Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL

Frank A. Siqueira

Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
E-mail: frank@das.ufsc.br

Resumo

Algumas aplicações distribuídas não funcionam adequadamente sem que garantias de qualidade sejam impostas aos serviços fornecidos pela rede de comunicação e pelas plataformas de execução. Estas aplicações necessitam de suporte a qualidade de serviço (QoS) para funcionar corretamente, obrigando que requisitos temporais, de confiabilidade e de segurança sejam observados pelo sistema. Este trabalho apresenta uma linguagem para especificação de QoS chamada QSL (*QoS Specification Language*), que é capaz de expressar os requisitos de diferentes domínios de aplicação, de mapear os requisitos em diferentes níveis de abstração e de especificar diferentes níveis de qualidade. QSL é integrada a métodos e ferramentas de especificação de software, o que permite que os requisitos de QoS sejam considerados ao longo de todas as etapas do desenvolvimento de software. Este trabalho também descreve um suporte de tempo de execução baseado no CORBA para obtenção dos requisitos de QoS especificados em QSL.

Abstract

Some distributed applications are unable to work correctly without imposing quality guarantees on services provided by the communication network and by the execution platforms. These applications require support for quality of service (QoS) to work correctly, implying that timing, realibility and security requirements must be enforced by the system. This paper presents a language for QoS specification called QSL (QoS Specification Language), which is able to express requirements for different application domains, to map requirements between different abstraction levels and to specify different quality levels. QSL is integrated with specification methods and tools, allowing that QoS requirements be taken into account during all phases of software development. This work also describes an execution support based on CORBA that is able to enforce QoS requiremets specified in QSL.

1 Introdução

Com a evolução das plataformas computacionais e de comunicação, passou a se popularizar entre os usuários de computadores uma série de aplicações que não aceita o modo como serviços são oferecidos pelos sistemas tradicionais disponíveis nos dias de hoje, que são baseados na política de melhor esforço (*best-effort*). Estas aplicações, que se situam em diversas áreas de aplicação como multimídia e trabalho cooperativo, impõem requisitos de qualidade sobre os serviços por elas utilizados. De maneira semelhante, uma série de sistemas corporativos, como os sistemas de manufatura, sistemas de compensação bancária e

plataformas de telecomunicações, devido às suas exigências de segurança, confiabilidade e correção temporal, estão migrando para plataformas comerciais abertas como forma de reduzir custos, simplificar a manutenção e facilitar a interoperabilidade entre sistemas. Deste modo, é necessário para estes sistemas se adequarem às limitações existentes nestas plataformas, observando os requisitos de qualidade impostos sobre os serviços utilizados.

De modo a atender os requisitos destas aplicações, engenheiros de software têm buscado fornecer serviços com garantias de qualidade. A chamada qualidade de serviço (ou simplesmente QoS, do inglês *Quality of Service*) vem assim obtendo maior atenção no fornecimento de serviços, englobando desde o hardware computacional e de comunicação, passando pelo sistema operacional e chegando até os serviços da camada de aplicação. Na última década, diversas arquiteturas foram desenvolvidas com o intuito de adicionar suporte de QoS às plataformas de execução de aplicações. No entanto, nenhum mecanismo para especificar QoS se mostrou flexível o bastante para ser utilizado nas várias aplicações e sistemas que exigem requisitos de qualidade [Siqueira 00].

Devido à natureza das técnicas de especificação de QoS propostas, o processo de desenvolvimento de aplicações tem falhado em capturar os requisitos de QoS durante o projeto de software, deixando que os requisitos de QoS sejam tratados no processo de implementação ou mesmo em tempo de execução das aplicações. Para sanar este problema, a especificação de qualidade de serviço deve estar integrada ao processo de especificação de software [Frolund 98].

A utilização de métodos de especificação e projeto de software tem demonstrado ser uma forma eficiente de obter uma maior qualidade de produto e de reduzir os custos de desenvolvimento e manutenção de software. Ferramentas que utilizam linguagens de especificação permitem a detecção de falhas no programa antes das fases de implementação e testes. No entanto, métodos tradicionais de especificação de software, como por exemplo a UML (*Unified Modeling Language*) [OMG 01b], lidam somente com os aspectos funcionais de sistemas, não levando em conta os aspectos de qualidade de serviço durante o processo de especificação. Portanto, se faz necessário definir métodos de desenvolvimento de software que levem em conta a qualidade dos serviços oferecidos.

Neste trabalho estamos propondo uma linguagem para especificação de QoS, chamada QSL (*QoS Specification Language*) que tem como objetivo suprir as limitações existentes em outras linguagens referentes à forma de especificar QoS em tempo de projeto. Os procedimentos necessários para a especificação de QoS e para sua obtenção em sistemas abertos são tratados em maior detalhe na seção 2 deste trabalho. A sintaxe de QSL e sua forma de utilização são demonstrados na seção 3. Na seção 4 apresentamos um modelo de integração de QSL com a linguagem UML e com ferramentas CASE baseadas nesta linguagem, o que fornece ao projetista de software um mecanismo de especificação não só dos aspectos funcionais da aplicação sendo desenvolvida mas também da qualidade que esta deve apresentar após sua implementação. Um suporte de tempo de execução para QSL baseado no CORBA [OMG 01a] é apresentado na seção 5. Na seção 6 apresentamos exemplos de utilização de QSL em ferramentas de trabalho cooperativo. Outras linguagens e suportes apresentados na literatura são comparados com a nossa proposta na seção 7. Finalmente, na seção 8, apresentamos nossas conclusões e perspectivas de futuros trabalhos relacionados ao tema aqui abordado.

2 Qualidade de Serviço

Qualidade de Serviço, ou simplesmente QoS (do inglês *Quality of Service*), pode ser conceituada como a qualidade que deve ser apresentada pelos serviços oferecidos pelo sistema, que é especificada pelo usuário destes serviços na forma de requisitos de qualidade no momento em que o serviço é requisitado.

Apesar ter sido aplicado inicialmente na definição da qualidade da comunicação através de redes de computadores, o conceito de QoS pode ser aplicado em sistemas computacionais para definir a qualidade de qualquer serviço fornecido pelo sistema. Processamento de instruções e transmissão de dados são exemplos de serviços oferecidos pelo sistema, enquanto tempo máximo de processamento e máximo erro de transmissão são respectivamente exemplos de requisitos de QoS aplicados a estes serviços. Também podemos especificar restrições temporais relativas às características de confiabilidade e de segurança de uma aplicação, como o máximo tempo de falha (*downtime*), ou o método usado para autenticar o usuário de um serviço.

2.1 Especificação de QoS

No processo de especificação é importante levarmos em conta os parâmetros usados para especificar a qualidade de serviço, os diferentes níveis de abstração nos quais QoS pode ser especificada, e os diferentes aspectos que podem ser considerados.

Os parâmetros de QoS são valores que podem ser quantificados de modo a descrever uma característica particular do serviço que está sendo fornecido. Com base no valor do parâmetro, se verifica se um determinado requisito está sendo atendido ou não pelo sistema. Outro fator importante a se ressaltar é que o conjunto de parâmetros utilizados depende diretamente do tipo de serviço cuja qualidade se pretende descrever. Por exemplo, para um serviço de transmissão de vídeo podemos utilizar o parâmetro quadros por segundo, mas para um sistema de manufatura, que também possui restrições de QoS, o mesmo parâmetro já não faz nenhum sentido. Ou seja, o conjunto de parâmetros utilizados para descreves os requisitos de QoS varia conforme o domínio de aplicação.

O nível de abstração incorre da forma como são descritos os requisitos de QoS. De modo geral, quanto mais clara for a noção de recursos fornecidos pela plataforma, mais distante se está do que é inteligível para a aplicação. Neste trabalho vamos considerar que os requisitos de QoS podem ser descritos em dois níveis: ao nível de sistema, levando em conta os recursos utilizados, e ao nível de aplicação, considerando as características do serviço fornecido [Siqueira 00].

Como existe um mapeamento entre parâmetros descritos em níveis diferentes, o processo de mapeamento deve ser efetuado pelo suporte de QoS presente no sistema. Devemos ainda levar em consideração que os parâmetros podem ter formatos completamente diferentes, dependendo do tipo de aplicação em questão, fazendo com que o suporte precise conhecer diferentes formatos e os consiga interpretar de forma adequada. Desta forma, é necessário prover um suporte bastante flexível que reconheça conjuntos de parâmetros diferentes e que permita que novos parâmetros sejam definidos.

Adicionalmente, vários aspectos de qualidade podem ser considerados de acordo com as características do sistema e dos serviços que estão sendo executados. Os aspectos considerados neste trabalho são os de correção temporal, presentes em sistemas tempo-real, de confiabilidade, presentes nos sistemas tolerantes a falhas, os aspectos de segurança. Os requisitos temporais presentes em sistemas tempo-real e os requisitos de confiabilidade

existentes em sistemas tolerantes a falhas, assim como os requisitos de segurança, não passam de um caso particular de requisitos de QoS [Fraga 01].

A especificação de requisitos de QoS pode ser feita tanto de maneira estática (*off-line*) quanto dinâmica (*on-line*), ou através de uma combinação das duas abordagens. A especificação estática utiliza linguagens de especificação de QoS, que permitem que os aspectos de QoS sejam levados em conta no projeto da aplicação. A especificação dinâmica utiliza rotinas de suporte para especificar os requisitos em tempo de execução, dando maior flexibilidade ao usuário na definição dos requisitos, mas permitindo a verificação do QoS somente no momento da execução da aplicação. Já a utilização de linguagens de QoS combinadas com mecanismos dinâmicos traz as vantagens de ambas abordagens, associando uma linguagem de especificação a um suporte de tempo de execução [Frolund 98] [Zinky 97].

Os requisitos de QoS devem ser associados a serviços através das abstrações de linguagem que os implementam, podendo ser aplicados a processos, módulos, objetos, dados, procedimentos e parâmetros de procedimentos. De modo a tornar a especificação de requisitos o mais flexível possível, mecanismos de especificação devem permitir que requisitos sejam associados de maneira individual para cada instância, chamada ou acesso ao provedor de um serviço ou a todas as instâncias/chamadas/acessos efetuados.

2.2 Negociação de QoS

Durante o processo de negociação, as aplicações são informadas da possibilidade de obterem o recurso em tempo de execução. Ou seja, sua obtenção não é garantida ainda neste momento, pois nada impede que a dinâmica do sistema venha a impedir a obtenção do recurso em tempo de execução.

A negociação de QoS implica na execução de verificações de conformidade entre a qualidade fornecida pelo serviço em questão e a qualidade requisitada pelo cliente deste serviço. Neste caso, são usadas linguagens de especificação para descrever a QoS tanto no servidor quanto no cliente, e efetuados cheques de conformidade *off-line*.

No caso de o processo de negociação não conseguir chegar a uma configuração de recursos que atenda os requisitos especificados pela aplicação, pode ser necessário reduzir o nível de qualidade. Desta forma, é desejável que a aplicação especifique claramente os níveis de QoS suportados para que o suporte saiba como degradar a qualidade sem prejudicar o funcionamento da aplicação.

2.3 Obtenção de QoS

A obtenção de QoS é efetuada com o auxílio de mecanismos disponíveis em nível de sistema operacional e de suporte para comunicação em rede. Tais mecanismos permitem que recursos sejam alocados para serem utilizados por determinadas aplicações em intervalos de tempo específicos. No entanto, muitas aplicações ainda não utilizam os protocolos de reservas de recursos de modo a obter a qualidade de serviço desejada por elas. Em alguns casos, isto se deve ao fato de existir uma grande diferença no nível de abstração no qual tais aplicações entendem QoS e a maneira como os protocolos permitem que recursos do sistema sejam reservados. Em outros casos, as aplicações utilizam algum suporte de processamento que não tira vantagens dos mecanismos de reserva de recursos presentes no sistema. Já nos casos em que a aplicação procura utilizar tais mecanismos, existem dificuldades em se lidar simultaneamente em nível de programação com os aspectos funcionais da aplicação e com os aspectos de qualidade.

Uma série de protocolos de comunicação e de sistemas operacionais com características de tempo-real permitem que recursos sejam alocados por aplicações de maneira precisa. A cada pedido de alocação de recursos é realizado um processo de avaliação dos recursos disponíveis que pode resultar tanto na aceitação quanto na rejeição da operação.

2.4 Monitoramento de QoS

Durante a execução de aplicações com requisitos de QoS, é necessário verificar o cumprimento destes requisitos mesmo após o processo de obtenção dos recursos do sistemas determinados como necessários para que os requisitos especificados sejam cumpridos. Esta necessidade advém do fato de, em alguns casos, ao executar o processo de mapeamento, o suporte poder subestimar a quantidade de recursos necessária para cumprir os requisitos de QoS da aplicação. Em outras situações, o próprio sistema pode comportar-se de maneira inesperada ao disponibilizar um recurso que apresenta falhas momentâneas, não suprindo a contento as necessidades da aplicação.

Deste modo, faz-se necessário efetuar procedimentos de monitoramento de QoS que, ao verificarem que um ou mais requisitos da aplicação não estão sendo atendidos, devem iniciar um processo de renegociação de QoS. Este processo deve verificar o mapeamento de recursos, evitando utilizar os recursos que apresentaram falhas, e redimensionando a quantidade de recursos a ser alocada para atender a determinados requisitos de QoS.

2.5 Adaptação de QoS

Em sistemas abertos temos que levar em conta o fato de que recursos podem tornar-se indisponíveis devido à reconfiguração dinâmica do sistema – máquinas podem ser ligadas e desligadas, segmentos de rede podem ficar sem comunicação durante um certo tempo e a mobilidade de máquinas pode mudar a qualidade dos serviços fornecidos.

Devido à dinâmica do sistema, pode ser necessário modificar o mapeamento de recursos e refazer o processo de negociação e de obtenção de QoS (ou seja, alocação de recursos). Nestes casos, ao remapear e renegociar um recurso, pode ser possível manter os requisitos da aplicação, fazendo a adaptação de maneira transparente [Siqueira 00]. Caso isto não seja possível, será necessário solicitar que a aplicação se contente com uma qualidade menor, adaptando seu funcionamento à qualidade que pode ser oferecida naquele momento [Gecsei 97]. Nestes casos, cabe à aplicação estar preparada para aceitar uma QoS menor, modificando seu funcionamento de acordo com o nível de qualidade suportado [Waddington 98].

3 A Linguagem QSL

Esta seção descreve em detalhes a linguagem proposta neste trabalho para especificação de requisitos de QoS. Esta linguagem, chamada QSL (*QoS Specification Language*), tem como objetivo facilitar o processo de especificação de requisitos de QoS efetuado pelo desenvolvedor de software no momento do projeto de uma aplicação distribuída em sistemas abertos. QSL foi projetada com o intuito de integrar-se facilmente a suportes para QoS baseados na obtenção de recursos e/ou na adaptação da aplicação.

3.1 Parâmetros e Requisitos de QoS

Os parâmetros de QoS são definidos com base em tipos básicos ou definidos pelo usuário. Já os requisitos de QoS associam valores a parâmetros. Estes valores, que quantificam a qualidade desejada, devem ser observados no momento da utilização do serviço.

Os parâmetros de QoS definidos em QSL podem ser associados a valores booleanos, numéricos, temporais ou a listas enumeradas.

Os parâmetros booleanos, numéricos e temporais são declarados com a sintaxe:

```
Tipo NomeDoParâmetro;
```

Os parâmetros booleanos, do tipo **bool**, podem assumir os valores **true** ou **false**.

Os parâmetros numéricos podem ser de tipos simples ou compostos. Os tipos simples são inteiro curto (**int**) e longo (**long**), e real de ponto flutuante curto (**float**) e longo (**double**). Os tipos numéricos simples podem ser precedidos da palavra-chave **unsigned** para indicar que possuem valor sem sinal.

Os tipos compostos (inteiros ou reais precedidos da palavra-chave **composed**) são estruturas de dados que possuem campos para especificação de valores máximo, mínimo, médio, variância, e distribuição percentual.

Os parâmetros temporais, do tipo **time**, são usados para especificação de tempo absoluto ou relativo. Estes parâmetros possuem campos de tipo **long** para especificação do ano, de tipo **unsigned int** para especificar o mês, dia, hora, minutos, segundos, milissegundos e microssegundos, e de tipo **float** para especificar a diferença em horas em relação ao meridiano de Greenwich.

Já os tipos enumerados são definidos como uma lista de valores em ordem crescente de qualidade, declarada da seguinte forma:

```
enum NomeDoParâmetro { Valor1, Valor2, ... };
```

Valores constantes booleanos e numéricos simples podem ser declarados usando a seguinte sintaxe:

```
const Tipo NomeDaConstante = Valor;
```

A definição de requisitos de QoS segue a sintaxe:

```
NomeDoParâmetro = Valor;
```

na qual um identificador pode ser o nome de um parâmetro de tipo simples, com o valor correspondente; o campo de um parâmetro de tipo composto ou temporal, associado ao valor correspondente; ou um parâmetro de tipo enumerado, associado a um dos membros da lista de valores.

3.2 Especificações de QoS

QSL permite que sejam definidas especificações de QoS. Nestas especificações podem ser definidos parâmetros e requisitos de QoS. Especificações podem ser reaproveitadas, adaptadas e aprimoradas por meio de herança. Especificações também podem ser utilizadas como estruturas de dados, e aparecer como um parâmetro contido por outra especificação.

Uma especificação de QoS é descrita da seguinte forma:

```
QoS NomeDaEspecificação : EspecificaçãoBaseA, EspecificaçãoBaseB {
 // ... definições de parâmetros ...
 // ... definições de requisitos ...
};
```

Como vemos, QSL permite múltipla herança de especificações. Parâmetros herdados podem ser redefinidos. Neste caso, deve ser utilizado o nome da especificação seguido de um ponto para se referir ao parâmetro da especificação herdada.

Uma especificação pode possuir vários níveis de qualidade. Os níveis são diferenciados através de um valor booleano, inteiro ou enumerado declarado entre chaves:

```
QoS Especificação [Tipo] { ... };
```

Níveis distintos devem definir requisitos diferentes adotando um mesmo conjunto de parâmetros, através de declarações para cada nível de QoS suportado.

Especificações são associadas a interfaces de serviços utilizando a sintaxe:

```
QoS Interface = Especificação;
```

ou, no caso de ser requisitado um nível específico de qualidade:

```
QoS Interface = Especificação [Nível];
```

Deste modo, o suporte de tempo de execução sabe a que interface a especificação de requisitos de QoS deve ser aplicada.

Os requisitos descritos na especificação associada à interface são válidos para todos os acessos à interface. Caso se deseje especificar requisitos particulares de algum método ou procedimento, parâmetro de chamada ou dado da interface, deve ser usada a seguinte sintaxe:

```
Método :: Parâmetro1 = Valor1;       // Restrição válida para a chamada
Método.Param :: Parâmetro2 = Valor2; // Restrição aplicada somente ao
                                     // parâmetro da chamada
Dado.Get :: Parâmetro3 = Valor3;     // Restrição válida ao ler o dado
Dado.Set :: Parâmetro4 = Valor4;     // Restrição ao modificar o dado
```

Note que diferentes qualidades podem ser especificadas para uma mesma interface em diferentes especificações QSL (ou seja, em diferentes arquivos do tipo .QSL). A seleção da qualidade de aplicações diferentes que forneçam a mesma interface determinada na especificação pode então ser efetuada em tempo de compilação, criando diferentes estruturas de execução para uma mesma interface através do compilador QSL, e associando-as a diferentes aplicações. A seleção da qualidade também pode ser feita em tempo de execução, usando mecanismos dinâmicos de seleção de QoS, caso a especificação preveja diferentes níveis de qualidade. Adicionalmente, os mecanismos dinâmicos de especificação de QoS permitem que os valores dos parâmetros de QoS sejam definidos em tempo de execução, possibilitando que uma mesma aplicação forneça um determinado serviço com qualidades diferentes em instantes de tempo diferentes.

3.3 Scripts de Mapeamento

Relações entre parâmetros podem ser estabelecidas através de scripts de mapeamento de parâmetros. Os scripts permitem que valores de dois ou mais parâmetros sejam relacionados diretamente. Isto faz com que não seja necessário especificar o valor de todos os parâmetros de uma especificação, mas apenas os valores de um subconjunto de parâmetros através dos quais todos os outros possam ser determinados.

Scripts são criados utilizando operações lógicas e aritméticas e operadores condicionais do tipo **if**/**else**. As operações lógicas e aritméticas aceitas por QSL seguem a mesma sintaxe usada por C, C++ e Java.

Por exemplo, um parâmetro de QoS pode ser definido da seguinte forma:

```
Parâmetro1 = Parâmetro2 + (2 * Parâmetro3);
```

ou através de operadores condicionais, como no exemplo abaixo, no qual o valor de um parâmetro corresponde ao maior valor de outros dois parâmetros:

```
if ( Parâmetro1 >= Parâmetro2 )
 Parâmetro3 = Parâmetro1;
else
 Parâmetro3 = Parâmetro2;
```

O uso de scripts de mapeamento permite que sejam definidas em QSL diferentes especificações correspondentes a diferentes níveis de abstração de QoS, e que seja estabelecido o mapeamento entre estes. Definido o mapeamento, não são necessárias alterações no suporte de execução para que um novo conjunto de parâmetros seja utilizado.

Em seguida apresentamos um exemplo de especificação em QSL no qual são definidas uma especificação ATM e uma especificação de Vídeo CBR (com taxa de bits constante, sem nenhuma compressão), e mapeados os parâmetros entre as duas:

```
QoS ATM {
 unsigned int bandwidth;       // Largura de banda do canal ATM
 // … outros parâmetros ...
};

QoS VídeoCBR : ATM {
 unsigned int lines;           // Linhas de vídeo
 unsigned int columns;         // Colunas de vídeo
 unsigned int bits_per_pixel;  // Bits por ponto do vídeo
 unsigned int frames_per_sec;  // Quadros por segundo
 // … outros parâmetros …
 // Script de mapeamento para o parâmetro ATM.bandwidth
 bandwidth = lines * columns * bits_per_pixel * frames_per_sec;
};
```

Através do script de mapeamento, ao especificar a qualidade desejada para o vídeo, é determinada a largura de banda do canal ATM necessária para transmitir este vídeo.

3.4 Compilação

O compilador QSL é responsável por verificar a correção da sintaxe da especificação de requisitos de QoS. Através dos dados adquiridos na compilação da especificação, deve ser gerado o código das estruturas de tempo de execução responsáveis por negociar, obter, monitorar e adaptar a qualidade de serviço da aplicação.

As estruturas de tempo de execução geradas a partir da compilação da especificação dependem diretamente da linguagem na qual o código será gerado. Em linguagens procedurais deve ser gerado um conjunto de procedimentos que implementem os mecanismos de QoS necessários. Por outro lado, em linguagens orientadas a objeto, deve ser gerado um objeto QoS que encapsule dados e métodos necessários para efetuar todo o gerenciamento de QoS. Já a forma de ativação destas estruturas de tempo de execução depende diretamente da plataforma na qual a aplicação será executada. No caso de utilizar-se *middleware* para computação distribuída, como CORBA e DCOM, podem ser criados objetos CORBA e/ou DCOM que interajam com a aplicação e com outros objetos de forma a gerenciar o QoS da aplicação. Na seção 5 deste documento será apresentada uma proposta de implementação de um suporte de QoS sobre a plataforma CORBA baseado na linguagem QSL.

4 Integração com Técnicas de Especificação de Software

Métodos de especificação e projeto de software têm demonstrado ser uma forma eficiente de obter uma maior qualidade de produto e de reduzir os custos de desenvolvimento e manutenção. Ferramentas de que utilizam linguagens de especificação permitem a detecção de falhas no programa antes das fases de implementação e testes. No entanto, os métodos e

ferramentas de especificação de software tratam somente dos aspectos funcionais da aplicação, não prevendo formas de definir a qualidade de serviço desejada.

A especificação de requisitos de QoS, no entanto, se faz apropriada a partir momento do projeto das aplicações distribuídas, como forma de garantir que os requisitos serão levados em conta no desenvolvimento da aplicação. Deste modo, é desejável que uma linguagem de especificação de QoS seja integrada a métodos e ferramentas de especificação de software, dando ao projetista / desenvolvedor de software meios para efetuar o projeto da aplicação considerando não só os aspectos funcionais, mas também os aspectos de qualidade da aplicação [Frolund 98].

Adotamos aqui a linguagem que se estabeleceu como padrão para especificação de sistemas. A UML (*Unified Modeling Language*) [OMG 01b] consiste em uma linguagem unificada e padronizada de modelagem visual de objetos para especificação, construção e documentação de software. A integração com QSL foi efetuada tomando como base os princípios utilizados para definir o *profile* de tempo-real da UML [Selic 00]. O *profile* de tempo-real define contratos de QoS, que contém a especificação dos requisitos de QoS necessários de um recurso na interação entre um cliente e um servidor. Deste modo, uma especificação QSL pode ser associada aos elementos da aplicação distribuída conforme ilustrado na Figura 1.

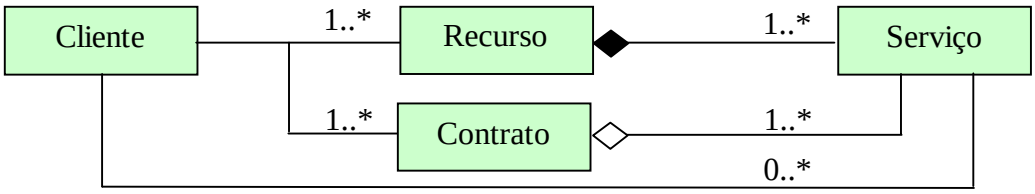

Figura 1 – Representação de Recursos e Contratos de QoS em UML

A Figura 1 mostra que o recurso fornece um ou mais serviços ao cliente, e sobre a qualidade destes serviços vigora um contrato com as especificações de requisitos de QoS. No nosso caso, para cada interação entre um cliente e um serviço, estaremos associando um contrato que consiste na especificação QSL correspondente a esta interação. A noção de recurso utilizada nas especificações em QSL contidas nos contratos de QoS pode ser tão abstrata quanto desejado, ou seja, pode ser representado o sistema distribuído como um todo, cada máquina individualmente ou cada um dos componentes das máquinas, por exemplo. De modo geral, quanto mais abstrata a especificação dos requisitos de QoS no contrato, mas abstrata será a representação dos recursos.

Ferramentas CASE podem utilizar diretamente o compilador QSL para gerar o código necessário para garantir os requisitos de QoS da aplicação projetada com o auxílio da ferramenta. Várias ferramentas permitem que o processo de geração de código seja configurado pelo usuário, fazendo com que seja possível integrar o compilador QSL com o gerador de código a partir da UML utilizado pela ferramenta.

Com a integração da ferramenta CASE com o compilador QSL, o desenvolvedor de software pode obter todo o código de tratamento dos aspectos de qualidade de serviço da aplicação que, juntamente com o código gerado a partir da especificação em UML, serve de base para todo o processo de implementação da aplicação. Desta forma, o desenvolvedor de software preocupa-se somente em codificar a funcionalidade da sua aplicação, o que pode ser realizado manualmente ou com o auxílio de outros métodos que permitam a especificação de aspectos funcionais de aplicações distribuídas.

5 O Suporte de QoS em Tempo de Execução

Os mecanismos de suporte a QoS utilizados pelas aplicações em tempo de execução podem ser implementados diretamente sobre a plataforma, ou utilizar-se de *middleware* para sistemas distribuídos abertos. Nesta seção mostraremos como a linguagem QSL é integrada a um suporte para QoS baseado no CORBA [OMG 01a] e em seus serviços de notificação de eventos, tempo-real, tolerância a falhas e segurança.

5.1 QoS no CORBA

O CORBA não define um suporte global para qualidade de serviço. No entanto, alguns de seus serviços, como será visto em seguida, possuem algum suporte para QoS. Cada serviço do CORBA possui uma interface diferente para QoS, mas o formato dos parâmetros utilizado pelos serviços é baseado no serviço de propriedades.

```
// Especificação do Serviço de Propriedades do CORBA em IDL
struct Property {
  string     property_name;
  any        property_value;
};
typedef sequence<Property> Properties;
```

Os parâmetros de QoS definidos em QSL são representados como uma seqüência de propriedades conforme definido acima. Os tipos simples e o tipo enumerado são mapeados diretamente para os tipos correspondentes definidos na IDL do CORBA. Já os tipos compostos e o tipo temporal correspondem em IDL a estruturas de dados com campos do tipo correspondente em QSL.

Cabe ainda fornecer uma interface única para especificação e obtenção de QoS em um ambiente CORBA. Deste modo, estamos integrando a este ambiente um objeto responsável por todo o tratamento de QoS. Este objeto, chamado *Objeto QoS*, comporta-se como um serviço de QoS no CORBA. Através deste objeto, é possível especificar dinamicamente requisitos de QoS, negociar e obter QoS, além de monitorar e adaptar os requisitos de uma determinada aplicação.

A interface do Objeto QoS é descrita em seguida. Esta interface é especializada com base na QSL do objeto CORBA que fornece os serviços sujeitos a restrições de QoS, através do processo de compilação da especificação em QSL deste objeto.

```
// Interface do Objeto QoS  em IDL
interface  QoS {
    // Exceções
    exception ParamEx { Properties qos_params; };
    exception ParseEx { short num_linha; string qsl_expr; };
    // QoS do Objeto correspondente
    attribute Properties myQos;
    // Leitura de especificação QoS
    void   readQosSpec (in string filename) raises (ParamEx,ParseEx);
    // Métodos para definir/ler valor de um parâmetro
    void   setParam (in string name, in any val) raises (ParamEx);
    void   getParam (in string name, out any val) raises (ParamEx);
    // Iniciar a obtenção de QoS
    void   getQos() raises (ParamEx);
};
```

A Figura 2 mostra o acesso a um objeto CORBA sujeito a restrições de QoS. Chamadas são capturadas por interceptadores do CORBA, tanto no cliente quanto no servidor,

permitindo que código adicional seja executado durante a execução das chamadas. O código para interceptação de chamadas é gerado automaticamente pelo compilador QSL. O interceptador redireciona a chamada ao objeto QoS, que pode então verificar os requisitos de QoS aplicados a esta chamada e interagir com o suporte de execução. Neste momento, o objeto QoS verifica a disponibilidade dos recursos necessários para execução da chamada respeitando os requisitos de QoS em vigor. Caso os recursos disponíveis não sejam suficientes para cumprir os requisitos de QoS, o objeto QoS deve iniciar um processo de adaptação. Assim que forem obtidos os recursos necessários para execução da chamada de modo a atender os requisitos de QoS em vigor, dá-se continuidade à execução normal da chamada. Durante a execução, o objeto QoS deve monitorar a qualidade obtida e, caso necessário, adaptar a alocação de recursos e o comportamento da aplicação.

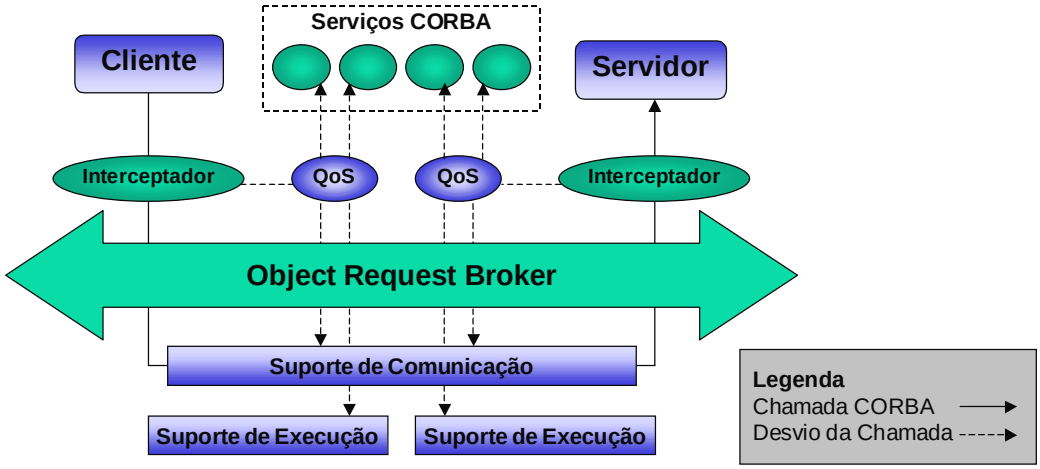

Figura 2 – Acesso a Objetos CORBA com Requisitos de QoS

5.2 Serviço de Notificação

O Serviço de Notificação [OMG 00] é uma extensão do Serviço de Eventos do CORBA. Ambos implementam um mecanismo de comunicação assíncrono e desacoplado para a troca de eventos via rede. O Serviço de Notificação adiciona ao Serviço de Eventos suporte para filtragem de eventos e para especificação e obtenção de qualidade de serviço.

Propriedades de QoS como confiabilidade, prioridade e *timeout*, podem ser especificadas de modo a influenciar a maneira como os eventos são entregues pelos canais de eventos. O serviço de notificação define um conjunto de propriedades que podem ser usadas para definir requisitos de QoS impostos sobre eventos. As propriedades aceitas são as seguintes:

- *Event Reliability*: confiabilidade do evento;
- *Connection Reliability*: confiabilidade da conexão;
- *Priority*: prioridade de entrega dos eventos;
- *Expiration Time*: tempo durante o qual o evento é válido;
- *Earliest Delivery Time*: tempo mínimo para entrega do evento;
- *Maximum Events per Consumer*: limita a quantidade de eventos armazenada no canal para um determinado consumidor;
- *Order Policy*: ordem na qual eventos são armazenados no canal;
- *Discard Policy*: política de descarte de eventos, para o caso de o canal estar cheio.

De modo a suportar a definição destes parâmetros, foi definido um conjunto de parâmetros correspondente em QSL.

```
QoS Notification {
   // Definições auxiliares
   enum reliability { BEST_EFFORT, PERSISTENT };
   // Parâmetros do serviço de notificação
   reliability event_reliability;
   reliability connection_reliability;
   int priority;
   time expiration_time;
   time earliest_delivery_time;
   unsigned int maximum_events_per_consumer;
   enum order_policy { FIFO, PRIORITY, EXPIRATION_TIME };
   enum discard_policy { NONE, PRIORITY, EXPIRATION_TIME };
   // Scripts de Mapeamento
   if (reliability != 0) event_reliability = reliability;
   if (reliability != 0) connection_reliability = reliability;
};
```

5.3 RT-CORBA

A falta de suporte de *middleware* para aplicações de tempo real levou a OMG a estender os padrões CORBA, levando à definição do Real-Time CORBA 1.0 [OMG 98a].

No RT-CORBA, a prioridade global CORBA é definida como um valor de prioridade transmitido em mensagens de invocação que pode atravessar diferentes plataformas. O RT-CORBA define dois modos de utilização: os modelos *client-propagated priority*, no qual a prioridade é definida pelo cliente, e o *server-set priority*, no qual o servidor define a prioridade de execução. Um *Pool* de *threads* pode ser criado para processar requisições de métodos recebidas por um determinado servidor, limitando a quantidade de recursos por ele utilizados. Uma fila de requisições pode ser criada e associada a um *pool* de *threads*.

Os requisitos temporais suportados pelo RT-CORBA também podem ser definidos através de QSL. Neste trabalho foi definida a seguinte especificação com os principais parâmetros utilizados pelo RT-CORBA:

```
QoS RT_CORBA {
  // Parâmetros do RT-CORBA
 int priority;
 const int MAX_PRIO = 65535;
 const int MIN_PRIO = - 65535;
 bool thread_pool;
 unsigned int thread_pool_size;
 unsigned int request_queue_size;
 enum priority_mode { CLIENT_PROPAGATED, SERVER_SET };
  // Script de Mapeamento Auxiliar
 if (thread_pool_size != 0) thread_pool = true;
};
```

Outros requisitos de tempo-real dependentes de implementação, como a política de escalonamento e a qualidade exigida no estabelecimento de conexões entre clientes e servidores, podem ser definidos estendendo esta especificação através de herança.

5.4 FT-CORBA

De modo a fornecer suporte a tolerância a falhas nos padrões CORBA, a OMG definiu a especificação FT-CORBA [OMG 99]. Esta especificação define um conjunto de serviços para construção de aplicações confiáveis em sistemas abertos.

A especificação FT-CORBA define interfaces e protocolos para:

- Gerenciamento de replicação: fornece serviços de gerenciamento de propriedades, de grupo de objeto e fábrica genérica;
- Gerenciamento de faltas: define interfaces para detecção de faltas, notificação de faltas e análise de faltas;
- Gerenciamento de recuperação e *logging*: define mecanismos para transferência de estado e recuperação de réplicas faltosas;
- Interoperabilidade: cria a referência de grupo de objetos, a IOGR (*Interoperable Object Group Reference*), uma extensão da IOR (*Interoperable Object Reference*) que contém referências para cada membro do grupo.

O suporte a tolerância a falhas fornecido pelo FT-CORBA pode ser configurado através de QSL, usando a especificação descrita abaixo.

```
QoS FT_CORBA {
 // Parâmetros do FT-CORBA
 enum replication_technique { PASSIVE_COLD, PASSIVE_HOT, ACTIVE };
 unsigned int number_of_replicas;
 bool detect_host_faults;
 bool detect_process_faults;
 bool detect_object_faults;
 bool detect_all_faults;
 // Script de Mapeamento Auxiliar
 if (detect_all_faults == true) {
    detect_host_faults = true;
    detect_process_faults = true;
    detect_object_faults = true;
 }
};
```

5.5 CORBASec

CORBA possui também um modelo de segurança para objetos distribuídos, o CORBASec [OMG 98b]. Mecanismos para autenticação e a verificação da autorização na invocação de métodos remotos, para segurança da comunicação entre objetos, além de aspectos relacionados com esquemas de delegação de direitos, não-repudiação, auditoria e administração da segurança são previstos pelo CORBASec.

Dentre as tecnologias que podem ser utilizadas para fornecer os serviços de segurança, está o SSL (*Secure Socket Layer*). O SSL fornece uma camada de transporte segura sobre o TCP/IP. Políticas baseadas em identidade são adotadas caso as características de autenticação opcionais do SSL sejam utilizadas.

Assim como foi feito com os outros serviços do CORBA que possuem suporte para especificação de QoS, foi criada uma especificação em QSL para as características de segurança suportadas pelo CORBA_SSL:

```
QoS CORBA_SSL {
 unsigned int ssl_version;
 enum encryption_algorithm { DES, BLOWFISH, DES3 };
 bool secure_communication = true;
 bool delegation = false;
};
```

6 Exemplo de Aplicação em Trabalho Cooperativo

Um dos desafios atuais para os desenvolvedores de software consiste em fornecer ferramentas de execução de trabalho cooperativo em sistemas abertos e distribuídos [Reinhard 94]. Para que isto seja possível, diversas formas de emular de maneira computacional o contato direto entre usuários devem ser possíveis através da plataforma de cooperação. Conseqüentemente, várias ferramentas devem ser colocadas à disposição do usuário para que este possa interagir com outros usuários e efetuar uma tarefa cooperativa.

Neste estudo consideramos a utilização de quatro ferramentas: *chat*, *whiteboard*, e aplicações de áudio e vídeo conferência. Apesar destas ferramentas serem utilizadas comumente, seu uso em atividades críticas é limitado devido a este tipo de ferramenta possuir requisitos de desempenho e de segurança que inexistem nas aplicações tradicionais.

As ferramentas de áudio e videoconferência, que fazem a transmissão de mídia contínua, devem utilizar um canal ATM exclusivo para transmitir a mídia. As ferramentas dirigidas a eventos, ou seja, o *chat* e o *whiteboard*, devem utilizar o serviço de notificação do CORBA para comunicação. Esse paradigma é adequado para a funcionalidade dessas aplicações, que devem enviar para um grupo de usuários as mensagens com informação de estado – no *chat*, uma nova frase emitida por um dos usuários, e no *whiteboard*, uma nova forma inserida na área de edição. Em todas as ferramentas, estamos considerando a utilização do CORBA com suporte a SSL para garantir a segurança dos dados.

As restrições temporais e de segurança existentes em cada uma das ferramentas podem ser especificadas através da linguagem QSL, com base nas especificações de QoS para o serviço de notificação, para o CORBASec usando SSL, e para redes ATM, que foram apresentados anteriormente. As especificações de QoS para as ferramentas de trabalho cooperativo seguem abaixo:

```
QoS Chat: Notification, CORBA_SSL {
 reliability = BEST_EFFORT;    // Eventos entregues com melhor esforço
 priority = 1000;              // Prioridade de entrega dos eventos
 expiration_time.sec = 5;      // Timeout no envio de eventos (5 seg.)
 ssl_version = 2;              // Versão do SSL
 encryption_algorithm = DES;   // Algoritmo de criptografia dos dados
};

QoS Whiteboard: Notification, CORBA_SSL {
 event_reliability = PERSISTENT;     // Eventos serão persistentes
 priority = 5000;              // Prioridade de entrega dos eventos
 expiration_time.sec = 3;      // Timeout no envio de eventos (3 seg.)
 ssl_version = 2;              // Versão do SSL
 encryption_algorithm = DES3;  // Algoritmo de criptografia dos dados
};

// Lista enumerada para definição dos níveis de qualidade de áudio
enum AudioQuality { PHONE, CD };

QoS AudioConference [AudioQuality] : ATM, CORBA_SSL {
 unsigned int sample_freq;     // Freqüência de amostragem do áudio
 unsigned int bits_per_sample; // Bits por amostra de áudio
 ssl_version = 2;              // Versão do SSL
 encryption_algorithm = DES;   // Algoritmo de criptografia do áudio
 // Script de Mapeamento para o parâmetro ATM.bandwidth
 bandwidth = sample_freq * bits_per_sample;
};

// Lista enumerada para definição dos níveis de qualidade de vídeo
enum VideoQuality { VHS, NTSC, PAL, HDTV };
```

```
QoS VideoConference [VideoQuality] : ATM, CORBA_SSL {
 unsigned int lines;           // Linhas de vídeo
 unsigned int columns;         // Colunas de vídeo
 unsigned int bits_per_pixel;  // Bits por ponto do vídeo
 unsigned int frames_per_sec;  // Quadros por segundo
 ssl_version = 2;              // Versão do SSL
 encryption_algorithm = DES;   // Algoritmo de criptografia do vídeo
 // Script de Mapeamento para o parâmetro ATM.bandwidth
 bandwidth = lines * columns * bits_per_pixel * frames_per_sec;
};
```

Note que para áudio e vídeo conferência são definidos níveis de qualidade, e desta forma os valores de alguns parâmetros devem ser definidos independentemente para cada nível (especificações dos níveis de qualidade não são apresentadas por limitação de espaço).

Todo o controle das aplicações é efetuado através dos mecanismos usuais de comunicação entre objetos do CORBA, utilizando o protocolo IIOP [OMG 96] e adotando o paradigma cliente/servidor. No entanto, neste processo não é necessário aplicar requisitos de qualidade, bastando utilizar o serviço *best-effort* fornecido pelo CORBA.

7 Trabalhos Relacionados

Vários outros trabalhos que objetivam permitir a especificação de QoS e a obtenção de QoS em tempo de execução foram apresentados na literatura., como QDL (*QoS Description Language*) da arquitetura QuO [Zinky 97], e as linguagens QML (*QoS Modeling Language*) [Frolund 98] e HQML (*Hipertext QoS Markup Language*) [Gu 00].

As linguagens citadas acima não possuem mecanismos para mapeamento de parâmetros como os existentes em QSL, o que limita o poder de expressão de requisitos de QoS dado ao desenvolvedor de software, já que o suporte de tempo de execução é capaz de interpretar somente um conjunto limitado de parâmetros. Neste caso, o suporte de tempo de execução precisaria ser modificado para compreender um novo parâmetro de QoS, o que não é necessário em QSL sempre que conseguirmos mapear o novo parâmetro em algum parâmetro já usado pelo suporte.

As linguagens QML e HQML não permitem a especificação de níveis de QoS como é possível em QSL. Já a linguagem QDL possui suporte para níveis de qualidade, chamados por ela de regiões de QoS. A definição de níveis de qualidade facilita a adaptação da aplicação.

QDL e HQML não são integradas a linguagens e métodos de especificação de software ou com ferramentas CASE, como acontece com QML e com QSL. O uso de técnicas de especificação de software e de ferramentas CASE permite que os requisitos de QoS sejam considerados ao longo de todo o processo de desenvolvimento das aplicações.

Devido à completude de suas características, QSL pode ser considerada uma linguagem de especificação mais amigável, mais expressiva e mais flexível para o usuário que as outras linguagens de especificação de QoS propostas na literatura.

8 Conclusões

Neste trabalho nós apresentamos uma linguagem de especificação de QoS, a QSL (*QoS Specification Language*). Devido ao fato de ser integrada a linguagens de modelagem de software e a ferramentas CASE, QSL permite que requisitos de QoS sejam especificados durante o processo de modelagem de aplicações. Desta forma, os requisitos de QoS são considerados durante o projeto e a implementação das aplicações, o que evita que problemas

relativos a QoS venham a aparecer somente nas fases finais do processo de desenvolvimento. A detecção tardia dos requisitos de QoS pode atrapalhar o andamento de todo o processo de desenvolvimento, tendo em vista que um determinado requisito pode exigir modificações na plataforma de execução e na estrutura da aplicação.

O suporte de tempo de execução proposto neste trabalho permite que os requisitos de QoS especificados através da linguagem QSL sejam negociados, observados, monitorados e adaptados caso necessário. O suporte é baseado em objetos CORBA e integrado ao *middleware* de forma a fornecer um suporte completo para computação distribuída em sistemas abertos. Ao longo deste trabalho descrevemos ainda como QSL pode ser utilizada para modelar os requisitos de aplicações de trabalho cooperativo, e mostramos as vantagens de QSL em relação a outras linguagens de especificação de QoS.

Bibliografia

[Fraga 01] J. Fraga, R. Rabelo, F. Siqueira "Suporte a Sistemas Abertos de Larga Escala para Empresas Virtuais", Workshop Tendências em Objetos Distribuídos (WTOD'2001), São Paulo – SP, November 2001.

[Frolund 98] S. Frolund, J. Koistinen, "Quality of Service Aware Distributed Object Systems", White Paper, Hewlett-Packard, 1998.

[Gecsei 97] Jan Gecsei "Adaptation in Distributed Multimedia Systems", IEEE Multimedia, Vol. 4, No. 2, April-June 1997.

[Gu 00] Xiaohui Gu, Klara Nahrstedt "Visual Quality of Service Specification of Distributed Heterogeneous Systems", Tech. Report UIUC DCS-R-2000-2190, Department of Computer Science, University of Illinois at Urbana - Champaign, November 2000.

[OMG 96] Object Management Group, "The Common Object Request Broker 2.0/IIOP Specification", Revision 2.0, OMG Document 96-08-04, August 1996.

[OMG 98a] Object Managemet Group, "Realtime CORBA 1.0: Revised Submission", OMG Document orbos/98-12-10, December 1998.

[OMG 98b] OMG, "Security Service: v1.2 Final", Object Management Group, Document 98-01-02, in CORBAServices, November 1998.

[OMG 99] Object Management Group "Fault-Tolerant CORBA", Joint Revised Submission Document orbos/99-12-08, December 1999.

[OMG 00] Object Management Group "Notification Service Specification", OMG Document formal/00-06-20, June 2000.

[OMG 01a] Object Management Group "The Common Object Request Broker: Architecture and Specification", Revision 2.5, OMG Document formal/01-09-01, September 2001.

[OMG 01b] Object Management Group "OMG Unified Modeling Language Specification", Version 1.4, Document formal/01-09-67, September 2001.

[Reinhard 94] W. Reinhard, J. Schweitzer, G. Völksen, M. Weber "CSCW Tools: Concepts and Architectures", IEEE Computer, May 1994.

[Selic 00] Bran Selic "A Generic Framework for Modeling Resources with UML", IEEE Computer, June 2000.

[Siqueira 00] Frank Siqueira, Vinny Cahill "Quartz: A QoS Architecture for Open Systems", Proceedings of the IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, April 2000.

[Waddington 98] Daniel Waddington and David Hutchison "End-to-End QoS Provisioning through Resource Adaptation", MPG Internal report number MPG-98-10, May 1998.

[Zinky 97] J. Zinky, D. Bakken, R. Schantz "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems, Vol. 3(1), January 1997.