# Provendo aplicações com requisitos não-funcionais dinâmicos através de contratos

**Leonardo Cardoso[1], Alexandre Sztajnberg[2], Orlando Loques[1]**

[1]Instituto de Computação - Universidade Federal Fluminense
[2]DICC/IME e PEL/FEN - Universidade do Estado do Rio de Janeiro

{lcardoso, loques}@ic.uff.br, alexszt@ime.uerj.br

***Abstract.*** *This paper presents an approach to facilitate the specification, deployment and management of component-based applications having non-functional requirements. The approach is centered on architectural descriptions and associated high-level contracts, which are used to guide configuration adaptations in the supporting infrastructure required to enforce and monitor non-functional requirements. To illustrate details of the approach we use a videoconference application that has dynamic non-functional requirements. A prototype of the supporting infrastructure developed in Java was used to implement the videoconference application, allowing a real-case performance evaluation.*

***Resumo.*** *Este artigo apresenta uma abordagem que visa facilitar a especificação, implantação e gerência de aplicações baseadas em componentes com requisitos não-funcionais. A abordagem é centrada em descrições arquiteturais e contratos de alto nível. Tais contratos são utilizados para guiar as adaptações necessárias na infra-estrutura de suporte requerida para impor e monitorar os requisitos não-funcionais. Para ilustrar detalhes da proposta, foi utilizada uma aplicação de videoconferência, com requisitos não-funcionais dinâmicos. Um protótipo da infra-estrutura de suporte, desenvolvido em Java, foi utilizado na implementação da aplicação de videoconferência, propiciando avaliar questões de desempenho em um caso real.*

## 1. Introdução

O funcionamento e a utilidade de muitos sistemas distribuídos dependem de requisitos não-funcionais[1] que se adicionam aos requisitos funcionais. Tal idéia permeia várias áreas de interesse atual, dentre elas a computação pervasiva e a computação autônoma (*autonomic computing*). Como uma evolução natural da tecnologia de desenvolvimento destes sistemas, torna-se atrativa a composição de serviços implementados a partir de componentes. Atualmente, a tecnologia básica de componentes apresenta-se madura e a preocupação está focada em caracterizar os diversos aspectos não-funcionais, que

[1] Requisitos não-funcionais abrangem requisitos de suporte à operação (e.g., processador e memória), requisitos de QoS (*quality of service*) - tradicionalmente ligados a protocolos e redes de computadores, e quaisquer recursos (e.g., *streamers* e *codecs*) ou qualidades (e.g., tolerância a falhas e segurança) não necessariamente ligadas às funcionalidades essenciais da aplicação. Neste texto usamos uniformemente **requisitos não-funcionais** para expressar essa classe de preocupação.

podem ser associados a componentes e serviços. Esta caracterização fornece os fundamentos necessários para o estabelecimento de responsabilidades individuais e mútuas destes componentes, e permite configurar a infra-estrutura de suporte, visando garantir as propriedades não-funcionais para a aplicação, como um todo.

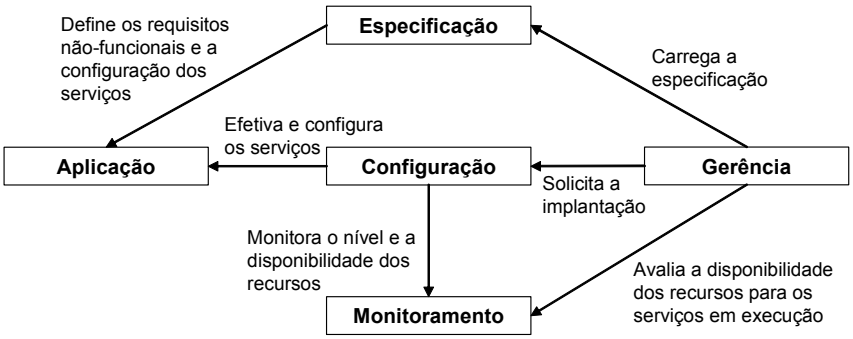



**Figura 1. Elementos das aplicações distribuídas com requisitos não-funcionais**

De uma forma geral, a concepção de sistemas com requisitos não-funcionais deve atender aos seguintes pontos (Figura 1):

- Uma forma de ***especificação*** de serviços deve possibilitar a formalização das exigências básicas para a instalação e operação dos mesmos. As especificações definem contratos onde perfis de qualidade a serem estabelecidos ou mantidos são indicados, permitindo que os aspectos não-funcionais desejados sejam monitorados durante a etapa de funcionamento do sistema.
- Uma vez especificado, um contrato de qualidade tem de ser aplicado e gerenciado. A maioria dos mecanismos de gerenciamento existentes é dirigida a requisitos de qualidade específicos e localizados. Em geral, esses mecanismos buscam impor as propriedades contratadas por meio de procedimentos codificados internamente, com forte acoplamento, como comumente observado em serviços e protocolos de comunicação. Contudo, torna-se atrativo a utilização de um mecanismo externo de ***especificação*** e ***gerenciamento*** de contratos, com baixa interferência na programação dos elementos funcionais, que possa ser reutilizado em diversas aplicações. Neste contexto, uma das questões relevantes é a capacidade de realizar ***adaptações dinâmicas*** na arquitetura visando dar continuidade à provisão do serviço.
- O ***monitoramento*** deve permitir identificar situações em que valores medidos de uma ou mais propriedades violem o contrato, degradando ou mesmo impedindo o oferecimento do serviço. Os valores monitorados podem ser usados para guiar ou negociar procedimentos adaptativos, visando dar continuidade à provisão do serviço e, opcionalmente, podem ser registrados em um repositório para uso futuro (e.g., redimensionamento do contrato ou estabelecimento de prêmios ou multas). A atividade de monitoramento, embora dirigida pelas especificações dos contratos, pode ser desempenhada por serviços independentes [Wol99 e Mat04].

Neste artigo apresentamos uma abordagem (CR-RIO) [Loq04] que visa facilitar a especificação, implantação e gerência de aplicações contendo características como as mencionadas anteriormente. A apresentação é centrada em uma aplicação de videoconferência, contribuição deste artigo, com requisitos não-funcionais dinâmicos,

que é utilizada para ilustrar detalhes da proposta. A abordagem inclui uma infra-estrutura de suporte, desenvolvida em Java, que foi utilizada para uma implementação da aplicação de videoconferência, propiciando avaliar questões de desempenho relevantes em casos reais.

Além da introdução e a conclusão, o texto apresenta os elementos de nossa abordagem (Seção 2), o exemplo de uma aplicação de videoconferência em que discutimos os vários aspectos de nossa abordagem (Seção 3), detalhes de implementação e uso de contratos (Seção 4). Apresentamos, também, trabalhos relacionados na Seção 5.

## 2. A abordagem CR-RIO

A abordagem CR-RIO é centrada em um modelo arquitetural e utiliza uma linguagem de descrição de contratos (CBabel) para expressar os requisitos não-funcionais das aplicações. Com base nestes elementos, uma infra-estrutura de suporte (*middleware*) foi desenvolvida para: (i) interpretar a especificação dos contratos e armazená-la como meta-informação associada à aplicação, (ii) prover mecanismos de reflexão e adaptação dinâmica, que permitem adaptar a configuração da aplicação (incluindo seus elementos de suporte), visando suprir as exigências de contratos, e (iii) prover um conjunto de mecanismos para impor, monitorar e manter os contratos associados à aplicação.

O diferencial da abordagem está na associação direta entre a arquitetura da aplicação, como um todo, ou das várias partes que a compõem, e os seus requisitos não-funcionais descritos por contratos. Por um lado, isso permite que o projetista consiga descrever, com a granulosidade requerida, em que parte da configuração de componentes da arquitetura determinado contrato deve ser imposto. Por outro, sempre que uma adaptação tenha que ser efetuada, para manter a qualidade descrita no contrato, ela é realizada através de operações de configuração da arquitetura da aplicação.

Na próxima seção apresentamos os elementos associados aos contratos e ao sistema de suporte concebido para a implantação dos mesmos.

### 2.1 Contratos

Em nossa proposta, um serviço funcional de uma aplicação é considerado uma atividade especializada, definida através da especificação de componentes arquiteturais, geralmente não permitindo negociação [Beu99]. Serviços não-funcionais são definidos através de restrições às atividades não-especializadas da aplicação e podem admitir alguma negociação envolvendo os recursos utilizados. Um contrato descreve em tempo de projeto os aspectos não-funcionais da aplicação, especificando o uso a ser feito de recursos compartilhados durante a operação, e variações aceitáveis na disponibilidade destes recursos. Uma aplicação pode depender de um ou mais contratos; a semântica definida por cada contrato é imposta em tempo de operação por um *middleware* composto por um conjunto padronizado de componentes (ver Seção 2.2). Um contrato possui os seguintes elementos:

a) **Categorias,** que descrevem propriedades de recursos ou aspectos não-funcionais específicos, separadamente dos componentes. Por exemplo, características de processamento, memória ou de comunicação podem ter associadas uma Categoria. Aspectos menos tangíveis como faixa de preço (“caro”, “barato”), tolerância a falhas,

ou qualidade (“boa”, “média”, “ruim”) também podem ser descritos. Para cada Categoria devem ser providos componentes ou serviços de suporte correspondentes, que irão alocar e monitorar os respectivos recursos, utilizando para isso a infra-estrutura disponível.

b) **Perfis**, que quantificam ou valoram as propriedades de uma Categoria. A quantificação restringe cada propriedade de acordo com a sua descrição, funcionando como uma instância de valores aceitáveis para determinada Categoria. Perfis podem ser definidos, com a granulosidade desejada, para restringir o contexto de operação de componentes individuais ou partes de arquitetura.

c) Um conjunto de **serviços**, onde cada serviço contém um conjunto de restrições que devem ser aplicadas à aplicação, no nível da arquitetura. Isso é feito associando-se um ou mais perfis aos componentes da aplicação, ou à forma de interação dos componentes. Assim, o nível de qualidade desejado/tolerado por um serviço é diferenciado de outro pelo conjunto das propriedades declaradas nos perfis. Um serviço pode ser implantado se todos os perfis associados ao mesmo forem válidos. O conjunto de serviços define os possíveis estados de operação para a aplicação.

d) Uma cláusula de **negociação** (*negotiation*), que descreve uma política, definida por uma máquina de estados, que estabelece uma ordem arbitrária para a implantação dos serviços. De acordo com o descrito na cláusula, quando um serviço de maior preferência não puder mais ser mantido, a infra-estrutura de suporte tentará implantar um serviço de menor preferência. O retorno para um serviço de maior preferência também pode ser descrito, permitindo que um serviço de melhor qualidade seja implantado, se os recursos necessários ao mesmo tornarem-se disponíveis.

### 2.1.1 Descrição de Categorias

A título de exemplo, descrevemos a seguir as categorias a serem utilizadas na aplicação de videoconferência apresentada na Seção 3. Cada categoria contém o nome das propriedades de interesse, e as características destas propriedades. A categoria *LocalResc* representa os recursos do sistema local a serem alocados/monitorados para dar suporte à qualidade esperada pelo usuário. A categoria *Transport* define características de transporte e comunicação, tais como banda, atraso e *jitter*.

```
category LocalResc {
  utilization: decreasing numeric %;
  clockFrequency: increasing
                    numeric MHz;
  memReq: increasing numeric Mbytes;
}
```

```
category Transport {
   bandwidth: increasing
                     numeric Mbps;
   delay: decreasing  numeric ms;
   jitter: decreasing numeric;
}
```

```
category VideoMedia {
  codec: enum (H261, H263, MJPEG);
  quality: enum (LOW, MEDIUM, HIGH);
  size: enum (CIF, QCIF);
  frameRate: increasing numeric fps;
}
```

```
category AudioMedia {
  codec: enum (G711, G723, GSM, DVI);
  sampleLenght: enum (8, 16);
  //tamanho da amostra (bits)
  sampleRate: increasing numeric Hz;
  channels: enum (MONO, STEREO);
}
```

As categorias relacionadas a recursos de áudio e vídeo contêm propriedades que podem ser selecionadas na maioria das implementações. Ao se instanciar um terminal de videoconferência, por exemplo, perfis de áudio e vídeo selecionam os valores específicos das propriedades de áudio (por exemplo, `codec:G711; sampleRate:8000`)

e vídeo (digamos, `codec:H261; frameRate:20`). Estes valores serão utilizados pelos RAs (ver próxima subseção), para alocar os recursos necessários, e pelo *Configurator*, para criar as instâncias com os parâmetros adequados.

## 2.2 Infra-estrutura de Suporte

As arquiteturas descritas em CBabel são mapeadas em um modelo de objetos [Cor05]. Este modelo é refletido em um repositório de meta-nível, que mantém as informações de configuração da arquitetura, as quais podem ser atualizadas e consultadas durante a vida da aplicação. Este repositório inclui também as representações dos contratos e mantém informações relacionadas a recursos (dispositivos, serviços, aplicações, etc) de interesse das aplicações. Com base nos contratos, que utilizam as informações contidas no repositório, uma infra-estrutura de suporte (*middleware*) é usada para gerenciar as configurações arquiteturais. Esse *middleware* é composto por um conjunto padronizado de componentes (Figura 2), os quais são descritos a seguir:

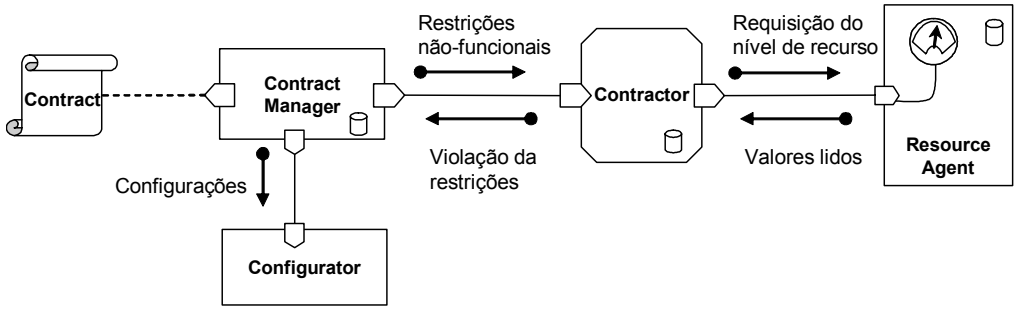



**Figura 2. Componentes da infra-estrutura (*middleware*) de suporte**

***Contract Manager*** (CM). Responsável pela implantação e gerência de contratos. O CM interpreta os contratos já mapeados no repositório de meta-nível e extrai dos mesmos as informações dos serviços, respectivos perfis e a máquina de estados de negociação de serviços. No processo de implantação de um serviço, o CM solicita aos *Contractors* a verificação das restrições exigidas pelos perfis do serviço selecionado. Se o CM for notificado por algum *Contractor* de que um perfil não pode ser (mais) atendido, o serviço atual é invalidado e um novo deve ser selecionado. O CM pode também iniciar uma nova negociação, quando os recursos para a implantação de um serviço de maior preferência tornam-se disponíveis.

***Contractor***. Gerencia o processo de monitoração das propriedades de elementos básicos (mecanismos, recursos ou serviços) especificadas nos perfis. Esse gerenciamento consiste do envio de requisições solicitando os valores dessas propriedades aos agentes de recursos (RA) e da avaliação das restrições dos serviços em face dos valores monitorados. Violações e validações de perfis são notificadas ao CM.

***Resource Agent*** (RA). Categorias são associadas a RAs que encapsulam o acesso aos elementos básicos de suporte (dispositivos, serviços, aplicações, etc), provendo interfaces para gerência e monitoração de valores de propriedades requeridas. Os valores monitorados são passados para o *Contractor* quando mudanças são detectadas.

***Configurator***. Elemento responsável por mapear as descrições arquiteturais (em CBabel) em ações que efetivam as configurações requeridas. Ele inicia a aplicação (se ela não estiver em execução) e recebe comandos do CM, relativos ao gerenciamento de

serviços. O *Configurator* provê duas APIs: configuração e reflexão arquitetural, através das quais as facilidades de configuração são acessadas. A API de configuração permite instanciar, ligar, parar e substituir componentes durante a operação da aplicação. Estas operações são refletidas num repositório persistente de meta-nível, que mantém o contexto da aplicação e pode ser consultado através da API de reflexão arquitetural.

## 3. Aplicação de Videoconferência

Nesta seção usamos uma aplicação de videoconferência para expor detalhes da abordagem proposta. Inicialmente, identificamos os requisitos básicos desta aplicação e apresentamos um contrato que descreve estes requisitos. Em seguida, discutimos a mesma aplicação com requisitos mais dinâmicos. A aplicação considerada contém os elementos usuais: (i) um serviço de diretório/sessão, que faz o registro e o controle de sessões (salas) de videoconferência, tal como o *Gatekeeper* do padrão H.323 [Tog99]; (ii) um elemento de redistribuição de fluxos de áudio e vídeo, equivalente ao MCU do padrão H.323 (sem a função de transcodificação), ou aos refletores de sistemas como o VRVS [Ada03]; e (iii) terminais de usuário que fazem a captura, exibição e transmissão de mídias de áudio e vídeo.

### 3.1 Cenário de Base

O cenário de base para a aplicação de videoconferência é constituído por usuários distribuídos em uma rede como a Internet, sendo uma rede *overlay* usada para prover comunicação multiponto, através de canais ponto-a-ponto. A arquitetura da rede *overlay* é formada por refletores, interligados por conectores de comunicação, que fazem o encaminhamento de fluxos de áudio e vídeo.

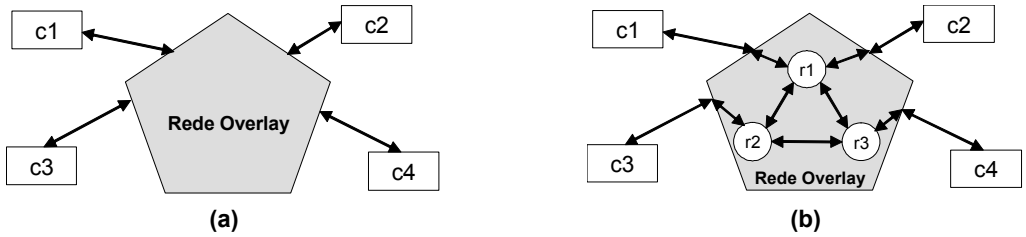



**Figura 3. (a) Arquitetura da aplicação; (b) Rede *overlay* detalhada**

A Figura 3(a) ilustra a arquitetura da aplicação, com usuários (terminais) que desejam participar de uma sessão de videoconferência utilizando a rede *overlay*. A rede *overlay* é "enxergada" como uma entidade arquitetural única, permitindo diferentes formas de configuração e adaptação, tais como: (i) para um conjunto estático de clientes participantes de uma sessão, selecionar um conjunto de refletores visando minimizar o retardo da comunicação entre os participantes; (ii) para um conjunto dinâmico de clientes, na admissão de um novo cliente, o suporte pode selecionar, do conjunto de refletores disponíveis, aquele que, baseado em alguma política, otimize parâmetros de desempenho, e.g., atraso e vazão de rede. A Figura 3(b) mostra mais detalhes da arquitetura da rede *overlay*, constituída de três refletores *r1*, *r2* e *r3* instanciados em diferentes domínios administrativos. Os clientes *c1* e *c2* estão ligados à rede *overlay* pelo refletor *r1* e os clientes *c3* e *c4* através dos refletores *r2* e *r3*, respectivamente.

Além dos aspectos funcionais da aplicação (componentes, topologia de interconexão, divisão de funcionalidade, etc.) estamos interessados em aspectos não-funcionais, relacionados a requisitos operacionais, tais como as qualidades dos fluxos de vídeo e áudio requeridas pelo usuário, e a capacidade da rede *overlay* encaminhar fluxos de várias sessões. Neste contexto, tais requisitos devem ser especificados, monitorados, e também impostos pelo ambiente de suporte, conforme discutido anteriormente. Em nossa abordagem isso é obtido com o estabelecimento de contratos associados a diferentes partes da arquitetura, especificamente: a rede *overlay*, o canal de comunicação entre o terminal e a rede *overlay*, e os recursos locais onde os terminais são executados.

### 3.2 Contrato para a Rede *Overlay*

Ao se planejar a rede *overlay* um contrato deve ser definido. Este contrato especifica os recursos mínimos necessários para a execução dos refletores, (CPU e memória) e os recursos mínimos para os canais de comunicação interligando os mesmos. No caso de uma rede *overlay* implantada com o objetivo de prover um serviço robusto, os refletores seriam instalados em máquinas dedicadas, com recursos suficientes. Neste caso, a especificação dos requisitos relacionados a recursos locais funcionaria como um controle de admissão, para assegurar que os recursos necessários existem. No caso mais geral, em que nas máquinas existem outros sistemas concorrendo com o refletor, os estados destes recursos também são monitorados para verificar se estão dentro dos limites especificados. Este contrato é independente das sessões de videoconferência, bem como dos terminais individuais ligados à rede *overlay* sob seu controle.

```
01 contract {
02   service {
03    // instâncias dos refletores em 3 nós distribuídos
04    instantiate reflector as refUFF at caueira.uff.br with refLProf;
05    instantiate reflector as refUERJ at blackcat.uerj.br with refLProf;
06    instantiate reflector as refLAMPADA at lampada.uerj.br with refLProf;
07    // ligação de pares de refletores
08    link refUFF  to refUERJ    by comSock with refCProf;
09    link refUERJ to refLAMPADA by comSock with refCProf;
10   } viaInternet;
11   negotiation { not viaInternet -> out-of-service; }
12 } cViaInternet;
```

**Figura 4. Contrato para a implantação da rede *overlay***

A Figura 4 apresenta um contrato para uma rede *overlay*. O serviço *viaInternet* (linha 2-10) apresenta os componentes da arquitetura e os requisitos não-funcionais associados: (i) instanciação dos refletores em máquinas específicas, obedecendo aos requisitos locais de cada refletor (no caso, o perfil *refLProf* – linhas 4-6); (ii) a ligação desses refletores, ponto-a-ponto, também de acordo com os requisitos de transporte necessários para prover os serviços (perfil *refCProf*, linhas 8-9), estabelecendo assim uma topologia de encaminhamento de dados através dos enlaces apropriados. A cláusula de negociação neste caso é bem simples (linha 11). Se o serviço *viaInternet* não puder ser implantado ou mantido, o mesmo torna-se indisponível. Num cenário mais robusto, poderíamos especificar no contrato configurações parciais que permitam manter o fornecimento do serviço, mesmo no caso de falhas de refletores (Seção 3.3).

A demanda por recursos locais para cada refletor é definida em termos de características de processamento e memória, especificadas no perfil *refLProf*, Figura 5. Neste perfil estão definidos valores para algumas características da categoria *Processing*. Assim, por exemplo, ao se instanciar o refletor *refUFF* (linha 4, Figura 4) estas características são previamente verificadas. As características desejadas para os canais de comunicação, que interligam cada par de refletores, são definidas no perfil *refCProf*. Os requisitos de banda são configurados proporcionalmente à quantidade máxima de usuários e sessões simultâneas e o tipo de codificação de áudio e vídeo usados. No exemplo, definimos, a partir de uma estimativa, as características deste perfil somando a banda necessária para transportar os dados de áudio e vídeo, e multiplicando o resultado pelo número médio de usuários previsto.

```
profile {
   LocalResc.utilization: 50;
   LocalResc.clockFrequency: 2800;
   LocalResc.memReq: 512;
} refLProf;
```

```
profile{
   Transport.delay: 50; //ms
   Transport.bandwith: 16000;// Kbps
} refCProf;
```

**Figura 5. Perfil de processamento e comunicação para os refletores**

Observa-se que nem sempre é possível fazer reserva de recursos. Na maioria dos sistemas é possível fazer apenas a alocação inicial dos recursos que podem então ser compartilhados, ou mesmo re-alocados, para aplicações de maior prioridade. Assim sendo, durante a operação, as características definidas podem ser monitoradas e, em caso de violação, mecanismos de adaptação dinâmica podem ser acionados, e.g., para introduzir um novo refletor na rede visando redistribuir a carga.

### 3.3 Tolerância a Falhas e Auto-reparo da Rede *Overlay*

A integração de contratos e conceitos de arquiteturas de software permite tratar os diversos aspectos da aplicação de videoconferência autonomamente. Ao associar um ou mais contratos à arquitetura definem-se políticas de operação e adaptação dinâmica que serão aplicadas automaticamente pela infra-estrutura de suporte. Assim sendo, podemos introduzir na arquitetura da rede *overlay* o atendimento a aspectos mais dinâmicos. Para exemplificar exploramos alguns casos:

```
01 contract {
02   context @teleconf;
03   service {... @teleconf = _viaInternet} viaInternet;
11   service {... @teleconf = _viaGIGA} viaGIGA;
18   negotiation {
19     not viaInternet -> viaGIGA;
20     not viaGIGA -> out-of-service; }
21 } cAlt1Ovl;
```

**Figura 6. Contrato com duas rede *overlay* prevendo tolerância a falhas**

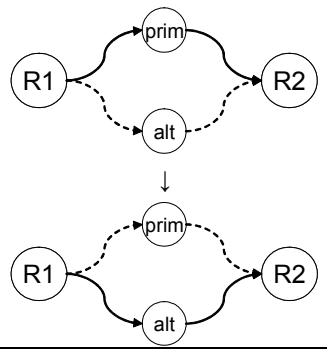
**Rede *overlay* alternativa**. Em nossos experimentos trabalhamos com dois conjuntos de refletores: um deles utiliza enlaces da Internet "commodity" e o outro, canais de Giga bps, providos pelo projeto GIGA [Loq04+], interligando a UFF, a UERJ e o Projeto LAMPADA. No contrato da Figura 6, além do serviço *viaInternet* (descrito na Figura 4), introduzimos o serviço *viaGIGA*, que também é associado a perfis, indicando seus requisitos não-funcionais. A idéia é que "provedores de *overlay*" ofereçam serviços

diferenciados e o usuário, através do contrato, especifique a política de uso destes serviços. Por exemplo, o serviço *viaInternet* seria utilizado preferencialmente - dado que os requisitos não-funcionais estejam sendo atendidos. Caso contrário, o serviço *viaGIGA„* com melhores recursos, porém mais caro, é negociado e passa a ser utilizado. Se nenhum dos serviços puder ser implantado, a sessão de videoconferência não poderá ser estabelecida. Esta política é definida na cláusula de negociação do contrato (linhas 18-21).

A rede *overlay* escolhida define o contexto sobre o qual as decisões de gerenciamento são realizadas. Para permitir a composição com outros contratos da aplicação de videoconferência (por exemplo, os contratos para terminais, ver Seção 3.4), a variável *@teleconf* (linha 2) é atualizada com a referência da rede selecionada (linhas 3 e 11), estabelecendo um caminho para acesso às suas informações, e.g., identidade de seus refletores.

**Enlace alternativo**. No caso de saturação ou falha de um enlace ligando dois refletores, a ligação poderia ser re-configurada utilizando um enlace alternativo (mantido em *cold* ou *hot standby*), Figura 7(b). Este aspecto pode ser contemplado em um contrato específico, que descreve as possíveis configurações de enlace entre dois refletores, como mostra a Figura 7(a), para a rede *overlay* provida pelo serviço *viaInternet.*

```
11  service {
12    link refUFF  to refUERJ  by
           comSock1 with refCProf;
13    } primeLink;
14  service {
15    link refUFF  to refUERJ  by
           comSock2 with refCProf;
16    } altLink;
17  negotiation {
18    not primeLink -> altLink;
19    altLink -> primeLink;
20    not altLink -> out-of-service; }
21 } cAltOvl;
```




**(a)** **(b)**

**Figura 7. Contrato para a rede *overlay* prevendo auto-reparo**

O contrato da Figura 7(a) descreve dois serviços de enlace. Através de conectores de comunicação diferentes (*comSock1* e *comSock2*) especifica-se que rotas independentes são utilizadas em cada serviço. A cláusula de negociação indica que o serviço preferencial é *primeLink.* Se este serviço não estiver disponível, a arquitetura é reconfigurada para usar o serviço *altLink* (linha 18). Se o serviço *primeLink* voltar a ser disponível, ele volta a ser estabelecido (linha 19). Se nenhum serviço puder ser implantado, esta parte passa a ser inoperante (linha 20), o que vai ser também refletido na rede *overlay*, como um todo.

Uma vez implantado e disponível o serviço de *overlay*, os terminais de usuário podem se conectar a esta rede para participar de uma sessão. O acesso de terminais é tratado em uma outra visão da arquitetura.

### 3.4 Contratos de Terminal

Nesta seção apresentamos alguns cenários onde contratos são empregados para definir os requisitos de terminais. Cada terminal-usuário conecta-se independentemente a um

refletor da rede *overlay*. Na descrição do contrato de usuário vários perfis devem ser definidos de forma a cobrir os requisitos envolvidos: (i) características dos recursos locais; (ii) qualidade das mídias, que dependem dos *codecs* de áudio e vídeo utilizados; (iii) características de comunicação, a serem observadas na ligação entre o módulo *Terminal* e o refletor associado; (iv) gerenciamento da sessão de videoconferência. Por brevidade, omitimos os detalhes deste último perfil, considerando que o serviço de diretório realiza um controle de admissão que limita o número de clientes por sessão.

### 3.4.1 Contrato Estático de Terminal

A instanciação de um terminal é regida pelo contrato descrito na Figura 8. Neste primeiro exemplo, o serviço *sSimpleTerm* contém os requisitos de recursos locais e de mídia, vinculados à instância do módulo *Terminal*, descritos nos perfis *termLProf*, *termVProf* e *termAProf* (linha 3); as restrições para a comunicação estão descritas no perfil *termCProf*. Estes perfis são valorações das propriedades das categorias *Processing*, *Transport, VideoMedia* e *AudioMedia*, definidas previamente. Observa-se também que está definido que se o serviço *sSimpleTerm* não (mais) puder ser estabelecido, o terminal sairá de operação (linha 6).

```
01 contract {
02   service {
03     instantiate Terminal with termLProf, termVProf, termAProf;
04     link Terminal to refUFF by commCon with termCProf;
05   } sSimpleTerm;
06   negotiation { not sSimpleTerm -> out-of-service; }
07 } cSimpleTerm;
```

**Figura 8. Contrato para um Terminal**

A especificação de cada perfil é mostrada na Figura 9. Por exemplo, no perfil *termVProf* (linhas 11-16) está definido que o *codec* de vídeo a ser utilizado é o H261, a resolução desejada é QCIF, e o *frame rate* mínimo tolerado é de 14 fps. Assim, a implantação do serviço *sSimpleTerm*, que requer a instanciação do *Terminal* (linha 3, Figura 8) só será realizada se estas características puderem ser impostas.

```
01  profile {
02    LocalResc.cpuSlice: 30;
03    LocalResc.utilization: 40;
04    LocalResc.clockFrequency: 400;
05    LocalResc.memReq: 128;
06  } termLProf;

07  profile{
08    Transport.delay: 80; //ms
09    Transport.bandwith: 128; //kbps
10  } termCProf;
```

```
11  profile {
12    VideoMedia.codec: H261;
13    VideoMedia.resolution: QCIF;
14    VideoMedia.frameRate: >= 14;
15    VideoMedia.quality: MEDIUM;
16  } termVProf;

17  profile {
18    AudioMedia.codec: DVI;
19    AudioMedia.sampleLenght: 8;
20    AudioMedia.sampleRate: 800;
21    AudioMedia.channels: MONO;
22  } termAProf;
```

**Figura 9. Perfis para um Terminal**

### 3.4.2 Contrato Dinâmico de Terminal

Observe-se que no contrato anterior, para conectar o terminal à rede *overlay*, um dos três refletores disponíveis foi estaticamente selecionado (Figura 8, linha 4). Contudo, o usuário pode desempenhar um papel pró-ativo, quanto ao acesso ao serviço de videoconferência, como o descrito a seguir:

(a) Inicialmente, a seleção do refletor de acesso pode ser feita através de políticas definidas nos perfis. Várias estratégias podem ser utilizadas levando em consideração as propriedades de interesse, descritas nos perfis do serviço, tais como: atraso de rede; capacidade de processamento; custo monetário, ou uma combinação dessas propriedades. Numa opção mais sofisticada, a seleção poderá ser feita através de uma função utilidade (referenciada no contrato) provida pelo projetista como proposto em [Hua04]. Assim, ponderações sobre conjuntos de propriedades (e.g., *clockFrequency* e *utilization* da categoria *Processing, delay* e *bandwidth* da categoria *Transport*) orientariam a escolha do melhor refletor.

```
01   service {
02     instantiate Terminal with hqTermLProf, hqTermVProf, hqTermAProf;
03     link Terminal to @ref by commCon with hqTermCProf,
                      @ref=select(hqRef, reflector@teleconf);
04   } sHQTerm;
05   service {
06     instantiate Terminal with lqTermLProf, lqTermVProf, lqTermAProf;
07     link Terminal to @ref by commCon with lqTermCProf;
08   } sLQTerm;
09 negotiation {
10   not sHQTerm -> sLQTerm;
11   not sLQTerm -> out-of-service; }
```

**Figura 10. Serviços prevendo adaptação dinâmica de Terminal**

(b) Após a seleção inicial, durante a operação, o enlace ligando o *Terminal* ao refletor começa a apresentar um atraso maior do que o tolerado, ou uma banda menor do que a necessária. Prevendo este caso, o contrato pode incluir um serviço de menor qualidade que consome menos recursos (H261, por exemplo), viabilizando a continuidade da interação.

Para esta nova versão do contrato, dois serviços são especificados (Figura 10):

*sHQTerm*: serviço preferencial, que permite a utilização de áudio e vídeo de alta-qualidade. A instanciação do módulo *Terminal* (linha 2) e a ligação do *Terminal* ao refletor (linha 3) são restritas por perfis dimensionados adequadamente.

*sLQTerm*: serviço de menor qualidade, a ser utilizado em último caso. Equivalente ao serviço *sHQTerm,* sendo que os perfis (linhas 6-7) contêm as restrições de propriedades com valores apenas suficientes para dar suporte a este serviço de qualidade inferior.

Os perfis utilizados para indicar os requisitos dos serviços *sHQTerm* e *sLQTerm* são apresentados na Figura 11 (a) e (b), respectivamente. Os valores para cada propriedade foram selecionados de modo consistente com o serviço ao qual está associado. Na cláusula de negociação, Figura 10, o serviço *sHQTerm* (linha 10) é inicialmente negociado. Caso não seja possível manter este serviço (considerado que neste exemplo será possível implantá-lo inicialmente), o CM negociará a implantação do serviço *sLQTerm* (linha 11), de menor qualidade. Se nenhum serviço puder ser implantado a aplicação é terminada.

Em tempo de implantação do contrato, o CM deve selecionar um refletor específico. Assim, na descrição da ligação do *Terminal*, no serviço preferencial (Figura 10, linha 3) é usada a variável de contexto *@ref*, que vai conter a referência ao refletor a ser efetivamente usado na ligação. Essa referência é resolvida, quando da implantação do serviço inicial, pela execução da primitiva *select*, que tem como parâmetros uma

função de seleção (escolhida dentre as descritas na categoria *Refl*, Figura 12(a)) e uma referência ao contexto/domínio, a ser considerado na seleção (no caso, *reflector* do contexto *@teleconf)*. Neste exemplo, a função utilidade (*optim* - enumerada no perfil *hqRef*, Figura 12(b)) é aplicada aos perfis de cada refletor, que faz parte do contexto, sendo os resultados classificados de modo determinar qual o refletor mais adequado. A referência da instância específica escolhida é então utilizada nas operações de configuração.

```
profile {
  LocalResc.utilization: 30;
  LocalResc.clockFrequency: 1600;
  LocalResc.memReq: 512;
} hqTermLProf;

profile {
  VideoMedia.codec: MJPEG;
  VideoMedia.resolution: CIF;
  VideoMedia.frameRate: 24;
  VideoMedia.quality: HIGH;
} hqTermVProf;

profile {
  AudioMedia.codec: G711;
  AudioMedia.sampleLenght: 16;
  AudioMedia.sampleRate: 32000;
  AudioMedia.channels: MONO;
} hqTermAProf;

profile{
  Transport.delay: 80;
  Transport.bandwith: 1600;
} hqTermCProf;
```

**(a)**

```
profile {
  LocalResc.utilization: 40;
  LocalResc.clockFrequency: 400;
  LocalResc.memReq: 128;
} lqTermLProf;

profile {
  VideoMedia.codec: H261;
  VideoMedia.resolution: QCIF;
  VideoMedia.frameRate: 14;
  VideoMedia.quality: MEDIUM;
} lqTermVProf;

profile {
   AudioMedia.codec: DVI;
   AudioMedia.sampleLenght: 8;
   AudioMedia.sampleRate: 800;
   AudioMedia.channels: MONO
} lqTermAProf

profile{
  Transport.delay: 80;
  Transport.bandwith: 128;
} lqTermCProf;
```

**(b)**

**Figura 11. Perfis dos serviços (a) *sHQTerm* e (b) *sLQTerm***

```
category {
   selPol (random, lowCost, bestPerf, optim);
} Refl;
```

**(a)**

```
profile {
   Refl.selPol: optim;
} hqRef;
```

**(b)**

**Figura 12. (a) Categoria *Refl e* (b) Perfil *hqRef***

**Observações.** Além dos cenários e contratos apresentados, outras possibilidades poderiam ser exploradas, como adicionar dinamicamente um refletor à rede *overlay*, para permitir o balanceamento da carga de terminais durante a operação. Em um outro caso, em uma versão refinada do contrato do cliente da Figura 10, estando em vigor o serviço *sLQTerm* (de menor qualidade), o serviço *sHQTerm* (de maior qualidade) poderia (voltar a) ser implantado, caso seus perfis fossem atendidos.

## 4. Implementação e Avaliação

Os mecanismos básicos de interpretação e gerenciamento de contratos são recorrentes para a classe de aplicação de interesse, constituindo um padrão de projeto (*design pattern*) descrito em [Lis03]. Com base neste padrão, o suporte específico para contratos foi implementado na forma de um conjunto de classes pré-compostas (um *framework* de classes) [Cor05], no qual objetos *hot-spot* permitem especializar algumas classes para atender a requisitos específicos de cada aplicação. Utilizando esse *framework*, a aplicação de videoconferência foi implementada com o objetivo de avaliar a abordagem de contratos e realizar experimentos relacionados ao chaveamento entre serviços. Especializações do *Contractor* foram implementadas, de forma tratar as

restrições específicas aos serviços dos contratos relacionados a essa aplicação. Da mesma forma, RAs foram especializados para oferecer acesso às métricas associadas a cada contrato. Especificamente, foram desenvolvidos três RAs:

- Processamento: realiza a descoberta e monitoração de recursos locais como CPU e memória; implementado utilizando a ferramenta Ganglia [Mat04].
- Rede: provê medidas de banda passante e atraso para os enlaces de comunicação; baseado na ferramenta Abing [Nav03].
- *Real Time Protocol*: encarregado da monitoração de propriedades dos fluxos de mídia. Desenvolvido a partir da biblioteca *commons* distribuída com o programa VIC [Mbone]. Na versão atual, monitora a perda de pacotes RTP.

O refletor utiliza como base o código descrito em [Hod98], que faz apenas o repasse de pacotes UDP. A versão original foi ampliada com uma API que permite fazer o gerenciamento dos participantes e a criação dinâmica de sessões. Para o experimento, configuramos cenários com fluxos de áudio e vídeo pré-gravados. Foram utilizadas máquinas (Pentium 4 de 2.80 GHz/ 512 MB de RAM) distribuídas entre a UFF e a UERJ, conectadas através da RedeRio, "estrangulada" com um HUB de 10 Mbps (para viabilizar a injeção de fluxos de *background*, sem impactar os nós sob observação), com uma conexão redundante através da rede GIGA, utilizando *switches* de 100/1000 Mbps.

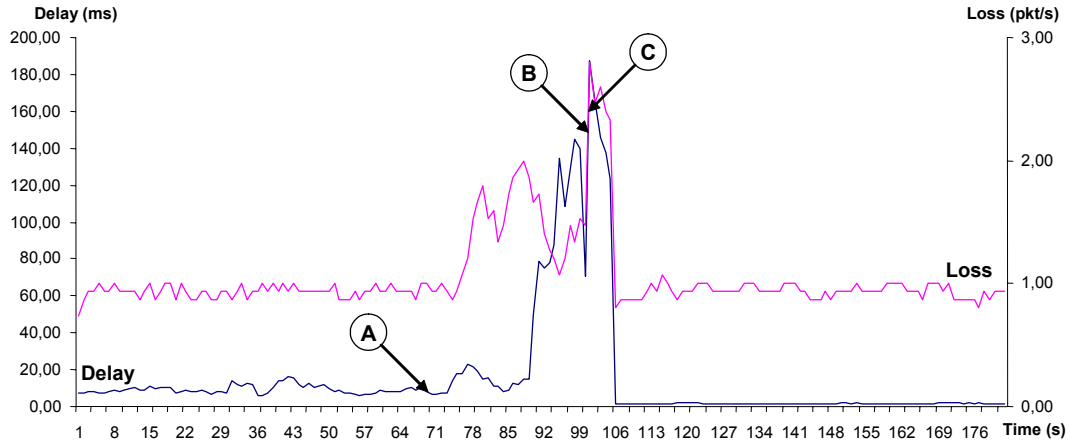



**Figura 13. Medidas (média móvel) de atraso e perda de pacotes RTP**

A Figura 13 apresenta o resultado de um experimento associado ao contrato de comunicação entre dois refletores, *refUFF* e *refUERJ*; serviço *primLink*, descrito na Figura 7. Um fluxo de áudio foi transmitido de *refUFF* para *refUERJ* e um fluxo UDP injetado como *background*. A medida de atraso (*delay*) foi utilizada para identificar a necessidade de adaptação. Também foi monitorada a perda de pacotes RTP em *refUERJ,* para avaliar a qualidade do fluxo de áudio sob teste. O ponto *A* mostra o início da introdução do fluxo UDP. O ponto *B* indica o instante em que o RA no *refUERJ* monitora um valor acima do estabelecido no perfil (150 ms), fato reportado ao CM através do *Contractor*. O ponto *C* aponta o instante em que o serviço alternativo (*altLink*) é implantado (entre B e C, a máquina de estados é avaliada, e um novo serviço é selecionado e implantado). O RA continua a reportar perdas de pacotes, que são descartadas pelo *Contractor*, até que o serviço alternativo tenha tempo suficiente para

entrar em regime permanente; verifica-se que a partir deste ponto o valor do atraso retorna ao patamar desejado.

De modo a evitar instabilidades, provocadas por medidas fora do padrão temporário do fluxo monitorado, que poderiam causar falsas transições entre serviços, foi usado um filtro de média móvel. No experimento da Figura 13, foi adotado um período de amostragem de 3 segundos, sendo a média móvel calculada sobre os 5 últimos valores. Por outro lado, segundo testes executados em [Cor05], o tempo médio gasto entre o recebimento de uma notificação de violação de perfil e o estabelecimento de um próximo serviço é cerca de 2,3 ms. Adicionalmente, o tempo gasto para trocar/adicionar um componente (o que pode ser necessário para efetivar uma adaptação, e.g., trocar um *codec*), utilizando um configurador integrado ao *middleware* [San05], varia entre 10,4 ms (melhor caso: execução local/objeto pré-instanciado) e 28,7 ms (pior caso: execução remota/com criação de objeto). Essas últimas medidas demonstram que o gargalo, que limita o tempo total de adaptação, é explicitamente ligado ao intervalo necessário para identificar de forma consistente a violação dos perfis, associados ao serviço em atividade. Desta forma, fica constatada a necessidade de desenvolver técnicas mais sofisticadas, que permitam disparar, rápida e consistentemente, o chaveamento entre serviços; [Bha99] apresenta uma investigação mais detalhada deste tópico.

## 5. Trabalhos Relacionados

A abordagem descrita neste texto abrange um grande espectro de tópicos [Szt05]. Aqui relacionamos três trabalhos relevantes na área.

Rainbow [Gar04] é uma proposta que também adota elementos semelhantes aos descritos na seção de introdução. Ela utiliza *invariantes da aplicação* em conjunto com *estratégias de adaptação*, que são, respectivamente, equivalentes a perfis e serviços dos contratos. As adaptações em Rainbow, em contraste com a nossa abordagem, que as realiza explicitamente no nível da arquitetura, são feitas através de procedimentos que embutem as ações de configuração, o que dificulta a verificação formal dos contratos, como factível em CR-RIO [Rad05].

A abordagem do projeto Tapas [Lam03] aproxima-se de CR-RIO, permitindo a especificação de restrições não-funcionais a partes específicas da arquitetura da aplicação através da linguagem SLAng [Ske04]. Ela utiliza conceitos similares a categorias e perfis, os quais são restritos a requisitos não-funcionais específicos às arquiteturas de aplicações de *e-business*, estruturadas sobre a WWW, e baseadas na tecnologia de componentes e containeres. Além disso, a proposta limita-se ao nível de especificação, apenas sugerindo que as restrições podem ser verificadas em tempo de execução e eventualmente usadas para guiar adaptações.

Huang apresenta uma proposta para auto-configuração de serviços baseada em receitas (*recipes*) ligadas à arquitetura da aplicação [Hua04]. No entanto, as receitas consideram apenas aspectos ligados ao nível de comunicação (retardo e vazão) e a associação com o nível arquitetural é apenas ilustrada através de um simulador. Um atrativo dessa proposta é o uso de funções utilidade, que identificam conjuntos adequados de recursos, para configurar inicialmente a aplicação. Após a iniciação, adaptações locais podem ser rapidamente decididas e efetivadas para dar continuidade à

aplicação. Nossa abordagem generaliza essa proposta contemplando aspectos não-funcionais diversos, bem como permite a descrição explícita das adaptações a serem realizadas.

## 6. Conclusão

Apresentamos uma abordagem que integra diversos elementos necessários para a concepção e implementação de aplicações com requisitos não-funcionais. Segundo essa abordagem, as especificações não-funcionais das aplicações são expressas através de contratos, associados às descrições de suas arquiteturas. A partir dos contratos, diversos tipos de requisitos podem ser automaticamente impostos, através de adaptações sobre a configuração da aplicação ou sobre sua infra-estrutura de suporte. Nesse contexto, deve ser notado que uma aplicação pode de fato ser gerenciada por diversos contratos, cada qual associado a um dos diferentes elementos (recursos, aplicações ou serviços) que a constituem, tornando-a auto-gerenciável. Desta forma, o objetivo de obter sistemas autônomos, com capacidade de realizar internamente e automaticamente as adaptações requeridas para atender as especificações contratuais, pode ser alcançado.

Uma questão relevante, no contexto de adaptação, visando qualidade de serviço, é a separação entre políticas e mecanismos na implementação dos componentes (ou recursos) utilizados na provisão de requisitos não-funcionais. Tradicionalmente, como em protocolos de comunicação, por exemplo, as políticas e mecanismos de adaptação são embutidos em uma implementação, restringindo as possibilidades de adaptação. Desta forma, um contrato pode apenas ser empregado para especificar o tipo de suporte, ou protocolo, a ser utilizado, e alguns parâmetros associados a ele. De modo a tornar efetivo e facilitar o uso das técnicas de adaptação aqui descritas, espera-se que uma separação explicita seja adotada futuramente, no projeto dos artefatos comumente usados no suporte a requisitos não-funcionais. Especificamente, seus agentes de recurso devem prover APIs (padronizadas) para gerência e monitoramento de seus mecanismo internos. Isto facilitará a integração com o nível alto de gerenciamento, permitindo mais flexibilidade para impor políticas adequadas para atender as exigências das aplicações.

**Agradecimentos.** Os autores gostariam de agradecer o apoio e a infra-estrutura da RNP/CPqD/Finep (Projeto GIGA - CARAVELA). Alexandre Sztajnberg agradece o apoio da Faperj (PAEP 04/2005 E-26/171.130/2005).

## Referências

[Ada03] Adamczyk, D., Collados, D., Denis, G., Fernandes, J. et al., "VRVS 3 - Global Platform for Rich Media Conferencing and Collaboration", Third Annual Workshop on Advanced Collaborative Environments, Seattle, Washington, Junho, 2003.

[Beu99] Beugnard, A., Jézéquel, J.-M., Plouzeau, N. e Watkins, D., "Making Components Contract Aware", IEEE Computer, Vol. 32, N.7, pp. 38-45, Julho 1999.

[Bha99] Bhatti, S. N. e Knight, G., "Enabling QoS adaptation decisions for Internet applications", Computer Networks, Vol. 31, No. 7, pp. 669-692, Março, 1999.

[Cor05] Corradi, A. S., "Um Framework de Suporte a Requisitos Não-Funcionais para Serviços de Nível Alto", Dissertação de Mestrado, Inst. de Computação, UFF, 2005.

[Gar04] Garlan, D., Cheng, S., Huang, A., et al., “Rainbow: Architecture-Bases Self Adaptation with Reusable Infrastructure”, IEEE Computer, Vol. 37, No. 10, Outubro, 2004.

[Hod98] O. Hodson, O., Varakliotis, S. e Hardman, V., “A software platform for multi-way audio distribution over the Internet”, Audio and music technology: the challenge of creative DSP, IEE Colloquium, London, Novembro 1998.

[Hua04] Huang, A.-C.; “Building Self-configuring Services Using Service-specific Knowledge”, PhD Thesis, Carnegie Mellon Univ., Pittsburgh, PA, Dezembro, 2004.

[Lam03] Lamanna, D. D., Skene, J. e Emmerich, W., "Specification Language for Service Level Agreements". D2 - TAPAS Project, http://www.newcastle. research.ec.org/tapas /deliverables/D2.pdf, 2004.

[Lis03] Lisbôa, J. C., Loques. O., “Um Padrão Arquitetural para Gerenciamento de Qualidade de Serviço em Sistemas Distribuídos”, The 4th SugarLoafPLoP Conference, Porto das Dunas, 2004.

[Loq04] Loques, O., Sztajnberg, A., Cerqueira, R. C. e Ansaloni, S., “A contract-based approach to describe and deploy non-functional adaptations in software architectures”. JBCS, Vol. 10, No. 1, pp. 5-18, Julho, 2004.

[Loq04+] Loques, O., Sztajnberg, A., Abelém, A., Braga, C. e Stanton, M., “CARAVELA - Contratos para Aplicações em Redes de Alta Velocidade”, Projeto GIGA - Coordenações Temáticas RNP, RNP/CPqD/Finep, 2004.

[Mat04] Matthew, M. L., Brent, C., e David, C., “The Ganglia Distributed Monitoring System: Design, Implementation, and Experience”, Parallel Computing, Vol. 30, No. 7, Julho, 2004.

[MBone] UCL Network and Multimedia Research Group, “Mbone Conferencing Applications”, http://www-mice.cs.ucl.ac.uk/multimedia/software

[Nav03] Navratil, J. e Cottrell, R. L., “A Practical Approach to Available Bandwidth Estimation”, Passive / Active Measurement Wks. pp.1-11, La Jolla, CA, Abril, 2003.

[Rad05] Rademaker, A., Braga, C. e Sztajnberg, A.; “A Rewriting Semantics for a Software Architecture Description Language”, Procs. of the Brazilian Symposium on Formal Methods (SBMF 2004), ENTCS, V.130, pp. 345-377, Maio, 2005.

[San05] Santos, A. L. G., “Técnicas de Configuração Dinâmica de Arquiteturas de Software”, Dissertação de M.Sc. (em andamento), Inst. de Computação, UFF, 2005.

[Ske04] Skene, J., Lamanna, D. e Emmerich W., "Precise Service Level Agreements", 26th Int. Conference on Software Engineering, Edinburgh, UK. pp. 179-188, 2004.

[Szt05] Sztajnberg, A., Corradi, A. M., Santos, A. L., Barros, F. A., Cardoso, L. X. T. e Loques, O. G., “Especificação e Suporte de Requisitos Não-Funcionais para Serviços de Nível Alto”, Minicursos do 23o SBRC, p. 223-279, Fortaleza, CE, 2005.

[Tog99] Toga, J. e Ott, J., “ITU-T standardization activities for interactive multimedia communications on packet-based networks: H.323 and related recommendations”, Computer Networks, v. 31, n. 3, p. 205-223, Fevereiro, 1999.

[Wol99] Wolski, R., Spring, T. N. e Hayes, J., “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, Future Generation Computer Systems, Vol. 15, No. 5-6, pp. 757-768, 1999.