Um Modelo de Interconexão de Componentes e sua Implementação em um Middleware para Sistemas de Televisão Digital Interativa

Carlos Eduardo da Silva, Diogo Fagundes de Oliveira, Frederico Lopes, Frederico Amaro, Adilson Barboza Lopes, Guido Lemos de Souza Filho, Gledson Elias

Resumo—Sistemas de televisão digital interativa devem incorporar conceitos de middleware para abstrair especificidades de hardware e sistemas operacionais. Aplicações nestes sistemas podem ser executadas em diferentes tipos de plataformas. Os componentes destes sistemas precisam interagir com outros, de modo a cumprir os requisitos da aplicação. Neste contexto, este artigo descreve um modelo consistente para interconexão de componentes e a arquitetura de implementação que está sendo usada para o desenvolvimento do Middleware AdapTV, um middleware para sistemas de televisão digital interativa.

***Palavras-Chave*—middleware, televisão digital interativa, componentes de software.**

***Abstract*—Interactive digital television systems must incorporate concepts of middleware to abstract hardware and operational systems details. Applications in these systems can be executed in different platforms. The components of these systems need to interact with one another, in order to fulfill the application requirements. In this context, this article describes a consistent model for components interconnection and the implementation architecture that has been used to the development of the Middleware AdapTV, an interactive digital television systems middleware.**

***Index Terms*—middleware, interactive digital television, software components.**

I. INTRODUÇÃO

Televisão digital interativa (TVDI) é uma tecnologia que está cada vez mais presente no cenário mundial. No Brasil em particular, o Governo está financiando um projeto para definição do Sistema Brasileiro de Televisão Digital (SBTVD). O desenvolvimento deste projeto está sendo coordenado pelo Ministério das Comunicações envolvendo um grupo de diferentes Ministérios, da Agência Nacional de Telecomunicações (ANATEL) e de vários segmentos da sociedade civil. O projeto encontra-se em sua fase inicial, com investimentos sendo aplicados para viabilizar a integração da comunidade acadêmica, centros de pesquisas e indústrias no sentido de analisar e definir um padrão viável tecnologicamente e economicamente para a realidade nacional. Neste contexto, a TVDI abre perspectivas bastante interessantes para o mercado de desenvolvimento de software nacional, assim como para exportação.

Apesar de já existirem no mundo diversas tecnologias para TVDI, há espaço para discutir a padronização de dispositivos, sistemas operacionais, ambientes de execução, linguagens, serviços e APIs no contexto da realidade brasileira.

Uma aplicação para TV interativa pode manipular diversas mídias com requisitos temporais críticos, tais como: áudio e vídeo, além de manipular fluxos de dados que devem ser executados em sincronia com o fluxo multimídia. Além disso, a complexidade destas aplicações é incrementada, uma vez que poderão ser executadas em diversos dispositivos, com capacidades variadas de processamento e armazenamento, e diferentes interfaces com o usuário.

Neste contexto, foi realizada a concepção de uma arquitetura conceitual de um middleware adaptativo para TVDI denominado AdapTV [1]. Este middleware foi baseado no framework Cosmos [2], cujo objetivo é prover uma arquitetura genérica baseada em componentes para a camada de configuração e gerenciamento de recursos em sistemas multimídia distribuídos abertos, servindo de prova de conceito para o framework. Explorando estas propostas, o presente artigo define um modelo de interconexão de componentes onde são abordados aspectos de comunicação entre componentes multimídia distribuídos.

De modo a instanciar o modelo, o artigo apresenta uma arquitetura para implementação do mesmo, usando algumas técnicas e padrões de programação propostos pela engenharia de software. O principal requisito consiste em prover um mecanismo de interconexão genérico, de modo a possibilitar a integração de uma variedade de tipos de componentes em ambientes distribuídos e heterogêneos. Esta arquitetura está sendo utilizada na implementação do middleware AdapTV, como uma extensão do protótipo apresentado em [1].

O artigo introduz na seção 2 o modelo de interconexão proposto apresentando o conceito de portas e o modelo de componentes adotado. A seção 3 descreve a arquitetura de implementação do mesmo. Na seção 4 são feitas algumas considerações sobre os principais trabalhos relacionados. Por fim, a seção 5 apresenta as conclusões e algumas considerações finais.

Carlos Eduardo da Silva, Diogo Fagundes de Oliveira, Frederico Lopes, Frederico Amaro, Adilson Barbosa Lopes, Departamento de Informática e Matemática Aplicada, UFRN, Brasil, E-Mails: kaduardo@natalnet.br, diogo@natalnet.br, sepolderf@natalnet.br, famarojr@natalnet.br, adilson@dimap.ufrn.br. Guido Lemos de Souza Filho, Gledson Elias, Departamento de Informática, UFPB, Brasil, E-mail: guido@di.ufpb.br, gledson@di.ufpb.br.

II. O MODELO DE INTERCONEXÃO

O Cosmos tem como objetivo atuar como um framework genérico para projeto e desenvolvimento da camada middleware para uma variedade de sistemas multimídia distribuídos abertos. De modo a atender este propósito, o framework concentra seus esforços na representação e manipulação da diversidade de conceitos, requisitos e componentes relacionados com os sistemas multimídia, fornecendo uma visão arquitetural do sistema, abstraindo os detalhes de implementação desses componentes.

No modelo proposto pelo Cosmos, os componentes possuem interfaces de comunicação para transmitir e receber dados, de forma a reduzir o acoplamento inter-componentes. Tais interfaces são denominadas portas. As portas são caracterizadas dinamicamente por propriedades (metadados) que descrevem os requisitos de um canal de comunicação. As portas também são tratadas como componentes internos, que estão integrados no contexto de outros componentes de escopo maior.

Usando a representação definida em [4,5], a Figura 1 apresenta uma visão geral do modelo com um diagrama de componentes (nível de especificação), mostrando os principais elementos envolvidos, bem como suas interfaces providas e requeridas.

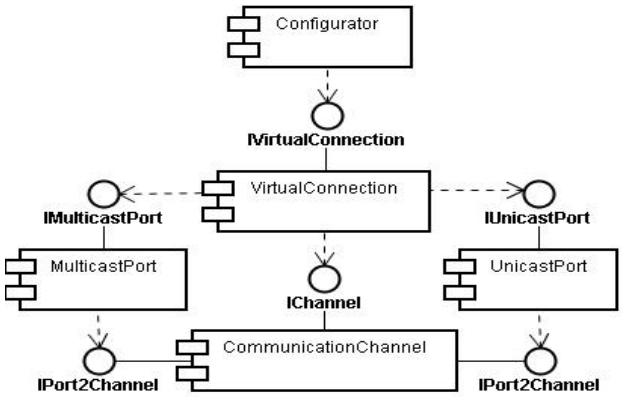

Figura 1: Diagrama de Componentes com uma Visão Geral do Modelo de Interconexão.

É através do componente VirtualConnection que o modelo trata de forma abstrata os aspectos da comunicação relacionados com formatos de mídia, topologia de interconexao, sincronização, protocolos e QoS. O modelo introduz também o componente CommunicationChannel que trata os detalhes relacionados com os tipos de tecnologia que dão suporte à comunicação entre portas.

O *Configurator* é o componente crítico responsável por iniciar, configurar e gerenciar os recursos e componentes do sistema. Cada aplicação é definida através de uma especificação envolvendo os componentes e os elementos da arquitetura, bem como os vários requisitos de QoS necessários à sua execução. É papel do *Configurator* interpretar e processar uma especificação de uma aplicação, de seus componentes e respectivas interconexões [3].

Cosmos incorpora a noção de reflexividade para dar suporte à adaptação e à reconfiguração dinâmica. O conceito de reflexividade do Cosmos foi inspirado em [6], estando presente em vários middlewares atuais Em particular, no contexto de uma comunicação envolvendo componentes remotos, onde o meio de transmissão está naturalmente sujeito a flutuações de comportamento e sobrecarga, o ajuste dinâmico do sistema é um requisito essencial. O suporte à reflexividade no Cosmos se dá em nível de operações nas meta-interfaces, consistindo na definição e modificação de valores de propriedades.

Pelo fato dos ambientes de programação atuais utilizarem o paradigma de objetos, pode-se presumir que componentes serão constituídos por objetos. Logo, pode-se afirmar que componentes consistem de classes.

A Figura 2 apresenta a descrição de algumas classes que foram definidas para implementação dos principais componentes do modelo, destacando-se os componentes portas.

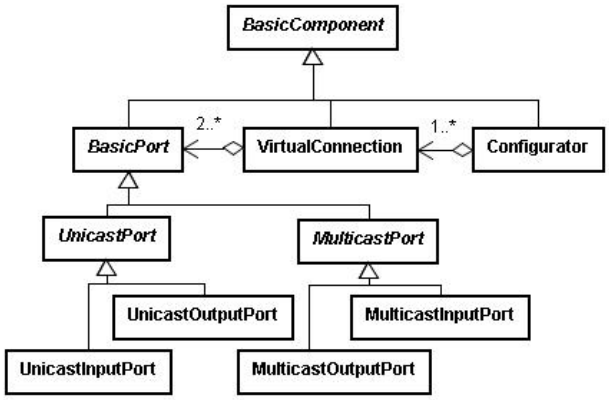

Figura 2: Principais Classes dos Componentes do Modelo de Interconexão.

Uma porta é uma abstração usada para definir um ponto de interação de um componente, podendo ser utilizada para conectá-lo com outros componentes no ambiente de execução. A declaração de uma porta em um componente consiste em prover metadados que definem as características do fluxo associado, envolvendo o tipo da porta (Entrada e Saída), o formato do fluxo, bem como os requisitos de qualidade de serviço (QoS), que são usados pelo configurator para negociação, monitoramento e gerenciamento de QoS.

Partindo da classe *BasicPort*, uma porta pode ser especializada para se comportar como porta de comunicação unicast ou multicast (*UnicastPort* ou *MulticastPort*); a partir daí, na hierarquia, ela pode ser definida como uma porta de entrada ou de saída

Um componente pode ter várias portas de tipos diferentes, que podem ser usadas para interagir com o ambiente - quer provendo informações, ou recebendo. No modelo são consideradas questões de topologia (como, conexão um-para-um, um para vários, vários para um e vários para vários), de compatibilidade (formatos de mídia), multi-protocolos (identificação e escolha dos tipos de conexão adequados) e de controle (não exploradas no presente artigo - envolvendo operações típicas para sincronização entre uma operação de envio e recepção, bem como para sincronização entre diferentes fluxos). Alguns atributos da porta são definidos

estaticamente e estabelecidos no processo de codificação do componente, como por exemplo, se a porta é de entrada ou saída; outros são vinculados durante a configuração, em função da topologia da conexão, e outros dinamicamente, quando houver ajustes e negociações para QoS.

O componente *VirtualConnection* representa uma conexão entre duas ou mais portas, abstraindo aspectos relacionados com a agregação e manipulação dos fluxos elementares associados, isolando-os para a aplicação, ao nível de componente e de porta. O modelo da Figura 2 possibilita tanto conexões unicast, quanto multicast e conexões entre componentes locais e/ou remotos.

É papel do *Configurator* estabelecer e realizar a negociação de acordos para configurar as propriedades das portas com valores de propriedades coerentes. O objetivo da negociação é cumprir os requisitos requeridos e especificados na aplicação, observando as potencialidades e limitações da plataforma. Como o foco do artigo está direcionado a aspectos de interconexão entre componentes, o texto discute algumas questões relevantes da arquitetura proposta, descrevendo algumas das operações que foram definidas na API de interconexão.

III. ARQUITETURA DE IMPLEMENTAÇÃO

A arquitetura de interconexão definida explora o conceito de propriedades [2,3]. O conceito de propriedades foi introduzido no framework Cosmos para descrever características e requisitos de componentes, sendo, portanto usadas para fins de gerenciamento de configuração.

Como elementos principais do modelo, foram definidos os componentes porta e canal A API de uma porta estabelece uma interface comum, independentemente de que tipos de comunicação sejam suportados pela plataforma local aonde o correspondente componente venha a ser instanciado.

Na arquitetura proposta para implementação do middleware o canal tem como papel abstrair os detalhes referentes à tecnologia de comunicação usada para a troca de dados, realizando a ligação entre duas portas.

A Figura 3 apresenta uma visão geral com a descrição de algumas classes que foram definidas na arquitetura de implementação para o modelo de interconexão proposto. Os canais de comunicação são representados pela classe Channel, podendo ter diversas especializações para representar as várias estratégias de comunicação que podem ser utilizadas. Uma conexão virtual (*VirtualConnection*) pode ter diversos canais de comunicação. É responsabilidade do *Configurator* manter estas conexões, gerenciando portas e intercâmbio de dados.

A interface *IInputPortOperation* oferece uma API unificada para que os componentes possam operar com uma porta de entrada, ao passo que a interface *IOutputPortOperation* prove uma API para as portas de saída.

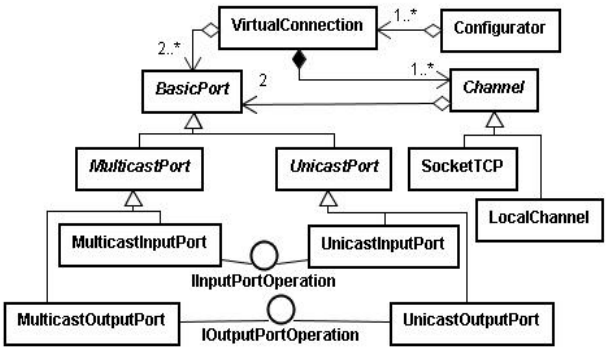

Figura 3. Visão Geral da Arquitetura de Implementação.

A definição do número de canais em uma *VirtualConnection* depende da topologia de interconexão estabelecida na especificação. Seguindo o modelo proposto, podemos ter tanto conexões unicast (cardinalidade 1:1 no relacionamento *VirtualConnection-Channel*), quanto multicast (cardinalidade 1:N no relacionamento *VirtualConnection-Channel*). Ou seja, para realizar uma conexão multicast, eventualmente o sistema criará diversos canais de comunicação, podendo envolver diversas tecnologias de comunicação simultaneamente.

É através da configuração de uma aplicação, onde conexões são estabelecidas, que uma determinada topologia de interconexão será definida. O processo de escolha de uma determinada tecnologia depende de fatores dinâmicos, tais como, a localização dos componentes envolvidos (por exemplo, se estão instanciados no mesmo espaço de endereçamento ou em computadores distribuídos). O processo para conectar dois ou mais componentes consiste em verificar se existe compatibilidade entre os tipos de dados (formatos) das portas, as capacidades de cada componente envolvido e a topologia requerida. Para isto ele estabelece valores de propriedades que satisfaçam os requisitos descritos na especificação.

Através da classe *Channel* consegue-se uma abstração acerca da tecnologia de comunicação a ser utilizada, fornecendo uma interface bem definida para o processo de estabelecimento, configuração e encerramento de uma conexão. As subclasses de *Channel* são definidas observando as possíveis tecnologias de comunicação existentes atualmente, que poderão ser utilizadas para a implementação do modelo.

A figura 4 apresenta a API definida para as classes *UnicastPort* e *MulticastPort*. A classe abstrata *UnicastPort* oferece uma interface única para a configuração de uma porta unicast, provendo métodos para se indicar o elemento com o qual a porta efetivamente realizará a troca de dados (setPartner) e para recuperar este elemento (getPartner).

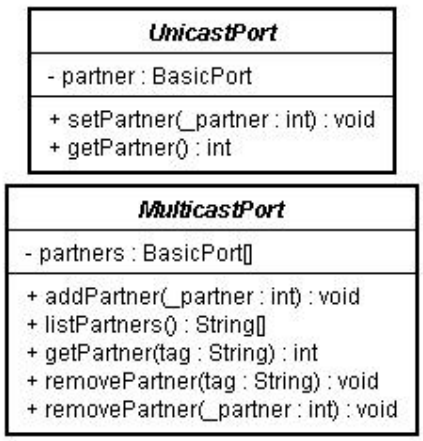

Figura 4. API das Portas.

De maneira análoga, a classe abstrata *MulticastPort* oferece uma API unificada para gerenciamento de portas multicast. Esta API provê métodos para adicionar um parceiro para a troca de dados (*addPartner*), para listar os elementos que estão no momento trocando dados com a porta em questão (*listPartners*), para recuperar um dos elementos (*getPartner*) e para remover parceiros de comunicação (*removePartner*), tanto por sua identificação (tag) quanto por sua referencia direta (_partner).

A classe abstrata *Channel* apresentada na Figura 5 oferece uma API para o estabelecimento de um canal de comunicação entre duas portas.

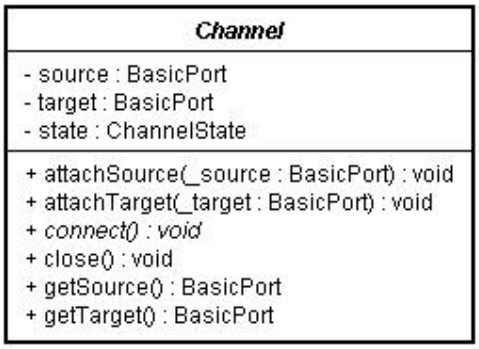

Figura 5. API do Canal.

Na API do canal foram definidos métodos para indicar quais são as portas participantes da interconexão (*attachSource* e attachTarget) e para recuperar estas portas (getSource e getTarget). Nota-se também a presença do método *connect*, que realiza todos os passos necessários para efetivamente conectar as portas envolvidas, alocando os recursos necessários para o estabelecimento do canal de comunicação entre duas portas; já o método *close* é utilizado para encerrar um canal de comunicação, liberando todos os recursos que foram alocados na instanciação deste canal.

A API da classe *VirtualConnection*, apresentada na Figura 6, provê, entre outros, métodos para se anexar uma porta (*attachPort*) de entrada (elementos que provêem uma interface do tipo *IInputPortOperation*) ou saída (elementos que provêem uma interface do tipo *IOutputPortOperation*), listar as portas envolvidas na interconexão (*listOutputPorts* ou *listInputPorts*), recuperar portas por sua identificação (*getOutputPort* ou *getInputPort*) e remover portas (*removeInputPort* ou *removeOutputPort*), tanto por sua identificação (tag) quanto por referência direta (*BasicPort*).

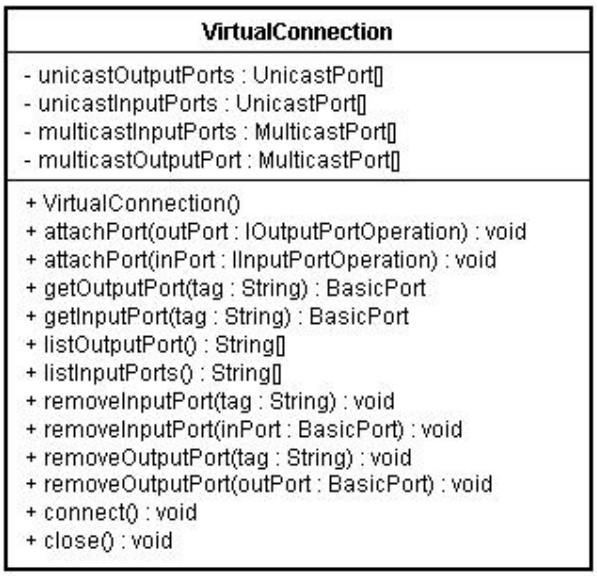

Figura 6. API da classe *VirtualConnection*.

Como explicitado anteriormente, o modelo suporta diferentes topologias de interconexão, permitindo o uso de diversos canais de comunicação simultaneamente, inclusive de diferentes tecnologias. A seleção de qual tipo de canal utilizar, por parte do middleware, pode ser em função, por exemplo, da localização dos componentes. Seguindo este critério, componentes residentes em uma mesma máquina podem se comunicar por intermédio de memória compartilhada. Outros critérios, poderiam ser baixo nível de perda de pacotes, ou menor jitter.

A Figura 7 apresenta uma visão arquitetural simplificada do modelo com um exemplo de uma conexão multicast envolvendo um componente produtor, que gera um fluxo de dados multimídia (component 1), e dois componentes clientes (component 2 e component 3), que recebem estes fluxos e realizam alguma ação. Neste exemplo, a conexão virtual mantém as referências dos canais de comunicação.

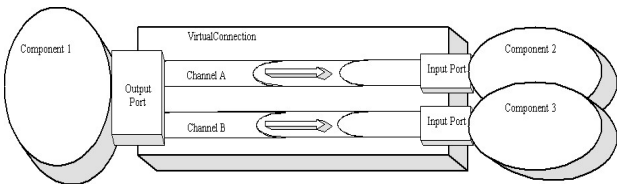

Figura 7. Visão Simplificada de uma arquitetura para uma conexão multicast

As portas de entrada (*InputPort*) e saída (*OutputPort*) estão representadas em seus respectivos componentes. Conexões (unicast ou multicast) são tratadas pelo Configurator através de componentes *VirtualConnection*s, constituindo-se assim no ponto de acesso para os elementos

envolvidos. Portanto, é através do componente *VirtualConnection* que o configurator tem acesso aos canais e portas. Os canais de comunicação incorporam a tecnologia utilizada para a realização da conexão física entre as portas envolvidas, fornecendo uma interface bem definida para as portas e para o Configurator.

A Figura 8 ilustra a fase inicial do processo definido para realizar uma interconexão no middleware AdapTV. Uma aplicação deve ser especificada na linguagem baseada em XML que foi definida em [7]. Um componente *XMLApplicationReader* processa esta especificação para obter as conexões e negociar as propriedades das portas definidas na especificação da conexão. Para simplificar a figura, algumas operações, bem como os parâmetros das mesmas foram abstraídos.

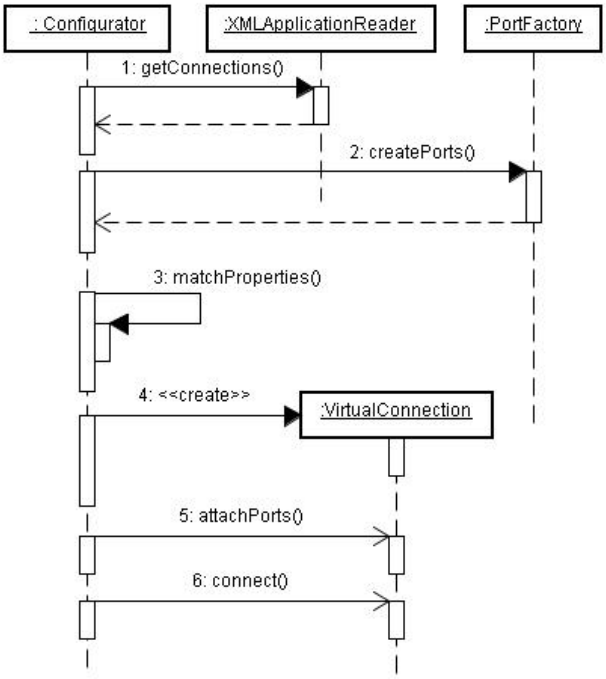

Figura 8. Fase inicial do processo de interconexão de componentes.

O *Configurator* solicita a criação das portas a um componente do tipo factory (*PortFactory*), conforme definido pelo Cosmos, e negocia então as propriedades objetivando definir valores compatíveis entre a especificação e a plataforma. Este processo produz os metadados que serão utilizados para as operações que irão efetivamente realizar a conexão das portas.

Logo em seguida o Configurator cria uma instância da classe *VirtualConnection*, que é usada para representar a conexão em questão, passando os metadados referentes à conexão. As portas criadas anteriormente são anexadas ao *VirtualConnection*, abstraindo o tratamento necessário para a ligação entre as mesmas. Após a configuração do *VirtualConnection,* o configurator realiza uma chamada para o método *connect* que efetivamente dispara o processo de alocação dos canais de comunicação que forem necessários.

A Figura 9 ilustra o processo de criação e alocação dos canais de comunicação, após o configurador acionar o método connect do *VirtualConnection.*

Após a chamada do método connect, O *VirtualConnection* realiza a instanciação dos canais de comunicação necessários, passando para cada um deles suas respectivas portas. Um canal de comunicação é representado por uma das subclasses da classe *Channel*, dependendo da tecnologia de comunicação utilizada.

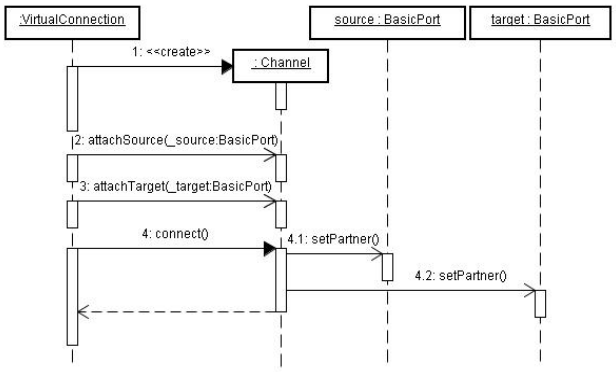

Figura 9. Criação e alocação dos canais de comunicação.

Ao chamar o método *connect* do canal de comunicação, o componente *VirtualConnection* passa o controle para o mesmo. O canal então realiza todos os procedimentos necessários para o estabelecimento de uma conexão entre as duas portas envolvidas utilizando a estratégia de comunicação que ele representa. Por conseguinte, ele tem conhecimento de como proceder para utilizá-las.

O canal de comunicação indica às portas envolvidas, através do método *setPartner*, sua referencia. Desta forma, as portas podem utilizar o canal para realizar a troca de dados.

IV. TRABALHOS RELACIONADOS

Na literatura existem vários trabalhos envolvendo middlewares com funcionalidades para sistemas multimídia. Entre eles podemos destacar o projeto NMM (Network-Integrated Multimedia Middleware) [8,9], que oferece uma arquitetura para a construção de aplicações multimídia distribuídas em ambiente Linux. Como elemento principal, destaca-se o conceito de nó (do inglês *node*) que encapsula processamento e funcionalidades. Os nós se conectam com outros nós através dos *jacks*. Os *jacks* são responsáveis por transportar os dados dos buffers de um nó para o outro, representando a entrada e saída de um nó. Através do uso de *jacks* ele explora o conceito de portas, possibilitando conexões 1x1 e 1xN de forma transparente para o componente que esta enviando os dados. Questões relacionadas com a adaptação e reconfiguração dinâmica são indicadas como trabalhos futuros, não tendo até o presente momento nenhum tipo de suporte.

No contexto desta proposta, foi também analisado o framework PREMO [10] da ISO, que leva em consideração um extenso conjunto de requisitos para aplicações multimídia

distribuídas. PREMO possui um grande nível de complexidade, o que inviabiliza a sua aplicação para o contexto de sistemas de Televisão digital interativa. Talvez, devido a esta complexidade, é incomum encontrar referencias para sua implementação na literatura.

Outro framework relacionado é o DirectShow da Microsoft [11]. Ele permite a construção de aplicações multimídia em ambientes de desenvolvimento Microsoft. Nesse framework, os componentes são implementados como filtros que se comunicam através de pinos. Os pinos fazem o papel das portas. O framework DirectShow é restrito a aplicações locais, não suportando nenhum tipo de adaptação ou reconfiguração, e no caso de aplicações distribuídas, o programador tem que explicitamente configurar as aplicações envolvidas em máquinas distintas, para que elas possam efetivamente trocar dados.

No modelo de interconexão definido neste trabalho, utiliza-se o conceito de porta explorado no AdapTV. As portas no AdapTV funcionam de maneira análoga aos pinos do DirectShow e aos jacks do NMM. Entretanto no middleware AdapTV elas podem ser configuradas explorando o conceito de propriedades introduzido pelo framework Cosmos. Além disso, elas podem ser reconfiguradas automaticamente em tempo de execução, suportando a comunicação distribuída de maneira transparente para a aplicação.

V. CONCLUSÃO

Este trabalho apresenta um modelo de interconexão de componentes que foi projetado para ser usado em ambientes multimídia distribuídos abertos. Como prova de conceito deste modelo, o artigo define uma arquitetura para sua implementação que está sendo realizada em um middleware para Sistemas de Televisão Digital Interativa. O principal objetivo desta arquitetura é prover um mecanismo de interconexão genérico, de modo a possibilitar a integração de uma variedade de tipos de componentes em ambientes heterogêneos. O artigo apresenta uma API relativamente simples e flexível, envolvendo as principais operações suportadas pelos componentes definidos no middleware AdapTV e seus relacionamentos com os conceitos definidos pelo framework Cosmos.

O conceito clássico de portas, que tem sido amplamente utilizado pela engenharia de software, constitui-se num elemento de grande flexibilidade da proposta, permitindo tratar conexões unicast e multicast com abstrações similares.

A API definida permite que o processo de configuração possa ser adaptado dinamicamente, possibilitando inclusive, que as portas sejam reconfiguradas automaticamente em tempo de execução, por exemplo, através da mudança de um canal de comunicação.

A implementação da extensão do protótipo do middleware AdapTV está em andamento, onde estão sendo incorporadas as idéias propostas pelo modelo de interconexão, com a adição de novas classes e novos métodos nos componentes envolvidos. Atualmente está sendo iniciada uma implementação do modelo para interconexão de portas unicast com duas estratégias para comunicação local: memória compartilhada e sockets. No protótipo, a adaptação interna de componentes consiste em mudar os formatos dos dados apresentados e a tecnologia de suporte a comunicação.

REFERÊNCIAS

[1] Borelli, F., Lopes A. B., Elias, G, Lemos, Guido. *Projeto e Implementação de um Middleware para Sistemas de Televisão Digital Interativa*. In: IV Workshop de Teses e Dissertações em Multimídia, Hipermídia e Web. Webmidia 2004, Ribeirão Preto, SP.

[2] Lopes, A. B., Borelli, F., Elias, G., Magalhães, M. F. A Component-based Configuration and Management Framework for Open, Distributed Multimedia Systems. In: 18th International Conference on Advanced Information Networking and Application (AINA'04), Fukuoka, Japão, 2004.

[3] Lopes A. B, Borelli, F., Elias, G., Lemos, Guido, Magalhães, Maurício. *Uma Arquitetura para Configuração e Gerenciamento de Recursos em um Middleware para Sistemas de Televisão Digital Interativa*. In: XXI Simpósio Brasileiro de Telecomunicações, 2004, Belém, PA.

[4] Szyperski C. . *Component Software-Beyond Object-Oriented Programming*. Addson –Wesley, Longman Limited, First Edition, 1998.

[5] Chesman J., Daniels J. UML Components-A Simple Process for Specifying Component-Based Software. Addson –Wesley, 2001.

[6] Blair, G. S., Cuolson, G et al, "The Design and Implementation of Open ORB Version 2. IEEE Distributed Journal, 2001, vol 2, no 6.

[7] Borelli, F. Lopes, A., Elias, G.. *A XML-Based Component Specification Model for an Adaptive Middleware of Interactive Digital Television Systems*. In: 18th IEEE Internacional Conference on Adavanced Information Networking and Applications (AINA), Fukuoka, Japão, 2004.

[8] Lohse, M., Repplinger, M., and Slusallek, P. *An Open Middleware Architecture for Network-Integrated Multimedia*. IDMS/PROMS'2002, Coimbra, Portugal, 2002.

[9] Lohse, M. Network-Integrated Multimedia Middleware. PhD These. Universitat des Saarlands.Saarbrucken. Germany. 2005.

[10] Herman, I. and Reynolds, G. PREMO: An Emerging Standard for Multimedia Presentation. In IEEE Multimedia Systems, Vol. 6, 1998.

[11] Pesce, M., *Programming Microsoft DirectShow for Digital Video and Television*. Microsoft Press. 2003.