Um Mecanismo Semântico de Busca de Componentes de Software Baseado em Qualidade de Serviço

Gustavo Fortes Tondello, Frank Siqueira

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

{fortes,frank}@inf.ufsc.br

Resumo. *Este artigo apresenta um mecanismo de busca semântica de Componentes de Software voltado para a descoberta de Componentes que atendam a um conjunto definido de restrições de QoS. Nossa abordagem se baseia no uso da Ontologia QoS-MO, que provê meios de especificar semanticamente as características de QoS de Componentes de Software ou Serviços Web Semânticos; e na linguagem SPARQL, que permite especificar consultas para encontrar Componentes a partir de um conjunto de restrições de QoS. Um protótipo deste mecanismo foi desenvolvido, contando tanto com uma interface de programação como uma interface Web, resultando em uma implementação simples e eficiente para validar o mecanismo proposto.*

Abstract. *This paper presents a semantic search engine for Software Components that focus on the discovery of components that fulfill a defined set of QoS constraints. Our approach relies on the use of the QoS-MO ontology, which provides means for specifying the QoS characteristics of Software Components or Semantic Web Services, and the SPARQL language, which allows the specification of queries for discovering Components based on a set of QoS constraints. A prototype of this search engine has been developed, with both a programming and a Web-based interface, resulting in a simple and efficient implementation which validates the proposed mechanism.*

1. Introdução

No processo de desenvolvimento de software baseado em componentes, freqüentemente são criadas grandes bibliotecas de Componentes de Software. Isto faz com que seja necessária a existência de mecanismos de busca eficientes, que permitam ao desenvolvedor encontrar o componente que procura dentro de uma grande biblioteca ou até mesmo na Internet.

Com o desenvolvimento da Web, também se torna cada vez mais comum a utilização de Serviços Web como Componentes para a construção de um software. Recentemente, após o surgimento da Web Semântica – definida como "uma extensão da Web atual, na qual a informação ganha significado bem definido, permitindo que computadores e pessoas trabalhem melhor em cooperação" [Berners-Lee et al. 2001] – passaram a existir ainda os Serviços Web Semânticos, ou seja, Serviços Web especificados semanticamente. A semântica bem definida destes serviços permite que eles sejam publicados, descobertos, compostos e executados automaticamente na Web Semântica.

Quando existe uma biblioteca com diversos Componentes ou Serviços Web que realizam determinada funcionalidade, pode ser necessário encontrar um componente que, além da funcionalidade desejada, atenda a determinados requisitos de qualidade de serviço (QoS, do inglês *Quality of Service*). Por QoS entende-se a qualidade apresentada por um determinado componente, especificada pelo seu usuário através de um conjunto de requisitos de QoS.

Neste artigo apresentamos um mecanismo de busca de Componentes de Software construído a partir da ontologia QoS-MO, uma ontologia para a especificação semântica de QoS de Componentes de Software e Serviços Web Semânticos [Tondello e Siqueira 2008]. Embora um mecanismo completo de descoberta de componentes de software deva ser capaz de filtrar tanto as características funcionais quanto as características não-funcionais do componente, o trabalho aqui apresentado tem como foco a busca das características não-funcionais de QoS.

Nosso mecanismo combina o uso de um mecanismo de inferência, capaz de identificar as hierarquias de classes e composições presentes na ontologia, com o uso de consultas em linguagem SPARQL [W3C 2007] para encontrar Componentes que satisfaçam um critério específico. Esta abordagem torna possível a descoberta semântica de Componentes de Software ou Serviços Web a partir de requisitos de QoS utilizando uma implementação simples, já que não há necessidade de desenvolver um complexo algoritmo semântico: é necessário apenas converter os requisitos de busca para uma consulta SPARQL.

O restante deste artigo está estruturado da seguinte forma. A seção 2 apresenta as tecnologias que foram usadas para a definição da ontologia QoS-MO. A seção 3 apresenta uma descrição resumida da ontologia QoS-MO, que é a base para a construção do mecanismo de busca. A seção 4 descreve o mecanismo semântico de busca que usa as características de QoS expressadas usando QoS-MO para encontrar os Componentes de Software. A seção 5 apresenta os trabalhos relacionados nesta área de pesquisa. Finalmente, a seção 6 apresenta as conclusões e perspectivas para trabalhos futuros.

2. Contextualização

Esta seção apresenta as tecnologias, linguagens e ontologias que foram utilizadas para construir o *framework* semântico para a descrição de QoS de Componentes de Software que foi utilizado para a construção do mecanismo de busca baseado em qualidade de serviço.

2.1. Ontologias

O termo *ontologia* surgiu na Filosofia, onde se refere a um ramo da metafísica ocupado da existência. Na Engenharia de Software, uma ontologia pode ser descrita como uma "especificação de uma conceitualização", ou, mais especificamente, como "uma descrição dos conceitos e relacionamentos que podem existir para um agente ou uma comunidade de agentes" [Gruber 1993].

Ontologias vêm sendo utilizadas em Engenharia de Software com o propósito de permitir o compartilhamento e o reuso de conhecimento. Quando existe uma ontologia que descreve o conhecimento de um determinado domínio, agentes de software podem ser escritos seguindo o que Gruber chamou de um "comprometimento ontológico", ou

seja, um comprometimento em utilizar o vocabulário definido pela ontologia. Desta forma, diferentes agentes comprometidos com a mesma ontologia conseguem comunicar-se, já que seu vocabulário é entendido por todos, mesmo que os dados que cada agente tenha disponível não sejam os mesmos.

2.2. OWL

OWL (*Web Ontology Language*) [W3C 2004] é uma linguagem para descrição de ontologias, projetada para ser compatível com a *World Wide Web* e a Web Semântica em particular.

A linguagem OWL é usada para definir Classes e Propriedades, relacionamentos entre classes, cardinalidades, relações de igualdade, tipo e características das propriedades e classes enumeradas. Uma classe é definida em OWL como um conjunto de requisitos que um indivíduo precisa preencher para ser considerado uma instância da classe. Estes requisitos são descritos como propriedades da classe e restrições da classe e suas propriedades.

Existem três sublinguagens de OWL, que crescem em expressividade, podendo ser utilizadas segundo as necessidades da aplicação: *OWL Lite*, *OWL DL* e *OWL Full*.

- *OWL Lite* é a mais simples, suportando apenas hierarquias de classificação e restrições simples.
- *OWL DL* suporta o máximo de expressividade possível, mantendo a completude computacional e a decidibilidade. Esta sublinguagem contém todas as construções de OWL, porém existem restrições para o seu uso. A *OWL DL* é particularmente interessante, pois sua expressividade corresponde à da lógica de descrição [Baader et al. 2003], que forma a base formal da OWL.
- *OWL Full* permite a expressividade máxima, sem nenhuma restrição sintática, porém não há nenhuma garantia computacional. É improvável que possa ser desenvolvido um software que suporte inferência para todos os recursos de *OWL Full*.

2.3. Linguagens de especificação de QoS

Diversas linguagens de especificação de QoS foram analisadas, com o propósito de identificar as melhores características de cada uma e adotá-las para serem empregadas na ontologia QoS-MO, que é usada como base para descrição de características de QoS de Componentes de Software.

As linguagens de especificação de QoS analisadas foram: *Quality of Service Modeling Language* (QML) [Frølund e Koistinen 1998], *Web Service Level Agreement* (WSLA) [Ludwig et al. 2003], *QoS Specification Language* (QSL) [Siqueira 2002] e *QoS Description Language* (QDL) [Zinky et al. 1997]. Apesar de fornecerem mecanismos para a especificação de QoS, estas linguagens não possuem a flexibilidade proporcionada por uma ontologia para o tratamento semântico das restrições de QoS.

2.4. Metamodelo de QoS da OMG

O metamodelo do *framework* de QoS da OMG [OMG 2006] define uma linguagem abstrata para uma linguagem de modelagem que suporta a modelagem de conceitos de

QoS. Ele foi desenvolvido como um metamodelo para a definição do *Profile* de QoS para UML. Este metamodelo é largamente aceitado em sua área de pesquisa devido à sua adequação para a modelagem dos conceitos de QoS e é também adotado como a referência básica para a definição da ontologia QoS-MO.

O metamodelo de QoS da OMG define três pacotes. O pacote *QoS Characteristics* (características de QoS) contém os elementos de modelagem para a descrição de características de QoS, dimensões de QoS (diferentes formas de quantificar uma característica) e contextos de QoS (restrições de qualidade que combinam diversas características de QoS). O pacote *QoS Constraints* (restrições de QoS) inclui os elementos de modelagem para a descrição de contratos e restrições de QoS. O pacote *QoS Levels* (níveis de QoS) inclui os elementos de modelagem para a especificação de níveis e transições de QoS.

2.5. Ontologias de Modelagem de QoS

Antes de decidir pela definição da ontologia QoS-MO, os autores analisaram algumas ontologias existentes para a especificação de QoS, e identificaram uma série de limitações. As ontologias analisadas foram:

- A ontologia QoSOnt [Dobson et al. 2005], que define um modelo para a especificação de QoS de Serviços Web Semânticos. Ela permite a associação de uma especificação de QoS com um perfil de Serviço Web Semântico.
- A ontologia DAML-QoS [Zhou et al. 2004], que define um modelo para a especificação de parametros e perfis de QoS. Esta ontologia permite a especificação de um perfil de QoS e um conjunto de características de QoS.
- A ontologia OWL-Q [Kritikos e Plexousakis 2006], que é uma ontologia de alto nível que estende OWL-S [OWL-S 2004] para descrever as métricas e restrições de QoS. Usando OWL-Q é possível definir especificações de QoS completas e ela ainda fornece meios para especificar conversão de unidades e métricas compostas. Além disto, um algoritmo semântico de descoberta e seleção que usa a ontologia OWL-Q foi desenvolvido pelos autores.

Em nossa análise, estas ontologias ou não possuem a habilidade para especificar restrições complexas de QoS como mapeamento de dimensões ou perfis complexos de QoS, ou dependem de complexos algoritmos semânticos para encontrar especificações de QoS que preencham determinados requisitos.

3. A Ontologia QoS-MO

A *Quality of Service Modeling Ontology* (QoS-MO, Ontologia de Modelagem de Qualidade de Serviço) [Tondello e Siqueira 2008] é uma ontologia de alto nível que contém elementos para a descrição de características e restrições de QoS de Componentes de Software ou Serviços Web Semânticos.

Para instanciar uma descrição de QoS de um Componente em particular, é preciso criar uma nova ontologia que importa a ontologia QoS-MO, estender as classes básicas da QoS-MO conforme necessário e criar os indivíduos que descreverão as características de QoS. As especificações funcionais e não-funcionais dos componentes ou serviços podem ser descritos em uma única ontologia ou em ontologias separadas. A Figura 1 ilustra a hierarquia de ontologias utilizadas para a definição de um

Componente, considerando que as descrições funcionais e não-funcionais tenham sido feitas em ontologias separadas.

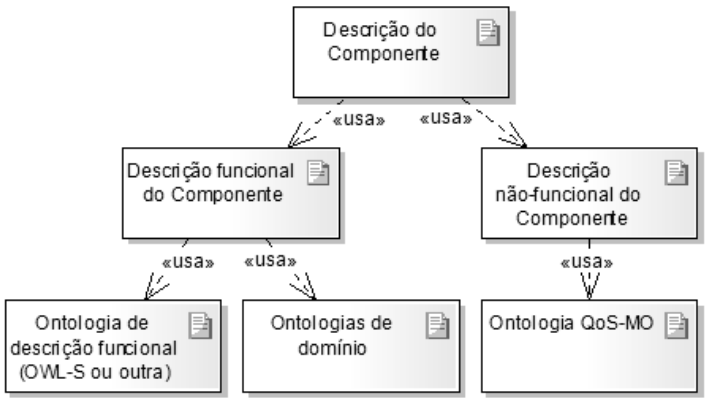

Figura 1. Ontologias para a especificação de um Componente de Software.

A ontologia QoS-MO é composta de três diferentes modelos que atendem todos os requisitos para uma especificação completa de QoS. A seguir será apresentada uma descrição resumida de cada um destes modelos. A especificação completa pode ser encontrada em [Tondello e Siqueira 2008].

3.1. Características de QoS

Este modelo contém as classes para especificar as características de QoS, ou seja, características não-funcionais, de Componentes de Software ou Serviços Web.

A classe *QoSCharacteristic* é a classe principal da ontologia QoS-MO para a definição de características quantificáveis de QoS. As características podem ser gerais, como latência, capacidade de fluxo, disponibilidade, confiabilidade ou segurança, ou específicas de um domínio particular. As características podem ser agrupadas em categorias utilizando a classe *QoSCategory*. A classe *QoSCharacteristic* é uma subclasse de *QoSContext*, sendo que um contexto de QoS é uma definição de um conjunto de uma ou mais características.

As classes *QoSCharacteristic* e *QoSCategory* podem ser estendidas ou especializadas, formando sistemas hierárquicos de classificação de características ou categorias.

As dimensões para a quantificação de características são modeladas usando a classe *QoSDimension*. Uma característica pode ter mais de uma forma de medi-la, ou pode necessitar de mais de uma dimensão para sua quantificação. Exemplos de dimensões para latência são: latência mínima, latência máxima e *jitter* [OMG 2006].

A classe *QoSCharacteristic* não tem um relacionamento direto com a classe *QoSDimension*. Toda dimensão é criada como uma subclasse de *QoSDimension* e uma subclasse de *QoSCharacteristic* definirá novas propriedades cujo alcance será a subclasse criada de *QoSDimension*. Isto permite a definição de dimensões como propriedades das características. Uma *QoSDimension* também pode ser definida como uma referência a uma *QoSCharacteristic*, permitindo, assim, o reuso de características de QoS através de composições de características.

O modelo de características de QoS ainda inclui a classe *QoSValue*, para definir os valores medidos de uma dimensão, e a classe *QoSDimensionMapping*, que é

empregada para definir scripts de mapeamento entre características. Os scripts de mapeamento permitem que mecanismos de busca ou mecanismos automáticos de negociação de QoS possam identificar uma dimensão mesmo se o seu valor não é definido para uma característica, desde que exista um script de mapeamento que permita calcular o seu valor a partir dos valores de outras dimensões.

3.2. Restrições de QoS

Este modelo contém as classes para especificar as restrições de QoS que representam as características de um Componente ou Serviço Web em particular. Há três tipos de restrições de QoS: QoS ofertado, QoS fornecido e Contrato de QoS.

As restrições de QoS ofertado e QoS fornecido (classes *QoSOffered* e *QoSRequired*) são usadas pelo fornecedor ou cliente do serviço ou componente para especificar suas restrições de QoS. Um Contrato de QoS (classe *QoSContract*) representa a qualidade de serviço final acordada entre as partes envolvidas em uma composição.

Uma restrição de QoS precisa referenciar os Contextos de QoS que forneçam as expressões de referência e os valores associados à restrição.

3.3. Níveis de QoS

Este modelo representa os diferentes níveis de QoS que um serviço ou componente suporta. As restrições de QoS são associadas com um nível de QoS (representado pela classe *QoSLevel*) que representa as restrições que devem ser satisfeitas para que seja possível a operação neste nível.

Quando o componente ou serviço não consegue mais operar em determinado nível, por exemplo, quando está recebendo mais requisições simultâneas e seu tempo de resposta máximo terá que ser alterado, uma transição de QoS ocorre. Quando isto ocorrer, é provável que os componentes envolvidos precisem renegociar seus parâmetros de execução e Contratos de QoS. A classe *QoSTransition* define as possíveis transições entre os níveis de QoS e as ações de adaptação que precisam ser executadas quando a transição ocorre.

3.4. Exemplo de Especificação

A ontologia QoS-MO estende as ontologias de especificação de Componentes ou Serviços Web Semânticos com informações de QoS. A especificação funcional dos Componentes ou Serviços Web pode ser feita com qualquer ontologia específica para este fim, como, por exemplo, a ontologia OWL-S [OWL-S 2004] para Serviços Web Semânticos ou a ontologia COSC [Ontoware 2008] para Componentes de Software.

As restrições de QoS ofertado e QoS fornecido especificadas em QoS-MO podem se referir a um único componente ou serviço ou a uma operação específica de um componente ou serviço. Os contratos de QoS serão associados com os dois ou mais componentes ou serviços envolvidos no contrato.

A Figura 2 mostra um exemplo completo de especificação de um perfil de QoS para um componente de software, que teria sido especificado utilizando a ontologia COSC. O componente, denominado apenas *Component*, possui um perfil de QoS, representado pelo indivíduo *Component_QoSProfile*. Este perfil de QoS define uma

restrição *Component_QoSOffered*, cujo contexto é definido por uma característica *Component_ResponseTime* com uma dimensão *Component_MaxReponseTime*, à qual foi atribuído o valor de 10 e a unidade *seconds*. Este perfil de QoS define que o componente em questão tem um tempo de resposta máximo de 10 segundos.

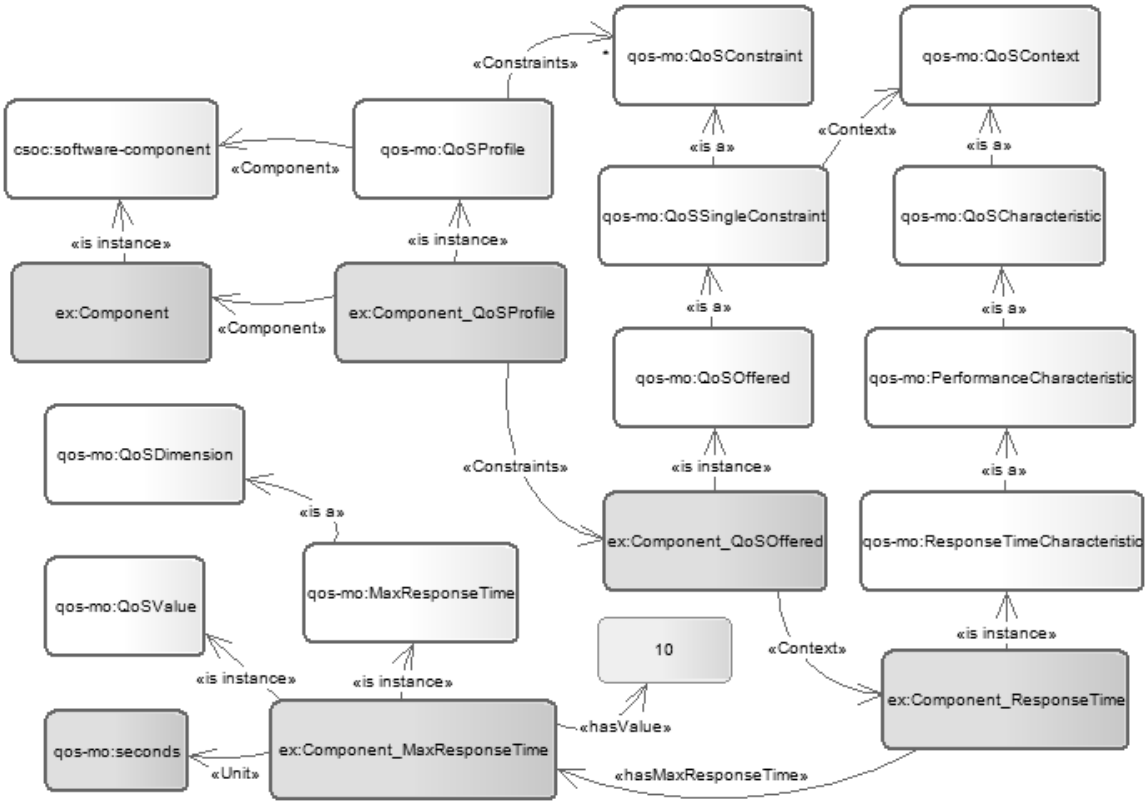

Figura 2. Exemplo de definição de um perfil de QoS para um componente de software.

4. O Mecanismo de Busca

Foi desenvolvido um mecanismo de busca que permite a localização de serviços ou componentes especificados utilizando a ontologia QoS-MO, levando em conta um conjunto de requisitos de QoS especificados pelo usuário. O mecanismo desenvolvido gera consultas SPARQL [W3C 2007] com o intuito de localizar modelos de QoS que satisfaçam os requisitos especificados e então retornar uma lista com os Serviços Web ou Componentes de Software correspondentes para o usuário.

O mecanismo de busca consiste em uma API (*Application Programming Interface*), que permite ao usuário identificar as características de QoS disponíveis na ontologia que descreve os perfis de QoS dos componentes – baseada na ontologia QoS-MO – e executar consultas para localizar Serviços ou Componentes que obedeçam a um conjunto de requisitos de QoS.

Foi desenvolvida também, utilizando a referida API, uma interface Web de busca de Serviços Web, através da qual o usuário especifica os requisitos de QoS que deseja que sejam satisfeitos pelo Serviço Web. Tais requisitos são enviados à API de busca, que executa a consulta SPARQL com o intuito de localizar serviços que satisfaçam os requisitos do usuário. O resultado da consulta é exibido na interface Web.

A Figura 3 dá uma visão geral do processo de publicação e busca de Componentes utilizando a ontologia QoS-MO para descrição de QoS.

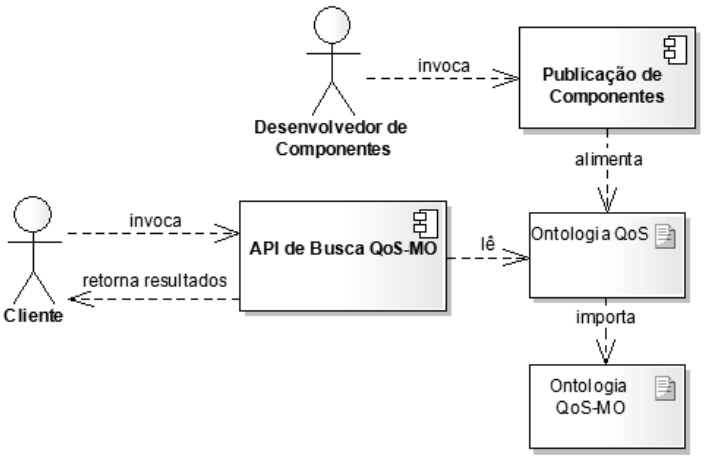

Figura 3. Visão geral do processo de publicação e busca de Componentes.

4.1. A API de Busca

A API de busca foi desenvolvida na linguagem Java, utilizando as APIs Jena (para manipulação dos modelos OWL) e ARQ (para execução de consultas SPARQL) [Jena 2008]. A API de busca pode ser utilizada diretamente por aplicações Java ou por qualquer outra aplicação, através de sua interface, já que ela também é um Serviço Web. A API é subdividida em três grupos funcionais:

- O primeiro grupo é composto por um componente que efetua a leitura da árvore de características de QoS disponíveis na ontologia e a retorna ao cliente. Com isso, é possível construir um suporte para negociação automática de QoS, que possa identificar dinamicamente as características disponíveis no sistema. Tal funcionalidade também torna possível a construção de uma interface gráfica para que o usuário possa escolher, dentre as características de QoS, quais ele deseja que estejam presentes nos Serviços Web ou Componentes que irá utilizar. Essa funcionalidade é demonstrada na interface Web de busca, que será descrita na seção 4.4.
- O segundo grupo funcional permite que o cliente especifique uma consulta e a execute. A execução de consultas é realizada por um componente que converte os requisitos de QoS especificados pelo cliente em uma consulta SPARQL e executa a consulta resultante sobre a ontologia que contém a descrição da qualidade de serviço suportada pelos Serviços Web ou Componentes disponíveis.
- O terceiro e último grupo é formado por um componente que recupera todos os Serviços Web ou Componentes de Software que satisfazem a consulta e os retorna, na forma de uma lista, ao cliente.

Antes de poder executar as consultas SPARQL, o mecanismo de busca carrega a ontologia para a memória e realiza as inferências (*reasoning*) necessárias. Considerando que as consultas SPARQL não possuem a capacidade de efetuar qualquer tipo de inferência, é necessário especificar a realização de inferência durante o carregamento da ontologia. Em nosso caso, é necessário efetuar inferências com o intuito de identificar a

hierarquia de classes da ontologia para classificar os tipos de características, assim como as hierarquias de composições de contextos, restrições e níveis de QoS. Isto permite que as consultas possam localizar os perfis de QoS apropriados, não importando quantos níveis de subclasses ou sub-composições existam nas suas definições.

Este processo de inferência emprega um subconjunto das regras de inferência da OWL, presente na API Jena, chamado *Micro OWL*. Este subconjunto se mostrou suficiente para a implementação do mecanismo de busca, pois ele suporta as duas regras de inferência necessárias para sua construção: a regra *rdfs:subClassOf*, para identificar as hierarquias de classe, e a regra *owl:TransitiveProperty*, para identificar as composições (que são definidas em propriedades transitivas).

4.2. Utilização da API de Busca

Para realizar uma busca, um requisito de QoS é definido como um conjunto de quatro informações: a característica desejada, a dimensão desejada da característica, o operador e o valor desejado para restringir o valor da dimensão. Uma consulta de QoS é definida como uma lista de vários requisitos de QoS.

De modo a exemplificar como as consultas são efetuadas, mostramos abaixo um trecho de código que tem o intuito de localizar Componentes que possuam uma característica de QoS chamada *ResponseTime*, com uma dimensão chamada *MaxResponseTime* cujo valor seja menor que 10 segundos.

```
QoSRequirement req = new QoSRequirement();
req.setCharact(new QoSCharacteristic("ResponseTime");
req.setDimension(new QoSDimension("MaxResponseTime");
req.setOperator(OperatorEnum.LESS);
req.setValue(10.0);
QoSQuery query = new QoSQuery();
query.addRequirement(req);
QoSQueryExecution exec =
    QoSSearchFactory.createQueryExecution();
List<Component> components = exec.executeQuery(query);
```

A partir do requisito definido, o componente de execução de consultas da API irá gerar uma consulta SPARQL semelhante à apresentada abaixo – que aqui foi um pouco simplificada por restrições de espaço (a consulta completa considera as composições de contexto e restrições que possam estar definidas).

```
SELECT
  ?constraint ?context ?charact ?dimension ?value
WHERE {
  ?constraint rdf:type qos-mo:QoSOffered.
  ?constraint qos-mo:Context ?context.
  ?context qos-mo:BasedOn ?charact.
  ?charact rdf:type qos-mo:ResponseTime.
  ?charact ?predicate ?dimension.
  ?predicate rdfs:range qos-mo:MaxResponseTime.
  ?dimension qos-mo:value ?value.
  FILTER (?value < 10)
}
```

4.3. Classificação dos perfis encontrados

No caso da especificação de consultas de QoS com mais de um requisito, o mecanismo de busca é capaz de realizar dois tipos diferentes de consulta: atendimento completo ou atendimento parcial dos requisitos. Na consulta de atendimento completo, são retornados somente os componentes ou serviços que atendam plenamente a todos os requisitos especificados. Na consulta de atendimento parcial, são retornados todos os componentes ou serviços que atendam a pelo menos um dos requisitos. O mecanismo de busca é capaz de executar as duas consultas separadas, para exibir os resultados classificando-os nas duas categorias.

No caso de haver sido encontrados mais de um componente ou serviço na mesma categoria (atendimento completo ou parcial), não é feita nenhuma classificação para tentar identificar qual atenderia melhor, pois isto implicaria em algum tipo de classificação dos requisitos através de pesos.

Na realização da consulta, caso sejam definidos níveis de QoS na especificação dos componentes pesquisados, o mecanismo de busca identifica componentes que tenham pelo menos um nível de QoS que atenda aos requisitos especificados. Nesse caso, a informação dos níveis suportados deverá ser levada em conta pelo ambiente de execução no momento de negociação dos contratos de QoS entre os componentes envolvidos.

4.4. A Interface Web de Busca

A interface Web de busca é capaz de apresentar as características disponíveis na ontologia de QoS para o usuário, utilizando os mecanismos de descoberta de características de QoS da API de busca. Com base nessa informação, o usuário é capaz de especificar uma consulta empregando esta interface. Após a construção da consulta, o botão de busca (*Search*) ativa os mecanismos de execução da consulta e de listagem de Serviços Web providos pela API, e é mostrada em uma nova página uma lista com os Serviços Web que satisfazem os critérios de busca.

4.5. Avaliação do Mecanismo de Consulta

Testes foram realizados para avaliar o desempenho do mecanismo de busca, de modo a permitir uma análise da eficiência da estratégia empregada na sua construção. Os testes foram realizados em um computador com processador Intel Core 2 Duo 2.4 GHz e 2 GB de RAM.

Os primeiros testes realizados avaliaram o tempo de carregamento da ontologia de domínio. Os testes mostraram que a carga utilizando o conjunto de inferência *Micro OWL* da API Jena é efetuada em um tempo aceitável. Quando uma ontologia com 10.000 componentes é utilizada, o tempo de carga é de pouco mais de 10 segundos. Como esta operação só é realizada na carga do sistema, este tempo é considerado baixo.

Um segundo teste foi efetuado com o intuito de avaliar o desempenho da busca. Foi possível verificar nesse teste que, ao aumentarmos o número de componentes registrados em uma determinada proporção, o tempo de busca cresce em uma proporção menor. Conforme mostrado na Figura 4, a busca em uma ontologia com 1.000 componentes leva aproximadamente 1,5 segundo para ser realizada; se aumentarmos dez vezes o número de componentes registrados, o tempo de busca passa a ser de

aproximadamente 5 segundos, menos de quatro vezes mais. Os resultados dos testes, sumarizados na Figura 4, também mostram a quantidade de memória necessária para armazenar a ontologia e os resultados da busca. Também se verificou que o uso de memória não sobe na mesma proporção que o número de componentes: com 1.000 componentes, o uso foi de 45 MB, aumentando dez vezes o número de componentes, o uso de memória foi para 126 MB, quase três vezes mais.

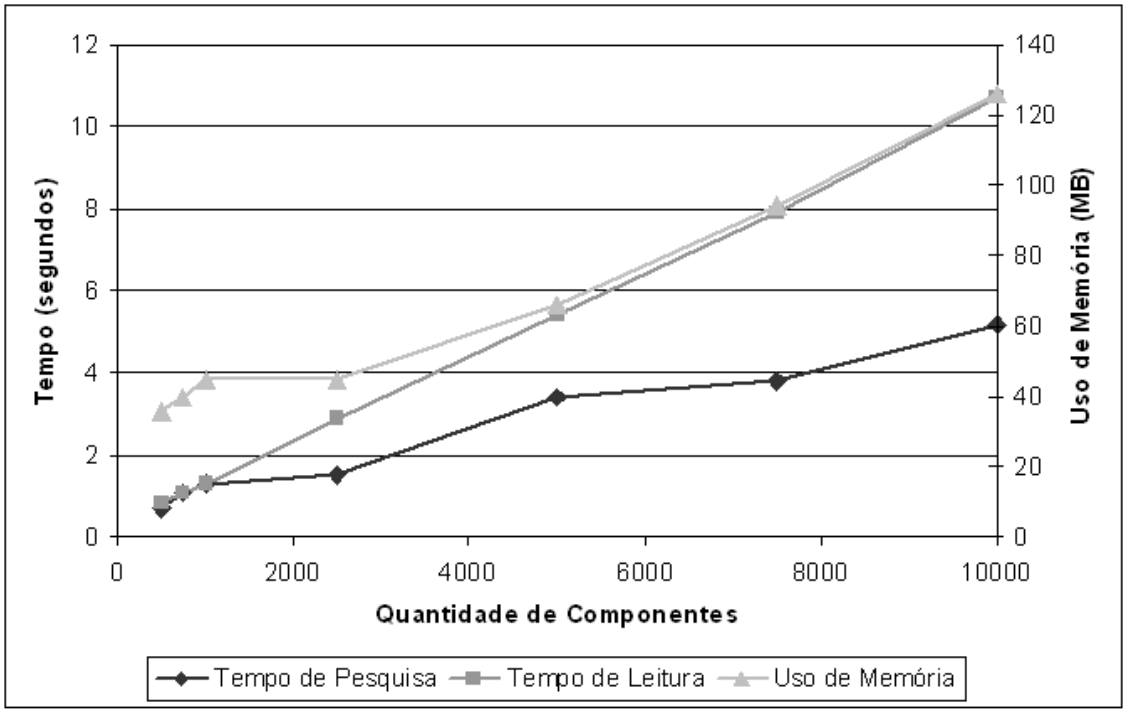

Figura 4. Resultados dos testes de desempenho e consumo de memória para execução de consultas SPARQL sobre a ontologia de QoS de Componentes.

Testes adicionais foram realizados para medir a eficiência do mecanismo de busca para a realização de diversas consultas simultaneamente. Os resultados dos testes, que não serão apresentados aqui por limitação de espaço, mostram que a eficiência da busca diminui conforme vai aumentando o tamanho da Ontologia ou a quantidade de consultas simultâneas, porém em proporção menor do que o aumento registrado da carga. Embora a máquina de teste não fosse um servidor preparado para suportar um grande número de requisições simultâneas, ele foi capaz de manter um bom nível de consultas por segundo. Para realizar um total de 1.000 consultas simultâneas, o desempenho máximo alcançado foi de 77 consultas por segundo em uma Ontologia com 2.500 componentes e o mínimo foi de 7 consultas por segundo para uma Ontologia com 10.000 componentes. A implementação de um mecanismo de *pooling* de threads, comumente utilizado em servidores preparados para suportar altas taxas de requisições simultâneas, provavelmente melhoraria ainda mais o desempenho obtido.

5. Trabalhos Relacionados

Dobson et al. (2005) definem um modelo para especificação de QoS para serviços Web semânticos chamado QoSOnt e uma solução para descoberta de serviços baseada em UDDI e XML. No entanto, o modelo proposto não provê meios para especificar um

perfil de QoS de um serviço Web com base em um conjunto de características de QoS e também não permite que uma especificação existente seja estendida ou reutilizada.

Zhou et al. (2004) propõem um modelo para especificação de parâmetros e perfis de QoS, chamado DAML-QoS, e um algoritmo de inferência baseado em lógica descritiva para encontrar serviços Web que satisfaçam os critérios de busca especificados pelo usuário. No entanto, não é possível estender perfis ou criar novos perfis através de composição. A principal limitação da DAML-QoS consiste no fato de ela utilizar restrições de cardinalidade entre um perfil e as suas métricas de QoS para definir os valores das características de QoS, devido a restrições no algoritmo de inferência, que as limita a valores inteiros não-negativos.

A ontologia OWL-Q [Kritikos e Plexousakis 2006] estende a OWL-S de modo a permitir a descrição de métricas e restrições de QoS, e provê um algoritmo de *matching* semântico para descoberta de serviços. Utilizando essa ontologia é possível definir especificações de QoS completas, incluindo a possibilidade de especificar regras de conversão de unidades e métricas compostas. Diferentemente da OWL-Q, a abordagem adotada nesse artigo é fundamentada na simples execução de uma consulta SPARQL, não exigindo o emprego de algoritmos semânticos complexos.

Vu et al. (2006) propõem um *framework* para descoberta de Serviços Web semânticos considerando suas características de QoS. Os autores definem um modelo completo para especificação e verificação de QoS, um mecanismo de classificação para indexar características de QoS, um algoritmo de descoberta de serviços e um ambiente distribuído para paralelizar a execução de consultas. O *framework* proposto é mais completo, e conseqüentemente mais complexo, que a proposta apresentada nesse artigo, o que o torna adequado para ambientes de grande escala. Por outro lado, a nossa abordagem é apropriada para ambientes de menor escala, pois requer uma menor infra-estrutura, o que a torna mais simples de implantar e manter.

Não foi possível encontrar descrições de testes de desempenho efetuadas sobre cada um dos mecanismos de busca proposto por estes autores, para que pudéssemos compará-los com nossos resultados e avaliar a eficiência do mecanismo de consultas com linguagem SPARQL em relação aos algoritmos propostos.

Sugumaran e Storey (2003), assim como Yao e Etzkorn (2004), também apresentaram modelos de mecanismos de busca semântica de componentes de software. No entanto, o foco principal de seus trabalhos é para encontrar componentes que atendam à determinada funcionalidade através de consultas em linguagem natural, enquanto nosso trabalho é focado na descoberta de componentes a partir de suas características não-funcionais. Entendemos que a união de trabalhos nestas áreas permitiria criar um mecanismo completo de busca e descoberta de Componentes e Serviços Web que considere a pesquisa por uma determinada funcionalidade em linguagem natural, em conjunto com a especificação das restrições de QoS a serem suportadas pelos Componentes encontrados.

6. Conclusões

Nesse artigo foi apresentado um mecanismo semântico de busca com base em requisitos de qualidade de serviço, que pode ser utilizado para a descoberta de Componentes de Software ou Serviços Web Semânticos. O mecanismo foi construído sobre uma API de busca que permite a identificação de características de QoS em uma ontologia e executa

buscas para localização de Componentes ou Serviços Web que satisfaçam determinados requisitos de QoS. A ontologia QoS-MO [Tondello e Siqueira 2008] é empregada para descrever as características e restrições de QoS.

A abordagem adotada neste trabalho é simples e de fácil implementação e manutenção. As buscas são executadas montando uma consulta SPARQL com base nos requisitos de QoS especificados pelo usuário e executando-a sobre a ontologia que descreve os componentes ou serviços Web disponíveis. A ontologia de descrição de QoS passa previamente por um processo de inferência para identificar as estruturas de especialização e composição de especificações.

Uma interface Web de consulta foi desenvolvida para demonstrar a viabilidade da abordagem proposta, e testes foram realizados com o intuito de avaliar o desempenho do mecanismo de busca. Os resultados dos testes mostram que a abordagem proposta é viável, e foi possível obter tempos de resposta aceitáveis mesmo para ontologias com milhares de Componentes e executando centenas de consultas simultaneamente.

Futuramente a ferramenta de busca deverá ser submetida a mais testes em ambientes de maior escala, nos quais seja necessário atender a uma grande quantidade de requisições em paralelo, com o intuito de melhor avaliar e otimizar o seu desempenho em situações de sobrecarga. Também seria interessante a realização de estudos de desempenho com a Ontologia de Componentes mantida em um Banco de Dados relacional ao invés de memória, recurso suportado pela API Jena – utilizada para construção do mecanismo de busca – para permitir a escalabilidade da solução proposta. Finalmente, a realização de estudos de desempenho das demais propostas de trabalhos relacionados e a comparação entre estes seriam interessantes para poder avaliar a eficiência de cada uma das idéias existentes na área.

Referências

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. e Patel-Schneider, P., editores. (2003) "The Description Logic Handbook: Theory, Implementation, and Applications". Cambridge University Press.

Berners-Lee, T., Hendler, J. e Lassila, O. (2001) "The Semantic Web". Scientific American, Maio.

Dobson, G., Lock, R. e Sommerville, I. (2005) "QoSOnt: a QoS Ontology for Service-Centric Systems". 31st Euromicro Conference on Software Engineering and Advanced Applications (Euromicro SEAA '05). Porto, Portugal.

Frølund, S., Koistinen, J. (1998) "QML: A Language for Quality of Service Specification". http://www.hpl.hp.com/techreports/98/HPL-98-10.html

Gruber, T. R. (1993) "A translation approach to portable ontologies". Knowledge Acquisition, 5(2):199-220. http://tomgruber.org/writing/ontolingua-kaj-1993.htm

Jena. (2008) "Jena – A Semantic Web Framework for Java". http://jena.sourceforge.net/

Kritikos, K., Plexousakis, D. (2006) "Semantic QoS Metric Matching". 4th European Conference on Web Services (ECOWS '06), Dezembro, pág.265-274.

Ludwig, H., Keller, A., Dan, A., King, R.-P., e Franck, R. (2003) "Web Service Level Agreement (WSLA) Language Specification". http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

Object Management Group. (2006) UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification, v1.0.

Ontoware. (2008) "Core Ontology of Software/Software Components/Services". http://cos.ontoware.org/

OWL-S Coalition. (2004) OWL-S 1.1 Release. http://www.daml.org/services/owl-s/1.1/

Siqueira, F. (2002) "Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL". 20º Simpósio Brasileiro de Redes de Computadores (SBRC'2002). Búzios - RJ, Brasil.

Sugumaran, V. e Storey, V. C. (2003) "A Semantic-Based Approach to Component Retrieval". The DATA BASE for Advances in Information Systems, Vol. 34, Nº 3, pág.8-24.

Tondello, G. F. e Siqueira, F. (2008) "The QoS-MO Ontology for Semantic QoS Modeling". Proceedings of the 23rd Annual ACM Symposium on Applied Computing, Fortaleza, Brasil.

Vu, L.-H., Hauswirth, M., Porto, F., e Aberer, K. (2006) "A Search Engine for QoS-enabled Discovery of Semantic Web Services". Special Issue of the International Journal on Business Process Integration and Management (IJBPIM), Vol. 1, Nº 4, pág.244–255.

W3C. (2007) SPARQL Query Language for RDF. Candidate Recommendation. http://www.w3.org/TR/rdf-sparql-query/

W3C. (2004) Web Ontology Language (OWL). http://www.w3.org/2004/OWL/

Yao, H. e Etzkorn, L. (2004) "Towards A Semantic-based Approach for Software Reusable Component Classification and Retrieval". Proceedings of the 42nd Annual ACM Southeast regional conference. Huntsville, Alabama, EUA, pág.110-115.

Zhou, C., Chia, L. e Lee, B. (2004) "DAML-QoS ontology for Web services". IEEE International Conference on Web Services (ICWS'04). San Diego, California, EUA, pág.472-479.

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