

**Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação**

**Uma abordagem na camada de *middleware* para troca
dinâmica de componentes em sistemas multimídia
distribuídos baseados no *framework Cosmos***

Ivanilson França Vieira Júnior

**Natal
Março de 2009**

Catálogo da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial Especializada
Centro de Ciências Exatas e da Terra – CCET.

Vieira Júnior, Ivanilson França.

Uma abordagem na camada de middleware para troca dinâmica de componentes em sistemas multimídia distribuídos baseados no framework cosmos / Ivanilson França Vieira Júnior. – Natal, 2009.

82 f. : il.

Orientador: Prof. Dr. Adilson Barboza Lopes.

Tese (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada. Programa de Pós-Graduação em Sistemas e Computação.

1. Multimídia – Tese. 2. Middleware – Tese. 3. Engenharia de software – Tese.
I. Lopes, Adilson Barboza. II. Título.

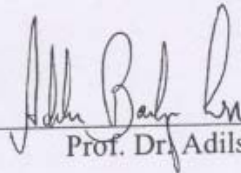
RN/UF/BSE-CCET

CDU: 004.4'27

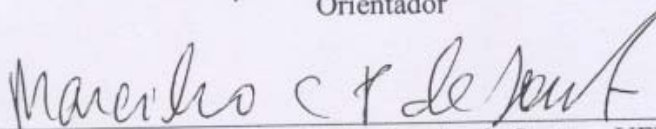
IVANILSON FRANÇA VIEIRA JÚNIOR

UMA ABORDAGEM NA CAMADA DE MIDDLEWARE PARA TROCA DINÂMICA DE COMPONENTES EM SISTEMAS MULTIMÍDIA DISTRIBUÍDOS BASEADOS NO FRAMEWORK COSMOS.

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

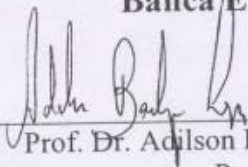


Prof. Dr. Adilson Barboza Lopes – UFRN
Orientador

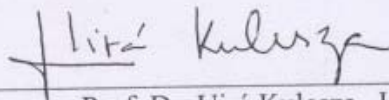


Prof. Dr. Marcílio Carlos Pereira de Souto – UFRN
Vice-Coordenador do Programa

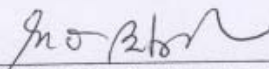
Banca Examinadora



Prof. Dr. Adilson Barboza Lopes – UFRN
Presidente



Prof. Dr. Uirá Kulesza – UFRN



Prof. Dr.ª Thaís Vasconcelos Batista – UFRN



Prof. Dr.ª Maria da Graça Campos Pimentel – USP

Março, 2009

Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação

**Uma abordagem na camada de *middleware* para troca
dinâmica de componentes em sistemas multimídia
distribuídos baseados no *framework Cosmos***

Tese de mestrado submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do grau de Mestre em Sistemas e Computação (MSc.).

Ivanilson França Vieira Júnior

Prof. Dr. Adilson Barboza Lopes
Orientador

Natal, Março de 2009

Este trabalho é dedicado à todos que me ajudaram a chegar até aqui, Nara, meus pais,
Adilson e meu amigos Carlos e Marjory.

AGRADECIMENTOS

Queria inicialmente agradecer a Deus, pois sem a sua força eu nem teria começado, segundo, a minha esposa Nara pois, como eu acredito, o amor verdadeiro é um sentimento que a cada dia só cresce e sem o suporte dela eu não conseguiria, a minha mãe Dulce, meu pai Ivanilson, meu irmão Ivanilton e minha irmã Ilana.

Ao professor Adilson, que me orientou e não deixou em nenhum momento que eu me desmotivasse, sempre contribuindo e ajudando no trabalho. Mesmo a distância, a sua dedicação e preocupação foram exemplares, um verdadeiro pai, exemplo e guia nesse processo de aprendizagem.

Ao grupo do Cosmos com que eu trabalhei, eu só tenho os mais sinceros agradecimentos. Carioca, Daniel, Felipe, Gustavo, Magaiver e Madson. Em especial Daniel que construiu o AnaMoC e contribuiu bastante para este trabalho.

Ao casal Carlos e Marjory, que mesmo distantes sabem que eles que me incentivaram a me inscrever no programa e a aceitar o desafio proposto, como também me ajudaram a conhecer esse lado fantástico da academia.

Aos meus amados e queridos amigos do CEFET, que me proporcionaram momentos memoráveis, conversando e tomando um café, professores: Fellipe, Minora, Jorgiano, Robinson, Erica, Alex, Bruno Pontes. Mas não esqueço do ano de 2004, durante a disciplina de Estruturas Discretas... prof. Ribamar chegou no final de uma aula onde eu apresentei um seminário e com sua sutileza habitual disse: "... Você devia pensar em ser professor, você leva jeito para isso...", desde então invisto nesta carreira.

Aos moradores da Natalnet (Agora LaSIC), Bruno, Tadeu, Alba, Camila que me proporcionaram momentos agradáveis... e também aos frequentadores esporádicos, Raphael, Diego, Aninha, Senegal, Daniel Sabino, dentre outros.

Aos servidores do DIMAp, não esquecendo das professoras Thais e Flávia que contribuíram bastante para o trabalho, participando da minha banca de qualificação; especialmente a professora Thais pelo apoio dado na ausência do professor Adilson.

A CAPES por ter me dado o apoio financeiro por 15 meses, onde me deu a tranquilidade para desenvolver esse trabalho.

Resumo

Para tratar a complexidade associada ao gerenciamento dos sistemas multimídia distribuídos, uma solução deve incorporar conceitos de middleware de forma a abstrair especificidades de hardware e sistemas operacionais. Aplicações nesses sistemas podem ser executadas em diferentes tipos de plataformas, e os componentes destes sistemas precisam interagir uns com os outros. Devido à variabilidade dos estados das plataformas de execução, uma abordagem flexível deve permitir a troca dinâmica de componentes visando garantir o nível de QoS da aplicação em execução. Neste contexto, o presente trabalho apresenta uma abordagem na camada de middleware para a realização de troca dinâmica de componentes no contexto do framework Cosmos, iniciando com a escolha do componente alvo para a troca, passando pela tomada de decisão de qual, entre os componentes candidatos, será escolhido e concluindo com o processo definido para a troca. A abordagem foi definida com base nos requisitos de QoS considerados no framework Cosmos de maneira a suportar a reconfiguração dinâmica de componentes.

Área de Concentração: Multimídia

Palavras-chave: Multimídia; Engenharia de Software; Componentes; Troca de Componentes; Engenharia de Componentes.

Abstract

To manage the complexity associated with the management of multimedia distributed systems, a solution must incorporate concepts of middleware in order to hide specific hardware and operating systems aspects. Applications in these systems can be implemented in different types of platforms, and the components of these systems must interact each with the other. Because of the variability of the state of the platforms implementation, a flexible approach should allow dynamic substitution of components in order to ensure the level of QoS of the running application . In this context, this work presents an approach in the layer of middleware that we are proposing for supporting dynamic substitution of components in the context the Cosmos framework , starting with the choice of target component, rising taking the decision, which, among components candidates will be chosen and concluding with the process defined for the exchange. The approach was defined considering the Cosmos QoS model and how it deals with dynamic reconfiguration .

Area of Concentration: Multimedia

Key words: Multimedia; Software Engineer; Software components; Substitution of Components; Components Engineer.

Sumário

1	Introdução	1
1.1	Objetivos	4
1.2	Resultados Obtidos	4
1.3	Organização do Trabalho	5
2	Modelos e Tecnologias para Suporte à Adaptação Dinâmica em Sistemas Multimídia Distribuídos	6
2.1	Adaptação em Sistemas Multimídia Distribuídos	6
2.2	Componentes de Software	7
2.2.1	Definição de Componente	8
2.2.2	Desenvolvimento Baseado em Componentes	9
2.2.3	Computação Reflexiva	10
2.3	QoS (Qualidade de Serviço)	11
2.4	O <i>Framework</i> Cosmos	12
2.4.1	Modelo de Componentes do Cosmos	13
2.4.2	Modelo de Recursos Virtuais	15
2.4.3	Modelo de Interconexão de Componentes	17
2.4.4	Arquitetura Geral de um Sistema Baseado no Cosmos	18
2.4.5	Modelo de QoS	21

2.4.6	ADL definida para o Framework Cosmos	22
2.4.7	O <i>Middleware</i> AdapTV	23
2.5	Considerações Finais	23
3	A Abordagem proposta para troca dinâmica de componentes em sistemas multimídia distribuídos baseados no <i>framework</i> Cosmos	24
3.1	Introdução	24
3.2	Modelo de Adaptação Dinâmica do Cosmos	25
3.2.1	Arquitetura Geral da Abordagem Inicial	25
3.2.2	Arquitetura Geral da Abordagem Proposta	28
3.3	Extensão da ADL do Cosmos	29
3.4	Seleção de Componentes	32
3.4.1	Identificação de Critérios de QoS para Análise de Componentes Candidatos	33
3.4.2	Classificação de Critérios	39
3.4.3	Modelo para Definição de Critérios	41
3.4.4	AnaMoC - Um Módulo para Análise e Seleção de Componentes	43
3.5	Mecanismo de Troca	49
3.5.1	Implantação da Proposta no Framework Cosmos	50
4	Trabalhos Relacionados	59
4.1	Frameworks para o desenvolvimento de aplicações Multimídia	59
4.1.1	PREMO	60
4.1.2	NMM	60
4.1.3	O Sistema de Componentes SATIN	61
4.1.4	O Framework QDD (Quality-Driven Delivery)	63

4.1.5	Comparativo	64
4.2	Seleção de Componentes	65
4.2.1	Método OTSO	65
4.2.2	Método PORE	66
4.2.3	CARE/SA	67
4.2.4	Component Rank	68
5	Considerações Finais	70
	Appendices	80
A	XSD	80

Lista de Figuras

2.1	Uma visão de sistemas reflexivos.	10
2.2	Interfaces do Componente CosmosComponent.	14
2.3	Principais elementos arquiteturais do Cosmos.	16
2.4	Relacionamento entre os Elementos	18
2.5	Arquitetura de Alto Nível do Cosmos.	19
2.6	Arquitetura geral de um middleware baseado no framework Cosmos.	20
2.7	Visão geral do modelo de QoS do Cosmos.	21
3.1	Arquitetura para Adaptação dinâmica no Cosmos com duplicação de canais de comunicação.	26
3.2	Diagrama de atividades relativo a um processo de adaptação no <i>framework</i> Cosmos	27
3.3	Relacionamento entre os Elementos.	29
3.4	Tempo médio de uma adaptação local com adaptações realizadas a cada 10 mensagens enviadas.	35
3.5	Tempo médio de uma adaptação remota com adaptações realizadas a cada 10 mensagens enviadas.	35
3.6	Tempo de Adaptação vs Tamanho do Buffer Cenário Local.	37
3.7	Tempo de Adaptação vs Tamanho do Buffer Cenário Remoto.	37
3.8	Tempo de Adaptação vs Velocidade do Processador Cenário Local.	38
3.9	Tempo de Adaptação vs Velocidade do Processador Cenário Remoto.	39

3.10	Modelo para definição de critérios e políticas de avaliação e seleção de componentes	42
3.11	Cenário de uso para a escolha de um componente decodificador de fluxo de vídeo	44
3.12	Componente AnaMoC	47
3.13	Interface <i>IAnamoc</i>	47
3.14	Opção 1 de uso do AnamoC	48
3.15	Opção 2 de uso do AnamoC	49
3.16	Interface IContext	54
3.17	Visão Funcional do Modelo de Interconexão.	55
3.18	Diagrama de Seqüência da Abordagem para a Troca de Componentes. . .	57
4.1	Visão Geral do processo do CARE/SA	68
4.2	Arquitetura do Component Rank	69

Lista de Tabelas

3.1	Tamanhos de buffers utilizados no experimento	36
3.2	Faixa de valores para o critério de Skew.	41
3.3	Faixas de valores dos critérios de rede. A última coluna mostra os valores dos mesmos para o componente.	42
3.4	Componentes e faixas possíveis	53
4.1	Comparativos entre os frameworks multimídia	65

Lista de Códigos-fonte

3.1	Trecho de código onde somente um componente era originalmente especificado.	29
3.2	Trecho de código da nova versão, onde existem dois componentes especificados e definidos na aplicação como possíveis provedores de fluxo de vídeo.	30
3.3	Trecho de código da especificação de uma política.	44
3.4	XML representando componentes.	46
3.5	XML representando componentes após a troca do AnaMOC.	50
3.6	Trecho de Código da especificação de QoS de uma aplicação.	51
A.1	XML Schema definindo os elementos propostos e incorporadas a ADL do Cosmos.	80

Glossário

- ADL *Architecture Description Language*.
- AnaMoC: *Analyser Module for Component Selection*
- CPU: *Central Processing Unit*
- COTS: *Commercial Off-The-Shelf*.
- DBC: Desenvolvimento Baseado em Componentes.
- ES: Engenharia de Software.
- ESBC: Engenharia de Software Baseada em Componentes.
- GHz: Giga Hertz.
- GPL: *General Public License*.
- ISO: *International Organization for Standardization*.
- LGPL: *Lesser General Public License*.
- MHz: Mega Hertz.
- MoP: *Metaobject Protocol*.
- QDD: *Quality-Driven Delivery*
- QoS: *Quality of Service*.
- RAM: *Random Access Memory*.
- WebMedia: *Brazilian Symposium on Multimedia and the Web*
- WTA: *Workshop on Tools and Applications*
- XML: *eXtensive Markup Language*.

Capítulo 1

Introdução

Atualmente, tem-se observado uma demanda de uso crescente de áudio e vídeo em aplicações multimídia distribuídas. Uma das características principais de uma aplicação multimídia é que ela, diferentemente de uma aplicação tradicional, tem que gerenciar o processamento integrado de múltiplas mídias, e, como descrito por [1], para gerenciar elementos áudio, vídeo, imagem, gráficos e animações há a necessidade de tratar questões complexas associadas por exemplo com: a distinção entre fluxos orientados a dados e orientados a controle; ciclos de vida para objetos de processamento de mídia e para elementos de comunicação; composição estática e dinâmica de componentes; semânticas específicas de domínio para os vários tipos de elementos envolvidos.

Sistemas multimídia distribuídos são caracterizados por utilizarem computadores distribuídos ao longo de uma rede, implicando dessa forma em tráfego de informações multimídia. Esses sistemas envolvem tipicamente recursos específicos de hardware (placas de captura de áudio e vídeo, por exemplo), bem como software para codificação, processamento, transmissão, decodificação e apresentação de dados de mídia. Estes sistemas são complexos porque, além dos aspectos temporais envolvidos com as mídias áudio e vídeo das aplicações, deve-se considerar que eles poderão ser executados em diversos tipos de dispositivos, com capacidades variadas de processamento e armazenamento, e diferentes interfaces com o usuário. Aliado a isso, o sistema precisa lidar com as limitações e variações da qualidade dos meios físicos de transmissão que ainda estão aquém das necessidades, cada vez mais exigentes dessas aplicações.

Estes fatos impõem a adoção de diferentes interfaces de interação, dependendo do tipo de dispositivo e da tecnologia de acesso empregada. Como exemplo, podem-se citar os set-top boxes, que são dispositivos atualmente utilizados para receber e decodificar

siniais de TV. Estes dispositivos incorporam conceitos e mecanismos dos sistemas operacionais modernos, provendo assim interfaces para uso de serviços comuns como acesso à memória, discos rígidos e interfaces de rede.

Em adição aos serviços de televisão convencional, estes dispositivos permitem a exibição de conteúdos de mídia armazenada localmente, ou a execução de programas distribuídos juntamente com o fluxo multimídia. Com estas facilidades, um telespectador pode, por exemplo, realizar interações com a estação produtora / transmissora de conteúdo de TV, de forma a estabelecer um canal de interatividade. Outra questão que gera complexidade no desenvolvimento de sistemas multimídia é a sincronização de fluxos. O problema consiste em assegurar que os dados de cada fluxo trafeguem numa cadência tal que possam ser exibidos no tempo esperado (sincronização intrastream), e mais ainda, que sejam sincronizados fluxos inter-relacionados (sincronização interstream). Desta forma, a sincronização neste contexto envolve relações temporais entre dados de mídia contínua. Na sincronização intrastream, a relação temporal está associada com os elementos do próprio fluxo, como por exemplo, os quadros de um fluxo de vídeo. Normalmente, esta relação é expressa em número de quadros por segundo, implicando em manter a taxa e reduzir o jitter (variação estatística do retardo).

Na sincronização interstream a relação temporal estabelece uma vinculação entre elementos de diferentes fluxos com o objetivo de minimizar o deslocamento entre referências temporais associadas. Um exemplo clássico consiste na sincronização labial envolvendo fluxos de áudio e vídeo. Uma aplicação multimídia distribuída precisa lidar com essa variedade de requisitos provendo suporte e capacidade de adaptação dinâmica considerando principalmente os requisitos de QoS (Qualidade de Serviço) das aplicações. O suporte à adaptação dinâmica e gerenciamento de QoS pode ser considerado como um requisito essencial para um sistema multimídia ser aceito em um mundo cada vez mais competitivo.

Atualmente, a maioria dos trabalhos na área de Engenharia de Software trata a adaptação de uma forma geral. Ou seja, questões associadas a domínio específicos precisam ser analisadas. Nesta direção, Lopes em [2], e Silva em [3], têm realizado um esforço no sentido de encontrar soluções em nível arquitetural, na camada de middleware, para dar suporte ao desenvolvimento de sistemas multimídia flexíveis, com capacidade de se adaptarem dinamicamente às possíveis variações do ambiente e dos requisitos dos usuários. Como resultado desses esforços foram propostos e desenvolvidos o Framework Cosmos [2] e o middleware AdapTV [3], um middleware baseado no Cosmos para sistemas de Televisão Digital. A definição da arquitetura e da estrutura de implementação do AdapTV foram direcionadas para o suporte à interconexão de componentes.

A solução proposta e implementada por Silva[3] tratou a questão da adaptação de

QoS, buscando manter o sistema com um nível aceitável de funcionamento a partir da troca propriedades internas de seus componentes. Entretanto, existem situações onde a simples troca de propriedades não é suficiente para atender aos requisitos da aplicação. Por exemplo, em alguns casos existe a necessidade de substituição de componentes internos da aplicação, seja pelo fato da mudança de um protocolo, ou até mesmo do modo de exibição.

Um modo simples para realizar uma adaptação (substituição de componentes) em uma aplicação poderia ser parar a execução da aplicação por completo, proceder a substituição do componente, onde muitas vezes se faz necessário realizar novamente um processo de compilação, e, após essas etapas, reiniciar a aplicação. Uma adaptação deste tipo seria inviável para a grande maioria das aplicações multimídia. Nesse contexto, o presente trabalho dá continuidade ao projeto Cosmos, tendo como foco a adaptação com a substituição dinâmica dos componentes.

A abordagem proposta define critérios flexíveis para serem usados na seleção de componentes substitutos levando em consideração o modelo de QoS do framework Cosmos (introduzido na Capítulo 2). Dessa forma, o processo de substituição precisa estabelecer um relacionamento e mapeamento entre os requisitos de QoS dos componentes definidos no modelo arquitetural do sistema e o mecanismo de seleção e substituição dinâmica, considerando o estado e os requisitos atuais do sistema. Outra questão que a literatura tem discutido recentemente e que deve ser considerada nas abordagens para sistemas multimídia distribuídos está relacionada com os requisitos de auto-adaptação. A questão da auto-adaptação deve ser levada em conta nesses sistemas dada a as variações de carga, requisitos e a mobilidade de componentes envolvidos nesses sistemas, que podem provocar mudanças na Qualidade dos Serviços oferecidos, e, em consequência, ter que dinamicamente realizar mudanças nos requisitos e/ou na arquitetura do sistema.

Abordagens para identificar mudanças no comportamento do sistema que requeiram a adaptação são feitas normalmente por um processo (interno ou externo) que deve continuamente monitorar o sistema. A questão da auto-adaptação no contexto do presente trabalho é considerada através da utilização de um módulo desenvolvido para seleção dinâmica e automática de componentes. Assim, para que a substituição de um componente possa ser feita, é preciso realizar um processo de preparação, que envolve a definição de critérios para a escolha de componentes substitutos (podendo ser estática ou dinâmica), a escolha do componente (que também pode ser estática ou dinâmica), e, por fim, concluindo com a substituição efetiva (que, neste caso precisa ser dinâmica).

A presente abordagem trata estas questões de forma integrada aos conceitos do framework Cosmos, com o objetivo de aumentar a flexibilidade do framework relacionada

com os requisitos de reconfiguração dinâmica de sistemas.

1.1 Objetivos

O principal objetivo desta proposta consiste em definir um processo para a realização de substituição dinâmica de componentes em plataformas baseadas no framework Cosmos e definir um mecanismo de suporte para a efetivação da substituição dinâmica de componentes.

Como objetivos secundários temos:

- Estender o Cosmos, de forma a permitir a especificação de um conjunto de componentes candidatos a fornecer o mesmo serviço dentro da aplicação;
- Propor um conjunto de critérios de QoS a serem utilizados como base para a definição do processo de escolha de componentes substitutos; e,
- Definir a estrutura de implementação para o processo de escolha e substituição de componentes

1.2 Resultados Obtidos

O ponto de partida que serviu de motivação para o presente trabalho foram as limitações do framework Cosmos com relação à adaptação dinâmica. Nessa direção, o trabalho foi realizado com uma perspectiva de estender o Cosmos com novas capacidades. De forma a tornar a proposta viável, foram definidos critérios para realizar o processo de adaptação dinâmica. Especificamente, o presente trabalho se concentrou no processo de troca dinâmica de componentes, onde foram definidos, a partir de uma análise experimental, alguns critérios para a escolha de componentes substitutos, os quais são utilizados para ajustar o comportamento do sistema, mediante possíveis situações de falhas de QoS. A partir desses critérios, o trabalho propôs um mecanismo, no contexto do framework Cosmos, para realizar a troca dinamicamente.

1.3 Organização do Trabalho

Este trabalho está organizando da seguinte maneira: no Capítulo 1 apresentamos esta introdução, motivação e demais considerações relacionadas ao contexto do trabalho; o Capítulo 2 apresenta uma visão geral envolvendo abordagens para suporte a adaptação dinâmica em sistemas multimídia distribuídos; o Capítulo 3 apresenta e discute os principais pontos do mecanismo; o Capítulo 4 relaciona e disserta sobre alguns trabalhos relacionados, e, por fim, o capítulo 5 conclui trazendo algumas considerações finais.

Capítulo 2

Modelos e Tecnologias para Suporte à Adaptação Dinâmica em Sistemas Multimídia Distribuídos

Este capítulo discute algumas abordagens e conceitos usados em soluções que visam dar suporte à adaptação dinâmica em sistemas multimídia distribuídos. O Capítulo inicia ressaltando a importância da utilização de abordagens para adaptação dinâmica e auto-adaptação para o domínio desses sistemas. Em seguida, são discutidas algumas propostas da área de engenharia de software para dar suporte à adaptação dinâmica. Por fim, o capítulo apresenta o Framework Cosmos, um framework para gerenciamento e configuração de componentes de sistemas multimídia distribuídos. O Cosmos é o resultado do esforço do grupo de pesquisa onde o presente trabalho está inserido. O capítulo termina apresentando algumas considerações finais destacando, no contexto do framework Cosmos, onde está a contribuição do presente trabalho, que consiste na definição de um processo para realização de substituição dinâmica de componentes com base nos requisitos de QoS definidos pelo framework Cosmos.

2.1 Adaptação em Sistemas Multimídia Distribuídos

O conceito de adaptação no contexto de auto-ajuste de código é bastante antigo. Esta idéia já estava presente no projeto do Computador ENIAC ao explorar o conceito de programa armazenado de Von Neuman. No modelo de Neuman, tanto instruções quanto dados podiam ser armazenados em memória. Isto permitia que instruções pudessem ser

trocadas rapidamente de forma que novos programas pudessem ser executados, e mais ainda, que programas pudessem gerar novos programas.

Recentemente, algumas abordagens clássicas que dão suporte à adaptação de sistemas, foram discutidas por McKinley [4, 5], onde são relacionadas as principais tecnologias e analisados aspectos importantes como a flexibilidade relacionada ao momento da adaptação (tempo de projeto, de implantação ou execução). Ainda, segundo McKinley, as principais tecnologias exploradas atualmente para dar suporte a adaptação, são as seguintes:

- Separação de aspectos da computação por interesses (concerns);
- Computação reflexiva; e,
- Projeto baseado em componentes.

Dado que o presente trabalho está focado na troca dinâmica de componentes no Cosmos, nossa discussão está concentrada nos aspectos de adaptação baseada na tecnologia de componentes associadas ao conceito de computação reflexiva.

2.2 Componentes de Software

O conceito de componentes não é uma idéia muito recente; já no final de década de 60, McIlroy [6], um dos primeiros pesquisadores que levantou a questão sobre o desenvolvimento de componentes, vislumbrava a possibilidade de reutilização de componentes em várias aplicações, podendo estes serem desenvolvidos por outros desenvolvedores, ou pela própria empresa.

Entretanto, apenas recentemente os modelos de desenvolvimento baseados em componentes de software têm se encaminhado para níveis de amadurecimento que viabilizam o seu uso em diversas áreas. A idéia tem sido explorada por várias abordagens para construir programas adaptativos obedecendo a um modelo que dê suporte e consistência às operações de adaptação.

Seguindo esta linha evolucionária, na década de 70, o trabalho de [7] propôs e apresentou um paradigma modular para o desenvolvimento de sistemas, onde o sistema era desmembrado em partes, também chamadas de módulos; neste paradigma módulos eram produzidos separadamente e posteriormente interligados.

A seguir, um grande impulso foi observado na década de 80 com a introdução da orientação a objetos [8]; houve uma esperança de que um novo paradigma pudesse finalmente aumentar o reuso na camada do software. Trabalhos como o [9] chegaram a colocar a classe como uma unidade básica de reutilização. A partir desse momento foi notado que a reutilização de software não era uma coisa inerente exclusivamente ao paradigma de programação e sim de técnicas utilizadas para este fim.

2.2.1 Definição de Componente

Não existe na literatura um consenso sobre a definição de componentes; desta forma, iremos destacar algumas definições clássicas que são frequentemente referenciadas.

Um componente é descrito por Brown [10] como uma unidade funcional significativa de um sistema, podendo representar desde uma função de alto nível até uma tarefa de baixo nível. Podem-se classificar componentes como atômicos ou compostos. Bosch [11] diz que “Um componente é um pacote desenvolvido e testado separadamente, distribuído como uma unidade que pode ser integrada com outros componentes para construir algo com maior funcionalidade”. E ainda complementando, Bosch afirma que o componente é identificado pela sua interface, fazendo com que qualquer dispositivo de software possa ser um componente, contando que este tenha uma interface bem definida.

Em seu trabalho, Chambers [12] diz que os componentes são módulos ou estruturas de dados, desenvolvidos e compilados separadamente, e unidos para formar aplicações que interagem através de interfaces.

Uma abordagem interessante é a de Brown [10] que define que um componente é uma unidade funcional significativa de um sistema, podendo representar desde uma função de alto nível até uma tarefa de baixo nível. Pode-se classificar um componente como atômico ou composto. Portanto, a arquitetura de um sistema completo apresenta-se como uma organização hierárquica de componentes compostos ou atômicos. Além disso, define-se também componentes como unidades reutilizáveis e compartilhadas, podendo ser utilizadas em diversos projetos.

Embora não exista uma definição clássica sobre componentes, a grande maioria dos autores definem que o ponto principal do componente é a sua interface, que será por onde esse componente será utilizado. Dessa forma, para uma possível adaptação envolvendo a troca de componentes, bastaria que o novo componente oferecesse a mesma interface, ou uma interface com serviços compatíveis.

Com isso, surgiu a Engenharia de Software Baseada em Componentes; um dos primeiros trabalhos que apresentou esta definição foi o de Council [13] no ano de 2001.

2.2.2 Desenvolvimento Baseado em Componentes

Resende [14] define o Desenvolvimento Baseado em Componentes (DBC) como uma abordagem de criação de sistemas de software em que todos os artefatos (do código-fonte até as especificações das interfaces, arquiteturas e modelos de negócios) podem ser construídos, montando-se, adaptando-se e interconectando-se componentes existentes em diversas configurações. Nessa abordagem, alguns componentes devem ser intencionalmente desenvolvidos, enquanto outros descobertos e adaptados.

Com isso existe um esforço dentro da ES de se amadurecer o DBC para chegar próximo aos níveis de reutilização da área de hardware, onde os componentes são amplamente reutilizados para a criação de novos produtos e equipamentos.

Abordagens como [15] estudam as vantagens de se utilizar o DBC para a criação de produtos; no mais, percebe-se ainda que o tempo de desenvolvimento pode ser reduzido, como também, o tempo de vida do software.

Ainda relacionado sobre o DBC, a abordagem de [16] define um modelo de estruturação de componentes para sistemas baseados em arquiteturas; nesta abordagem também define-se um mapeamento para materializar componentes, conectores e interações, descritos arquiteturalmente, em elementos do modelo de objetos.

O nível de flexibilidade de um sistema poderá ser incrementado se o processo de montagem de componentes requeridos for retardado de modo a ser realizado em tempo de execução, sob demanda, à medida que as necessidades forem identificadas. Esta forma de abordagem incremental permite por exemplo, a inclusão de novas características e funcionalidades eventualmente não identificadas em tempo de projeto. Por outro lado, apesar desta abordagem proporcionar níveis altos de flexibilidade, retardar o processo poderá introduzir problemas relacionados ao gerenciamento da consistência da aplicação.

Na análise apresentada por Emmerich [17] e [18] são exploradas algumas idéias envolvendo a introdução de novos componentes ao modelo em tempo de execução. Nessas análises são mencionadas várias preocupações relacionadas com o nível de estabilidade da arquitetura para o tratamento consistente dos requisitos não-funcionais. Por exemplo, quando a composição utilizada na montagem do componente ocorre na fase de desenvolvimento, ou seja, em tempo de compilação, ou mesmo em tempo de implantação do componente, o dinamismo, apesar de limitado, é seguro, no sentido de que permite o sistema

analisar a consistência da aplicação observando as características e restrições dos modelos de componentes e de configuração definidos previamente. Desta forma, uma solução de compromisso que oferece bons níveis de flexibilidade consiste em definir componentes com características configuráveis, onde os níveis de variabilidade dessas características são estabelecidos em tempo de projeto, com possibilidades de escolha de propriedades para estas características no momento da implantação ou da instanciação dos componentes, bem como dinamicamente, de acordo com as necessidades da aplicação.

2.2.3 Computação Reflexiva

O conceito de computação reflexiva refere-se à habilidade de permitir que a aplicação possa consultar e eventualmente alterar o seu próprio comportamento. Estas idéias são abordadas por [19, 20, 21] juntamente com os conceitos de reificação e metaprogramação. O conceito de reificação está associado à preservação de informações de compilação do componente de modo a permitir o seu uso dinamicamente pelo sistema. A técnica de suporte ao uso e modificação destas informações é chamada por Szypersky de metaprogramação.

A Figura 2.1, utilizada em [2], mostra um modelo que explora o conceito de reflexividade. No plano inferior, a figura representa componentes de nível básico, enquanto que no plano superior, a figura mostra metacomponentes onde são descritas as propriedades dos respectivos componentes de nível básico.

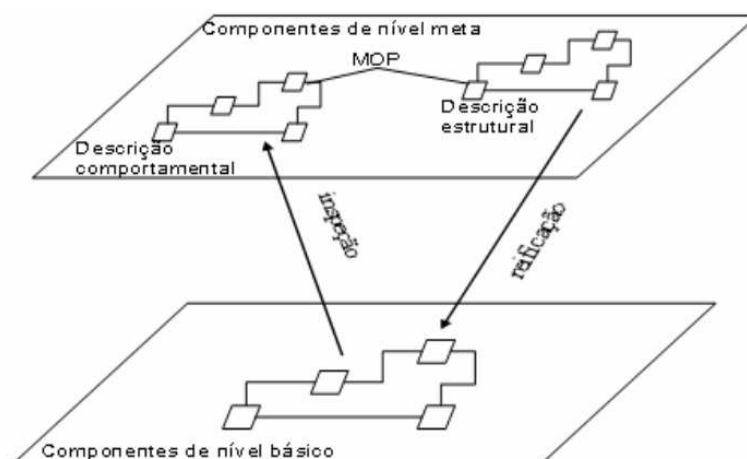


Figura 2.1: Uma visão de sistemas reflexivos.

Um modelo reflexivo normalmente envolve dois tipos de operação: introspecção, onde o sistema ou a própria aplicação inspecionam o seu comportamento, e reificação, onde

alterações nos componentes básicos do sistema podem ser realizadas de acordo com as descrições nos respectivos metacomponentes.

O conceito de protocolo de metaobjetos (MOP - sigla oriunda do termo *Metaobject Protocol* em inglês) [22], consiste na definição de interfaces que permitem a realização de operações para introspecção e atualização de propriedades de metacomponentes.

Estas operações normalmente podem tratar aspectos estruturais, como por exemplo relação de dependência, hierarquia de classes e localização de componentes, assim como aspectos comportamentais, como o conjunto de interfaces providas por um componente e os protocolos de comunicação suportados.

2.3 QoS (Qualidade de Serviço)

A QoS é um conceito usualmente empregado em redes para expressar os níveis de qualidade relacionados com: *delay*, perdas, etc. Os sistemas multimídia distribuídos são fortemente afetados por esses problemas.

Para dar suporte à adaptação dinâmica, o sistema deve permitir a realização de ajustes no seu comportamento em resposta, por exemplo, a mudanças dos requisitos da aplicação ou do ambiente, capturados pelo processo de monitoramento das condições do ambiente e de mudanças de requisitos do sistema. Existem várias situações em que adaptações poderão ocorrer, incluindo, entre outras: mudanças nos parâmetros de QoS da aplicação, mudanças nos parâmetros de QoS da rede, inclusão de novos componentes, detecção de desvios da QoS estabelecida e solicitação de adaptação originada por iniciativa da própria aplicação.

Mudanças originadas por decisão do sistema estão normalmente relacionadas a variações de QoS, ou à indisponibilidade temporária de recursos ou a adaptações pré-programadas.

Em particular, as questões de QoS são críticas no contexto dos sistemas multimídia distribuídos. Em razão disso, elas são vastamente discutidas na literatura; entretanto, o suporte ao gerenciamento desses aspectos ainda se constituem em desafios no contexto dos modelos de desenvolvimento baseados em componentes.

A qualidade de serviço está fortemente relacionada com políticas de gerenciamento de recursos [23, 24]. Os recursos necessários à execução de serviços multimídia envolvem tipicamente dispositivos de captura e exibição, bem como recursos de sistema (memória, CPU, rede, etc.) que normalmente são interconectados adotando um estilo arquitetural do

tipo pipe-filter [24, 25]. É possível que estes recursos sejam compartilhados, o que pode ocasionar situações de conflito. Cabe à política de gerenciamento de recursos antecipar-se e solucionar essas situações. É tarefa do sistema operacional fornecer mecanismos para gerenciamento de recursos, porém esta abordagem pode ser considerada de baixo nível.

A gerência sobre esses recursos pode, no entanto, ser tratada em um nível mais alto de abstração através de serviços de middleware (camada de software entre o sistema operacional e a aplicação). Tais serviços propiciam à aplicação simplicidade e transparência no uso e compartilhamento de recursos. A representação lógica de um recurso do sistema é freqüentemente denominada no middleware como recurso virtual [26], ao passo que a interconexão entre recursos, por conexão virtual [3].

A caracterização dos requisitos de QoS de uma aplicação depende das características das mídias envolvidas e dos requisitos dos usuários. É natural que exista uma variação inerente à sensibilidade de cada indivíduo. Isto implica que o mesmo sistema instanciado em diferentes contextos pode se comportar de formas diferentes. No entanto, para a execução de um serviço de qualidade, existe um limite mínimo de requisitos de apresentação cuja não obediência causa desconforto a qualquer indivíduo, bem como um limite superior a partir do qual é indiferente. Desta forma, o gerenciamento de QoS deve levar em conta o conceito de faixas de valores para associar a critérios de gerenciamento de QoS. Além disso, ele precisa verificar se há disponibilidade dos recursos necessários à execução dentro desses limites, o que nem sempre é possível, uma vez que os sistemas multimídia distribuídos estão sujeitos a sobrecargas, falhas na rede e à própria capacidade dos recursos no local de destino.

2.4 O *Framework* Cosmos

Considerando que o presente trabalho foi desenvolvido no escopo do framework Cosmos, cujo esforço foi direcionado para o desenvolvimento de um processo de suporte à troca dinâmica de componentes, faz-se necessário apresentar as principais características do framework.

O framework Cosmos é um framework genérico baseado em componentes que poderá ser usado para o projeto e desenvolvimento da camada de middleware de uma variedade de sistemas multimídia distribuídos. Considerando que a maioria das vezes que um sistema precisa realizar dinamicamente adaptações em sua arquitetura e/ou comportamento deve-se a problemas associados com possíveis flutuações da QoS da rede, a presente introdução está focada no modelo de interconexão do Cosmos [3].

Para dar suporte à adaptação dinâmica o Cosmos define um modelo arquitetural para gerenciamento dinâmico de configuração de componentes e recursos de sistemas. Este modelo foi definido com foco na visão arquitetural do sistema, onde componentes abstratos incorporam as funcionalidades básicas dos elementos arquiteturais de um sistema multimídia distribuído. A idéia consiste em utilizar estes elementos abstratos para definição de extensões e instâncias de acordo com as necessidades e contextos particulares do sistema alvo. A essência do framework consiste na definição dos modelos de recursos virtuais, interconexão de recursos e de gerenciamento de QoS.

O modelo de interconexão do Cosmos foi proposto para tratar os aspectos envolvidos com a comunicação entre componentes multimídia.

Uma instanciação do Cosmos foi realizada como prova de conceito para o framework. Esta instanciação resultou na definição de um middleware adaptativo para sistemas de televisão digital interativa denominado AdapTV [26, 3]. O AdapTV envolve os principais elementos e modelos definidos pelo framework Cosmos.

Nesta seção, são apresentadas as principais características dos modelos acima citados (recursos virtuais, interconexão e QoS do framework) delimitando a apresentação ao contexto do projeto Cosmos no estágio anterior ao desenvolvimento do presente trabalho. Dessa forma, nosso objetivo é deixar clara a apresentação do processo proposto para dar suporte à troca dinâmica de componentes no Cosmos (Capítulo 3), ao tempo que facilita a identificação da principal contribuição do trabalho dentro de um contexto mais amplo.

2.4.1 Modelo de Componentes do Cosmos

O Cosmos concentra seus esforços na representação e manipulação da diversidade de conceitos, requisitos e componentes relacionados com os sistemas multimídia. O framework definido consiste de um conjunto de componentes abstratos que podem ser estendidos, instanciados e customizados para contextos particulares, fornecendo uma visão arquitetural do sistema onde são abstraídos os detalhes de implementação desses componentes.

Com o objetivo de tratar os aspectos de generalidade e reusabilidade, o framework define um conjunto de interfaces e regras para gerenciamento e interação entre os diversos componentes do sistema. Como base para esta abordagem, o Cosmos define um componente abstrato, denominado CosmosComponent, com as interfaces básicas que devem ser providas obrigatoriamente por qualquer componente concreto do framework. A Figura 2.2 apresenta um modelo UML simplificado que descreve as interfaces que devem ser providas por qualquer componente Cosmos (componente abstrato CosmosComponent)

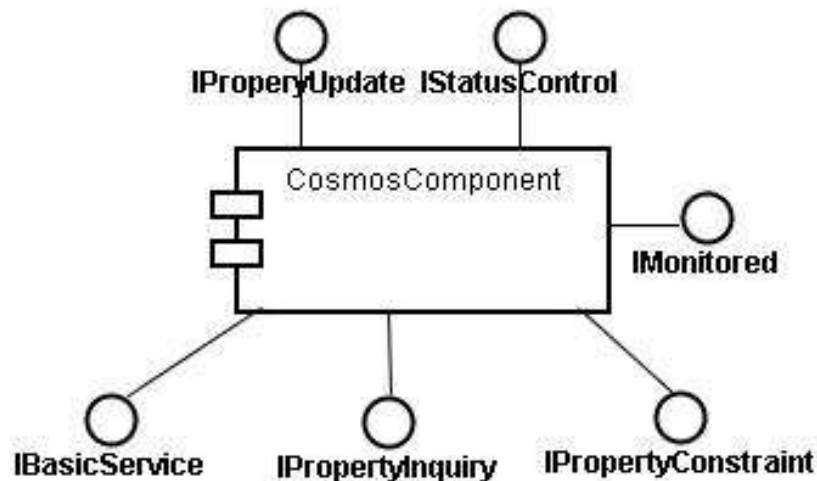


Figura 2.2: Interfaces do Componente CosmosComponent.

A decisão de definir um modelo de componentes próprio tem o objetivo de prover um modelo neutro que trate, de maneira independente, a diversidade de tecnologias existentes. Para tal, definiram-se um conjunto de interfaces, denominadas interfaces básicas, que são providas por todos os componentes do modelo. Dentre as interfaces deste conjunto encontram-se as interfaces de propriedades. Estas interfaces incorporam os conceitos de propriedades definidos e explorados em [27], provendo características de flexibilidade e adaptabilidade. Uma propriedade pode ser definida dinamicamente como um par de elementos (chave, valor), no qual a chave corresponde a uma cadeia de caracteres (string), usada, por exemplo, para identificar uma característica do componente, e valor pode ser uma instância de qualquer tipo de dados, utilizada para descrever a característica.

Explorando estes conceitos, diversos componentes podem ser configurados e customizados de maneira uniforme através da definição de valores específicos de propriedades, consistindo num mecanismo de suporte genérico para o gerenciamento de adaptação dinâmica. Além das interfaces de propriedades, as interfaces básicas contemplam operações que permitem o gerenciamento de recursos e ciclo de vida.

As interfaces de propriedades dão suporte às tarefas de configuração, gerenciamento e adaptação, definindo aspectos não-funcionais de componentes. Estas interfaces consistem em um mecanismo de suporte ao conceito de reflexividade [21], possibilitando a realização de consultas e definição de características de componentes. Desse modo, o sistema pode consultar e definir mudanças no comportamento dos seus componente através de operações das interfaces de propriedades.

As operações básicas definidas para a manipulação de propriedades foram agrupadas em diferentes interfaces do componente abstrato `CosmosComponent`, conforme ilustrado na Figura 2.2.

A interface `IBasicService` fornece operações para obtenção de informações básicas do componente, como por exemplo, as interfaces por ele providas. A interface `IpropertyInquiry` é definida para consultas de valores de propriedades. A interface `IPropertyConstraint`, fornece operações para a restrição de valores para efeito de configuração de uma determinada propriedade em uma instância do componente. Por exemplo, uma instância do componente pode restringir determinados valores de propriedades, indicando desta forma que estes valores não sejam considerados na operação do componente.

As interfaces de propriedades que foram introduzidas no componente básico `CosmosComponent` permitem a realização de mudanças nas propriedades dos componentes. Para permitir mudanças consistentes de configuração, o gerente do sistema deve supervisionar as mudanças verificando se a mudança é coerente com o estado atual da plataforma.

A interface `IStatusControl` oferece uma operação para gerenciamento e controle de mudança de estado do componente. Cada componente tem uma máquina de estados associada que determina os possíveis comportamentos e transições de estado válidas. A interface `IPropertyUpdate` é utilizada para a atualização de valores selecionados de propriedades dos componentes quando houver eventuais alterações em decorrência, por exemplo, de negociações dinâmicas, e para a ativação e desativação do componente.

2.4.2 Modelo de Recursos Virtuais

O conceito de recurso virtual, um tipo de componente instanciável do `Cosmos`, constitui-se numa das características fundamentais do framework. Recursos virtuais (`VirtualResources`) são componentes que efetivamente representam e gerenciam os elementos associados ao processamento de um fluxo de mídia, provendo abstrações para a representação de recursos de hardware ou software da plataforma. Além de implementar as interfaces básicas definidas pelo framework (`CosmosComponent`), eles podem definir outras interfaces, de modo a customizá-los de acordo com as funcionalidades do recurso efetivo que ele representa. A Figura 2.3 descreve o conceito de recurso virtual apresentando os principais relacionamentos envolvidos com os demais conceitos e elementos arquiteturais do `Cosmos`.

Um `VirtualResource` pode ter várias portas de comunicação, e cada porta tem um tipo associado que descreve as características do fluxo de dados que passa por ela. O fra-

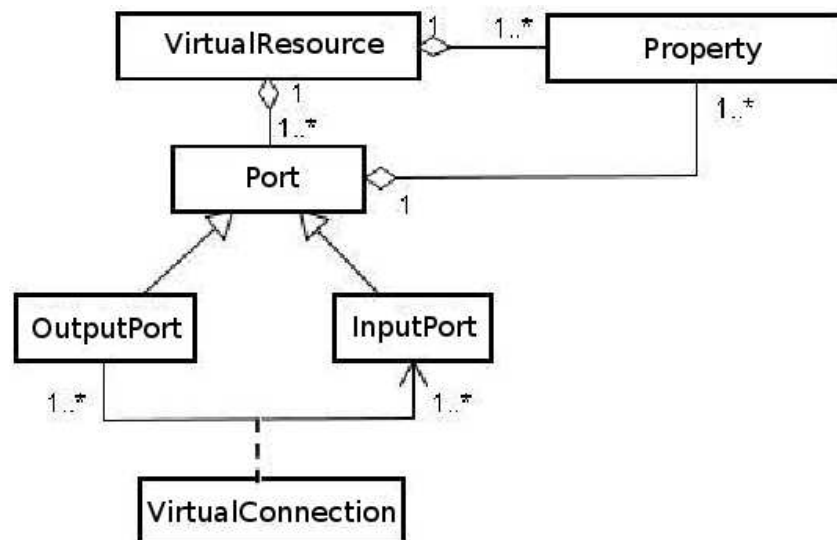


Figura 2.3: Principais elementos arquiteturais do Cosmos.

network Cosmos define diferentes tipos de interfaces a serem oferecidas pelas portas, cada uma com semântica distinta, como por exemplo, interfaces operacionais utilizadas pelos VirtualResources, caracterizando a porta como sendo do tipo porta de saída (OutputPort) ou porta de entrada (InputPort).

Portas de saída implementam interfaces que provêm suporte para comunicações que fluem do VirtualResource, enquanto que portas de entrada recebem fluxos destinados ao VirtualResource. No modelo simplificado da Figura 2.3, retirada de [2], foi introduzido um elemento Port para tratar os conceitos e comportamentos comuns às portas. As portas também são componentes, devendo possuir, além de suas interfaces operacionais, as interfaces básicas definidas pelo framework (CosmosComponent) que são utilizadas para fins de configuração, permitindo a definição, monitoramento ou alteração dinâmica de propriedades associadas às mesmas. Os tipos concretos de componentes OutputPort e InputPort definem respectivamente comportamentos para portas de saída e de entrada.

Os componentes locais ou remotos são interconectados através de ligações entre portas de comunicação. Estas ligações são gerenciadas por uma conexão virtual, denominada VirtualConnection (o modelo detalhado de uma conexão virtual é apresentado na seção 2.4.3 adiante). A conexão virtual provê uma interface bem definida para configuração e gerenciamento das ligações entre as portas, abstraindo os detalhes inerentes ao fluxo de dados multimídia. Um componente VirtualConnection não realiza a transferência dos dados do fluxo; seu papel é criar, configurar e gerenciar os recursos e componentes do middleware que realizam a comunicação. A configuração e o gerenciamento dos componentes que compõem os mecanismos de comunicação acontecem de forma transparente

pelo VirtualConnection.

2.4.3 Modelo de Interconexão de Componentes

O modelo de Interconexão de componentes para o framework Cosmos foi definido em [3] motivado pela necessidade de um mecanismo de comunicação específico para sistemas multimídia distribuídos que pudesse ser agregado pelo framework Cosmos. O objetivo foi definir um mecanismo de comunicação flexível que ofereça o suporte necessário à adaptação e reconfiguração dinâmica em sistemas multimídia distribuídos.

2.4.3.1 Conceitos Principais

Os componentes do modelo definido pelo framework Cosmos interagem uns com os outros, como origem ou destino de dados, através de portas de comunicação, utilizando-as para a troca de fluxo de dados multimídia. Para tratar o gerenciamento de interações, o framework define o conceito de conexão virtual.

Uma conexão virtual não é utilizada para a troca de dados efetivamente. Seu papel é servir de ponto de acesso para as operações de gerenciamento dos elementos que compõem uma interconexão, permitindo a abstração dos detalhes referentes à topologia de interconexão, QoS, sincronização e protocolos de comunicação.

As portas de comunicação são registradas em uma conexão virtual que se responsabiliza por seus respectivos gerenciamentos. A porta de comunicação é uma abstração que define um ponto de interação de um componente. Ela oferece uma interface padrão, independente das tecnologias de comunicação suportadas pela plataforma onde o componente venha a ser instanciado. De forma a abstrair os detalhes destas tecnologias, foi explorado o conceito de canal de comunicação [28, 29, 30]. O canal de comunicação realiza a ligação entre duas portas, sendo utilizado pelas mesmas para a troca de dados. A Figura 2.4, retirada de [3], apresenta o relacionamento entre os recursos virtuais, portas de comunicação, canal e conexão virtual.

As portas de comunicação são acessadas pelos recursos virtuais e registradas junto à conexão virtual. Por sua vez, a conexão virtual se encarrega de gerenciar os elementos da interconexão, criando, por exemplo, o(s) canal(ais) de comunicação necessário(s) para a interligação entre as portas.

A utilização do canal de comunicação para estabelecer ligações entre portas permite ao middleware escolher diferentes tecnologias e protocolos de comunicação. Com isso, a

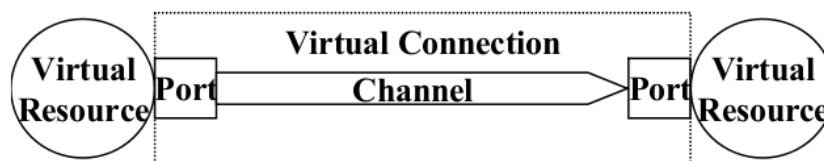


Figura 2.4: Relacionamento entre os Elementos

escolha da tecnologia de comunicação a ser empregada na ligação entre duas portas pode levar em consideração o tipo de dado a ser transmitido, parâmetros de rede e a localização dos respectivos componentes participantes (recursos virtuais).

O modelo suporta diversas topologias de interconexão, permitindo conexões 1x1, 1xN, Nx1 e NxN. As portas de comunicação podem funcionar como portas de entrada ou portas de saída. A conexão entre uma porta de saída e várias portas de entrada caracterizam uma topologia 1xN. O componente VirtualConnection representa uma conexão entre duas ou mais portas; a definição do número de portas envolvidas depende das ligações estabelecidas pela especificação da aplicação.

A porta de comunicação opera com dados em formato de seqüências de bytes. Cada porta de comunicação possui um conjunto de propriedades que define os parâmetros do fluxo associado, ou seja, que transita por ela. Isto acontece devido ao fato de um recurso poder possuir diversas portas de comunicação, podendo ser portas de entrada ou portas de saída, com possibilidades de tratar diferentes formatos, como acontece, por exemplo, com componentes conversores de fluxos de um formato para outro.

2.4.4 Arquitetura Geral de um Sistema Baseado no Cosmos

O Cosmos dá suporte às tarefas de configuração e gerenciamento de recursos e componentes de um sistema multimídia distribuído. Uma visão arquitetural de alto-nível do framework relacionando os principais conceitos e componentes envolvidos é apresentada na Figura 2.5; uma breve indicação com o papel de cada elemento na arquitetura é discutida nos parágrafos seguintes.

Como elemento central desta arquitetura, o Cosmos define o Configurator, cujo papel consiste em coordenar e gerenciar todas as fases do ciclo de vida dos componentes do middleware e da aplicação. O Configurator é um componente crítico, sendo responsável pela iniciação, configuração e gerenciamento dos recursos do sistema, dos componentes do middleware e da aplicação. Este componente realiza operações de negociação e ajuste

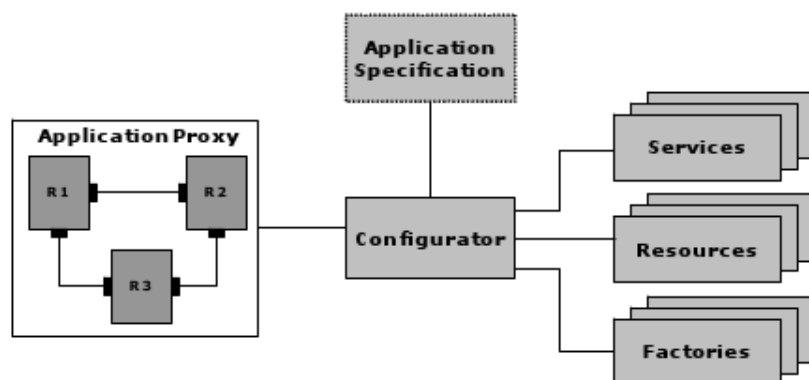


Figura 2.5: Arquitetura de Alto Nível do Cosmos.

dinâmico de propriedades, que estão associadas com os recursos e fluxos multimídia envolvidos no sistema.

Uma aplicação no Cosmos deve ser definida através de uma especificação (Application Specification), que descreve os recursos e os elementos da arquitetura envolvidos, assim como os vários requisitos de QoS necessários para a sua execução. Esta especificação é realizada usando uma linguagem de configuração, que possui uma estrutura similar a uma ADL [31] (Architecture Description Language). O Cosmos não define uma tecnologia, nem tampouco uma linguagem específica para a descrição de aplicações e componentes. No entanto, o Configurator precisa interpretar especificações de modo a poder construir e representar as respectivas aplicações. Para isso, o framework define um modelo de meta-componentes, apresentado em [26], que serve de base para a descrição e representação de aplicações no middleware.

Dessa forma, a definição da linguagem a ser utilizada é de responsabilidade do middleware que implementa o framework, devendo esta estar baseada no modelo de meta-componentes definido. Isso possibilita que a configuração dos componentes e aplicações possa ser gerada e processada pelo middleware independentemente de que linguagem venha ser utilizada em uma instanciação do framework Cosmos.

A descrição de cada aplicação, conforme o modelo definido no Cosmos, é representada no middleware por um componente denominado ApplicationProxy. É através do componente ApplicationProxy que o Cosmos dá suporte ao conceito de reflexividade. Este componente atua como um repositório em um meta-nível, onde são descritas a estrutura dinâmica da aplicação e as propriedades dos respectivos componentes de nível base.

A representação da aplicação no ApplicationProxy é realizada através de um grafo

com os meta-componentes que representam os componentes do sistema e da aplicação. Estes meta-componentes funcionam como repositórios de meta-dados (propriedades) associados aos recursos, descrevendo as propriedades e os estados dos componentes associados, e dando suporte ao gerenciamento, configuração e adaptação [2].

O ApplicationProxy facilita o suporte à consistência em processos de adaptação. Cada operação associada a uma mudança dinâmica de propriedades de um componente deve ser encaminhada ao componente ApplicationProxy que verifica a coerência desta operação, atualiza a descrição da arquitetura e repassa estas alterações para os respectivos componentes afetados.

O modelo definido para o desenvolvimento e configuração de QoS de um sistema baseado no Cosmos é apresentado na figura 2.6

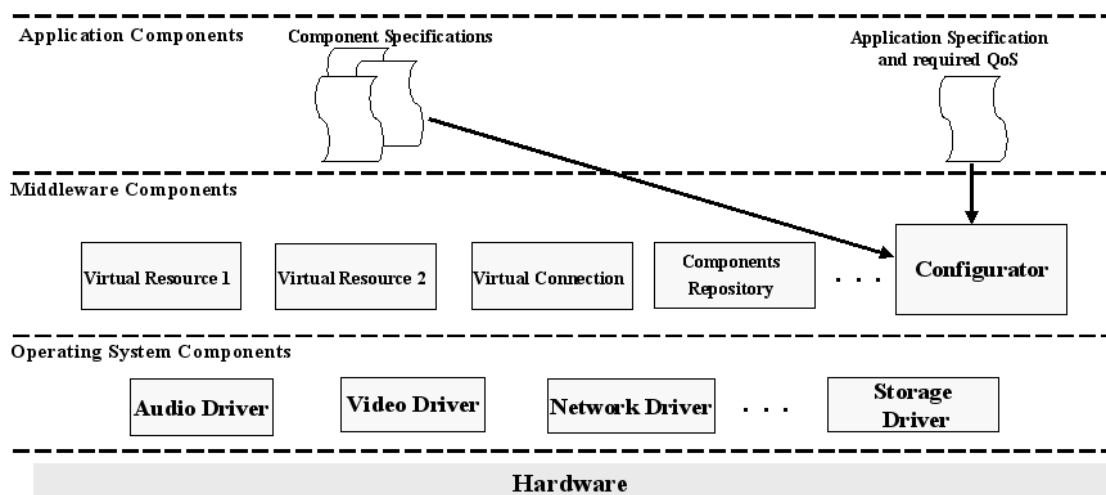


Figura 2.6: Arquitetura geral de um middleware baseado no framework Cosmos.

A especificação de uma aplicação segundo o framework Cosmos consiste na configuração dos componentes do sistema (camada de middleware), cuja descrição deve ser definida de acordo com a linguagem de especificação disponível na plataforma middleware que implementa o framework. Uma plataforma middleware aderente ao modelo Cosmos deve disponibilizar os componentes clássicos de gerenciamento de recursos de mídia em um repositório.

A configuração do sistema, realizada pelo Configurator, deve estabelecer critérios para negociações e seleção de propriedades e componentes (quando houver diferentes possibilidades de configuração de propriedades internas de componentes e/ou de escolha de componentes de funcionalidades e QoS compatíveis e aceitáveis considerando os requisitos especificados).

A interconexão entre componentes envolve a verificação de compatibilidade entre a especificação, os tipos dos componentes envolvidos nas interconexões definidas e as capacidades da plataforma.

Por fim, quando todos os componentes especificados estiverem instanciados e configurados, o configurador dispara o início a da execução da aplicação. Considerando que a plataforma pode sofrer variações na Qualidade dos Serviços oferecidos, durante a execução da aplicação o sistema deverá fazer o monitoramento da QoS de forma a verificar se os requisitos especificados estão sendo satisfeitos, e, quando detectada possíveis anomalias, o sistema possa fazer a adaptação. Para isso o Cosmos define um modelo de QoS(descrito a seguir, na Seção 2.4.5), que constitui-se no elemento base para a definição das diretrizes e critérios para o processo de ajustes e eventuais trocas de componentes.

2.4.5 Modelo de QoS

Os elementos envolvidos com o gerenciamento da QoS no Cosmos e seus relacionamentos são apresentados na Figura 2.7, retirada de [2]. Um recurso virtual pode ser monitorado por um QoSMonitor que por sua vez é gerenciado por um QoSManager. Políticas para gerenciamento de recursos envolvem aspectos estáticos, como especificação de QoS, negociação para escolha de formatos para codificação/decodificação de mídia e controle de admissão, bem como aspectos dinâmicos relacionados com monitoramento, manutenção e renegociação de QoS, para fins de adaptação.

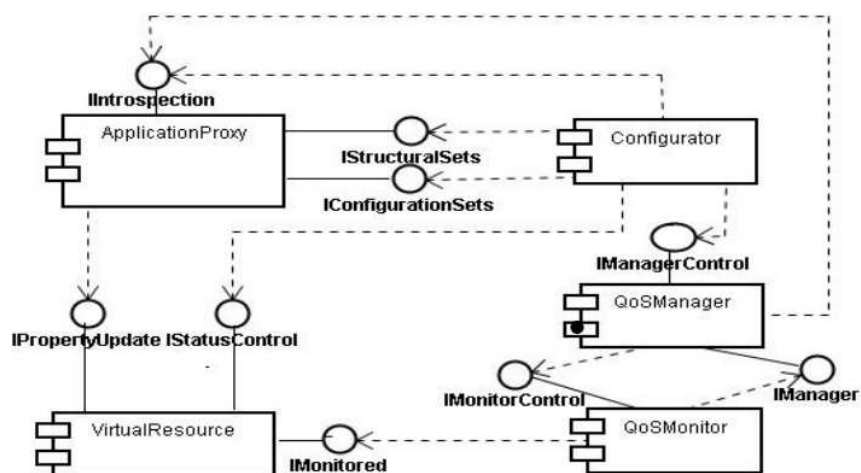


Figura 2.7: Visão geral do modelo de QoS do Cosmos.

A especificação de QoS, presente an figura 2.6 tem como função descrever os requisitos de QoS da aplicação, ou seja, uma descrição tem um papel análogo ao de um contrato

que rege as normas de configuração e operação da aplicação. A negociação tem como objetivo selecionar valores apropriados para os parâmetros de QoS que possam cumprir os requisitos da especificação.

O monitoramento de QoS consiste em verificar, em tempo de execução, se os recursos alocados à aplicação estão compatíveis com o contrato, ou seja, se os valores dos parâmetros de QoS estão dentro dos requisitos estabelecidos na especificação, relatando eventuais violações. Caso ocorram violações, as discrepâncias em relação ao contrato especificado devem ser consideradas pelo processo responsável por realizar a adaptação, de acordo com a indicação do QoS Manager.

O modelo de QoS do Cosmos utiliza o conceito de intervalos de QoS, no qual a definição dos requisitos ocorre por faixas de valores. Além disso, o modelo permite também a definição de um valor *default*, servindo como referência para dar início à operação dos recursos nas respectivas regiões. Se durante a execução da aplicação for detectada alguma situação de não cumprimento da QoS negociada, um processo de adaptação é iniciado observando os requisitos estabelecidos na descrição da aplicação.

2.4.6 ADL definida para o Framework Cosmos

Linguagens de Descrição Arquitetural (ADLs) são usadas para definir e modelar a arquitetura do sistema antes da implementação dos seus componentes. ADLs podem ser consideradas como ferramentas que dão suporte à programação de granularidade grossa (*programming in the large*), onde são descritos os elementos arquiteturais do sistema, como componentes, conexões, composição, propriedades não-funcionais, restrições e paradigmas de comunicação, entre outros. As ADLs normalmente apresentam construções e conceitos comuns em relação à maioria dos conceitos explorados nos modelos de desenvolvimento baseados em componentes[32].

De acordo com o modelo definido pelo Cosmos em [2], cada aplicação e componente precisa ser especificado utilizando uma linguagem de descrição arquitetural. Esta linguagem tem um papel similar à ADL. Assim, esta linguagem é usada para descrever a especificação de aplicações indicando os componentes requeridos, as conexões, os parâmetros e as propriedades associadas, assim como os requisitos de QoS necessários à sua execução. Baseado nesta especificação, o Configurator localiza e aciona fábricas capazes de instanciar os respectivos componentes. Na verdade, o framework não define uma tecnologia, nem tampouco uma linguagem específica para descrição de aplicações e componentes. Ele apresenta um modelo de descrição independente de tecnologias, com regras e relacionamentos comuns para definição de aplicações e componentes.

2.4.7 O *Middleware AdapTV*

A definição de um middleware baseado no Cosmos denominado AdapTV foi proposta e implementada com a arquitetura acima [3]. Este trabalho definiu uma arquitetura para a conexão virtual e permitiu a realização de adaptações trocando dinamicamente as propriedades dos componentes de forma a ajustar a QoS de um fluxo.

2.5 Considerações Finais

O presente projeto se acrescenta aos esforços atuais na linha da suporte à adaptação dinâmica para sistemas multimídia distribuídos, consistindo na continuidade do trabalho desenvolvido por [3].

O modelo proposto por [3], foi direcionado para a adaptação com a troca de propriedades dos componentes localizados nas extremidades dos canais de comunicação e o ajuste realizado com a troca dos respectivos canais. O presente trabalho tem como foco a adaptação com a troca dos componentes (VirtualResouces).

Capítulo 3

A Abordagem proposta para troca dinâmica de componentes em sistemas multimídia distribuídos baseados no *framework* Cosmos

3.1 Introdução

Este capítulo descreve as várias etapas que compõem a abordagem proposta para dar suporte à substituição dinâmica de componentes em sistemas multimídia distribuídos desenvolvidos com base no *framework* Cosmos e a arquitetura de implementação definida no escopo do middleware AdapTV. Conforme foi mencionado nas considerações finais do Capítulo 2, a abordagem dá continuidade a outros trabalhos realizados anteriormente na busca de soluções para adaptação dinâmica nesses sistemas. O Capítulo descreve como o processo foi proposto e integrado ao modelo de QoS definido pelo Cosmos no contexto do middleware AdapTV.

Para o desenvolvimento da abordagem foi necessário adaptar a arquitetura originalmente proposta pelo Cosmos para adaptação dinâmica. Para dar suporte à adaptação envolvendo a troca dinâmica de componentes, foram desenvolvidas as seguintes atividades: extensão da ADL do Cosmos de modo a suportar a especificação de aplicações com indicação de potenciais componentes substitutos, identificação de possíveis critérios para seleção de componentes, definição de um modelo para classificação e escolha de critérios, definição de um módulo para implementação de um processo de escolha de componentes

substitutos, e, concluindo, com a definição da estrutura de implementação de um mecanismo para troca dinâmica de componentes.

Deve-se destacar que para atingir o objetivo fim, que no escopo do presente trabalho consiste na definição de um mecanismo de suporte à substituição dinâmica de componentes, era necessário realizar todas essas etapas. Assim, a apresentação da abordagem envolve a descrição de cada uma dessas atividades.

3.2 Modelo de Adaptação Dinâmica do Cosmos

Segundo o *framework* Cosmos, um processo de adaptação dinâmica deve considerar os requisitos de QoS e as políticas de adaptação descritas na especificação da aplicação, de acordo com o modelo de gerenciamento de QoS introduzido na Seção 2.3. Neste modelo, monitores (Figura 3.1) iniciam o processo de adaptação ao detectarem parâmetros de QoS definidos na especificação fora da faixa de valores aceitáveis especificados. Ou seja, caso algum monitor detecte uma situação de anomalia que requeira algum tipo de adaptação, um evento é indicado ao gerente do sistema, QoSManager, que se encarrega de interagir com o Configurator, (componente central da arquitetura Cosmos) de modo que ele possa decidir o que fazer.

Para decidir o que fazer, o Configurator precisa adotar critérios. Por sua vez, a definição de critérios requer o conhecimento do comportamento do sistema, como também, das atividades do processo nas mais diferentes situações; como este trabalho está inserido no contexto do Cosmos reforçamos a necessidade de se realizar um estudo aprofundado destas situações.

O modelo de QoS originalmente proposto para Cosmos é limitado no sentido de não permitir a especificação de uma aplicação com dois ou mais componentes que realizem a mesma tarefa. Outra limitação se dá na descrição de requisitos para a aplicação, que não permite estabelecer indicadores do melhor nível no contexto atual para o correto funcionamento da aplicação. No decorrer deste trabalho realizamos vários testes para identificar alguns critérios-chaves de modo a estender a ADL estrutural do Cosmos com o objetivo de incluir estes critérios na gerência da QoS do sistema.

3.2.1 Arquitetura Geral da Abordagem Inicial

Na abordagem inicial proposta por [3], a adaptação dinâmica se concentrou no escopo da conexão virtual, sendo realizada duplicando os canais de comunicação envolvidos,

conforme ilustrado na Figura 3.1. Para esse caso, as atividades do processo de adaptação são descritas no Diagrama de Atividades da Figura 3.2.

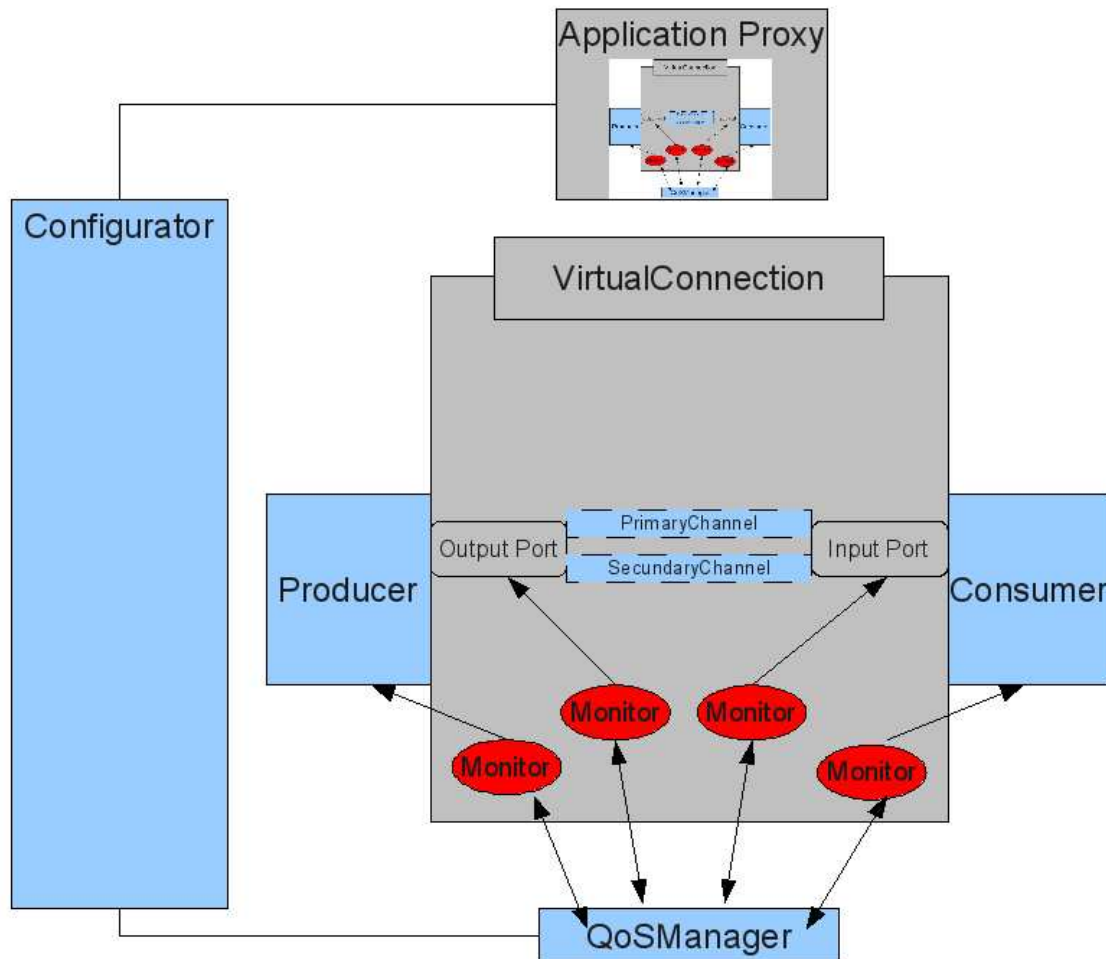


Figura 3.1: Arquitetura para Adaptação dinâmica no Cosmos com duplicação de canais de comunicação.

Ao iniciar um processo de adaptação, o configurador (elemento principal do Cosmos) precisa analisar a solicitação de forma a não deixar o sistema em um estado inconsistente. Após essa verificação, o configurador efetivamente inicia o processo de adaptação notificando os componentes envolvidos para que estes realizem uma etapa de reserva de recursos e se preparem para a reconfiguração. O Configurator notifica então o correspondente componente **VirtualConnection**, informando-o que um processo de adaptação foi iniciado. A conexão virtual realiza a mesma operação de alocação de recursos e preparação para a reconfiguração nas portas associadas (denominadas **Output Port** e **Input Port**), para em seguida realizar a duplicação do canal de comunicação. Após a configuração do canal secundário e das portas de comunicação envolvidas, a conexão virtual notifica o Configurator, informando-o que o processo de preparação e alocação de recursos para

a reconfiguração foi concluído com sucesso e que a reconfiguração pode continuar. O configurador, após receber a notificação da conexão virtual, identifica os componentes transmissores, para em seguida notificá-los, de modo que estes possam ativar a adaptação previamente preparada. Neste ponto, o componente transmissor interrompe a sua transmissão e altera seus parâmetros operacionais. O componente realiza então uma notificação à sua correspondente porta de saída e recomeça a transmissão. A porta de saída (Output Port), por sua vez, realiza a troca do canal de comunicação, marcando o canal primário como livre e continua a operar normalmente enviando dados, agora, pelo canal secundário. Caso ainda tenham dados do fluxo em trânsito no canal de comunicação primário, até esses eventuais dados chegarem à porta de entrada, o sistema realizará, paralelamente, a transmissão dos novos dados (fluxo adaptado), saindo do componente transmissor através do canal secundário.

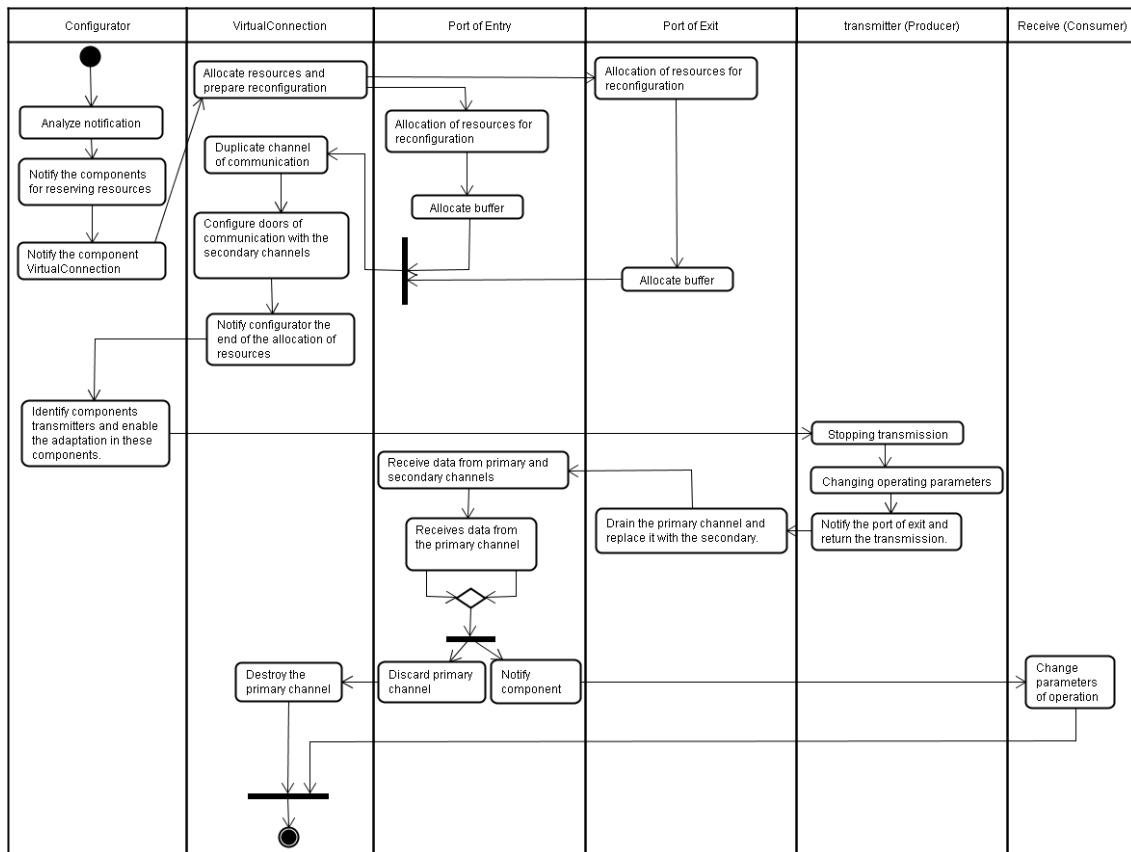


Figura 3.2: Diagrama de atividades relativo a um processo de adaptação no *framework* Cosmos

O componente receptor continua sua operação solicitando dados à porta de entrada (Input Port). A porta de entrada, por sua vez, continua entregando os dados do buffer correspondente ao canal primário, até que o mesmo se esvazie. Quando o buffer primário da porta de entrada não contiver mais dados, assume-se que o fluxo de dados do canal de

comunicação primário terminou, e que este canal não irá entregar mais nenhum dado para a porta de entrada.

Neste momento, esta porta notifica o componente associado e inicia o processo de troca dos canais de comunicação e de seus buffers de recepção. Após a notificação, o componente receptor realiza os procedimentos necessários à reconfiguração dinâmica, trocando seus parâmetros de operação e retornando ao seu funcionamento normal.

No final da troca dos canais, a porta de entrada notifica ao canal primário que ele não é mais necessário. O canal primário, com as notificações de suas duas portas envolvidas realiza uma chamada callback à conexão virtual correspondente, informando que o canal primário não está mais sendo utilizado e que o mesmo pode ser desalocado. A conexão virtual realiza a troca dos canais de comunicação, fazendo com que o canal de comunicação secundário passe a ser o canal primário da interconexão. Uma visão geral do processo de adaptação pode ser vista na Figura 3.2 por meio de diagrama da UML 2.0.

3.2.2 Arquitetura Geral da Abordagem Proposta

O processo para realizar trocas dinâmicas de componentes em sistemas, sem afetar o funcionamento da aplicação, e, principalmente, mantendo os critérios de qualidade especificados previamente, envolve várias atividades; o gerenciamento da sincronização entre estas atividades requer um esforço de grande dimensão.

Para que a troca de um componente seja realizada é necessário que exista um mecanismo para selecionar um componente que melhor se adeque dentro da necessidade gerada pelo novo contexto de execução, denotado pelos novos parâmetros necessários para se manter a aplicação funcionando. Portanto a abordagem proposta no presente trabalho se utiliza de um mecanismo para a seleção de componentes, que foi introduzido na arquitetura como um novo serviço para o Cosmos e definido através de um componente denominado `ComponentChooserService`.

A Figura 3.3 apresenta uma arquitetura simplificada para o Cosmos prover suporte à adaptação dinâmica com troca de componentes. Nesta arquitetura, novos serviços foram adicionados ao Cosmos: o `ComponentChangeService`, que engloba toda a parte relacionada com as conexões e os demais componentes da aplicação, e, o `ComponentChooserService` que é responsável por selecionar componentes substitutos.

Dado que a arquitetura é uma evolução da arquitetura proposta originalmente pelo Cosmos, a simplificação tem como objetivo direcionar o foco da apresentação para os

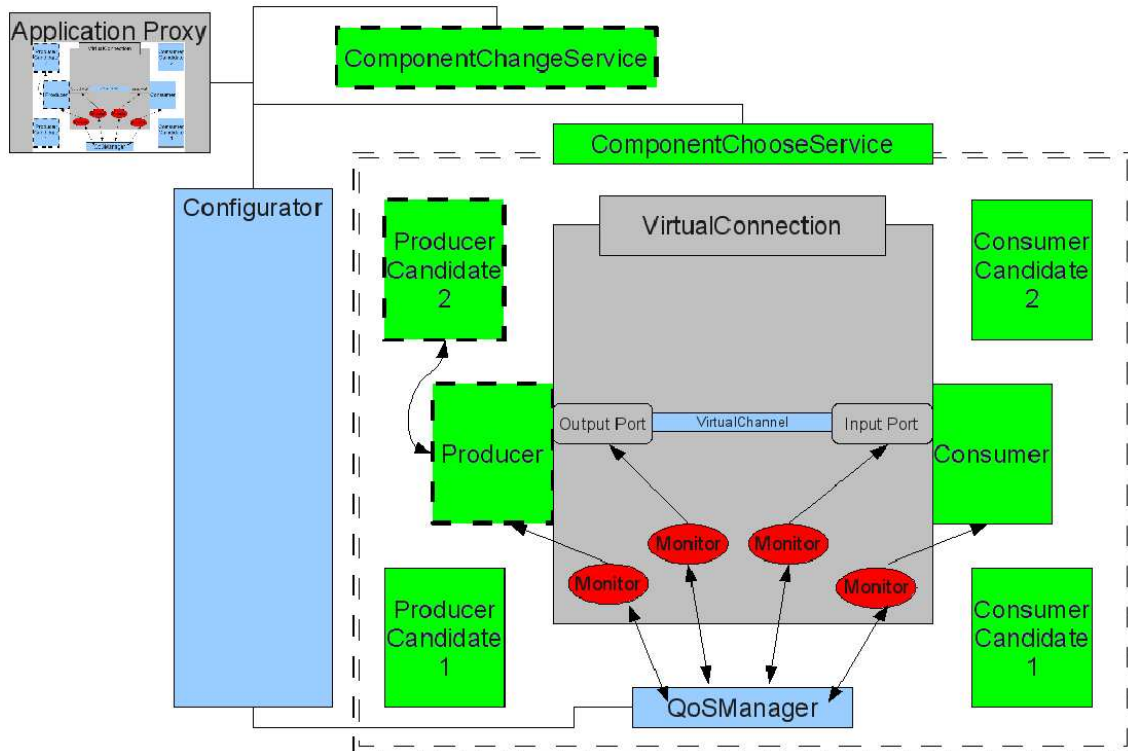


Figura 3.3: Relacionamento entre os Elementos.

elementos principais da proposta, e, ao mesmo tempo, ocultar detalhes que não fazem parte do presente trabalho. Considerando que o modelo originalmente proposto encontra-se descrito em [2] e [3], descrições mais detalhadas dos demais componentes envolvidos na arquitetura podem ser obtidas nas respectivas referências.

3.3 Extensão da ADL do Cosmos

Como apresentado em [2] e explanado no Seção 2.4, o Cosmos definiu uma ADL para a especificação de uma aplicação. O Código 3.1 apresenta alguns trechos da descrição da ADL apresentada em [2] e [3]. Este segmento de código descreve a criação da instância de um componente `VideoFlowProducer`, com as suas respectivas propriedades. De acordo com esta especificação, nesta aplicação, apenas o componente `VideoFlowProducer` poderá ser usado para geração de fluxos multimídia.

Código-fonte 3.1: Trecho de código onde somente um componente era originalmente especificado.

```
<COMPONENT name="VideoFlowProducer" instance="videoFlowProducer"
  location="comp1, services.lasic.ufrn.br:8080/enquiring, services.
  dimap.ufrn.br/enquiring">
```

```
<PROPERTIES>
    <PROPERTY name="AllocatedMemory" value="5000" />
</PROPERTIES>
5 <ATTRIBUTES>
    <ATTRIBUTE name="Title" value="Example1" />
</ATTRIBUTES>
<PORTS>
    <PORT name="OutputFlow">
10     <CONSTRAINT property="Encoding" remove="MPEG-2" />
    </PORT>
</PORTS>
</COMPONENT>
```

A presente proposta estende esta ADL adicionado algumas construções com o objetivo de dar subsídios para o mecanismo de troca. O segmento de código adiante (Código-fonte 3.2) descreve como introduzir na arquitetura a especificação de dois componentes com capacidade de fornecer fluxos de vídeo, cabendo ao mecanismo de gerenciamento proposto escolher dinamicamente qual deles utilizar.

Código-fonte 3.2: Trecho de código da nova versão, onde existem dois componentes especificados e definidos na aplicação como possíveis provedores de fluxo de vídeo.

```
<COMPONENT name="VideoFlowProduceren" instance="videoFlowProducer"
    location="br.natalnet.adaptv.producers.VideoFlowProduceren">
<PROPERTIES>
    <PROPERTY name="AllocatedMemory" value="5000" />
    <PROPERTY name="QoSRegion" value="Normal,Low" />
5 </PROPERTIES>
<ATTRIBUTES>
    <ATTRIBUTE name="Title" value="Example in english" />
</ATTRIBUTES>
<PORTS>
10 <PORT name="OutputFlow">
    <CONSTRAINT property="Encoding" remove="MPEG-2" />
    </PORT>
</PORTS>
</COMPONENT>
15 <COMPONENT name="VideoFlowProducerpt_BR" instance="videoFlowProducer"
    location="br.natalnet.adaptv.producers.VideoFlowProducerpt_BR">
<PROPERTIES>
    <PROPERTY name="AllocatedMemory" value="5000" />
    <PROPERTY name="QoSRegion" value="High,Normal" />
</PROPERTIES>
20 <ATTRIBUTES>
    <ATTRIBUTE name="Title" value="Exemplo em Portugues" />
```

```
</ATTRIBUTES>
<PORTS>
  <PORT name="OutputFlow">
    <CONSTRAINT property="Encoding" remove="MPEG-2" />
  </PORT>
</PORTS>
</COMPONENT>
```

Assim, de acordo com esta especificação, tanto o componente VideoFlowProducer quanto VideoFlowProducerpt_BR poderão ser usados para geração de fluxos multimídia. Conforme pode ser visto no código exemplo, ambos componentes estão associados à instância videoFlowProducer

Assim como na especificação original do Cosmos, valores default para propriedades podem ser definidos, servindo como referências para dar início à operação dos recursos nas respectivas regiões. As faixas associadas às regiões podem ser modificadas dinamicamente em função das políticas de gerenciamento especificadas pela aplicação. A definição envolvendo a seleção da região de QoS adotada também pode mudar dinamicamente, por exemplo, quando um monitor observar uma violação em relação à faixa associada à região corrente. Neste caso, o gerente de QoS, em conjunto com o Configurator, escolhem uma nova região, iniciando a operação nesta região com o respectivo valor de propriedade default definido para esta região. O modelo também permite a retomada automática para regiões de QoS melhores, a partir de tentativas a serem realizadas periodicamente, onde a periodicidade definida para realizar estas tentativas é especificada na descrição da aplicação. Após serem instanciados e configurados, os monitores aguardam o início da operação do recurso.

Essa proposta de ADL estendida permite a definição de conjuntos de componentes que podem ser substituídos em tempo de configuração ou execução da aplicação. De forma a formalizar essa ADL, foi desenvolvido um XML Schema definindo os elementos propostos e incorporados à ADL do Cosmos; este XML Schema pode ser visto no Apêndice A.1.

O princípio consiste em definir um conjunto de elementos (XML) que poderão ser usados por arquitetos de software para construir sistemas especificando uma série de componentes candidatos (alternativas com diferentes componentes que provêm por exemplo um mesmo serviço dentro da aplicação); com isso, o sistema se torna menos susceptível a falhas, e, com um nível de adaptabilidade e disponibilidade mais alto.

3.4 Seleção de Componentes

Na arquitetura apresentada na seção 3.2.2 foi introduzido o componente `ComponentChooserService`. O papel deste novo componente na arquitetura consiste em selecionar, quando solicitado, o componente que melhor se adequa considerando o contexto de execução atual e os parâmetros indicados para que a aplicação se mantenha funcionando com os níveis de QoS definidos.

Para realizar a seleção de um componente o sistema deve definir quais critérios serão utilizados na escolha, ou seja, deve considerar a especificação fornecida pelo arquiteto do sistema. Um dos esforços envolvidos no desenvolvimento deste trabalho consistiu em definir e classificar, sob a visão de nossa experiência, alguns os critérios que entendemos como relevantes para o domínio dos sistemas multimídia distribuídos.

O foco da abordagem foi direcionado para a definição de um modelo capaz de tratar a especificação de requisitos do sistema e analisar, no contexto atual, qual dos componentes disponíveis é o melhor para atender os requisitos de QoS no escopo do sistema.

A idéia consiste em passar para o serviço `ComponentChooserService`, como parâmetros, o conjunto de componentes candidatos capazes de prover as funcionalidades e a QoS necessária à execução e/ou continuidade do serviço; ou seja, é necessário estabelecer critérios que sejam coerentes com a política de QoS do sistema, para escolher qual componente candidato deve substituir o elemento alvo da troca (normalmente disparada porque o componente alvo não está contribuindo para a consecução dos objetivos do sistema). Assim, o objetivo da substituição é melhorar a QoS do sistema sob a ótica dos critérios de avaliação que serão utilizados.

Para o processo de seleção foi analisada a possibilidade de gerar um conjunto de componentes ordenados por algum critério de qualidade ao invés de indicar apenas um, como sugerido em [33]. A opção de manter uma fila ordenada aparentemente facilitaria futuras decisões, porém, ela não foi adotada devido à dinamicidade desses sistemas, os quais mudam frequentemente de estado, fazendo com que esta lista ficasse rapidamente desatualizada.

A definição do conjunto de critérios pode ser feita estaticamente ou dinamicamente. Entretanto, deve-se lembrar que critérios de qualidade podem ser frequentemente modificados, bem como novos componentes podem ser inseridos ou até mesmo mudanças habituais no nível físico da plataforma podem afetar critérios relacionados com a latência da rede, por exemplo.

Cada vez que houver a indicação da possível troca de um componente, deve-se disparar o processo para avaliar e selecionar o novo componente substituto. Entretanto, considerando que algumas propriedades dos componentes avaliadas anteriormente ainda poderão ser consideradas, o modelo deve considerar estas informações para fins de otimização do processo, evitando assim repetições de avaliações que continuam consistentes

3.4.1 Identificação de Critérios de QoS para Análise de Componentes Candidatos

Na literatura, muitos trabalhos buscam identificar os requisitos que devem ser considerados em sistemas multimídia distribuídos [24]. Entre os requisitos, podemos destacar: baixos níveis de latência, jitter e skew, os quais estão associados ao comportamento das plataformas de rede. Porém, esses elementos avaliados isoladamente não são suficientes para definir os critérios de adaptação dado a ocorrência de eventuais conflitos de objetivos.

Nesse contexto, o presente trabalho procurou de forma empírica avaliar alguns critérios identificados como potenciais critérios que efetivamente podem influenciar na eficácia do processo de adaptação no Cosmos. Para estabelecer o que é relevante para o processo de escolha de critérios de seleção de componentes é necessário conhecer como os componentes candidatos se comportaram em situações passadas.

Assim, com o objetivo de identificar potenciais critérios de seleção de componentes no contexto de componentes do *framework* Cosmos, decidimos fazer várias baterias de testes envolvendo os componentes de uma aplicação anteriormente desenvolvida por [3]. A idéia consistiu em provocar de forma controlada variações nos parâmetros do sistema, de forma a disparar a adaptação. Os resultados dos testes mais significativos são apresentados adiante. De acordo com esses testes a eficiência pode ser afetada por fatores específicos da aplicação bem como da plataforma em que a mesma está inserida. Esses resultados foram considerados na definição dos critérios chaves que propomos em [34], os quais estão sendo utilizados em processos de seleção de componentes substitutos.

Com relação a características específicas da aplicação, foram analisados requisitos associados ao momento da adaptação, tamanho do buffer e à latência de adaptação. Por outro lado, com relação a características da plataforma considerou-se nos testes a capacidade/velocidade de processamento. Vale salientar que esses requisitos não contemplam totalmente os critérios que precisam ser considerados no modelo de adaptação, porém a abordagem descrita na Seção 3.5 pode ser considerada como uma solução representativa para efeito de análise e composição de critérios.

O tamanho do buffer é um elemento fundamental que poderá afetar o desempenho da eficiência de comunicação entre componentes que manipulam o streaming de dados. Já a latência de adaptação reflete o tempo consumido pelo sistema para realizar a adaptação da aplicação, seja ela por troca de propriedades ou até mesmo pela eventual troca de um de seus componentes; ela pode ser considerado como elemento crítico para a viabilidade do modelo de adaptação. Por fim, outro critério importante para o modelo é a análise da capacidade dos processadores no processo de adaptação.

A seguir, os itens abaixo apresentam e descrevem os principais testes realizados e analisam os resultados obtidos. Os cenários de teste foram construídos usando duas máquinas Pentium IV 2.8 GHz, 512 MB de memória RAM mais uma máquina com dois núcleos Turion de 1.9 GHz com 2GB de memória RAM processador e uma máquina Duron 900MHz com 512MB de memória RAM, todas elas sobre a plataforma Windows. O componente escolhido para realizar as medições foi o receptor (consumidor) do fluxo. Esse componente é parte integrante do middleware AdapTV [2, 35, 26, 3]. Vale destacar que o foco dos testes foi direcionado para a análise dos impactos relativos às mudanças dinâmicas de propriedades.

Embora as adaptações realizadas tenham sido limitadas ao ajuste de valores de propriedades, o objetivo dos experimentos foi direcionado para a questão da troca dinâmica de componentes.

3.4.1.1 Análise relacionada com o momento de adaptação

Para avaliar o impacto do momento da adaptação na troca de um componente em relação ao tempo necessário para realizar a adaptação, foram feitos alguns testes variando a quantidade de mensagens antes de disparar o processo. Para realizar essa análise foram considerados envios de mensagens com tamanho de 512 bytes e com eventos de adaptação ocorrendo a cada 10 mensagens enviadas. Após inúmeros testes, pode-se concluir que a quantidade de mensagens antes de se iniciar um processo de adaptação não interferia no tempo de realização da adaptação. A Figura 3.4 mostra o resultado obtido com esses testes preliminares num contexto de comunicação local, onde cada valor é representado por uma média de 10 execuções. Durante as experiências, nota-se que à medida que se aumenta a quantidade de mensagens enviadas antes de iniciar a adaptação o valor do tempo do processo é praticamente estável, atingindo uma média de 15,55 ms.

Uma análise semelhante foi realizada considerando as mesmas características da comunicação local, porém realizada num contexto envolvendo componentes remotos, com

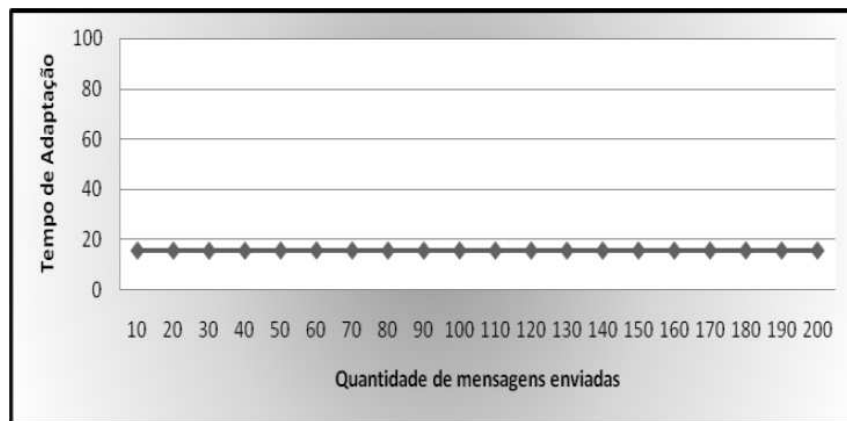


Figura 3.4: Tempo médio de uma adaptação local com adaptações realizadas a cada 10 mensagens enviadas.

duas máquinas desempenhando respectivamente os papéis de produtora e consumidora. As medições observadas e apresentadas na Figura 3.5 demonstram que a quantidade de mensagens enviadas antes da adaptação também não interferiu no tempo de realização da adaptação, consumindo em média 69,28 ms. Para fins de avaliação foram considerados envios de mensagens com tamanho de 512 bytes e verificados os resultados para cada critério mencionado. Uma vez que a quantidade de mensagens processadas antes da realização da adaptação não interfere no tempo de adaptação, o trabalho decidiu simplificar os testes analisando apenas disparos de adaptações a cada 10 mensagens enviadas. Os cenários de testes consideraram os critérios discutidos em duas situações: adaptação local envolvendo um componente produtor e um componente consumidor, e adaptação remota contendo um produtor e um consumidor.

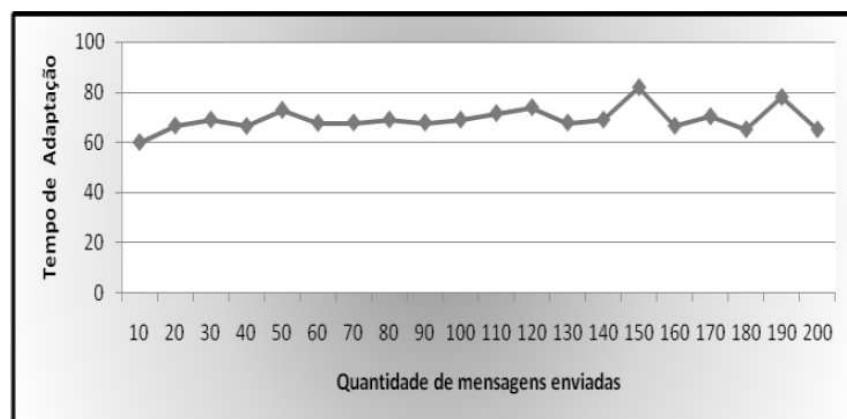


Figura 3.5: Tempo médio de uma adaptação remota com adaptações realizadas a cada 10 mensagens enviadas.

Tabela 3.1: Tamanhos de buffers utilizados no experimento

Cenário	Buffer(bytes)
1	16
2	32
3	64
4	128
5	256
6	512
7	1024
8	2048
9	4096
10	8192

3.4.1.2 Análise relacionada com o tamanho do buffer

As análises a seguir vão considerar o tamanho do buffer, de acordo com as 10 situações, conforme os valores indicados na Tabela 3.1. Os testes realizados consideraram o tempo de adaptação observada no lado do componente consumidor da aplicação.

Estes cenários de testes permitem verificar o comportamento do sistema em diferentes situações: o buffer de recepção com espaço menor do que o tamanho da mensagem enviada (< 512); buffer tem espaço disponível com capacidade igual ao tamanho da mensagem enviada (512); e, buffer maior que o tamanho da mensagem enviada (> 512):

Contexto Local - Nesse contexto, a avaliação dos testes relativos à definição de buffers com os tamanhos indicados na Tabela 3.1 produziram os tempos de adaptação indicados na Figura 3.6.

De acordo com as informações mostradas na Figura 3.6, quando o tamanho do buffer disponível estiver na faixa entre 16 e 512 o tempo de adaptação observado no lado do consumidor consome em média 35,7ms. Por outro lado, quando o espaço disponível para o buffer for maior que do que o tamanho da mensagem enviada, o tempo de adaptação diminui para próximo dos valores medidos na Figura 3.4, de 15,55ms. Esse comportamento deve-se ao fato de que, quanto menor o tamanho do buffer, mais requisições ao mesmo são feitas para a coleta dos dados que chegam ao consumidor.

Contexto Remoto - Os testes realizados no contexto remoto considerando os tamanhos de buffer indicados na Tabela 3.3, produziram os tempos de adaptação mostrados na

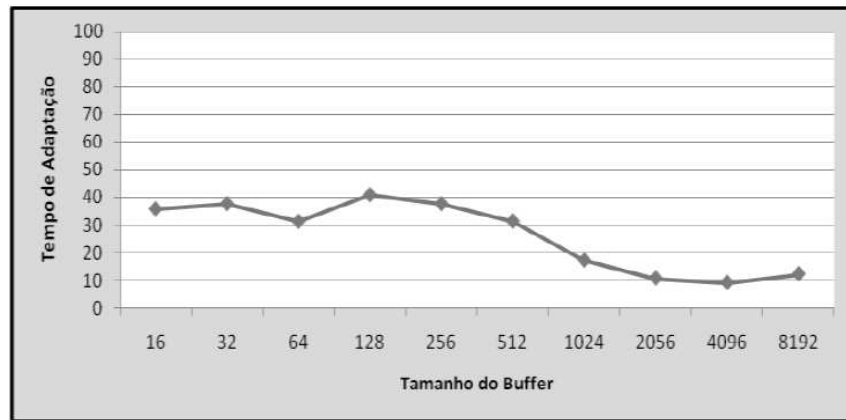


Figura 3.6: Tempo de Adaptação vs Tamanho do Buffer Cenário Local.

Figura 3.7. Dessa forma, podemos perceber novamente que, para as faixas de tamanho de buffer entre 16 e 512, o tempo de adaptação observado pelo lado do consumidor consumiu em média 75,05 ms; porém, quando o tamanho do buffer foi maior do que o tamanho da mensagem enviada, percebe-se uma pequena diferença em seu tempo de adaptação, mas mesmo assim, permanecendo inferior em relação a buffers menores.

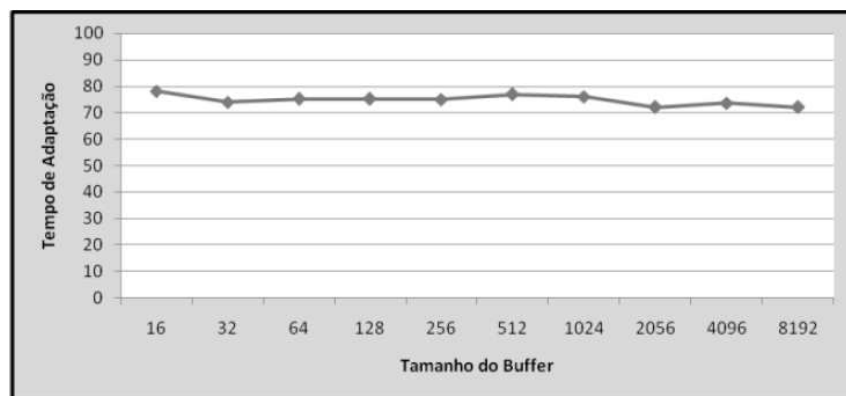


Figura 3.7: Tempo de Adaptação vs Tamanho do Buffer Cenário Remoto.

3.4.1.3 Análise relacionada com a capacidade do processador

As análises de capacidade de processador referem-se à frequência de operação do processador. Nesse caso, os testes se limitaram a realização de experimentos em três arquiteturas. O cenário foi construído usando máquinas com processadores de diferentes fabricantes com 950 MHz, 1.9 GHz e 2 GHz. Para esse cenário o tamanho de memória não foi considerado na medição.

Contexto Local - Nesse contexto, pedidos de adaptação foram gerados a cada 10

mensagens enviadas considerando o tamanho do buffer igual ao tamanho da mensagem enviada, ou seja, 512 bytes. Os resultados obtidos são mostrados na Figura 3.8.

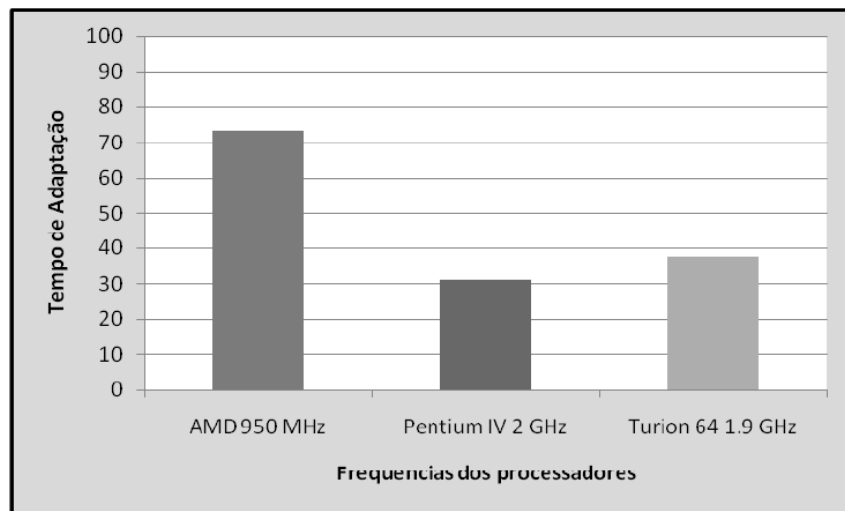


Figura 3.8: Tempo de Adaptação vs Velocidade do Processador Cenário Local.

De acordo com as informações apresentadas da Figura 3.9, podemos notar uma diferença significativa entre os diferentes processadores analisados no experimento. Podemos perceber que ao usar um processador com frequência menor que 1 GHz o tempo de adaptação teve uma média de 73,5 ms, enquanto uma máquina com processador de frequência igual a 1.9 GHz, a média alcançada na adaptação atingiu 37,5 ms. Por fim, a máquina de processador 2 GHz (o mesmo usado nos testes envolvendo o tamanho do buffer) foi alcançado uma média de adaptação igual a 35,7 ms; percebemos que a aproximação no tempo observadas nos dois últimos casos deve-se ao fato de possuírem frequências de processamento aproximada.

Contexto Remoto - Considerando os testes remotos feitos em sistemas com arquiteturas semelhantes aos testes locais acima, a Figura 3.9 mostra os tempos de adaptação obtidos em relação aos diferentes modelos de processadores.

A partir das informações apresentadas da Figura 3.9, podemos notar que o comportamento persiste entre os diferentes processadores usados no experimento. Dessa vez, ao usar um processador com frequência menor que 1 GHz o tempo de adaptação alcançou uma média de 148,35 ms, enquanto que usando uma máquina com processador de frequência igual a 1.9 GHz a média alcançada na adaptação atingiu uma média de 77,7 ms e, por fim, a máquina de processador 2 GHz (o mesmo usado nos testes envolvendo o tamanho do buffer) alcançamos uma média de adaptação igual a 74,79 ms. Portanto, a análise destes dois cenários mostrados poderá ser útil para as diversas fases do ciclo de vida do software, e de forma mais evidente, no momento da instanciação/execução,

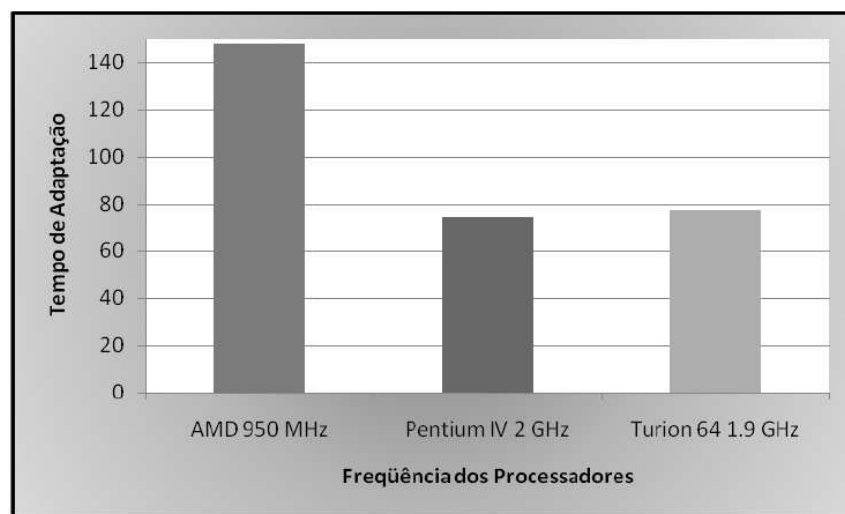


Figura 3.9: Tempo de Adaptação vs Velocidade do Processador Cenário Remoto.

as informações sobre comportamento e a eficiência dos componentes e da plataforma, podendo ajudar no processo de tomada de decisões que satisfaçam os requisitos de QoS do sistema.

3.4.2 Classificação de Critérios

A partir dos cenários e testes analisados na seção anterior, foi definido um conjunto preliminar de critérios que poderão ser utilizados na composição de processos de seleção dinâmica de componentes em sistemas baseados no Cosmos.

Para o Cosmos, políticas de QoS são especificadas juntamente com a configuração de uma aplicação através de uma ferramenta de especificação com características de uma ADL (*Architectural Description Language*). Estas políticas deverão indicar as informações necessárias para a extração dos critérios.

Para a definição, classificação e uso dos critérios de seleção a análise considera os fatores discutidos adiante, os quais foram definidos a partir dos resultados dos experimentos realizados. A abordagem classifica os critérios de QoS em quatro classes de critérios no contexto; vale salientar que o processo deve ser flexível no sentido de aceitar quantidades variadas de critérios ou até mesmo de classes. Porém, esta proposta direcionou o foco para as quatro classes mostradas a seguir:

- *Critério Individual*: está associado com características de cada componente em relação a seu comportamento em aplicações multimídia. Os critérios associados a esta classe estão relacionados com as seguintes propriedades:

- Latência de carga: determina o tempo para o componente alocar os recursos necessários para o seu funcionamento no escopo do sistema;
 - Tamanho do buffer: estimativa de buffer necessário para manipular o stream.
 - Latência de adaptação: tempo médio que o componente gasta para ajustar seu estado (contexto) e de suas tarefas associadas considerando a realização de adaptações com ajustes de propriedades ou com troca de componentes.
- *Critério de rede*: está associado com as características referentes ao estabelecimento de interconexões de componentes, onde os fatores mais críticos referem-se ao uso de tecnologias de comunicação envolvendo infra-estruturas de redes de computadores:
 - Latência: é o tempo que um pacote leva da origem ao destino;
 - Jitter: mede a variação da latência causada na rede;
 - Skew: mede a diferença entre os tempos de chegada de diferentes mídias que deveriam estar sincronizadas.
 - *Critério de estabilidade*: está associado ao tempo médio de uso do componente sem realização de trocas, podendo ser considerado como um indicador preliminar de confiabilidade para o uso no futuro observando a taxa de sucesso em seu funcionamento anterior:
 - Tempo médio de uso: indica o tempo médio de uso do componente.
 - Versão: indicador de versão e maturidade de desenvolvimento.
 - *Outros critérios*: estão associados à definição de critérios gerais que eventualmente poderão ser considerados como fatores adicionais a serem usados pelo processo de seleção de componentes, como, por exemplo:
 - Custo: preço correspondente ao componente que se deseja usar.
 - Fornecedor: entidade responsável pela distribuição do componente.

Para fins de ordenação de prioridades no uso dos critérios, foram estabelecidos três fatores determinantes, os quais estão associados respectivamente ao peso do critério (importância em relação às demais classes de critérios), à informação se o critério é mandatório e aos valores associados a cada critério. Os pesos são estabelecidos pela aplicação para cada classe apresentada de acordo com a sua visão, podendo variar de 0 a 1. A soma dos pesos associados às quatro classes de critérios deve totalizar 1. Estes critérios

Tabela 3.2: Faixa de valores para o critério de Skew.

Faixa de Valores	Níveis de Faixas
0ms a 50ms	Ótimo
51ms a 70ms	Bom
71ms a 80ms	Regular

foram inspirados na teoria da utilidade [36]. Os critérios mandatórios referem-se às condições consideradas como necessárias para o componente; ou seja, caso um componente seja reprovado em algum critério estabelecido como mandatório, automaticamente este componente estará fora da seleção, mesmo que para os outros critérios ele apresente as melhores notas.

Os valores associados a cada critério de QoS podem ser representados como intervalos de valores ou atômica, devendo estar vinculados aos três níveis de QoS considerados no modelo como aceitáveis para a aplicação: regular, bom e ótimo. Esses três níveis são usados na pontuação do componente, com as notas 1, 3 e 5 respectivamente associadas aos correspondentes critérios.

Por exemplo, se para o critério skew (critério de rede) o valor máximo estabelecido como aceitável sem nenhum desconforto para a percepção humana seja 80 ms como descrito em [37], o projetista poderia definir as faixas para esse critério de acordo com os níveis apresentadas na Tabela 3.2, retirada de [37].

Vale destacar que a manipulação das classes de critérios está associada à política de QoS definida para o sistema. Com o objetivo de tratar as questões relacionadas com a descrição e manipulação desses critérios de forma independente de tecnologia, a seção seguinte apresenta um meta-modelo para definição de critérios e políticas de seleção de componentes.

3.4.3 Modelo para Definição de Critérios

De forma a permitir a definição de modelos de seleção flexíveis e adaptáveis, com diferentes formas de composição e/ou configuração, definimos um metamodelo (Figura 3.10) que relaciona os principais elementos que podem afetar o processo de seleção. Este metamodelo permite que o processo proposto possa ser ajustado para ser instanciado em diferentes plataformas. A partir deste metamodelo e do conhecimento das propriedades dos componentes, assim como do comportamento prévio dos mesmos em outras situa-

Tabela 3.3: Faixas de valores dos critérios de rede. A última coluna mostra os valores dos mesmos para o componente.

	Ótimo (5)	Bom (3)	Regular (1)	Crítérios
Jitter	20ms a 23ms	24ms a 30ms	31ms a 34ms	25ms
Latência	15ms a 17ms	18ms a 23ms	24ms a 26ms	16ms
Skew	0ms a 50ms	51ms a 70ms	71ms a 80ms	75ms

ções (por exemplo, em outras plataformas e/ou aplicações), pode-se ajustar um modelo de critérios e seus respectivos pesos no processo. Na Seção 3.4.4 a seguir, apresentamos um modelo que definimos para uma instanciação do processo de seleção, o qual serviu de base para a implementação de um módulo de seleção de componentes, denominado AnaMoc, proposto e implementado por [38].

Na descrição de um modelo de processo de seleção de componentes, a definição de uma política consiste na escolha das classes de critérios que deve ser considerada para efeito de avaliação da QoS dos componentes candidatos. Para cada classe, a política descreve os fatores determinantes e os valores de QoS associados aos respectivos critérios. Conforme pode ser observado na Figura 3.10, uma especificação pode tratar várias classes de critérios. Cada política pode ter diferentes versões que identificam mudanças de abordagens de QoS dependendo, por exemplo, da plataforma onde a aplicação vier a ser implantada.

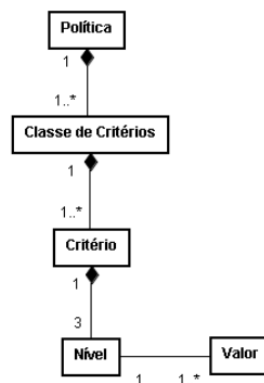


Figura 3.10: Modelo para definição de critérios e políticas de avaliação e seleção de componentes

Cada classe de critérios tem um peso(atributo de classe). As classes de critérios podem definir vários critérios. Cada critério é composto por três níveis que identificam os níveis aceitáveis para o sistema, correspondendo aos níveis ótimo, bom e regular. Cada

nível é descrito por um único valor atômico ou intervalos de valores. Numa instanciação do processo de avaliação, cada valor terá suas respectivas notas configuradas pelo gerenciador de QoS do sistema.

3.4.4 AnaMoC - Um Módulo para Análise e Seleção de Componentes

O AnaMoC (*Analyzer Module for Component Selection*) foi construído para funcionar como um módulo de apoio dentro de plataformas baseadas no *framework* Cosmos [2]. Seu projeto foi baseado no modelo de seleção discutido na seção anterior, no qual, dada a descrição dos componentes candidatos à seleção (normalmente solicitada por um processo de adaptação dinâmica com troca de componentes), o módulo deve indicar para o sistema, dentre os componentes fornecidos, aquele que é considerado como o melhor no momento para realizar a substituição, ou até mesmo servir como um indicador de quais componentes utilizar na fase de *deployment* do projeto da aplicação. Para isso, o módulo considera a política de QoS definida para o sistema que será interpretada e analisada pelo AnaMoC.

O cenário de uso descrito na Figura 3.11 supõe, num dado momento que a aplicação deseja melhorar os índices de QoS após perceber que esses valores não condizem com a política definida, e, para isso, ele precisa realizar a troca de um certo componente, responsável, por exemplo, pela codificação de fluxo de vídeo, por outro, dentre as opções possíveis, correspondentes aos componentes A, B, C e D. O AnaMoC obtém as informações dos critérios dos componentes candidatos para então fazer a análise e processar a seleção, indicando na saída, como selecionado, o componente B, o qual foi avaliado no exemplo como o melhor, segundo a política de QoS estabelecida para fazer a codificação do fluxo.

Esta seção discute, em detalhes, os principais aspectos relacionados com a arquitetura de implementação do módulo de seleção de componentes, envolvendo questões relacionadas com a estrutura e inserção na plataforma destino.

3.4.4.1 Estrutura da Política

A política utilizada pelo AnaMoC foi descrita considerando a linguagem ADL (*Architecture Description Language*) definida originalmente no Cosmos [2]. Entretanto, mesmo considerando que o Cosmos possui construções para especificação de QoS para a comunicação entre seus componentes da aplicação, a política definida pelo AnaMoC deve ser

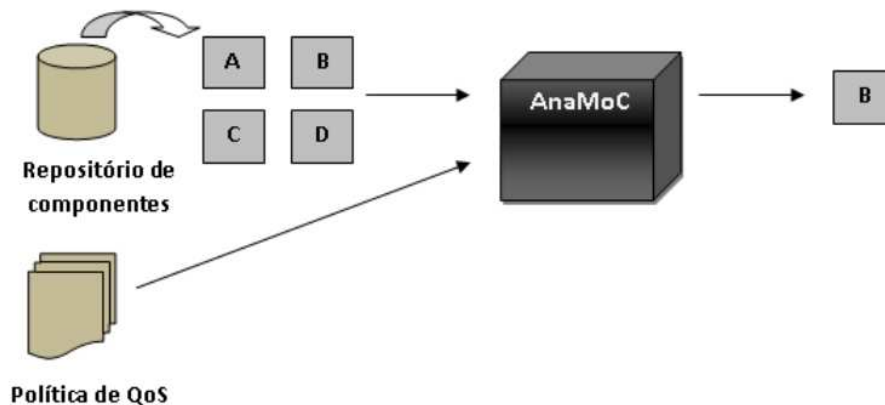


Figura 3.11: Cenário de uso para a escolha de um componente decodificador de fluxo de vídeo

escrita de forma independente, de maneira que permita o módulo ser utilizado também por outras plataformas. Adicionalmente, isso permite que o projetista possa definir para cada aplicação políticas diferentes dentro da mesma plataforma ou aplicação. Por exemplo, poderia ser definida uma política para a transmissão de vídeo MPEG e outra para a transmissão de vídeo AVI, ou até mesmo, na transmissão de um filme, uma política para a transmissão do vídeo, outra para o áudio e outra para a legenda.

A descrição de políticas e de componentes é feita usando a linguagem XML, na qual alguns termos empregados derivam da ADL do Cosmos [2]. O XML Schema definido para descrição de políticas está apresentado no Apêndice A.

Baseado no XML Schema apresentado no Apêndice A, podemos exemplificar, através do código 3.3 abaixo, o documento que descreve uma política da aplicação a ser usada pelo módulo. Vale salientar que os valores mostrados servem apenas como demonstração para a compreensão da estrutura da política.

Código-fonte 3.3: Trecho de código da especificação de uma política.

```

<POLICY_QoS name="Policy_Cosmos" version="1" threshold="11.5" decide=
"note">
  <CLASS_OF_CRITERION name="Individual" weight="0.40">
    <CRITERION name="Load_Latency" type="range" unit="seconds"
5      mandatory ="false">
      <GREAT min="0.86" max="3"></GREAT>
      <GOOD min="4" max="6"></GOOD>
      <REGULAR min="7" max="9"></REGULAR>
    </CRITERION>
10    <CRITERION name="Buffer_Size" type="atomic" unit="bytes"
      mandatory ="true">

```

```

    <GREAT value="512"></GREAT>
    <GOOD value="1024"></GOOD>
    <REGULAR value="2056"></REGULAR>
15 </CRITERION>
    <CRITERION name="Adaptation_Latency" type="range" unit=
    "milliseconds">
        <GREAT min="1" max="2"></GREAT>
        <GOOD min="3" max="4"></GOOD>
20 <REGULAR min="5" max="6"></REGULAR>
    </CRITERION>
</CLASS_OF_CRITERION>
<CLASS_OF_CRITERION name="Network" weight="0.30">
    <CRITERION name="Latency" type="range" unit="milliseconds"
25 mandatory="false">
        <GREAT min="10" max="16"></GREAT>
        <GOOD min="17" max="19"></GOOD>
        <REGULAR min="20" max="22"></REGULAR>
    </CRITERION>
30 <CRITERION name="Jitter" type="range" unit="milliseconds"
    mandatory="false">
        <GREAT min="30" max="36"></GREAT>
        <GOOD min="37" max="42"></GOOD>
        <REGULAR min="43" max="46"></REGULAR>
35 </CRITERION>
    <CRITERION name="Skew" type="range" unit="milliseconds"
    mandatory="false">
        <GREAT min="0" max="50"></GREAT>
        <GOOD min="51" max="70"></GOOD>
40 <REGULAR min="71" max="80"></REGULAR>
    </CRITERION>
</CLASS_OF_CRITERION>
<CLASS_OF_CRITERION name="Stability" weight="0.20">
    <CRITERION name="Use_Time" type="range" unit="hours" mandatory
45 ="false">
        <GREAT min="500" max="700"></GREAT>
        <GOOD min="350" max="499"></GOOD>
        <REGULAR min="290" max="349"></REGULAR>
    </CRITERION>
50 <CRITERION name="Version" type="range" unit="decimal" mandatory
    ="false">
        <GREAT min="20"></GREAT>
        <GOOD min="5" max="8"></GOOD>
        <REGULAR min="3" max="4"></REGULAR>
55 </CRITERION>
</CLASS_OF_CRITERION>

```

```

        <CLASS_OF_CRITERION name="Others" weight="0.10">
            <CRITERION name="Cost" type="range" unit="monetary_unit"
                mandatory
                = "false">
60         <GREAT min="49" max="70"></GREAT>
            <GOOD min="71" max="80"></GOOD>
            <REGULAR min="81" max="100"></REGULAR>
        </CRITERION>
        <CRITERION name="Supplier" type="atomic" unit="
            certification_entity"
65     mandatory = "false">
            <GREAT value="0"></GREAT>
            <GOOD value="B"></GOOD>
            <REGULAR value="R"></REGULAR>
        </CRITERION>
70     </CLASS_OF_CRITERION>
</POLICY_QoS>

```

Isso significa que o parser existente na implementação do protótipo do *framework* Cosmos teve que ser ajustado para processar a leitura desse arquivo e repassar as informações ao AnaMoC. As informações referentes aos componentes são retiradas do repositório de componentes do Cosmos representado, por exemplo, no código 3.4 abaixo.

Código-fonte 3.4: XML representando componentes.

```

<REPOSITORY>
  <COMPONENT id="0">
    <PROPERTIES>
      <PROPERTY name="load_latency" value="9.63982"></PROPERTY>
5     <PROPERTY name="buffer_size" value="512"></PROPERTY>
      <PROPERTY name="adaptation_latency" value="9.020421"></PROPERTY>
      <PROPERTY name="latency" value="15.391293"></PROPERTY>
      <PROPERTY name="jitter" value="10.937062"></PROPERTY>
      <PROPERTY name="skew" value="78.23136"></PROPERTY>
10     <PROPERTY name="use_time" value="984"></PROPERTY>
      <PROPERTY name="version" value="44"></PROPERTY>
      <PROPERTY name="cost" value="2.6927462"></PROPERTY>
      <PROPERTY name="supplier" value="R"></PROPERTY>
    </PROPERTIES>
15   </COMPONENT>
  <COMPONENT id="1">
    <PROPERTIES>
      <PROPERTY name="load_latency" value="0.047543414"></PROPERTY>
      <PROPERTY name="buffer_size" value="1024"></PROPERTY>
20     <PROPERTY name="adaptation_latency" value="3.8972824"></PROPERTY>

```

```

25 <PROPERTY name="latency" value="29.960718"></PROPERTY>
    <PROPERTY name="jitter" value="25.696056"></PROPERTY>
    <PROPERTY name="skew" value="54.55846"></PROPERTY>
    <PROPERTY name="use_time" value="135"></PROPERTY>
    <PROPERTY name="version" value="14"></PROPERTY>
    <PROPERTY name="cost" value="256.84174"></PROPERTY>
    <PROPERTY name="supplier" value="B"></PROPERTY>
    </PROPERTIES>
    </COMPONENT>
30 </REPOSITORY>

```

3.4.4.2 Usando o AnaMoC para selecionar Componentes em Plataformas Cosmos

O módulo AnaMoC pode ser tratado como um componente caixa-preta, escondendo sua composição interior e mostrando apenas como se dá seu processo de seleção. Estrutura de implementação definida por [38] definiu o AnaMoc como um componente (Figura 3.12) que possui uma interface (Figura 3.13) bem definida a ser utilizada pelo middleware. Em sua implementação foi usada a linguagem de programação Java por manter consistente à implementação original do protótipo criado para executar sobre a plataforma do Cosmos, o AdapTV.

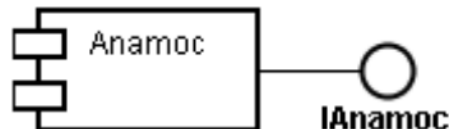


Figura 3.12: Componente AnaMoC

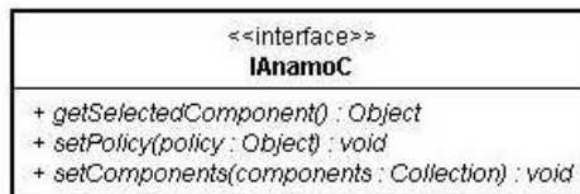


Figura 3.13: Interface *IAnamoc*

- *getSelectedComponent*: método responsável por devolver à aplicação o objeto que representa o componente escolhido no processo de seleção e por iniciar o processo de seleção.

- *setPolicy*: método responsável por receber a política da aplicação a ser interpretada pelo módulo
- *setComponents*: método responsável por receber os componentes armazenados no repositório da aplicação.

A seguir, descreve-se como se proceder durante a execução para escolher o componente considerado como o melhor, dentre um conjunto conhecido de possibilidades. A descrição e análise serão apresentadas utilizando diagramas de seqüência da UML 2.0, de acordo com os métodos definidos em cada interface dos componentes

As Figuras 3.14 e 3.15 mostram, de forma resumida, duas opções para usar o módulo AnaMoC e como obter o resultado do processo de seleção; neste caso, apenas está sendo considerada a comunicação direta do *middleware* AdapTV [3] com o módulo.

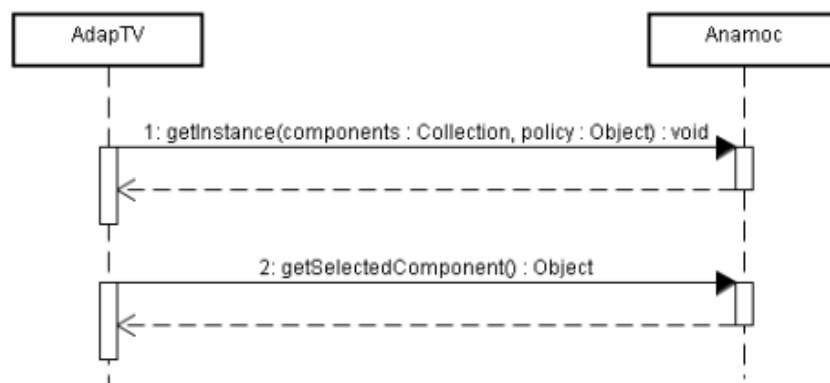


Figura 3.14: Opção 1 de uso do AnamoC

Na primeira forma de se utilizar o AnaMoC é passar para o método *getInstance* a política e a especificação dos componentes como argumento, após isso o método *getSelectedComponent* pode ser invocado, retornando o componente selecionado. Na segunda forma é possível utilizar o *getInstance* para recuperar um objeto do tipo AnaMoC e depois informar a partir de métodos de sua interface a política e os componentes para depois recuperar o componente selecionado.

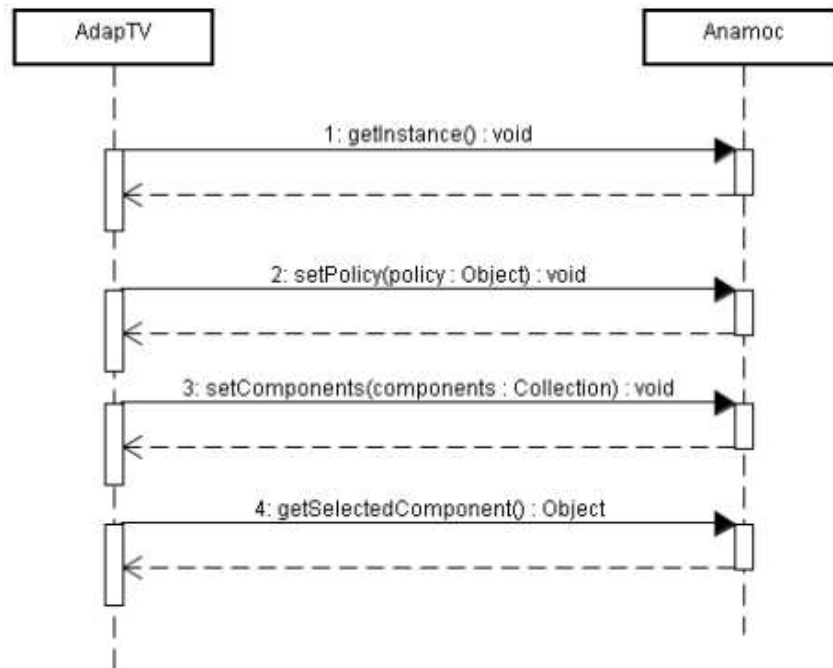


Figura 3.15: Opção 2 de uso do Anamoc

3.5 Mecanismo de Troca

Para concluir o processo de substituição de componentes proposto para o Cosmos, apresentamos o mecanismo definido para dar suporte a troca efetiva de componentes. As seções anteriores introduziram os passos que antecedem a troca de componentes segundo a abordagem proposta.

Antes de se decidir por realizar a troca, é necessário analisar se uma adequação de uma das propriedades do componente em uso, como explorado em [3], não seria suficiente. Ou seja, caso o componente não disponha de facilidades de ajustes, conforme a especificação fornecida na ADL da aplicação. Se esta análise detectar que esta ação é suficiente para manter o nível de QoS do serviço, a troca de propriedade é realizada; por sua vez, caso o componente atual não consiga manter um nível adequado de qualidade, o módulo seletor de componentes definido na seção 3.2.2 é instanciado para promover uma análise das propriedades dos componentes candidatos juntamente com o componente atual, retornando o componente candidato eleito como substituto para troca.

3.5.1 Implantação da Proposta no Framework Cosmos

Como introduzido na Seção 3.4.4, a primeira etapa da implementação consistiu-se na criação da classe `ComponenteChooserService`, incorporando assim um serviço para a escolha de componentes dentro do Cosmos. Para isso, foi necessário fazer ajustes na infraestrutura de forma a poder integrar a proposta com o AnaMoC.

3.5.1.1 Incorporação do AnaMoC

O trabalho [38], definiu um módulo para a seleção de componentes baseado nos critérios de QoS analisados na Seção 3.4.2. Para incorporar o AnaMoc ao `ComponenteChooserService` (foco da abordagem proposta), realizamos uma modificação numa classe deste módulo para ajustar o seu analisador sintático de forma a recuperar as propriedades dos componentes a partir de um arquivo XML . O código 3.4 descreve um repositório de componentes para o AnaMoC. Como descrito, o repositório recupera as propriedades dos componentes os diferenciando por uma propriedade `id`. A presente proposta insere no AnaMoC a propriedade `class`; esta propriedade deve conter o nome da classe do componente.

Devido a sua natureza genérica, o AnaMoC trabalha apenas com a descrição de propriedades dos componentes. Para atender a necessidade específica do serviço proposto por este trabalho no contexto do framework Cosmos, decidimos alterar o AnaMoC inserindo uma propriedade `class`. Essa alteração foi necessária para ajustar a localização da classe, e, com isso, viabilizar a sua instanciação. Para o desenvolvimento de aplicações segundo a versão original do AnaMoc, o projetista define o repositório de componentes em um único ponto de configuração, mapeando classes para `id`. Depois da escolha também é necessário buscar a classe para a sua instanciação. Abaixo apresentamos o código 3.5 que mostra como deve ficar o nome XML da especificação dos componentes para o AnaMoC.

Código-fonte 3.5: XML representando componentes após a troca do AnaMOC.

```
<REPOSITORY>
  <COMPONENT id="0">
    <PROPERTIES>
      <PROPERTY name="load_latency" value="9.63982"></PROPERTY>
5     <PROPERTY name="buffer_size" value="512"></PROPERTY>
      <PROPERTY name="adaptation_latency" value="9.020421"></PROPERTY>
      <PROPERTY name="latency" value="15.391293"></PROPERTY>
      <PROPERTY name="jitter" value="10.937062"></PROPERTY>
```

```

    <PROPERTY name="skew" value="78.23136"></PROPERTY>
10  <PROPERTY name="use_time" value="984"></PROPERTY>
    <PROPERTY name="version" value="44"></PROPERTY>
    <PROPERTY name="cost" value="2.6927462"></PROPERTY>
    <PROPERTY name="supplier" value="R"></PROPERTY>
    <PROPERTY name="class" value="br.natalnet.adaptv.ProducerPT_BR"><
        /PROPERTY>
15  </PROPERTIES>
    </COMPONENT>
    <COMPONENT id="1">
        <PROPERTIES>
            <PROPERTY name="load_latency" value="0.047543414"></PROPERTY>
20  <PROPERTY name="buffer_size" value="1024"></PROPERTY>
            <PROPERTY name="adaptation_latency" value="3.8972824"></PROPERTY>
            <PROPERTY name="latency" value="29.960718"></PROPERTY>
            <PROPERTY name="jitter" value="25.696056"></PROPERTY>
            <PROPERTY name="skew" value="54.55846"></PROPERTY>
25  <PROPERTY name="use_time" value="135"></PROPERTY>
            <PROPERTY name="version" value="14"></PROPERTY>
            <PROPERTY name="cost" value="256.84174"></PROPERTY>
            <PROPERTY name="supplier" value="B"></PROPERTY>
            <PROPERTY name="class" value="br.natalnet.adaptv.ProducerEN"></
                PROPERTY>
30  </PROPERTIES>
        </COMPONENT>
    </REPOSITORY>

```

3.5.1.2 Evolução do Modelo de QoS do Cosmos

Nos trabalhos iniciais, [35] definiu um modelo que permitia apenas a adaptação por troca de propriedades do componente em execução; para isso, o modelo de QoS descrito em [35] e discutido na seção 2.4.5 permite expressar intervalos de QoS para definir faixas de valores aceitáveis para o funcionamento da aplicação. Com base neste conceito, o desenvolvedor da aplicação deve definir faixas de utilização, como apresentado no exemplo da especificação de uma aplicação em XML no código seguinte:

Código-fonte 3.6: Trecho de Código da especificação de QoS de uma aplicação.

```

    <QOS parameter = "QoSBandwidth">
        <DEFAULTREGION> High </DEFAULTREGION >
        <FREQUENCY> 100.00</FREQUENCY>
        <TESTTIME> 1000.00</TESTTIME>
5  <REGIONS>

```

```
<REGION name = "High" >
  <RANGE min = "30" />
  <PROPERTY_DEFAULT name="framerate"
    values = "30" />
</REGION>
10 <REGION name = "Normal" >
  <RANGE max = "29" min = "10" />
  <PROPERTY_DEFAULT name="framerate"
    values = "18" />
</REGION>
15 <REGION name = "Low" >
  <RANGE max = "9" />
  <PROPERTY_DEFAULT name="framerate"
    values = "7" />
</REGION>
</REGIONS>
</QOS>
```

Intervalos consistem de valores máximos e mínimos que são utilizados para fins de monitoração pelos elementos de QoS do framework Cosmos. Quando existe uma mudança de faixa, o Configurador da aplicação é notificado devendo tomar uma decisão sobre qual atitude tomar. Nos trabalhos anteriores, os esforços foram direcionados para mudanças dinâmicas de propriedades [3], consistindo em tentar incrementar ou decrementar a qualidade de serviço para as faixas adjacentes.

Para o presente trabalho, o conceito de faixas foi mantido. Para facilitar a compreensão da mudança introduzida, vamos exemplificar o processo seguindo o exemplo de parâmetro de QoS especificado acima, que consiste em controlar a QoS relacionada com framerate (quantidade de quadros por segundo de um fluxo de vídeo). Quanto maior a quantidade de quadros por segundo, melhor será a sensação para quem estiver assistindo o vídeo.

No XML da aplicação vemos que existe um elemento chamado REGION que contém o atributo name, no caso da linha 6, o valor é High, significando que os valores existentes para a propriedade de nome framerate (Linha 8 do Código 3.6) que estejam neste intervalo serão os de melhor nível de qualidade; então, a aplicação deve prover a melhor qualidade possível, prover a melhor qualidade possível. De acordo com o projetista desta aplicação, o valor para que a aplicação funcione com o melhor nível de qualidade é, no mínimo, 30 quadros por segundo.

Na aplicação também existem outras regiões: Normal e Low, onde os valores para a aplicação funcionar na região Normal é de no mínimo 10 quadros por segundo e no

Tabela 3.4: Componentes e faixas possíveis

Componente	Faixa Normal	Faixa High	Faixa Low
C1		X	X
C2	X		
C3			X

máximo 29 quadros por segundo. Por sua vez, para a região Low, o valor máximo desta faixa é de 9 quadros por segundo.

Os monitores de QoS do Cosmos identificam, como exemplificado na Seção 2.3, e indicam qual a faixa atual de funcionamento, de acordo com o atual nível de QoS do sistema; assim, caso seja necessário uma adaptação, esta é realizada.

Nos trabalhos iniciais [35] o sistema permitia apenas a adaptação por troca de propriedades do componente em execução; uma adaptação consistia em tentar incrementar ou decrementar a qualidade de serviço para as faixas adjacentes.

Neste trabalho estendemos o procedimento de especificação de aplicações no Cosmos, inserindo uma alteração no mecanismo de especificação de QoS; nesta extensão são mantidos os conceitos de faixas, que eram focadas na configuração da conexão virtual, e, que por conta disso, os projetistas para proverem flexibilidade de adaptação, precisavam definir componentes que tivesse capacidades de operarem nas três faixas. Com a abordagem proposta, onde a adaptação pode envolver a troca de componentes, o arquiteto pode desenvolver um projeto onde, por exemplo, componentes sejam concebidos para funcionarem em somente uma faixa, ao invés das três faixas-alvo. Isto tem como impacto a simplificação do processo de desenvolvimento de componentes uma vez que ele pode se restringir a operar numa única faixa de valores, não sendo necessário que eles sejam desenvolvidos para trabalhar nas três faixas. Como exemplo podemos ter dois componentes que tenham capacidades de transmitir um fluxo em faixas diferentes. A Tabela 3.4 exemplifica esta situação onde os componentes podem transmitir em diferentes faixas de QoS.

Utilizando a Tabela 3.4 como base, podemos perceber que o componente C1 pode transmitir em 2 faixas, enquanto o componente C2 permite somente na faixa Normal; por sua vez o componente C3 pode transmitir somente na faixa Low.

Em um processo de adaptação, o gerenciador de QoS do Cosmos irá informar a necessidade de mudança de faixa para o Configurador, que tomará a decisão de qual componente escolher. Conforme descrito na seção 3.4.4, o mecanismo de escolha deve retornar

o componente alvo para a troca conforme explicação apresentada na seção seguinte.

Como exemplo no Código 3.2 temos a propriedade *QoSRegion*, que deve especificar quais faixas o componente trabalha. A partir desta propriedade do componente o configurador sabe se o componente consegue se adaptar a uma mudança de faixa trocando internamente a propriedade do componente [3] ou é necessário realizar a troca do componente.

3.5.1.3 Salvando e Recuperando o Contexto

Na abordagem de [3], durante a adaptação a responsabilidade de sincronização da mídia é do componente. Ou seja, ao se desenvolver o componente para o Cosmos, o programador deve se preocupar com todas as operações envolvidas nas etapas da troca de propriedades. Não havia nenhuma preocupação, nem tampouco necessidade de salvar e/ou recuperar contextos.

Como atualmente está sendo incorporada a troca dinâmica de componentes, se faz necessário introduzir uma nova interface para salvar e recuperar o contexto de componentes. Assim, um componente substituto pode ser configurado para iniciar sua operação no estado e contexto em que se encontrava o componente antigo.

Este trabalho define a interface apresentada na Figura 3.16, que deverá ser incluída em componentes Cosmos, de modo a permitir o salvamento e recuperação de informações de estado e contexto de componentes. A interface definida considera o contexto como um número inteiro longo (*long*), consistindo numa simplificação onde este valor denota a posição em milissegundos da mídia, que deverá ser passada para o componente substituto. Isto significa que, após a troca, o componente substituto deve iniciar a sua operação a partir desta posição.

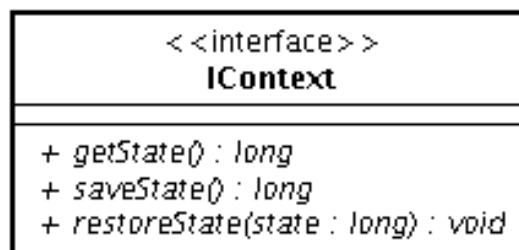


Figura 3.16: Interface IContext

Por exemplo, caso seja necessária a adaptação por troca de componentes, e, se o componente atual indicado como alvo da troca estiver transmitindo o fluxo na posição 293010

(4m53.010s), após a troca, o componente substituto deve começar a transmitir na posição 293011(4m53.011s), o serviço ComponentChangeService utiliza esta interface para salvar e restaurar o contexto dos componentes envolvidos.

Futuramente, a interface deverá evoluir para incorporar informações mais reais de contexto como nos estudos de caso explorados em [39].

3.5.1.4 Substituição do componente

O ultimo passo do processo consiste na troca do componente alvo pelo substituto. Após realizar a escolha do substituto, é chegada a hora de efetivar a substituição. Esta troca, conforme foi discutido no início do capítulo é realizada por um mecanismo definido no escopo de serviço de substituição definido para o Cosmos no contexto da classe ComponentChangeService.

Usando como base a Figura 3.17 vemos que componentes VirtualConnection em uma aplicação deverão ser interligados com outros através de uma Port, sendo esta de entrada ou de saída, como foi explicado na seção 2.4. Uma porta contém um buffer associado, a partir do qual os dados são consumidos pelo cliente

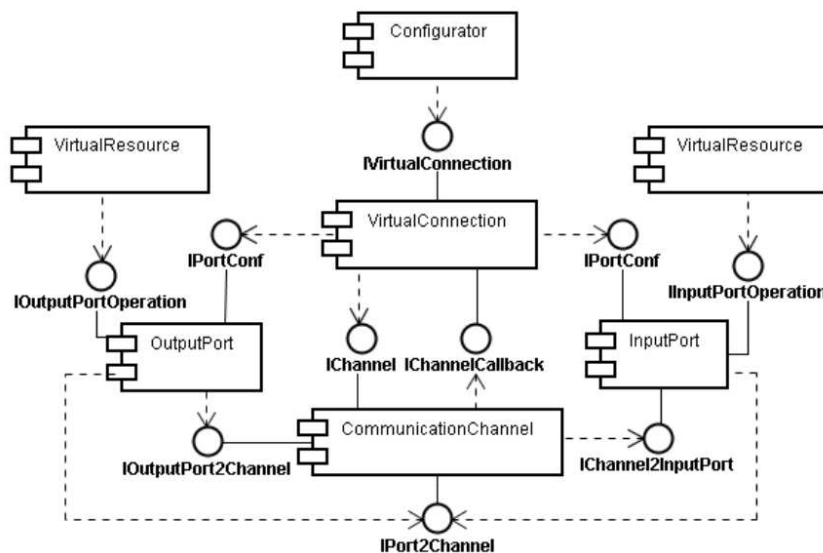


Figura 3.17: Visão Funcional do Modelo de Interconexão.

Os testes experimentais realizados com objetivo de analisar e definir critérios de substituição foram direcionados para os componentes produtores de fluxo. A composição do fluxo considerado no processo é descrita a seguir: o VirtualResource produtor gera o fluxo e envia para a porta de saída, que por sua vez através do canal de comunicação envia este

fluxo para a porta de entrada destino; a partir desta porta, o VirtualResource consumidor obtém os respectivos dados do fluxo.

Como o processo de adaptação está direcionado para a substituição do produtor, a referência, para efeito de monitoramento e gerenciamento da QoS, é a porta de saída (OutputPort). Observando então a porta referenciada, o mecanismo deve realizar um cálculo envolvendo o tamanho de buffer adequado para evitar que o buffer se esvazie antes da troca ser efetivada. Caso necessário, o buffer deve então ser redimensionado. Neste cálculo são considerados fatores como plataforma, latência de carga, latência de troca do componente, dentre outros observados.

Como o Cosmos foi projetado com um baixo acoplamento entre seus componentes é possível iniciar a troca do componente produtor independente da porta de saída.

Os principais eventos que determinam as ações estão descritos no diagrama de seqüência apresentado na Figura 3.18; este diagrama contém os passos mais significativos da abordagem.

A seguir, será apresentado um resumo dos passos mais relevantes para a abordagem.

1. O QoSMonitor notifica o seu manager de que existe uma anomalia;
2. O QoSManager notifica o Configurator, informando a mudança na faixa de QoS;
3. O Configurator analisa se é possível realizar uma adaptação no componente atual considerando a troca de propriedades (envolvendo a técnica de clonagem de canal); caso contrário o serviço de troca de componentes é invocado;
4. O serviço de troca recupera os metadados da aplicação;
5. O serviço de troca utiliza o serviço de seleção de componentes para selecionar um componente que se adeque a nova faixa de QoS;
6. O componente selecionado é instanciado;
7. A conexão virtual é informada para redimensionar o buffer com o tamanho necessário para a realização da troca;
8. O estado do componente atual é recuperado;
9. A negociação para a troca é iniciada;
 - (a) A conexão virtual é notificada da troca;

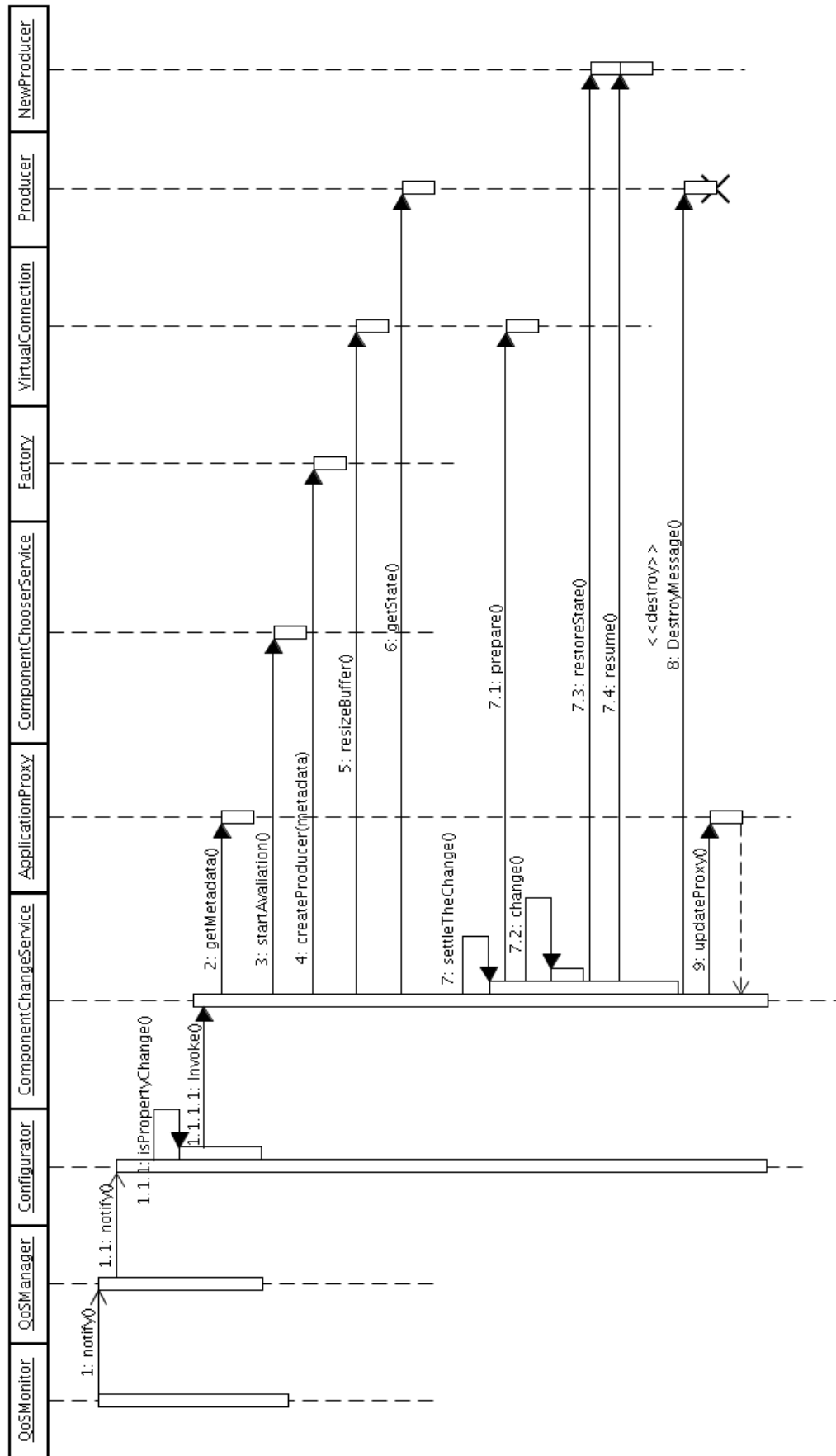


Figura 3.18: Diagrama de Sequência da Abordagem para a Troca de Componentes.

- (b) O novo componente é alocado;
 - (c) O estado é atualizado, e
 - (d) O novo componente é ativado.
10. O antigo componente é desalocado, e
11. O proxy é atualizado.

Capítulo 4

Trabalhos Relacionados

Este capítulo faz uma análise da nossa proposta comparando-a com alguns trabalhos relacionados. Considerando que a proposta envolve questões relacionadas com o processo definido para adaptação dinâmica com troca de componentes em sistemas multimídia distribuídos baseados no Cosmos, a estrutura do capítulo procura relacionar cada etapa do processo no contexto do estado da arte.

4.1 Frameworks para o desenvolvimento de aplicações Multimídia

A existência de diversas propostas de middleware adaptativos e de seus respectivos modelos de comunicação voltados para ambientes multimídia evidencia a necessidade da definição de mecanismos específicos para este tipo de ambiente. O framework Cosmos tem caminhado nesta direção, definindo uma abordagem arquitetural, baseada no conceito de componentes, para dar suporte ao desenvolvimento de uma variedade de sistemas multimídia distribuídos. O esforço que está sendo dedicado atualmente no escopo do Cosmos está direcionado para suporte a características de auto-adaptação.

Dado que o esforço do presente trabalho se agrega ao Framework Cosmos, cabe fazer algumas análises do mesmo relacionadas com outras propostas discutidas na literatura, a tabela 4.1 na seção 4.1.5 faz um comparativo entre eles.

4.1.1 PREMO

O framework PREMO (Presentation Environment for Multimedia Objects) [40], proposto pela ISO, foi baseado no modelo RM-ODP. Ele leva em consideração um extenso conjunto de requisitos para aplicações multimídia distribuídas. O PREMO define um modelo de objetos e de sincronização, assim como um conjunto de serviços multimídia. Seus principais elementos são as abstrações para dispositivos (Virtual Devices), portas (port) e conexões virtuais (Virtual Connections). Estas entidades suportam conexões ponto-ponto e ponto-multiponto. Um recurso virtual representa uma unidade de processamento, podendo ser uma entidade de software ou uma entidade de hardware. As portas são utilizadas para realizar a ligação entre recursos virtuais. A porta suporta diversos formatos de mídia associados que podem ser consultados ou configurados. A conexão virtual abstrai questões relacionadas ao gerenciamento e transferência de dados entre os dispositivos. Ela não realiza a transferência de dados entre os dispositivos. Ela é responsável por separar a conexão em diferentes elementos.

O estabelecimento de uma conexão no PREMO envolve a definição de um tipo (local ou remota), a negociação de formatos do fluxo, a QoS e as capacidades dos elementos envolvidos. Entretanto, o modelo para conexão virtual definido no PREMO não permite que a aplicação se envolva na definição de propriedades do processamento e transporte do fluxo, o que inviabiliza a realização de adaptação dinâmica, não se adequando portando aos requisitos perseguidos pelo projeto do framework Cosmos.

4.1.2 NMM

O projeto NMM (Network-Integrated Multimedia Middleware) [30] provê uma arquitetura baseada em objetos para a construção de aplicações multimídia distribuídas em ambiente GNU/Linux. O NMM foi desenvolvido utilizando a linguagem C++, e hoje se tornou um projeto de software livre sendo distribuído sob as licenças GPL e LGPL. Um sistema NMM consiste de uma coleção de nós (nodes), que são entidades de software independentes e compõem as unidades básicas do sistema. Os nós constituem um grafo, formando um pipeline onde cada nó realiza uma função específica no fluxo multimídia.

Os nós (nodes) são os elementos funcionais do NMM, representando dispositivos de hardware e software. Os nós se conectam através dos jacks, que representam os pontos de entrada e saída dos nós. Os buffers contêm os dados que são transportados através dos jacks de um nó para o outro, sendo gerenciados por um gerente (buffermanager). Um formato (format) é uma descrição da estrutura e função dos dados que compõem um fluxo

multimídia. Os formatos determinam os tipos de dados de um fluxo e seus parâmetros. Os nós são conectados formando um grafo, onde o formato de dois nós que se comunicam devem ser iguais.

Toda comunicação no NMM utiliza um sistema de mensagem unificada que é composta por dois tipos de objetos: Buffers, utilizado para a transmissão de dados multimídia, que podem estar em uma fila para ser processado; e eventos compostos (Composite Events), que são utilizados para controlar o comportamento de um nó. As mensagens de eventos são enviadas de duas maneiras: in-stream e out-of-band. Nos eventos in-stream, as mensagens são encaminhadas da mesma maneira que em buffers. Nos eventos out-of-band, as mensagens são enviadas se comunicando diretamente com os nós instanciados através de chamadas remotas. Um evento composto consiste de um conjunto de eventos simples (ou comandos) ordenados.

Através do jack de entrada o nó recebe buffers ou eventos in-stream, que são tratados de maneiras distintas. Ao mesmo tempo, o nó pode receber eventos out-of-band da aplicação ou diretamente de outros nós, que correspondem a eventos de controle. Através de seu jack de saída, ele envia buffers ou eventos que serão tratados pelo nó seguinte ou encaminhado para o próximo nó da cadeia.

Os jacks são responsáveis por transportar os dados dos buffers de um nó para o outro, representando a entrada e saída de um nó. Os nós são criados com seus formatos de entrada e de saída definidos, e, posteriormente, os jacks associados são criados e conectados. Diversos jacks de saída podem ser combinados em um container chamado Jack-Group, permitindo o envio de dados para diversos receptores de forma transparente para o emissor. Os jacks do NMM só são criados após a definição do formato a ser utilizado pelos nós, não suportando nenhum tipo de configuração.

Com a definição dos formatos suportados por um nó, o NMM suporta negociação antes da criação de um grafo, tendo como indicação de trabalho futuro a inserção de elementos adaptadores quando necessário.

4.1.3 O Sistema de Componentes SATIN

SATIN é um metamodelo que descreve um modelo de componentes para desenvolvimento de sistemas adaptáveis para dispositivos móveis. A base deste sistema consiste em definir primitivas para reconfiguração de sistemas a partir de processos de transferência de código. Uma descrição da implementação do metamodelo é apresentada no middleware SATIN - um middleware para computação móvel baseado em componentes [18]. Esta

implementação usa as primitivas definidas no metamodelo para reconfigurar o próprio sistema e a aplicação.

Algumas das idéias principais do SATIN são as seguintes:

- Tudo é um componente;
- O SATIN é estruturalmente um sistema reflexivo, e
- As aplicações desenvolvidas em SATIN utilizam primitivas lógicas para transferir objetos, classes, dados e componentes para outros nós.

O SATIN, assim como as suas aplicações derivadas, são representadas como um conjunto de componentes interligados. Um componente SATIN encapsula uma série de funcionalidades. Como exemplos podemos citar: interfaces de usuário; códigos de áudio; Os componentes separam a implementação da interface.

O SATIN, assim como as suas aplicações derivadas, são representadas como um conjunto de componentes interligados. Um componente SATIN encapsula uma série de funcionalidades, como exemplos podemos citar: interfaces de usuário; códigos de áudio; bibliotecas de compressão, etc. Os componentes separam a implementação da interface.

Os componentes são descritos utilizando uma coleção de atributos. As propriedades mapeam cada chave para cada tupla associada. O conjunto de todos os atributos de um componente é chamado de propriedades do componente. Atributos são as peças fundamentais do funcionamento do SATIN, eles são utilizados como consulta para descobrir quais componentes são disponíveis localmente ou remotamente.

Um componente SATIN também deve implementar `ComponentFacet` (ou uma especialização desta); com isso é permitido conhecer o componente e os valores dos seus atributos. Isso também expõe o ciclo de vida básico, incluindo o construtor e o destrutor. Um componente tem dois estados, `ENABLED` (habilitado) e `DISABLED` (desabilitado). O estado é controlado através da `ComponentFacet`.

As referências para todos os componentes SATIN estão disponibilizadas no container, o qual é uma especialização do componente. O registro e a remoção de um ou mais componentes é manipulado por um ou mais `Registers`; um `Registrar` também é uma especialização de um componente.

O SATIN provê primitivas de migração para classes, objetos, dados e componentes; esses são manipulados por um `Deployer`, que novamente é uma especialização de

um componente SATIN; o Deployer é responsável por enviar e receber Logical Mobility Units (LMUs). Um LMU é um container que encapsula um número arbitrário de classes, objetos, dados e componentes.

Embora o SATIN tenha algumas estruturas básicas similares ao Cosmos, as suas ações de adaptação são adequações e ações internas dos componentes. O SATIN também não possui um mecanismo de seleção como o proposto neste trabalho.

4.1.4 O Framework QDD (Quality-Driven Delivery)

O framework QDD [41] é direcionado para o gerenciamento de qualidade em transmissão de informações em sistemas multimídia. O framework foca a questão da modelagem das informações da QoS e de transformações envolvendo diferentes dimensões da QoS.

Modelos de informação de QoS são usados para mapeamento de requisitos a serem usados para definição de restrições associadas por exemplo a troca de informações de QoS, para efeito de compatibilidade e decisões sobre a QoS. A abordagem é interessante no sentido de trazer uma análise da QoS sob a perspectiva de diferentes visões. O trabalho tenta analisar as diferentes dimensões de QoS e definir modelos de mapeamento entre essas dimensões. Estas idéias poderão ser interessantes no sentido de que, no futuro, o processo de seleção de componentes pode lidar com um mapeamento automático de requisitos de diferentes dimensões, facilitando a abordagem de decisão mediante situações de conflito.

No trabalho, os autores utilizaram o modelo QDD para avaliar serviços de transmissão de vídeo em ambientes controlados, onde eram alterados determinados parâmetros de QoS do sistema, de forma a observar os efeitos no sistema. O objetivo desses testes foi o de definir estratégias para adaptação definindo regras para adaptação, como por exemplo:

- Quando um atraso for aumentado na transmissão, resultando num decréscimo da taxa de transmissão de vídeo, a decisão de adaptação de QoS pode ser:
 - Alocar mais banda no caminho atual
 - Trocar o servidor de transmissão
 - Trocar o Codec
- Quando for percebida uma grande variação no jitter:

- Incrementar temporariamente os buffers no lado no transmissor, e nos elementos intermediários do caminho
- Aumentar o tamanho dos buffers de recepção do cliente
- Quando houver um aumento na taxa de transmissão de pacotes, a decisão de QoS pode ser:
 - Trocar o protocolo de UDP para TCP
 - Trocar o Codec

Esta abordagem está mais centrada na questão do modelo de transformação, enquanto a nossa proposta teve uma visão semelhante em termos de avaliação empírica, no entanto nos preocupamos em estabelecer critérios de seleção de componentes, ao invés de estratégias de adaptação.

4.1.5 Comparativo

Esta seção tem por objetivo estabelecer um comparativo entre os frameworks multimídia citados. A tabela 4.1 apresenta uma síntese desta comparação.

Para realizar a comparação foram analisados os fatores descritos adiante observando a relevância dos fatores de acordo com a discussão apresentada no Capítulo 2.

- Baseado em componentes - Se o *framework* é baseado em componentes
- Reflexivo - Se o *framework* tem suporte a reflexão
- Suporte a Adaptação - Se o *framework* tem suporte a adaptação
- Adaptação Dinâmica - Se o *framework* tem suporte a adaptação dinâmica
- Possui Mecanismo de Seleção - Se o *framework* tem um serviço/mecanismo de seleção
- Linguagem de programação - Linguagem onde o *framework* foi desenvolvido

Com relação ao fator Baseado em componentes, o único que não tem essa característica é o NMM; todos os outros tem essa capacidade. Assim, relacionado com o Suporte a Adaptação o único com um esforço de adaptação dinâmica é o Cosmos. Os únicos frameworks com suporte a reflexão foram o Cosmos, SATIN e QDD.

Tabela 4.1: Comparativos entre os frameworks multimídia

Característica	Cosmos	PREMO	NMM	SATIN	QDD
Baseado em Componentes	X	X		X	X
Reflexivo	X			X	X
Suporte a Adaptação	X			X	X
Adaptação Dinâmica	X				
Possui Mecanismo de Seleção	X			X	
Linguagem de Programação	Java	Java	C++	Java	Java

4.2 Seleção de Componentes

Existem algumas propostas para a seleção de componentes apresentadas na literatura. A seguir, o texto apresenta uma análise de algumas delas relacionadas com a definição de critérios e processos para seleção de componentes.

4.2.1 Método OTSO

Uma das primeiras iniciativas para assistir e facilitar a seleção de componentes, principalmente do tipo COTS é o Método OTSO (*Off-The-Shelf Option*) [42]. Esse método suporta a localização, avaliação e seleção de componentes reusáveis, assim como fornece técnicas específicas para definir o critério de avaliação, comparando custos e benefícios das alternativas. Esse método decompõe o critério de avaliação em quatro áreas principais:

- Requisitos dos Usuários;
- Características de qualidade do produto;
- Compatibilidade de domínio e arquitetura;
- Interesses estratégicos (custos, estabilidade do vendedor, etc).

Uma das fases mais importantes do método é a análise dos resultados, essa fase faz uma estimativa dos custos para adquirir cada candidato COTS e compara custos e benefícios de cada alternativa. Além disso, usa a técnica AHP (Analytic Hierarchy Process) [43] para consolidar os resultados da avaliação e suportar o processo de tomada de decisão.

Apesar do método OTSO enfatizar que um dos problemas chave do processo de seleção de COTS é a falta de atenção aos requisitos dos *stakeholders*, esse método não provê nenhuma sugestão ou solução para atacar esse problema. O principal foco do método é definir o critério de avaliação, porém ele não oferece orientação de como adquirir e modelar os requisitos. O método assume que os requisitos já existem, pois ele parte da especificação dos requisitos para em seguida utilizá-los durante o processo de seleção.

4.2.2 Método PORE

O Método PORE (*Procurement-Oriented Requirements Engineering*) [44] é baseado num processo iterativo de aquisição de requisitos e seleção de componentes, essa é uma abordagem recente e bastante aceita [45]. O modelo de ciclo de vida PORE consiste de seis processos genéricos, são eles:

- **Processo de gerenciamento de aquisição do produto** - onde os objetivos são planejar e controlar o processo de aquisição;
- **Processo de aquisição de requisitos** - esse processo adquire e valida os requisitos dos usuários, ele também determina a arquitetura que os produtos deverão seguir;
- **Processo de seleção de fornecedor** - o objetivo é estabelecer um critério para selecionar o fornecedor e fazer um *ranking* de acordo com o critério das melhores alternativas;
- **Seleção de pacote de software** - o objetivo é identificar os pacotes candidatos, estabelecer o critério de seleção usando os requisitos dos usuários e fazer um *ranking* para selecionar o produto que melhor se ajusta aos requisitos chave dos usuários;
- **Produção do contrato** - onde são negociados os contratos legais com o fornecedor, além disso, são negociados os direitos e garantias;
- **Aceitação do pacote** - o objetivo é validar o pacote em relação aos requisitos chave dos usuários.

Uma das principais fraquezas desse método é a falta de uma análise mais detalhada de como deve ser realizado o processo de requisitos. Em particular, ele não aborda como elicitar e modelar os requisitos não-funcionais durante o processo de seleção de componentes. O Método PORE traz uma abordagem nova, porém não descreve como o critério de seleção deve ser obtido e utilizado, como também se esquece de considerar aspectos

não técnicos durante o processo seletivo, tais como: custo e treinamento. Outras limitações do método são a sua complexidade de uso e tempo necessário para realizar todas as fases propostas.

4.2.3 CARE/SA

O CARE/SA (*COTS-Aware Requirements Engineering and Software Architecting*) [33] é um *framework* que suporta as operações de comparação iterativa, *ranking* e seleção de componentes do tipo prateleira (COTS). Os componentes são representados como um agregado de seus requisitos funcionais, não-funcionais e arquiteturais.

Para realizar a tarefa de escolha de componentes o *framework* primeiramente define os conjuntos de metas para o sistema em questão. Nesse passo, o auxílio de uma ferramenta como o *Framework* NFR [46] pode ser usada no estabelecimento e organização dos requisitos não-funcionais. As metas não podem ser muito abstratas, pois se assim forem, podem causar inconsistência na escolha do componente pela falta de compreensão por parte do *framework*.

Depois de estabelecidas, as metas são analisadas a fim de verificar se não apresentam inconsistências, conflitos, erros ou até mesmo se não são redundantes. Por fim, considerando que as metas já estejam definidas e que os componentes já se encontram armazenados no repositório juntamente com suas características, a escolha é determinada a partir de buscas nesse repositório. Nesse momento é feita uma comparação entre as metas armazenadas e as que foram inseridas na busca. Antes de retornar o resultado, dois grupos de componentes são gerados: um observando os requisitos funcionais e o outro observando os requisitos não-funcionais. Em seguida, esses conjuntos se fundem para analisar que componentes atendem ambos os requisitos, formando um único grupo pronto para ser retornado como resposta da busca. Uma visão geral de todo o seu processo de avaliação pode ser observado através da Figura 4.1.

O CARE/SA retorna um conjunto de componentes selecionados, os quais são capazes de atender os requisitos definidos na busca. Um dos pontos negativos desse *framework* está no fato de que ele deixa para a aplicação toda a responsabilidade de escolher qual componente usar dentro do universo de componentes retornados. Porém ele apresenta como ponto positivo a mudança de metas dinamicamente de forma a auxiliar na escolha de um componente que possui, por exemplo, um alto custo de uso e manutenção, definindo dinamicamente, por exemplo, critérios mais amenos.

4.2.4 Component Rank

O sistema Component Rank [47] possui em sua essência a responsabilidade de analisar os componentes de acordo com suas relações e realizar a propagação do peso de cada relação entre os componentes usados no sistema. O alvo principal dessa abordagem é fazer uma análise para fins de *ranking* de componentes Java, cujo foco da avaliação está voltado para os aspectos relacionados com código-fonte. Dessa forma, o Component Rank está mais preocupado com a cadeia de dependências de um componente em relação a outros (código-fonte).

O Component Rank define um componente principal (*Component Graph*), cuja responsabilidade consiste em organizar os componentes do sistema numa estrutura de grafo onde são representadas as relações e dependências entre os mesmos. Cada nó do grafo representa um componente e as arestas representam suas relações de uso.

Os nós do grafo, como também suas arestas, são pontuadas com pesos que variam de 0 a 1, sendo que a soma de todos os pesos dos nós deve ser igual a 1. Cada aresta está associada a um raio de distribuição; este elemento é utilizado no estabelecimento de um peso a ser atribuído ao elemento correspondente.

Para realizar a escolha de um componente, além de analisar os pesos dos nós e de suas respectivas arestas, o sistema usa também a idéia de similaridade, onde a avaliação pode analisar, por exemplo, o quão semelhante em nível de linhas de código um componente é considerado com relação aos outros. Essa semelhança deve estar acima de um limiar estabelecido pelo sistema.

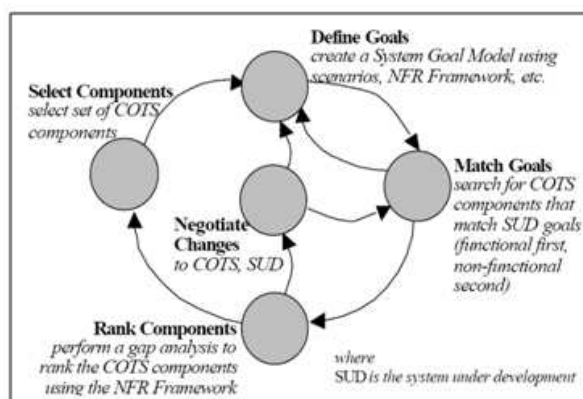


Figura 4.1: Visão Geral do processo do CARE/SA

Para a escolha de um componente, o Component Rank, como descrito na Figura 4.2, realiza as seguintes tarefas: o sistema recebe em sua entrada o conjunto de componentes (códigos-fonte Java), aplica o critério de similaridade, e, em seguida, considera os componentes que possuem uma taxa de similaridade acima da estabelecida pelo sistema, construindo assim um grupo de componentes similares. Paralelamente a essa tarefa, os componentes têm suas relações de uso e dependência analisadas. Em seqüência, o *Component Graph* gera o grafo que representa os componentes, calcula os pesos de seus nós e arestas, onde cada grupo é tratado como um único componente. Por fim, os grupos de componentes similares são separados, refazendo-se o cálculo dos pesos em seus nós e arestas.

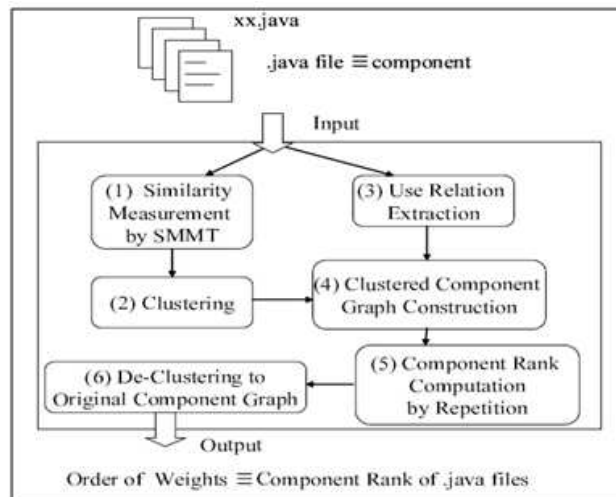


Figura 4.2: Arquitetura do Component Rank

O resultado retornado é um grafo representando os nós, suas relações de uso e dependências e os pesos respectivos. Mais uma vez, cabe à aplicação, de acordo com o grafo construído, fazer a escolha de qual componente usar na construção do sistema.

Capítulo 5

Considerações Finais

Este capítulo apresenta algumas considerações finais da dissertação discutindo os resultados obtidos em relação aos objetivos do trabalho e acenando com alguns possíveis trabalhos futuros.

O objetivo do presente trabalho consistiu em definir um processo para substituição dinâmica de componentes em plataformas baseadas no framework Cosmos e sua implementação no middleware AdapTV.

Considerando que uma troca dinâmica de componentes efetivamente só traz benefício para o sistema se o componente substituto contribuir para melhorar a Qualidade dos serviços providos pelo sistema, a análise precisava identificar alguns passos determinantes para que a eventual troca de componentes pudesse então efetivamente melhorar o comportamento do sistema diante dos requisitos formulados.

Alguns problemas então foram inicialmente identificados como cruciais, os quais são brevemente lembrados a seguir.

Para identificar a necessidade de substituição de um componente no sistema, precisamos conhecer claramente o estado de todas as variáveis relacionadas do sistema, de forma e decidir qual componente substituir, e, mais ainda, qual seria o componente substituto.

Para escolher com segurança um componente substituto seria portanto necessário conhecer ou ter uma previsão de como cada candidato a componente substituto se comportaria considerando o estado atual do sistema. Assim, conhecendo as qualidades de cada candidato, e, tendo critérios para avaliar qual deles se comporta melhor no estado atual do sistema, o processo de seleção consistiria em aplicar esses critérios usando os atributos de qualidade conhecidos de cada componente.

Entretanto, a literatura não apresentava critérios que pudessem ser utilizados diretamente para a escolha de componentes dentro do contexto de sistemas multimídia, e, mais especificamente, dentro do contexto do framework Cosmos.

Considerando que uma experiência no contexto do Cosmos para identificação de critérios poderia ser útil inclusive para outras abordagens, decidimos por propor um processo empírico para identificar potenciais critérios, e partir dessa experiência, definir um modelo de seleção de componentes.

Os resultados dessas experiências foram relatados em [34]. Utilizando estas experiências e o meta-modelo para definição de modelos de seleção de componente, foi definido o módulo AnaMoC e sua implementação [38], no contexto de projeto do framework Cosmos. Como o modelo de QoS do Cosmos não estava preparado para tratar a questão da especificação e substituição dinâmica de componentes, se fazia necessário também ajustar o conceito de faixas de valores QoS para o novo contexto; isto implicava na necessidade de ajustar a ADL original do Cosmos de forma a permitir uma especificação de sistema onde vários componentes semanticamente semelhantes pudessem desempenhar o mesmo papel em um sistema.

Durante o desenvolvimento deste trabalho tivemos algumas dificuldades relacionadas principalmente com: a atualização de plataforma Cosmos, onde foi demandado um grande esforço; utilização do gerenciador de código, ocasionando retrabalho e problemas de consistência, e, falta de documentação relacionada com alguns frameworks analisados.

Como principal contribuição consideramos a definição de um processo envolvendo todas as etapas que devem ser consideradas em abordagens de trocas de componentes no contexto do framework Cosmos.

Embora as atividades desenvolvidas tenham sido realizadas no contexto do Cosmos, sob a nossa visão, as principais idéias podem ser aplicadas em outras plataformas. Para isso, como trabalhos futuros, pode-se pensar em criar mecanismos de mapeamento automático de modelos de QoS, a exemplo de esforços na linha do framework QDD para uniformizar os conceitos de QoS, e, usando abordagens do tipo MDA para gerar código relacionado com decisão de que critérios utilizar, que pesos atribuir e gerar automaticamente processos de seleção. Um outro trabalho futuro pode ser trabalhar com a técnica de adaptação por mudança de canal onde os casos que o redimensionamento do buffer não se aplique. Por último, é necessário atualizar a ferramenta gCosmos para que ela, além de especificar as aplicações, especifique componentes e políticas para a avaliação dos componentes.

Para finalizar apresentamos as publicações que foram geradas durante o desenvolvimento do trabalho:

1. O trabalho [48] que foi publicado no XIII WebMedia, onde contribuimos com o gCosmos, que é uma ferramenta gráfica para a criação de aplicações Cosmos, e
2. O trabalho [34] que foi publicado no XIV WebMedia, onde contribuimos com o estabelecimentos de critérios de QoS.

Referências Bibliográficas

- [1] Amatriain X. *A Domain-Specific Metamodel for Multimedia Processing Systems*. IEEE TRANSACTIONS ON MULTIMEDIA 2007;9(6):2
- [2] Lopes AB. *Um framework para configuração e gerenciamento de recursos e componentes em sistemas multimídia distribuídos abertos*. Ph.D. thesis, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, 2006
- [3] da Silva CE. *Um modelo de interconexão de componentes para ambientes multimídia distribuídos*. Master's thesis, Departamento de Informática e Matemática Aplicada, Centro de Ciências Sociais e da Terra, Universidade Federal do Rio Grande do Norte, 2007
- [4] McKinley PK, Sadjadi SM, Kasten EP, Cheng BHC. *Composing Adaptive Software*. Computer 2004;37(7):56–64. ISSN 0018-9162. doi: <http://dx.doi.org/10.1109/MC.2004.48>
- [5] Mckinley PK, Sadjadi SM, Kasten EP, Cheng BHC. *A Taxonomy of Compositional Adaptation*. Tech. rep., Michigan State University, 2004
- [6] McIlroy MD. *“Mass Produced” Software Components*. In Naur P, Randell B, eds., *Software Engineering*. Scientific Affairs Division, NATO, Brussels, 1969; 138–155. Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968
- [7] Deremer F, Kron H. *Programming-in-the large versus programming-in-the-small*. In *Proceedings of the international conference on Reliable software*. ACM Press, New York, NY, USA, 1975; 114–121. doi:10.1145/800027.808431
- [8] Meyer B. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0136290493

- [9] Johnson RE, Foote B. *Designing Reusable Classes*. Journal of Object-Oriented Programming 1988;1(2):22–35
- [10] Brown AW, Wallnan KC. *Engineering of component-based systems*. In *ICECCS '96: Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '96)*. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-7614-0, 1996; 414
- [11] Bosch J, Szyperski C, Weck W. *Summary of the Second International Workshop on Component-Oriented Programming*. In Weck W, Bosch J, Szyperski C, eds., *Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP'97)*, no. 5 in TUCS General Publication. Turku Centre for Computing Science, 1997; 1–4
- [12] Chambers C. *Towards reusable, extensible components*. ACM Comput Surv 1996; 1:192. ISSN 0360-0300. doi:<http://doi.acm.org/10.1145/242224.242473>
- [13] Heineman GT, Councill WT. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-70485-4
- [14] de Lima Resende ARM, da Cunha AM, de Resende AMP. *Um modelo de processo para seleção de componentes de software*. UFLA, 1 ed., 2007
- [15] Crnkovic I, Larsson M. *A case study: demands on component-based development*. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. ACM, New York, NY, USA. ISBN 1-58113-206-9, 2000; 23–31. doi:<http://doi.acm.org/10.1145/337180.337185>
- [16] da Silva Júnior MC. *COSMOS - Um Modelo de Estruturação de Componentes para Sistemas Orientados a Objetos*. Master's thesis, Instituto de Computação - Universidade Estadual de Campinas, 2003
- [17] Emmerich W. *Distributed component technologies and their software engineering implications*. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. ACM, New York, NY, USA. ISBN 1-58113-472-X, 2002; 537–546. doi:<http://doi.acm.org/10.1145/581339.581405>
- [18] Zachariadis S, Mascolo C, Emmerich W. *The SATIN Component System: A Meta-model for Engineering Adaptable Mobile Systems*. IEEE Transactions on Software Engineering 2006;32(11):2

- [19] Szyperski CA. *Extensible Object Orientation*, 1992
- [20] Gorton I, Heinemann GT, Crnkovic I, Schmidt HW, Stafford JA, Szyperski C, Wallnau K. *Component-Based Software Engineering: 9th International Symposium, CBSE 2006, Västerås, Sweden, June 29 - July 1, 2006, Proceedings (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540356282
- [21] Maes P. *Concepts and experiments in computational reflection*. SIGPLAN Not 1987;22(12):147–155. ISSN 0362-1340. doi: <http://doi.acm.org/10.1145/38807.38821>
- [22] Costa FM, Blair GS. *Integrating Meta-Information Management and Reflection*. In *Middleware International Symposium on Distributed Objects and Applications 2000*;
- [23] Duran-Limon HA. *A Resource management framework for reflective multimedia Middleware*. Ph.D. thesis, University of Lancaster, 2002
- [24] Duke DJ, Herman I. *A standard for multimedia middleware*. In *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*. ACM, New York, NY, USA. ISBN 0-201-30990-4, 1998; 381–390. doi: <http://doi.acm.org/10.1145/290747.290806>
- [25] Sadjadi S. *A Survey of Adaptive Middleware*. Ph.D. thesis, Michigan State University, 2002
- [26] Lopes A, Elias G, Magalhaes MF. *Um Modelo de Metacomponentes para Suporte à Adaptação Dinâmica em um Middleware para Sistemas de Televisão Interativa*. In *XII Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia 2006*. SBC, Natal, RN, Brazil, 2006;
- [27] Lopes A, Borelli F, Elias G, Maurício Ferreira Magalhães. *A Component-based Configuration and Management Framework for Open, Distributed Multimedia Systems*. IEEE Computer Society 2004;II:465
- [28] Lohse M, Repplinger M, Slusallek P. *An Open Middleware Architecture for Network-Integrated Multimedia*. In *IDMS/PROMS 2002: Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems*. Springer-Verlag, London, UK. ISBN 3-540-00169-7, 2002; 327–338

- [29] Lohse M. *Network-Integrated Multimedia Middleware, Services, and Applications*. Ph.D. thesis, Department of Computer Science, Saarland University, Germany, 2005
- [30] Lohse M. *Network-Integrated Multimedia Middleware, Services, and Applications*. VDM Verlag, 2007
- [31] Medvidovic N, Taylor RN. *A Classification and Comparison Framework for Software Architecture Description Languages*. IEEE Trans Softw Eng 2000;26(1):70–93. ISSN 0098-5589. doi:<http://dx.doi.org/10.1109/32.825767>
- [32] Medvidovic N, Taylor RN. *A Classification and Comparison Framework for Software Architecture Description Languages*. IEEE Trans Softw Eng 2000;26(1):70–93. ISSN 0098-5589. doi:<http://dx.doi.org/10.1109/32.825767>
- [33] Chung L, Cooper K. *Matching, Ranking, and Selecting Components: A COTS-Aware Requirements Engineering and Software Architecture Approach*. Proc Intl Workshop on Models and Processes for the Evaluation of COTS Components 2004; 1:4. Edinburgh, Scotland
- [34] da Silva DC, Júnior IV, Lopes A, Pinto FP, Silva) A. *Estabelecimento de Critérios para Adaptação Dinâmica de QoS em Sistemas Multimídia Distribuídos com Base em Testes Experimentais Realizados no Framework Cosmos*. In SBC, ed., *Full Paper - XIV Brazilian Symposium on Multimedia and the Web, 2008, Vila Velha - ES. Full Paper - XIV Brazilian Symposium on Multimedia and the Web*. SBC, SBC, 2008;
- [35] Lopes A, Amaro F, Elias G, Lemos G, Magalhaes MF. *QoS Specification and Management in a Middleware for Distributed Multimedia Systems*. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2466-4-01, 2006; 959–964. doi: <http://dx.doi.org/10.1109/AINA.2006.280>
- [36] Cheng SW, Garlan D, Schmerl B. *Architecture-based self-adaptation in the presence of multiple objectives*. In *SEAMS '06: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. ACM, New York, NY, USA. ISBN 1-59593-403-0, 2006; 2–8. doi: <http://doi.acm.org/10.1145/1137677.1137679>
- [37] Júnior LAB, da Graça Campos Pimentel M. *Sincronização Multimídia: Aspectos Básicos*. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo São Carlos, São Paulo 2000;I:39

- [38] da Silva DC. *AnaMoC: Um módulo para seleção dinâmica de componentes baseado em critérios de QoS no framework Cosmos*. TCC 2008;111
- [39] Autili M, Benedetto PD, Inverardi P, Tamburri DA. *Towards Self-evolving Context-aware Services*. ECEASST 2008;11
- [40] Duke D, Herman I, Marshall M. *PREMO: a framework for multimedia middleware : specification, rationale, and Java binding*. Lecture Notes in Computer Sciences No. 1591. Springer Verlag, Heidelberg, 1999
- [41] Kerherve B, Nguyen KK, Gerbe O, Jaumard B. *A Framework for Quality-Driven Delivery in Distributed Multimedia Systems*. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2522-9, 2006; 195
- [42] Kontio J, fune Chen S, Limperos K, Tesoriero R, Caldiera G, Deutsch M. *A COTS Selection Method and Experiences of Its Use*. In *Proceedings of the 20 th Annual Software Engineering Workshop*. Maryland, USA, 1995;
- [43] Alves CF, Castro JFB. *CRE: A Systematic Method for COTS Components Selection*. In *Anais do XV Simpósio Brasileiro de Engenharia de Software*, 15. Rio de Janeiro, Brasil, 2001; 193–207
- [44] Ncube C. *A requirements engineering method for COTS-based systems development*. Ph.D. thesis, Thesis (Ph. D.)—City University, May 2000., 2000
- [45] Kunda D, Brooks L. *Identifying and classifying processes (traditional and soft factors) that support COTS component selection: a case study*. *European Journal of Information Systems* 2000;9(4):226–234(9)
- [46] Tran Q, Chung L. *NFR-Assistant: Tool Support for Achieving Quality*. In *ASSET '99: Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-0122-2, 1999; 284
- [47] Inoue K, Yokomori R, Fujiwara H, Yamamoto T, Matsushita M, Kusumoto S. *Component rank: relative significance rank for software component search*. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-1877-X, 2003; 14–24

- [48] Silva M, Júnior IFV, Brandão R, Silva C, Lopes AB. *Ferramenta visual para especificação de aplicações em um Middleware para Sistemas de Televisão Interativa*. In SBC, ed., *Workshop Tools & Applications - XIII Brazilian Symposium on Multimedia and the Web, 2007, Gramado. Workshop Tools & Applications - XIII Brazilian Symposium on Multimedia and the Web, 2*. SBC, SBC, 2007; 160–162.

Appendices

Apêndice A

XSD

Código-fonte A.1: XML Schema definindo os elementos propostos e incorporadas a ADL do Cosmos.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/schema_metadados"
xmlns:tns="http://www.example.org/schema_metadados"
5 elementFormDefault="qualified">
  <!-- definição de atributos -->
  <attribute name="name" type="string"></attribute>
  <attribute name="version" type="int"></attribute>
  <attribute name="type" type="string"></attribute>
10 <attribute name="unit" type="float"></attribute>
  <attribute name="value" type="string"></attribute>
  <attribute name="weight" type="float"></attribute>
  <attribute name="max" type="float"></attribute>
  <attribute name="min" type="float"></attribute>
15 <!-- definição de elementos complexos -->

  <element name="POLICY_QoS">
    <complexType>
      <sequence>
20 <element ref="tns:CLASS_OF_CRITERION"></element>
      </sequence>
      <attribute ref="tns:name" use="required"></attribute>
      <attribute ref="tns:version" use="required"></attribute>
    </complexType>
25 </element>

  <element name="CLASS_OF_CRITERION">
    <complexType>
```



```
<sequence>
30 <element ref="tns:CRITERIA"></element>
</sequence>
<attribute ref="tns:name" use="required"></attribute>
<attribute ref="tns:weight" use="required"></attribute>
</complexType>
35 </element>

<element name="CRITERIA">
<complexType>
<sequence>
40 <element ref="tns:GREAT"></element>
<element ref="tns:GOOD"></element>
<element ref="tns:REGULAR"></element>
</sequence>
<attribute ref="tns:name" use="required"></attribute>
45 <attribute ref="tns:type" use="required"></attribute>
<attribute ref="tns:unit" use="optional"></attribute>
</complexType>
</element>

50 <element name="GREAT">
<complexType>
<!-- Se type = atomic-->
<attribute ref="tns:value" use="required"></attribute>
<!-- Caso contrário -->
55 <attribute ref="tns:max" use="required"></attribute>
<attribute ref="tns:min" use="required"></attribute>
</complexType>
</element>

60 <element name="GOOD">
<complexType>
<!-- Se type = atomic-->
<attribute ref="tns:value" use="required"></attribute>
<!-- Caso contrário -->
65 <attribute ref="tns:max" use="required"></attribute>
<attribute ref="tns:min" use="required"></attribute>
</complexType>
</element>

70 <element name="REGULAR">
<complexType>
<!-- Se type = atomic-->
<attribute ref="tns:value" use="required"></attribute>
```

```
    <!-- Caso contrário -->
75  <attribute ref="tns:max" use="required"></attribute>
    <attribute ref="tns:min" use="required"></attribute>
  </complexType>
</element>
</schema>
```
