às conexões de outros tipos de porta no CCM, nessa abordagem a utilização de um sistema de tipos também garante conexões de portas de mesmo tipo entre instâncias de componentes. Os conectores nesse modelo podem ser de quatro tipos:

- ***Basic*** - Tipos de fluxo básicos são definidos como formatos de dados concretos para a especificação do conteúdo de um fluxo. Esses formatos não são necessariamente definidos em IDL e, portanto, podem ser codificados de alguma outra forma. Fluxos básicos são usados para o transporte de fluxos de dados codificados, tipicamente dados de áudio ou vídeo. A compatibilidade de conectores desse tipo pode ser avaliada comparando-se o nome do tipo descrito na IDL do conector, utilizando por exemplo, tipos MIME (*Multipurpose Internet Mail Extensions* [21]) ou tipos definidos pelo usuário.
- ***Value*** - Tipos de fluxo de valor são um subtipo de fluxos básicos que transportam tipos de dados especificados em um subconjunto da IDL serializados consecutivamente. O subconjunto suportado é definido na especificação.
- ***Constructed*** - Esse tipo representa um agrupamento hierárquico de vários tipos de fluxo básicos ou outros tipos *Constructed*, indicando a capacidade de produzir ou consumir qualquer um dos tipos básicos ou de valor.
- ***Raw*** - Esse tipo representa fluxos sem tipo, destinados a aplicações onde o formato do fluxo não influencia a funcionalidade de um componente. Exemplos para a aplicabilidade desse tipo podem ser um componente que criptografa ou comprime um fluxo de dados ou um componente que lê ou escreve em um arquivo.

A figura 2.2 apresenta o meta-modelo dos tipos como detalhado na especificação. O código 2.1 ilustra a declaração desses tipos em IDL.

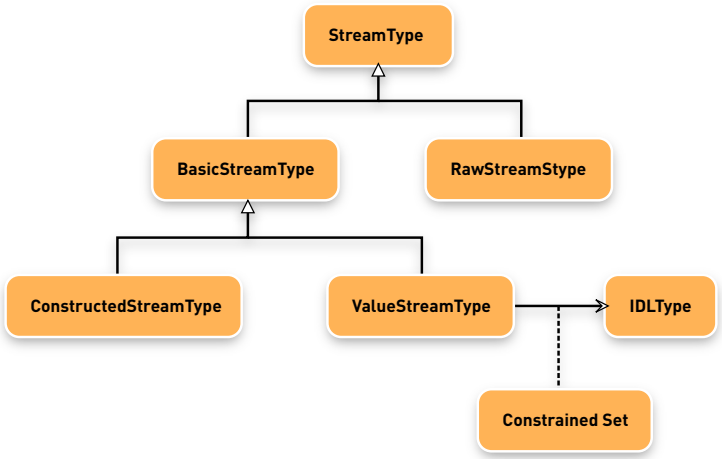

Figura 2.2: O meta-modelo do CCM StreamType

```
1  // Basic
2  streamtype <> MP3;
3  streamtype <> MyOwnFormatStreamType;
4  streamtype <> RGBVideo;
5
6  // Value
7  streamtype <long> MyLongStreamType;
8
9  struct PositionData {
10   long x;
11   long y;
12   long z;
13 };
14 streamtype <PositionData> MyPositionDataStreamType;
15
16 // Constructed
17 streamtype MyAudioStreamType : PCM, MP3;
18 streamtype MyVideoStreamType : RealVideo, WMV;
19 streamtype MyMultimediaStreamType : MyAudioStreamType, MyVideoStreamType;
```

Código 2.1: Declaração dos tipos de fluxos.

A conexão entre os dois tipos de porta (*Source* e *Sink*) é realizada através da interface *StreamPorts*. Essa interface em especial oferece operações de conexão e desconexão de portas, assim como operações de introspecção. Dessa forma, um componente pode ter conhecimento das portas providas e se conectar dinamicamente a outro componente em tempo de execução. O trecho de código 2.2 exibe a declaração de um componente com as novas portas.

```
1 component MyComponent {
2   sink MyPositionDataStreamType pos_data_input;
3   source Components::Streams::RGBVideo video_output;
4 };
```

Código 2.2: Declaração dos tipos de fluxos.

A programação do componente pode ser realizada utilizando diferentes modelos de execução. O código pode ser bloqueante ou baseado em eventos de leitura e escrita. A especificação descreve um terceiro modelo chamado *Run Condition* que é baseado em eventos de nível mais alto, como por exemplo, quantidade de dados ou tempo máximo. Uma descrição mais detalhada dos modelos de execução é realizada na seção 3.2.

2.2 OMG Audio / Video Stream Specification

A necessidade de suporte a conectores de fluxos de dados em aplicações distribuídas já estava em evidência há mais de uma década. Em um trabalho anterior ao *"Stream for CCM"*, um grupo da OMG especificou uma extensão de Conectores Fluxos de Dados no contexto de CORBA sem fazer qualquer referência à programação baseada em componentes [22]. Essa especificação visava atender aplicações que transmitiam dados de áudio e vídeo em sistemas distribuídos. A extensão proposta por esse grupo cobria um conjunto menor de requisitos e, como consequência, o modelo resultante é naturalmente mais simples que o modelo do CCM descrito na seção anterior.

Essa especificação apresenta um modelo arquitetural e interfaces IDL para a construção de *frameworks* de comunicação distribuída que suportem fluxos de dados multimídia. O suporte à extensão não requer alterações na sintaxe da IDL. Apenas a implementação de uma biblioteca de classes se faz necessária. Os principais objetivos dessa especificação são: padronizar os mecanismos de estabelecimento e controle de fluxos, suportar múltiplos protocolos de transporte e suportar diversos tipos de fontes e consumidores.

A figura 2.3 ilustra a arquitetura de um fluxo entre dois pontos de conexão (*Endpoints*), uma fonte (*Source*) e um consumidor (*Sink*). Cada *endpoint* é composto por três entidades lógicas:

1. ***Stream Interface Control Object*** - provê uma interface para controle e gerência do fluxo de dados.

2. ***Flow Data Endpoint* (*Source* ou *Sink*)** - objeto que representa o destino final de um fluxo de dados.

3. ***Stream Adapter*** - objeto que transmite e recebe dados em uma rede de computadores.

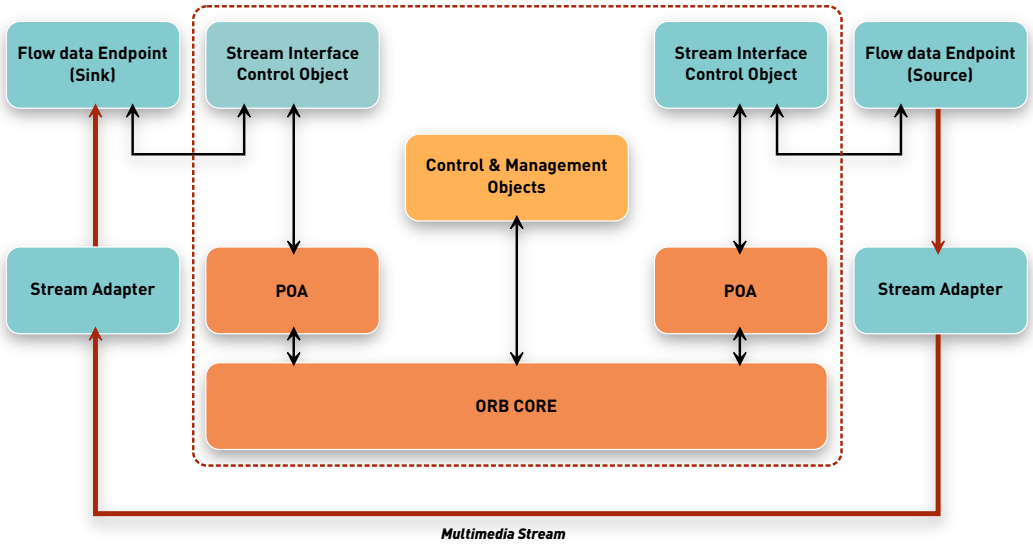

Figura 2.3: Arquitetura da OMG *Audio/Video Stream Specification*.

Os conectores são apresentados como interfaces de dispositivos virtuais e o conteúdo fornecido é identificado através de tipos MIME [21]. A figura 2.4 mostra um fluxo simples entre um microfone e um alto-falante. As principais entidades que compõem o *framework* são:

- **Dispositivos de multimídia normais e virtuais** - representados pelas interfaces *MMDevice* e *VDev*.
- **Fluxo de dados *(Stream)*** - representado pela interface *StreamCtrl*. Um fluxo pode ser composto por várias correntes (*Flows*).
- **Pontos de conexão de fluxos** - representados pela interface *StreamEndPoint*.
- **Correntes *(Flows)* e seus pontos de conexão** - representados por *FlowConnection* e *FlowEndPoint*.
- **Dipositivos de corrente** - Representado por *FDev*.

O estabelecimento e controle de conexões é realizado através de uma API definida pela interface *StreamCtrl*. Ela oferece métodos como *bind* e *bind_devs* para conectar um dispositivo ou múltiplos dispositivos. O código 2.3 demostra o procedimento de conexão entre dois videofones.

```
1 // (C++) Declare the local and remote videophone multimedia devices
2 videophone_ptr myPhone = ...;
3 videophone_ptr johnsPhone = ...;
4
5 //Declare the stream controller
6 videophone_StreamCtrl_ptr myStream;
```

```
7  // Some code here to initialize the local MMDevice (myPhone)
8  ...
9  // Bind johnsPhone
10 ...
11
12 myStream = myPhone->bind(johnsPhone,
13   someQoS,
14   &wasQoSMet,// Was requested QoS honored
15   nilFlowSpec); // Bind all flows
16
17 myStream->start(nilFlowSpec);
18
19 cout << "Hit any key to hang up..." << endl;
20 cin >> buf;
21
22 myStream->stop(nilFlowSpec);
23 myStream->destroy(nilFlowSpec);
```

Código 2.3: Conexão de dois *EndPoints*.

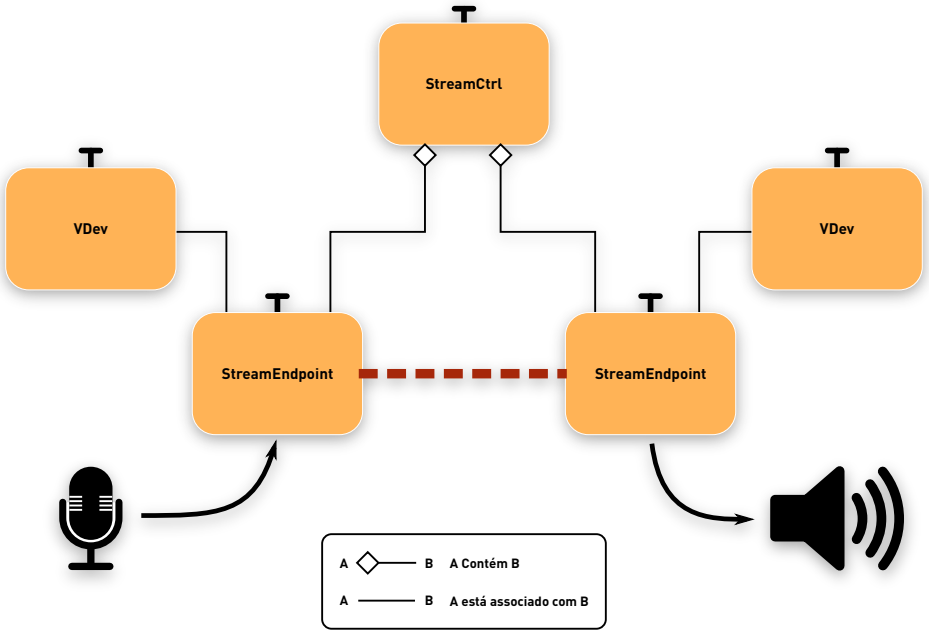

Figura 2.4: Uma configuração de fluxo básica.

Não são definidos na especificação modelos de execução ou regras de gerenciamento de *buffers*, ficando a cargo do desenvolvedor da aplicação definir e implementar tais decisões.

2.3 Self-organizing Distribution Structures for Streaming Media

No artigo publicado por *Rafaelsen et al.* [23], é descrito um *framework* de *gateways* de conteúdo multimídia para o desenvolvimento de gerenciadores de conexões de fluxos de dados. O arcabouço contribui para a gestão automatizada de serviços de *streaming* com múltiplos usuários, através do suporte à

configuração e reconfiguração automatizada de conexões de múltiplos receptores. A gestão automatizada é realizada por politicas providas por usuários e gestores do sistema.

O artigo de *Rafaelsen et al.* não se relaciona diretamente com o objetivo desse trabalho, mas sim com o cenário motivador, que visava produzir uma aplicação com suporte à execução de fluxos de algoritmos distribuídos. Esse artigo não descreve as características dos CFDs, entretanto, as políticas podem ser vistas como planos para construir e gerenciar conexões, um conceito relativamente similar aos planos de implantação propostos no trabalho desenvolvido por Barbosa [17].

A figura 2.5 ilustra conexões "heterogêneas" entre entidades do arcabouço. Essas são classificadas como heterogêneas pelo autor pois os receptores possuem características diferentes, como por exemplo, formato/tipo de dados suportado. Como pode ser observado, o arcabouço consiste de quatro tipos de entidades básicas:

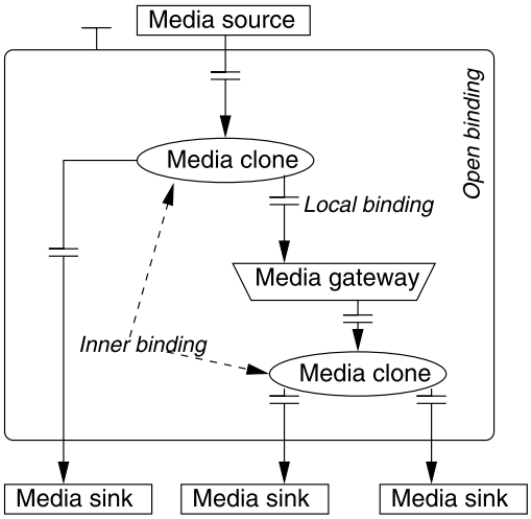

Figura 2.5: Conexões heterogêneas.

- ***Media Source*** - entidade capaz de produzir fluxos de mídia. Esses fluxos podem ser produzidas ao vivo de uma câmera ou de um arquivo de mídia existente. O produtor pode ter a opção de entregar o fluxo de mídia em diversos formatos mas, no entanto, para cada conexão ativa, apenas um formato pode ser selecionado.
- ***Media Sink*** - entidade capaz de receber fluxos de mídia produzidas por uma fonte de mídia (produtor).
- ***Media Gateway*** - entidade de rede que fornece a conversão de dados de mídia realizadas através da rede. Se um consumidor não é capaz

de receber a mídia no formato provido pelo produtor, essa entidade é inserida na conexão para fazer a conversão.

– ***Media Clone*** - entidade responsável pelo suporte a *multicast*. Sua tarefa é receber um fluxo de mídia como entrada e produzir multiplas cópias do fluxo como saída.

Para decidir a composição das conexões entre as entidades através de um algoritmo, o sistema desenvolvido utiliza os seguintes componentes:

– ***Type Checker*** - faz verificações de tipo. Informalmente, verificação de tipo significa garantir que as propriedades de cada interface de origem são como o esperado pela interface de destino correspondente.

– ***Gateway Trader*** - fornece um serviço de oferta e procura de *gateways*.

– ***Composition Meta-Level Model*** - responsável por manter uma representação do grafo de conexões.

– ***Binding Manager*** - responsável por construir e gerenciar conexões.

A conexão entre uma entidade provedora e uma consumidora é realizada através do serviço fornecido pelo *Binding Manager*. Para racionalizar sobre a compatibilidade de tipos e verificar se existe a necessidade de conversores de formatos entre uma fonte e um consumidor, são utilizados os serviços dos outros componentes e metadados associados às entidades. Esses metadados são descritos na linguagem *Flow Interface Description Language (FIDL)* desenvolvida em um trabalho anterior do mesmo grupo [24]. A *FIDL* é utilizada para descrever o conjunto de configurações possíveis de entrada e saída das interfaces de mídia das entidades.

2.4 Considerações Finais

Esse capítulo apresentou as principais características das soluções desenvolvidas nos trabalhos relacionados. O trabalho *Streams for CCM* é o que mais se assemelha com os desafios enfrentados por esta pesquisa. Entretanto, apesar dos outros dois trabalhos apresentados não serem relacionados à programação baseada em componentes, eles tratam partes dos mesmos problemas com abordagens diferentes.

A analise dos três trabalhos nos permitiu levantar características importantes relacionadas a Conectores de Fluxos de Dados. Nesses trabalhos, diferentes sistemas de tipos foram utilizados para verificar a compatibilidade dos conectores provedores com os consumidores. O primeiro utiliza um sistema de

tipos que permite a construção de tipos compostos derivados de outros tipos, introduzindo um nível de polimorfismo semelhante à programação orientada a objetos. A maior vantagem dessa abordagem é a possibilidade dos componentes expressarem a habilidade de processar conjuntos de tipos polimórficos de fluxos de dados, permitindo assim, a implementação de componentes mais reutilizáveis e adaptáveis.

O segundo trabalho utiliza um sistema de tipos baseado em relações um para um. A maior vantagem dessa abordagem é a simplicidade de implementação. No terceiro trabalho os conectores podem fornecer conteúdo em diversos formatos, então, uma linguagem é utilizada para descrever as possíveis configurações de cada conector. Com isso, um algoritmo baseado em uma politica pré-definida é utilizado para resolver a compatibilidade entre dois conectores. Essa abordagem parece ser tão flexível quanto a utilizada no primeiro trabalho mas, no entanto, só foi utilizada para descrever conteúdo multimídia e não fluxos de dados genéricos.

Uma segunda característica comum nesses trabalhos é a existência de uma API para introspecção e conectividade dinâmica. Todos eles oferecem uma interface para consulta de conectores existentes e métodos para conectá-los. Entretanto, dadas as diferentes abordagens, os passos envolvidos no estabelecimento de uma conexão são completamente diferentes em cada um dos sistemas. Apesar disso, todos fazem verificação de tipos.

Uma última característica importante é descrita apenas no primeiro trabalho, o modelo de execução. A extensão do CCM suporta três modelos diferentes, um bloqueante e dois reativos a eventos. A extensão de CORBA não especifica modelos de execução, mas em um artigo publicado por *Mungee et al.* [25] são descritos dois possíveis modelos, um baseado na concorrência de processos e um reativo. O último trabalho não entra nesse nível de detalhe sobre as implementações. Contudo, o suporte a diferentes modelos é fundamental para que os desenvolvedores sejam capazes de se expressarem adequadamente na implementação de um software desse domínio.

A especificação do suporte a fluxos de dados para o CCM apresenta uma solução interessante, entretanto nenhuma implementação completa da mesma foi encontrada até o momento da escrita deste texto. Além disso, CCM e CORBA têm sido alvos de diversas críticas (*Henning* [26] e *Chappel* [27]), tais como ter uma especificação grande e complexa, e o fato de nenhum fornecedor comercial de CORBA ter se comprometido a implementar CCM.

Finalmente, essa extensão requer modificações no *parser* de IDL e implementações razoavelmente complexas no ORB/contêiner. A falta de suporte adequado de fornecedores é um ponto a ser considerado.