“Estratégias de Ranqueamento para Seleção Dinâmica de Serviços Baseadas em Atributos de Qualidade”

Por

David Júnio Mota Cavalcanti

Dissertação de Mestrado

Universidade Federal de Pernambuco
Centro de Informática
posgraduacao@cin.ufpe.br
http://www.cin.ufpe.br/~posgraduacao

Recife, 2014

Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciência da Computação

David Júnio Mota Cavalcanti

"Estratégias de Ranqueamento para Seleção Dinâmica de Serviços Baseadas em Atributos de Qualidade"

Trabalho apresentado ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Nelson Souto Rosa*

Recife, 2014

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da Silva, CRB4-1217

C376e Cavalcanti, David Júnio Mota.
Estratégias de ranqueamento para seleção dinâmica de serviços baseadas em atributos de qualidade / David Júnio Mota Cavalcanti. – Recife: O Autor, 2014.
93 f.: il., fig., tab.

Orientador: Nelson Souto Rosa.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2014.
Inclui referências.

1. Sistemas distribuídos. 2. Composição de serviços. I. Rosa, Nelson Souto (orientador). I. Título.

004.36 CDD (22. ed.) UFPE-MEI 2014-144

Dissertação de Mestrado apresentada por **David Júnio Mota Cavalcanti** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Estratégias de Ranqueamento para Seleção Dinâmica de Serviços Baseadas em Atributos de Qualidade**" orientada pelo **Prof. Nelson Souto Rosa** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____

Prof. Fernando Antônio Aires Lins
Departamento de Estatística e Informática/ UFRPE

_____

Prof. Nelson Souto Rosa
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 14 de agosto de 2014.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedicado à minha Família.

Agradecimentos

À minha Família, especialmente minha mãe Suzana, pelo imenso amor, carinho e dedicação demonstrados ao longo de toda a minha existência. Agradeço também pelo exemplo de vida e por me ensinar que a educação é a riqueza mais importante que um ser humano pode ter.

Ao professor Nelson Rosa, pela oportunidade de realizar este trabalho, pela paciência na orientação e pelo grande incentivo que tornou possível a conclusão deste trabalho.

A Fábio Souza, pela incansável ajuda, dedicação e, sobretudo, paciência acompanhando todo o desenvolvimento deste trabalho, me auxiliando sempre que necessário.

A Tarcisio Coutinho, Luca Bezerra e a Eduardo Virães, amigos especiais, que ajudaram, cada um a sua forma, a tornar este trabalho possível.

A todos os amigos, companheiros de trabalho e irmãos na amizade que fazem parte da minha vida.

Finalmente, a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Por que o homem quer subir a montanha mais alta do mundo? Ora, a resposta é a mais simples possível, porque ela está lá. Ou seja, o desafio é uma virtude inerente ao ser humano. Foi assim que me senti diante da proposta de realizar este trabalho. Como algo totalmente novo, abracei esse repto, com o doce sabor de aventura e de fascinação, diante às novas sendas abertas por essa jornada através das fronteiras do conhecimento e da habilidade humana.

—ANÔNIMO

Resumo

Computação Orientada a Serviços (COS) permite o desenvolvimento de aplicações como composições de entidades básicas chamadas de serviços. Estes serviços oferecem operações de negócios que são usadas como critérios primários nos algoritmos de seleção de serviços. Em cenários distribuídos, devido ao grande número de serviços que podem oferecer funcionalidades semelhantes (ou mesmo idênticas), tem sido amplamente aceito que o processo de seleção para um novo serviço deve também levar em conta requisitos não funcionais (atributos de qualidade), tais como desempenho, disponibilidade, segurança e assim por diante. Abordagens para a seleção de serviços existentes são geralmente estáticas, ou seja, adotam uma estratégia (algoritmo de ranqueamento) para classificar os serviços candidatos que nunca é alterada. No entanto, diferentes estratégias podem ser utilizadas para classificar os serviços com base em atributos de qualidade e não é trivial escolher uma que seja adequada a todos os cenários possíveis. Neste contexto, este trabalho propõe uma solução de seleção de serviços que suporta a definição de novas estratégias de seleção em tempo de execução. A fim de validar a solução de seleção proposta, foi realizada uma avaliação experimental que mostra a utilização de diferentes estratégias de seleção e seus impactos na escolha dos serviços.

Palavras-chave: Computação Orientada a Serviço. Seleção de Serviços. Atributos de Qualidade. Algoritmos de Ranqueamento.

Abstract

Service-Oriented Computing (SOC) allows the development of applications as compositions of basic entities called services. These services offer business functions, which are used as primary criteria in the service selection algorithms. In distributed scenarios, due to the large number of services having similar (or even identical) functionalities, it has been widely accepted that the selection process for a new service should also take into account non-functional requirements (QoS attributes), such as performance, availability, security and so on. Existing approaches for service selection are usually static, i.e., they adopt a strategy (ranking algorithm) to rank the candidate services and it is never changed. However, different strategies can be used to classify services based on these attributes and it is quite difficult to select one that seems adequate in every scenario. In this context, this work proposes a service selection solution that supports the definition of new selection strategies at runtime. In order to validate the proposed service selection solution, was perform an experimental evaluation that shows the utilization of different service selection strategies and their performance impacts and effectiveness in selecting services.

Keywords: Service Oriented Computing. Service Selection. Quality Attributes. Ranking Algorithms.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

SOC Service-Oriented Computing

XML eXtensible Markup Language

QoS Quality of Service

SOAP Simple Object Access Protocol

WSDL Web Service Description Language

URI Uniform Resource Identifier

SOA Service-Oriented Architecture

UDDI Universal Description, Discovery and Integration

SLA Service Level Agreement

OSGi Open Services Gateway Initiative

API Application Programming Interface

SAX Simple API for XML

QSS Quality-Aware Service Selector

Sumário

1 Introdução

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito."

—MARTIN LUTHER KING

O objetivo deste capítulo é apresentar este trabalho no contexto da área de seleção de serviços com base em atributos de qualidade. Inicialmente são apresentados o contexto e a motivação para este trabalho, destacando a importância da seleção de serviços levando em consideração atributos de qualidade. Em seguida, são apresentados os principais desafios envolvidos e abordados no desenvolvimento do trabalho. Na sequência, o objetivo deste trabalho é apresentado. Por fim, é apresentada a estrutura desta dissertação.

1.1 Contexto e Motivação

Uma das grandes tendências atuais na academia e na indústria é o desenvolvimento de aplicações de software evolutivas, distribuídas e dinâmicas. A motivação para a construção de tais aplicações é a necessidade de softwares que possam ser modificados em tempo de execução, de forma rápida, e com baixo custo.

Neste contexto, aplicações evolutivas são aquelas que possuem a habilidade de absorver novos componentes de software, bem como, componentes podem deixar de existir ou ser modificados, para acrescentar novas funcionalidades, corrigir erros e assim por diante. A característica evolutiva está muito ligada aos sistemas abertos (*Open Systems*) e ao reuso de software. Sistemas abertos são definidos como sistemas onde novos componentes podem dinamicamente passar a compor a aplicação. Já o reuso de

software visa utilizar componentes de software já prontos para compor uma aplicação através de interfaces ou contratos bem definidos que devem ser respeitados entre quem os desenvolveu e quem está os usando.

A característica distribuída também está relacionada aos sistemas abertos, no entanto, leva-se em consideração que as aplicações podem usar componentes que estão localizados em máquinas diferentes, e que estes podem ser acessados através da rede.

Finalmente, a característica dinâmica significa que as aplicações possuem a habilidade de ser (re) configuradas em tempo de execução sem a necessidade de serem paradas e/ou recompiladas. Por exemplo, caso algum requisito de uma aplicação seja alterado, esta é capaz de alterar alguma biblioteca ou componente e a aplicação passa a atender a este novo requisito.

O suporte necessário ao desenvolvimento destas aplicações geralmente requer um paradigma de programação que suporte tais características, como a Computação Orientada a Serviços (COS), ou do inglês *Service-Oriented Computing* (SOC) [1][2][3].

Aplicações desenvolvidas no COS são baseadas na noção de serviços. Estes serviços são essencialmente modulares e dirigidos através de contratos (por exemplo, interface Java ou um arquivo *XML*), onde qualquer relacionamento entre os componentes que fazem parte de uma aplicação são descritos, tais como as suas dependências e seu nível de qualidade. Isto facilita a construção, manutenção e evolução das aplicações.

Além disso, aplicações orientadas a serviços são por natureza distribuídas, dinâmicas e fazem uso de um registro onde os serviços podem ser descobertos, selecionados e vinculados aos consumidores através de uma interface bem definida. Por isso, estas aplicações conseguem se adaptar às mudanças de requisitos com facilidade. Cada serviço assume a responsabilidade de uma funcionalidade específica e pode ser combinado com outros serviços para compor aplicações mais robustas, a partir de uma composição de serviços. Assim, é possível que as aplicações sejam construídas (ou quando necessárias modificadas) em tempo de execução, de forma rápida, com pouco esforço e baixo custo, a partir de serviços fracamente acoplados e flexíveis [1][2][4].

No COS, as aplicações tipicamente fornecem serviços (provedores) ou consomem serviços (consumidores). Além disto, provedores normalmente resgistram seus serviços em locais (registros) onde eles podem ser encontrados pelos consumidores. Estes resgistros contém as informações necessárias para os consumidores acessarem os serviços desejados.

Neste contexto, quando se deseja utilizar um serviço ou substituir os serviços pertencentes a uma composição, um desafio é como selecionar o serviço adequado a ser

utilizado.

Os consumidores acessam o registro buscando serviços que implementem uma determinada funcionalidade. No registro, é realizada uma filtragem para determinar quais serviços atendem à necessidade do consumidor [5]. O registro responde à requisição fornecendo o identificador do provedor de serviços que implementa a interface desejada. Finalmente, o consumidor invoca o serviço através do identificador fornecido pelo registro.

No entanto, com o crescente número de serviços disponíveis, os consumidores são forçados a escolher entre vários serviços funcionalmente similares (ou mesmo idênticos), por isso, a seleção tornou-se ainda mais complexa. Por exemplo, o seekda [1] oferece mais de trinta mil serviços que podem ser buscados de acordo com a funcionalidade desejada. Por esta razão, faz-se necessário fornecer aos consumidores outros critérios para selecionar o serviço desejado, como os requisitos não funcionais, geralmente referidos como atributos de qualidade de serviço, ou do inglês *Quality of Service* (QoS) [6][7][8][9].

Neste contexto, a seleção de um serviço adequado não é uma tarefa fácil, especialmente se diferentes critérios forem utilizados. Isto porque, o uso de critérios relacionados à funcionalidade é diferente daqueles associados aos atributos de qualidade. Por exemplo, a seleção usando critérios funcionais geralmente requer um comparativo sintático e semântico entre a funcionalidade requerida e a interface do serviço provido. Já na seleção considerando atributos de qualidade, além da funcionalidade dos serviços, os atributos de qualidade destes também devem ser analisados. No entanto, a natureza dinâmica dos serviços reflete diretamente na qualidade provida por estes, modificando frequentemente os valores dos atributos de qualidade. Então, a elaboração de um procedimento para selecionar os serviços considerando os atributos de qualidade dos serviços, além da funcionalidade, deve considerar alguns outros aspectos fundamentais:

1. Como é feita a especificação dos requisitos funcionais e atributos de qualidade do serviço requeridos pelo consumidor e oferecidos pelo provedor, através da requisição de serviço (*Service Request*) e do descritor de serviço (*Service Descriptor*);

2. Como é a avaliação dos atributos de qualidade dos serviços oferecidos e;

3. A agregação dos resultados da avaliação e a seleção adequada dos serviços com base nesses resultados.

Neste trabalho, todas estas questões são abordadas, no entanto, a ênfase consiste na agregação e seleção adequada dos serviços. De fato, selecionar o serviço adequado

[1] www.seekda.com

com base em seus atributos de qualidade é um aspecto a ser melhor explorado, estando relacionado com a descoberta e o ranqueamento dos serviços com base nestes atributos de qualidade [6][8]. Portanto, este constitui o foco principal do trabalho.

1.2 Caracterização do Problema

A seleção de serviços concentra-se em obter os serviços adequados à aplicação de modo que estes satisfaçam os requisitos funcionais e de qualidade exigidos pelo consumidor. Como foi dito anteriormente, em geral, os serviços são selecionados com base na sua funcionalidade, e muitas vezes, ainda de forma manual. No entanto, quando a qualidade do serviço está envolvida na seleção, geralmente, estes são ranqueados considerando seus atributos de qualidade e, em seguida, um serviço (ou mais) é selecionado com base neste *ranking*. Nestas situações em que a qualidade é usada, normalmente a seleção tem um algoritmo de ranqueamento definido estaticamente, de tal forma que, quando ele é definido, não é alterado em tempo de execução.

Porém, diversos são os algoritmos para determinar como o serviço deve ser selecionado. Cada algoritmo geralmente adota uma estratégia de ranqueamento específica para determinar o serviço mais adequado para o consumidor. Portanto, a utilização de um único algoritmo pode não ser sempre interessante. De fato, enquanto alguns algoritmos fazem distinção entre atributos de qualidade obrigatórios e opcionais a serem considerados no ranqueamento, outros permitem ao consumidor expressar suas preferências por alguns atributos de qualidade. Outros permitem aos consumidores especificar pesos para classificar os serviços candidatos, enquanto em alguns o consumidor pode utilizar valores históricos de qualidade de serviços, observados antes da seleção. Nestes casos, algoritmos de ranqueamento que envolvem dados a partir do consumidor do serviço devem ser utilizados [9].

Além disso, segundo Yu [9], alguns algoritmos usam estratégias que adotam métodos aritméticos e geométricos que podem ser eficientes e compreensíveis em algumas situações. No entanto, não são as melhores escolhas para situações mais complexas onde tem-se muitos atributos de qualidade. Porque, a quantidade de atributos pode influenciar no desempenho das estratégias.

Finalmente, o comportamento e o desempenho da estratégia de ranqueamento podem ser considerados na seleção: o consumidor pode requerer o serviço mais próximo à sua preferência ou o melhor serviço dentre os serviços candidatos, neste caso, alguns algoritmos são mais eficazes. Da mesma forma, os consumidores querem que a seleção

seja a mais rápida possível dado um número de atributos de qualidade.

Assim, a possibilidade de escolher o algoritmo de ranqueamento, e, eventualmente, substituí-lo para melhor atender as necessidades da aplicação é certamente uma característica interessante. Neste contexto, este trabalho concentra-se em mudar dinamicamente o algoritmo de ranqueamento para determinar o serviço mais adequado para o consumidor.

Embora a questão da pesquisa deste trabalho já tenha sido definida, é importante mencionar que outros aspectos relevantes dentro da classificação e seleção dos serviços são considerados:

- A especificação de qualidade oferecida e requerida pelos provedores e consumidores de serviços, respectivamente;
- A obtenção de atributos de qualidade utilizados no ranqueamento: oferecidos pelos provedores, valores históricos, dentre outros e;
- A utilização de diferentes algoritmos de ranqueamento.

Desta forma, pode-se abordar o problema das diversas situações encontradas na seleção de serviços usando vários algoritmos de ranqueamento, bem como, usando valores de qualidade diferenciados, levando em consideração as exigências do consumidor (aplicação).

1.3 Objetivos e Contribuições

O principal objetivo deste trabalho é desenvolver uma solução extensível, dinâmica e automática para seleção de serviços, que leve em consideração não só a funcionalidade dos serviços, mas também os atributos de qualidade dos mesmos.

Em particular, a solução deve ser capaz de mudar dinamicamente o algoritmo de ranqueamento de serviços, ou seja, sem a necessidade de interromper sua execução para que o melhor algoritmo possa ser escolhido.

As características, extensível, dinâmica e automática são obrigatórias nesta solução.

A característica extensível é justificada através da possibilidade de inclusão de novos algoritmos de ranqueamento de serviços e métodos de obtenção e certificação dos atributos de qualidade dos serviços. É importante ressaltar que a dinamicidade da solução elimina a necessidade de interrupções para assimilação de novos algoritmos de ranqueamento.

A característica dinâmica significa que a solução pode ser ativada e desativada a qualquer momento, bem como podem ser substituídos os algoritmos de ranqueamento em tempo de execução, sem afetar outros serviços.

Por automática, a solução não requer qualquer intervenção humana na seleção de serviço. É uma prática comum que a seleção de serviços seja feita de forma manual, um ser humano procura por serviços apropriados em um registro e toma decisões quanto a qual serviço escolher [9]. Nesta solução, ainda que o consumidor especifique os requisitos funcionais e os atributos de qualidade que o serviço deve atender, todo o processo de descoberta, análise e seleção do serviço é automática.

No entanto, para atingir o objetivo principal deste trabalho, outros objetivos secundários precisam ser alcançados:

- Identificar os requisitos das soluções de seleção de serviços, com o intuito de aplicá-los à solução proposta;
- Implementar diferentes algoritmos para serem utilizados na solução proposta e;
- Avaliar a solução proposta considerando a sua eficácia e o seu desempenho.

1.4 Organização do Trabalho

Visando atingir o objetivo proposto na Seção 1.3, esta dissertação foi organizada em capítulos, que se apresentam da seguinte forma:

- **Capítulo 2**: neste capítulo serão introduzidos os principais conceitos necessários ao entendimento do trabalho: Computação Orientada a Serviços, Qualidade de Serviço, OSGi, Algoritmos de Ranquemento e a Plataforma DSOA. Além disso, este capítulo também faz uma análise do estado da arte na área de seleção de serviços.
- **Capítulo 3**: este capítulo mostra com detalhes a solução de seleção de serviços proposta. São apresentadas as características da solução, sua arquitetura e seus componentes. Além disso, os detalhes da implementação e do funcionamento também são descritos.
- **Capítulo 4**: este capítulo apresenta a avaliação experimental da solução proposta.
- **Capítulo 5**: este capítulo conclui o trabalho, apresentando as principais contribuições à área de seleção de serviços. Além disso, são apresentadas algumas limitações da solução e possíveis trabalhos futuros.

2

Conceitos Básicos e Trabalhos Relacionados

"Se vi mais longe, foi por estar de pé sobre ombros de gigantes."

—ISAAC NEWTON

Este capítulo apresenta os conceitos básicos necessários ao entendimento da solução proposta e alguns dos principais trabalhos relacionados à área de seleção de serviços baseada em atributos de qualidade.

2.1 Conceitos Básicos

Nesta seção são apresentados os conceitos básicos. Inicialmente é apresentado o paradigma de orientação a serviços, que corresponde ao conceito mais básico utilizado para o desenvolvimento da solução. Posteriormente é apresentado o conceito de qualidade de serviços, bem como a plataforma *OSGi*, que é a tecnologia que forma o núcleo para a implementação da solução. O conceito algoritmos de ranqueamento, fundamental para o entendimento do trabalho também é apresentado. Por fim, é apresentada a plataforma *DSOA*, que representa a base onde este trabalho está inserido.

2.1.1 Computação Orientada a Serviços

Este trabalho está inserido na Computação Orientada a Serviços (COS), portanto é importante o entendimento deste conceito. COS consiste em um paradigma de programação

que propõe o desenvolvimento de aplicações distribuídas de forma rápida e dinâmica através de serviços reutilizáveis e fracamente acoplados [1][2][3][4][5][10][11][12].

Na literatura, o conceito de serviço é definido de várias formas [1][2][3][4][5][10][11][12]. Com base nessas definições, serviço pode ser conceituado como entidades computacionais autônomas que fornecem uma funcionalidade específica através de interfaces bem definidas para serem incorporadas dinamicamente à uma aplicação.

Atualmente, o exemplo mais promissor de tecnologia baseada em COS são os *Web services*, que consistem em serviços disponibilizados através da Internet que utilizam o protocolo *SOAP* (*Simple Object Access Protocol*) para a transmissão de dados e *WSDL* (*Web Services Description Language*) para descrição dos serviços. Além disso, *Web services* são identificados através de uma *URI* (*Uniform Resource Identifier*) para serem descobertos e utilizados.

Com base nas definições [1][2][3][4][5][10][11][12] foi possível identificar algumas características relevantes ligadas aos serviços:

- **Baixo Acoplamento**: o baixo acoplamento representa também um dos sucessos da abordagem de orientação a serviços. Ele fornece ao serviço um maior grau de flexibilidade, uma vez que este pode evoluir de forma independente. Este baixo acoplamento é caracterizado através da ligação entre o provedor e o consumidor de serviços. A ligação é realizada apenas em tempo de execução, ou seja, o consumidor apenas especifica o serviço que deseja utilizar e o serviço é ligado somente no momento da execução. Em geral, a aplicação não faz referência direta ao serviço e sim a uma interface para o serviço. Desta forma, consumidores e provedores podem ser desenvolvidos de forma independente;

- **Reusabilidade**: a reusabilidade é naturalmente um dos principais objetivos da orientação a serviços. Essencialmente, esta característica é a capacidade de reutilizar um mesmo serviço para executar uma determinada função sempre que esta for necessária e que um mesmo serviço possa ser utilizado várias vezes para construir diferentes aplicações;

- **Adaptabilidade**: adaptabilidade é outra característica marcante no paradigma de orientação a serviços. Através da troca de serviços, aplicações conseguem se adaptar as mudanças de requisitos com facilidade. Os serviços podem ser adicionados ou removidos das aplicações, em tempo de execução, sempre que necessário.

- **Composição**: a composição é a característica mais significativa do paradigma de orientação a serviços. Ela representa a possibilidade de construir aplicações através da junção de vários serviços para realizar determinada funcionalidade. O objetivo da composição é compor e recompor aplicações utilizando serviços já implementados e testados. Muitas vezes essa composição é em tempo de execução, utilizando novos serviços e invocando métodos específicos de uma interface existente.

Além das características apresentadas, para construir aplicações orientadas a serviço, COS conta com uma Arquitetura Orientada a Serviços, chamada *SOA* (*Service Oriented Architecture*) [1][4]. *SOA* é uma maneira de organizar as aplicações (arquitetura) e sua infraestrutura em um conjunto de serviços que interagem entre si.

Arquitetura Orientada a Serviços

SOA constitui uma concepção lógica de arquitetura de software que permite representar as funcionalidades de uma aplicação de software como um serviço através da rede. Na prática, *SOA* define um modelo de interação através de trocas de mensagens entre provedores e consumidores de serviços que permite a publicação, descoberta, seleção, utilização e combinação de serviços interoperáveis, de forma dinâmica.

O modelo básico de *SOA* consiste em uma relação entre três elementos fundamentais: o provedor (*Service Provider*), o consumidor (*Service Requestor*) e o registro de serviços (*Service Registry*). Através de um processo conhecido como estratégia "*Find-Bind-Execute*", apresentado na Figura 2.1, estes três elementos interagem entre si para construir as aplicações distribuídas.

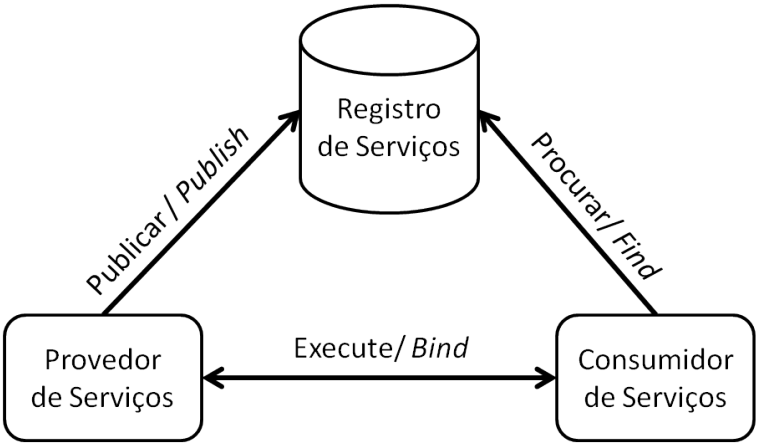

Figura 2.1: Estratégia *"Find-Bind-Execute"*.

De acordo com este paradigma, as aplicações que proveem serviços devem publicá-los em um registro, onde as aplicações baseadas em serviços, atuando como consumidores, podem dinamicamente encontrá-los e utilizá-los. Deste modo, o consumidor pode incorporar serviços à sua aplicação, ou ainda substituir o serviço consumido, de acordo com suas necessidades, por exemplo, em função de mudanças nos seus requisitos funcionais.

Para publicar um componente de software como um serviço o provedor inicialmente deve definir contratualmente este serviço através de um descritor (*Service Descriptor*) e utilizar uma operação de publicação (*Publish*) em um registro de serviços. Desta forma o serviço fica acessível para ser descoberto.

As descrições dos serviços contêm algumas combinações de informações que descrevem as capacidades do serviço, sua interface e eventuais propriedades, por exemplo, propriedades relacionadas à qualidade ou localização daquele serviço.

A publicação de tais informações fornece os meios necessários para a descoberta, seleção e composição de serviços. Em particular, a interface do serviço informa a assinatura do serviço, enquanto que a descrição da capacidade define o propósito e os resultados esperados para aquele serviço. Finalmente, a descrição da qualidade do serviço que informa os atributos de qualidade que aquele serviço possui, tais como o custo do serviço, as métricas de desempenho (tempo de resposta, disponibilidade), atributos de segurança (integridade, escalabilidade), assim por diante.

Por outro lado, para usar um serviço o consumidor acessa o registro, ou repositório de serviços (por exemplo, *UDDI*) e realiza uma operação para descobrir (*Find*) algum serviço com base nesse *Service Descriptor*. Caso haja algum provedor que atenda as necessidades do consumidor, então o registro enviará uma referência ao consumidor do provedor que possui aquele serviço. Em seguida, o consumidor realiza uma operação de ligação com o provedor (*Bind*). Este último, por sua vez disponibiliza uma implementação do serviço e então o consumidor pode utilizar tal serviço. Estas duas operações juntas formam uma etapa importante em COS, que é o processo de seleção do serviço.

Uma ressalva importante é que as funções de provedor e consumidor de serviços são construções lógicas e uma aplicação pode ter características de ambos, ou seja, é possível que um provedor de serviços possa também requerer serviços, caracterizando uma composição.

2.1.2 Qualidade de Serviços

Qualidade de serviço (*QoS*) também é um conceito importante para o entendimento deste trabalho. Na literatura existem várias definições de *QoS* [13][14][15][16]. Em particular

é assumida a definição proposta em Menascé [17], onde *QoS* consiste de uma combinação de várias qualidades ou propriedades do serviço, tais como, disponibilidade, tempo de resposta, precisão e vazão.

Enquanto os requisitos funcionais definem o que o serviço deverá fazer, os requisitos não funcionais, em particular, a *QoS*, especifica o nível de qualidade dos serviços, informando, por exemplo, o tempo que um serviço é necessário para responder a várias requisições (tempo de resposta) e a porcentagem de tempo que um serviço está funcionando (disponibilidade). Porém, *QoS* têm uma natureza muito distinta, compreendendo uma grande variedade de atributos de qualidade, tais como os ligados a desempenho (tempo de resposta, disponibilidade), custo e tolerância a falhas (dependabilidade, confiabilidade).

Considerando-se a diversidade dos atributos de qualidade, diferentes esquemas foram propostos para agrupá-los e classificá-los [15][16]. Uma classificação natural que pode ser facilmente considerada é agrupar os atributos de qualidade em duas categorias. A primeira compreende os atributos determinísticos que são aqueles cujos valores são conhecidos antes da execução da aplicação, tais como custo. A segunda refere-se aos atributos não determinísticos que engloba aqueles cujos valores são observados em tempo de execução, tais como tempo de resposta.

Em aplicações de software orientadas a serviços, os atributos de qualidade, principalmente os não determinísticos, representam uma questão importante (a qualidade do serviço) e, muitas vezes, são considerados na seleção e composição de serviços, devido principalmente a quantidade de serviços disponíveis com a mesma funcionalidade.

Devido às características da COS, provedores e consumidores de serviços geralmente pertencem a diferentes domínios e a rede tem um forte impacto na segurança e no desempenho do serviço. Logo, consumidores costumam mudar o provedor de serviços quando a qualidade esperada não é satisfeita [15]. Por isso, atributos ligados ao desempenho são os mais comuns aos trabalhos que relacionam serviços e *QoS*, principalmente em trabalhos ligados a descoberta e seleção de serviços. Eles consistem em um conjunto de atributos não determinísticos e quantitativos que descrevem o desempenho e a capacidade de resposta dos serviços em tempo de execução na visão do consumidor.

A seguir são descritos os principais atributos de qualidade ligados ao desempenho de serviços. Em particular, serão apresentados os atributos considerados nos experimentos realizados no Capítulo 4.

- **Tempo de resposta**: o tempo de resposta de um serviço ou de uma operação do serviço refere-se à duração do tempo para enviar uma mensagem a partir de um determinado consumidor até o recebimento da mensagem de resposta por este

consumidor. Por isso, o tempo de resposta é um atributo de qualidade específico para o consumidor e deve ser calculado individualmente.

- **Vazão**: a vazão é o número de solicitações que um provedor de serviços pode processar com sucesso e retornar ao consumidor em um determinado período de tempo.
- **Disponibilidade**: a disponibilidade representa a probabilidade do serviço estar disponível, funcionando e produzindo resultados corretos. Ou seja, ela refere-se à razão entre o tempo que o serviço está em atividade e o tempo de inatividade deste serviço.
- **Precisão**: a precisão refere-se à taxa de sucesso de serviço. Ela é calculada através da razão entre todas as invocações ao serviço e as invocações que falharam durante intervalo de tempo.
- **Preço**: o preço de um serviço é o valor monetário que o consumidor paga para invocar o serviço e, normalmente, é calculado por invocação.

É importante salientar que existem outros atributos de qualidade e que a lista apresentada inclui apenas os atributos de qualidade usualmente associados a serviços.

2.1.3 OSGi

OSGi (*Open Services Gateway Initiative*) constitui o núcleo para a construção da solução proposta e consiste em um conjunto de especificações que fornece o suporte para o desenvolvimento de aplicações dinâmicas a partir da composição de módulos de software colaborativos e reutilizáveis baseados em Java. No ambiente *OSGi*, estes módulos encapsulam aplicações que são utilizadas como serviços [18][19][20].

O principal componente da especificação *OSGi* é o *framework OSGi*. O *framework* é responsável por prover um ambiente de execução que suporta a implementação de aplicações extensíveis através de módulos configuráveis, que proveem e requerem serviços. Além disso, ele é responsável por manter a consistência dos serviços, ao controlar a relação de dependências entre os módulos instalados, pelo gerenciamento de módulos, permitindo a instalação, atualização e desinstalação destes módulos em tempo execução.

Para realizar todo o gerenciamento das aplicações a arquitetura do *framework OSGi* é composta por um conjunto de camadas como mostrado na Figura 2.2.

Estas camadas realizam as seguintes funções:

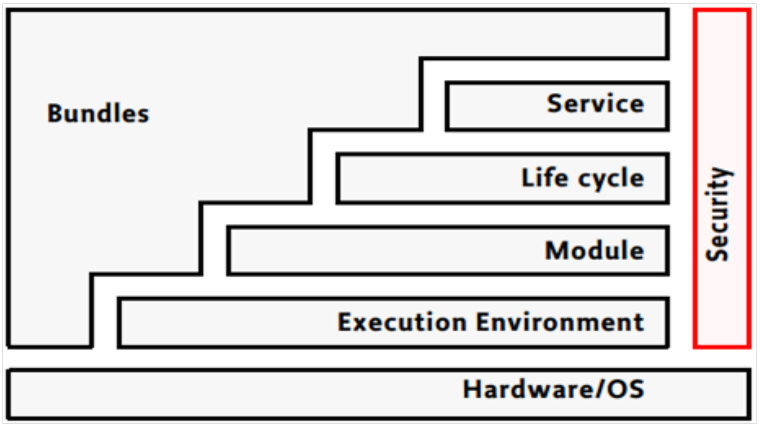

Figura 2.2: Arquitetura da plataforma *OSGi* [20]

- ***Execution Environment***: a camada *Execution Environment* especifica o ambiente de execução Java (por exemplo, Java EE ou Java SE) onde os módulos serão executados. Ela define quais os métodos e classes estão disponíveis em uma plataforma específica.

- ***Bundles***: os *Bundles* são componentes *OSGi* implementados pelos desenvolvedores e encapsulam aplicações que podem requerer ou prover serviços para outros *bundles*. Um *bundle* é um arquivo JAR com um conjunto de metadados definidos em um arquivo *Manifest*. Ele define um módulo lógico que pode ser combinado para a composição de uma aplicação [18]. Além disso, *bundles* podem declarar explicitamente dependências externas e pacotes exportados, ampliando o mecanismo de controle de acesso padrão de Java. Essa capacidade de declarar explicitamente pacotes importados e exportados traz um grande benefício ao desenvolvedor, uma vez que o próprio *framework* é responsável por gerenciar e verificar automaticamente a consistência da aplicação através de um processo chamado *bundle resolution* [18][21].

- ***Module***: esta camada define o conceitos de módulos configuráveis, providos através dos *Bundles*. Os metadados que ativam o *framework OSGi* para executar isto são configurados em um arquivo manifesto, que são processados por esta camada, e as dependências externas declaradas são supridas com as exportações por outros módulos. Assim, os módulos podem ser carregados e compartilhados entre *Bundles*, bem como, se tornarem invisíveis para outros determinados *Bundles*.

- ***Life cycle***: esta camada define todo o ciclo de vida dos *bundles* no *framework OSGi*. Ela é responsável pelo gerenciamento dinâmico dos *bundles*, determinando

em tempo de execução quando estes poderão ser iniciados ou parados, bem como, quando estes deverão ser instanciados, atualizados e desinstalados. Para isso, ela define uma *API* que permite instalar, iniciar, parar, atualizar e desinstalar os *bundles*. Além disso, ela também define toda interação entre o *bundle* e o *framework OSGi*.

Cada *bundle* tem um ativador provido através de uma interface chamada *BundleActivator*, que realiza uma função semelhante ao *main* de um programa Java. Esta interface é responsável pela inicialização e parada dos *bundles*.

- ***Service***: a camada de serviços incorpora características de *SOA* no *OSGi* incluindo um registro, provedor e consumidor de serviço (ver Seção 2.1.1). Ela expande o dinamismo provido pelos *bundles* na camada de ciclo de vida, combinando-o ao dinamismo provido por arquiteturas orientadas a serviço, onde serviços podem aparecer ou desaparecer a qualquer momento [18].

Como dito anteriormente, juntas, estas camadas formam o *framework OSGi*, que provê o ambiente de execução do *OSGi*, onde as aplicações podem fornecer e consumir serviços a partir do registro de serviços local ou remoto. Desta forma, é possível construir aplicações através de módulos que podem ser instalados, iniciados, parados, atualizados ou desinstalados em tempo de execução sem a necessidade de parar a aplicação.

2.1.4 Algoritmos de Ranqueamento

Outro conceito que deve ser observado para o entendimento deste trabalho é o conceito de algoritmos de ranqueamento.

A descoberta e a seleção de serviços são duas atividades relacionadas. A descoberta compreende o processo de encontrar os serviços que atendam às exigências requeridas pelos consumidores. Enquanto isso, a seleção é o processo que trata da escolha de uma instância do serviço que tenha sido previamente descoberta. Logo, a descoberta de serviços precede o processo de seleção [14][22].

Neste contexto, algoritmos de ranqueamento são utilizados para determinar qual serviço deve ser selecionado quando um conjunto de serviços que satisfaz os requisitos do consumidor é encontrado.

Em algumas situações, utiliza-se uma comparação sintática entre os requisitos, tanto funcionais quanto de atributos de qualidade, requeridos pelos consumidores com as descrições dos serviços fornecidas pelos provedores. Nesta situação, o consumidor acessa o registro de serviços, busca os serviços que satisfazem os seus requisitos e, com base na

similaridade sintática entre seus requisitos e seus atributos de qualidade com a descrição do serviço fornecida pelo provedor no registro, o serviço é escolhido.

No entanto, a seleção com base nesta similaridade sintática não é sempre a mais adequada para uma seleção eficiente, ou seja, considerar a similaridade da comparação entre os valores requeridos pelo consumidor com os declarados pelo provedor não quer dizer que o serviço escolhido é a melhor opção para o consumidor. Se por um lado, nem sempre é verdadeiro o fato de uma alta similaridade representar o serviço mais adequado, por exemplo, um consumidor pode requerer um serviço que tenha um tempo de resposta de 15 *ms* e o registro contenha dois serviços, um com 15 *ms* e outro com 10 *ms* de tempo de resposta, neste caso, o serviço com 15 *ms* seria escolhido por conta da similaridade com o requisitado pelo consumidor, no entanto, o serviço mais adequado seria o com 10 *ms* já que tem o menor tempo de resposta . Por outro lado, o uso apenas dos valores informados pelo provedor considera a confiança e a veracidade entre os valores reais e os firmados nos contratos, no entanto, os valores dos provedores podem não estar em conformidade com a situação real do serviço, que pode apresentar valores diferentes de atributos de qualidade devido a alguma incosistência no ambiente de execução. Além disso, é difícil determinar qual serviço deve ser selecionado quando mais de um serviço possui semelhanças sintáticas com os requisitos do consumidor.

Por isso, outras estratégias devem ser utilizadas para determinar qual serviço deve ser selecionado. Diante desse contexto, diversos algoritmos de ranqueamento têm sido propostos na literatura [7].

Estes algoritmos [6][23][24] determinam o ranqueamento do serviço através de uma métrica quantitativa que revela a importância do serviço dentro do processo de seleção. Ou seja, após encontrar os serviços que satisfazem funcionalmente os consumidores e quando todos os atributos de qualidade desejados forem avaliados, o próximo passo importante é agregar os valores desses atributos de qualidade individuais de cada serviço e obter uma pontuação final para eles [25].

Para determinar esta pontuação, os algoritmos utilizam estratégias de agregação aritméticas, geométricas e estatísticas. Através dessas estratégias, os valores dos atributos de qualidade são agrupados e os serviços são ordenados com base em cálculos realizados. Portanto, inicialmente, os serviços estão em um registro para serem descobertos a partir de sua funcionalidade. Após a descoberta desses serviços seus atributos de qualidade são analisados e então um algoritmo é utilizado para ranqueá-los.

Como mencionado na Seção 1.2 diversas situações podem acontecer na classificação dos serviços e o algoritmo de ranqueamento utilizado pode não ser a melhor opção dada

uma situação complexa. Por exemplo, o consumidor do serviço considera tanto o tempo de resposta do serviço quanto o preço como atributos de qualidade que um serviço deve atender. Neste caso, não podem ser utilizadas funções de agregação que somem os atributos, mas um produto resolveria tal problema. Portanto, é importante que algoritmos possam ser escolhidos e substituídos dinamicamente para uma escolha mais efetiva dos serviços [9].

2.1.5 Plataforma DSOA

O último conceito, mas não menos importante para o entendimento do trabalho proposto é a plataforma *DSOA* (*Dynamic SOA*). Na prática, a solução proposta neste trabalho está inserida na plataforma *DSOA*.

DSOA é uma extensão de *SOA* com suporte à adaptação dinâmica de aplicações baseada no uso de atributos de qualidade. *DSOA* é uma plataforma orientada a componentes, e baseada em serviços, que permite o desenvolvimento de aplicações através de composição de serviços dinamicamente adaptativos com base em atributos de qualidade.

Além disso, outra característica importante de *DSOA* é que ela baseia-se em conceitos de computação autonômica e usa composição dinâmica com loop adaptativo, injeção de dependência e processamento de eventos [26][27][28][29].

A adaptação é realizada através do monitoramento de atributos de qualidade dos serviços em tempo de execução. Quando um determinado serviço não atende a qualidade anunciada, os serviços são trocados sem a necessidade da composição ser parada.

Outra característica da *DSOA* é que ela foi desenvolvida a partir de uma abordagem orientada a modelos. Diferente das abordagens de desenvolvimento convencionais (não orientadas a modelos) que usam a modelagem de software apenas como "guia", para tarefas de desenvolvimento e manutenção, e obrigam o desenvolvedor a interagir manualmente com todo o código-fonte. Na abordagem orientada a modelos, o desenvolvedor produz modelos no intuito de descrever o conceito dos domínios da aplicação e, a partir dos modelos, através de um mecanismo automático, são gerados os códigos-fontes. Neste cenário, modelos também fazem parte do software, assim como o código-fonte.

Neste contexto, a plataforma apresenta uma coleção de modelos independentes que compreendem os diferentes domínios relacionados à concepção de aplicações baseada em serviços ciente de qualidade. Os modelos definidos em *DSOA* incluem a modelagem de: serviços, qualidade, monitoramento e eventos. Estes modelos são interpretados pelos componentes internos da plataforma, a fim de configurar dinamicamente a própria plataforma e as aplicações que ele suporta.

As Figuras 2.3 e 2.4 apresentam os modelos de qualidade e serviços propostos pela plataforma *DSOA*. Os modelos apresentados são os utilizados pela solução de seleção de serviços proposta por este trabalho.

O modelo de qualidade é responsável pelo suporte à solução para expressar os conceitos ligados à qualidade, ou seja, fornecer suporte ao consumidor para especificar os atributos de qualidade que ele deseja, bem como permitir aos provedores de serviços expressarem atributos de qualidade que suportam. Enquanto que o modelo de serviço permite a solução a expressar os conceitos ligados a serviços, por exemplo, como são feitas as requisições e como são ofertados os serviços.

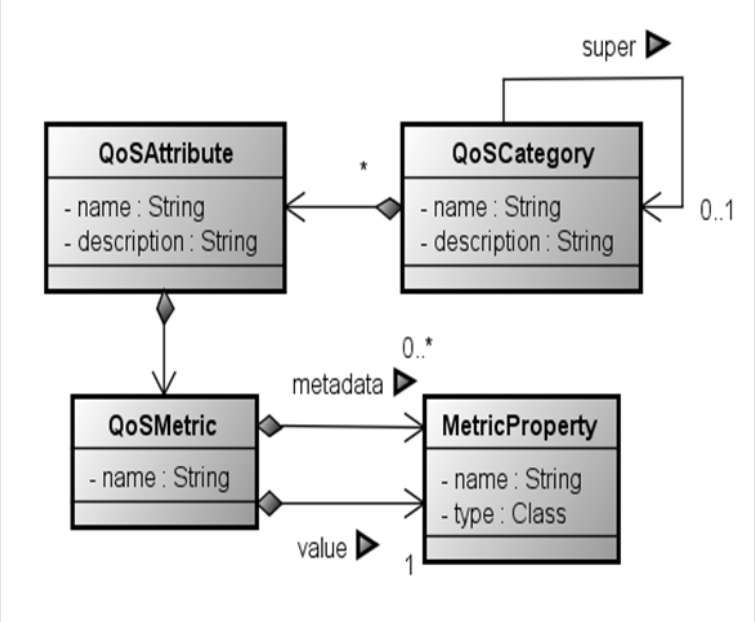

Figura 2.3: Modelo de Qualidade

Através deste modelo de qualidade é possível o uso de várias métricas de qualidade. Como pode ser observado, os atributos de qualidade (*QoSAttribute*) são agrupados em categorias de qualidade (*QoSCategory*), que são organizados em uma estrutura de árvore. Cada atributo de qualidade é caracterizado por um conjunto de métricas de qualidade (*QoSMetric*), que são definidas em termos de um conjunto de metadados e de dados. Os metadados servem para caracterizar as circunstâncias em que os dados foram coletados (por exemplo, *timestamp*), enquanto os dados são os valores calculados para as métricas de qualidade, por exemplo, tempo de resposta.

Além disso, os atributos de qualidade podem ser avaliados através de diferentes métricas, cada uma resumindo um ponto de vista diferente, por exemplo, valores médios e mínimos (ver Figura 2.3).

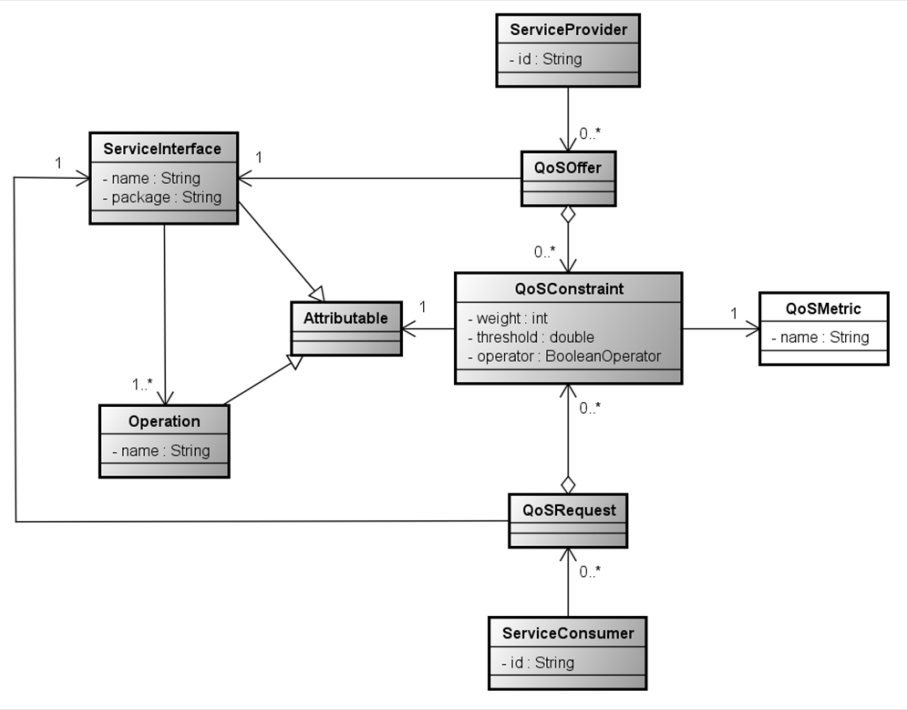

Figura 2.4: Modelo de Serviços

O modelo de serviço (ver Figura 2.4) quando associado com o modelo de qualidade permite associar os dois domínios e possibilita o uso de atributos de qualidade na seleção dos serviços. Neste contexto, a fim de anunciar um serviço um provedor deve construir uma oferta de serviço com qualidade (*QoSOffer*), especificando a interface funcional do serviço (*ServiceInterface*), e um conjunto de atributos de qualidade (*QoSConstraint*), que caracterizam o seu nível de qualidade. Cada atributo de qualidade define um limite (*threshold*) para o atributo de qualidade que é parte do modelo qualidade.

Por outro lado, quando um consumidor quer encontrar um serviço adequado, constrói uma requisição de serviço (*QoSRequest*) com a interface do serviço requerido e um conjunto de atributos de qualidade que ele requer. Esta requisição é recebida pela solução de seleção proposta por este trabalho, que utiliza um registro de serviços para localizar os serviços candidatos e selecioná-los.

Arquitetura DSOA

A Figura 2.5 apresenta uma visão de alto nível da arquitetura de *DSOA*.

Os componentes de *DSOA* são incluídos na plataforma dinamicamente como serviços e são responsáveis por diversas atividades, tais como: monitoramento de atributos de qualidade, gerenciamento das adaptações e seleção de serviços. Estes componentes são

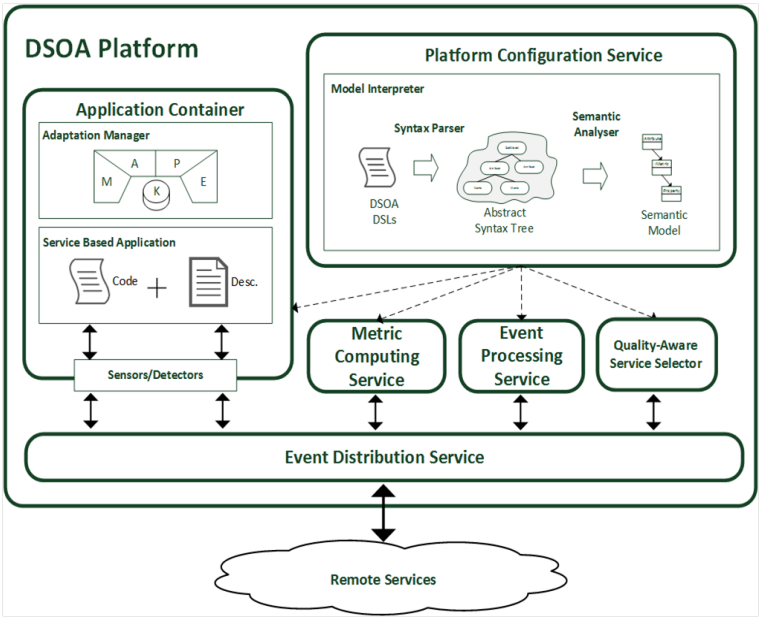

Figura 2.5: Arquitetura da Plataforma *DSOA*

descritos a seguir:

- ***Platform Configuration Service***: este componente é responsável pelo processamento dos modelos de eventos, qualidade e serviços através das DSLs que são especificadas por estes modelos. Estes modelos quando processados geram os modelos de objetos que são utilizados para (re) configurar os componentes da plataforma. A partir das configurações dos componentes é possível, por exemplo, a definição de novos tipos de eventos, agentes, serviços e atributos de qualidade. Ele é um componente central da plataforma, pois ajuda na característica dinâmica inerente da plataforma. Normalmente é inicializado quando a plataforma inicia sua execução, e partir desse momento escuta as definições das configurações dos modelos que são usados para configurar em tempo de execução os componentes da plataforma.

- ***Application Container***: este componente é responsável por todo o ciclo de vida das aplicações e por garantir que os serviços providos e requeridos estejam em conformidade com os acordos de nível de serviços (SLAs) estabelecidos. Para cumprir sua missão, ele medeia as interações entre as aplicações e o mundo externo. Como mostrado na Figura 2.5, através de sensores, o *container* controla todos os acessos as aplicações e notifica os demais componentes da plataforma sobre estes acessos. Estes sensores constroem eventos que contém um conjunto de dados

e metadados que descrevem a invocação de serviços, tais como parâmetros de chamadas, identificação do consumidor de serviços e assim por diante. Além disso, uma vez feita a ligação do consumidor a um serviço e, este último começar a ser executado, a plataforma irá monitorá-lo para verificar se o serviço selecionado satisfaz os atributos de qualidade da aplicação. Se este não for o caso, o *container* pode adaptar a aplicação substituindo os serviços.

- ***Event Distribution Service* (*EDS*)**: uma vez que um evento é produzido, ele deve ser publicado na plataforma, a fim de permitir que outros componentes possam descobri-lo. O componente responsável pela distribuição dos eventos na plataforma é o *EDS*. Ele funciona como um barramento de eventos e é responsável por manter a conexão de baixo acoplamento entre os componentes da plataforma *DSOA*. Esse baixo acoplamento, na verdade, é o baixo grau de dependência entre os componentes da plataforma. De fato, toda conexão na plataforma acontece através dos eventos. Estes últimos são transformados em mensagens que são distribuídas para os componentes que pediram para ouvir tais mensagens. Na prática, sempre que acontece algum evento na plataforma, este deve ser informado a todos os outros componentes, e o *Event Distribution Service* é o responsável por distribuir esta informação.

- ***Event Processing Service* (*EPS*)**: Este componente processa todos os eventos da plataforma, sejam os eventos externos que chegam, sejam os eventos produzidos pela própria plataforma. Além disso, ele é um componente central do monitoramento da plataforma *DSOA*. O *EPS* se registra no *EDS* para receber e processar os eventos de acordo com um conjunto de regras de processamento definidas pelos usuários. Estas regras são definidas a partir de agentes de processamento de eventos, que são utilizados para monitorar os atributos de qualidade dos serviços utilizados. Por exemplo, um agente é responsável por determinar a disponibilidade de um serviço, nesse caso, ele verifica periódicamente os serviços que estão online e em funcionamento, através dos eventos ligados a disponibilidade do serviço.

- ***Metric Computing Service***: é responsável por computar e manter os atributos e as definições de suas métricas na base de dados de qualidade interna da plataforma. Além disso, ele centraliza as informações sobre o mapeamento desses atributos de qualidade e suas métricas para cada serviço.

- ***Quality-Aware Service Selector* (*QSS*)**: é responsável por abstrair a busca ao registro de serviços com o objetivo de selecionar os serviços que irão fazer parte da

composição. Esta seleção é baseada em atributos de qualidade.

Em particular, este último componente é o foco deste trabalho.

2.2 Trabalhos Relacionados

Esta seção descreve os principais trabalhos relacionados ao desenvolvimento da solução proposta. Estes trabalhos contribuíram para a aquisição do conhecimento necessário sobre estado da arte atual, permitindo a construção de uma solução original que atendesse a todos os requisitos gerais de uma seleção com base em atributos de qualidade, bem como, apresentasse uma contribuição para a área de seleção de serviços.

2.2.1 Seleção de Serviços

A seleção de um serviço entre milhares com funcionalidade semelhante ou mesmo idêntica para compor uma aplicação é complexa e, por isso, é um tema bastante abordado atualmente na indústria e na academia.

Neste contexto, várias abordagens têm sido propostas para resolver este problema nos últimos anos [6][23][30][31][32][33]. Embora não sejam todas as soluções, a maioria adota os atributos de qualidade como critérios fundamentais para o sucesso da seleção.

Nesta seção são descritos alguns dos principais trabalhos relacionados com a seleção de serviços e é feita uma comparação da solução proposta com algumas abordagens existentes.

Serhani *et. al.* [31] apresenta uma arquitetura baseada em *QoS* que ajuda os consumidores a selecionar serviços Web (*Web services*), considerando os atributos de qualidade.

O objetivo foi estender *SOA* com suporte a *QoS*. O autor inclui uma descrição de atributos de qualidade durante a publicação de serviços e executa a invocação dinâmica com base nesses atributos de qualidade. As características importantes dessa solução incluem o apoio a seleção de serviços, com base nas exigências do consumidor, e o processo de verificação e certificação de *QoS* dos serviços.

A solução tem duas fases: a de verificação e a de certificação dos atributos de qualidade dos serviços. A fase de verificação é o processo de validação da veracidade das informações dos serviços, incluindo os aspectos sintáticos e semânticos da interface dos serviços, e os parâmetros de *QoS* descritos. A verificação é realizada através do uso de processos de testes para obter os valores dos atributos de qualidade reais dos serviços e comparar com os valores requeridos pelo consumidor.

A realização dos testes é feita por um componente da arquitetura chamado de *Client Generator*, que consiste em uma aplicação java responsável por gerar diversas cargas de instâncias de clientes que invocam os serviços. A partir dessas invocações, são calculados, os valores dos atributos de qualidade, tais como, tempo de resposta e disponibilidade, que são armazenados em uma base de dados. Com estes valores históricos, os serviços são verificados para determinar se atendem aos valores informados pelo consumidor.

Uma vez que a verificação foi bem sucedida, a fase de certificação é iniciada. A certificação tem como entrada os resultados do processo de verificação e trata-se da validação do resultado desta verificação com o objetivo de apontar se o serviço está em conformidade com as suas descrições. Caso o resultado seja positivo é emitido um certificado para o provedor do serviço. Este certificado indica que o serviço tem a qualidade desejada pelo consumidor e está apto a ser selecionado, caso contrário este é descartado e um novo serviço é analisado. Além disso, no processo de certificação é possível realizar testes adicionais para se certificar de que as exigências de *QoS* sejam cumpridas.

Esta solução concentra sua contribuição no processo de verificação e certificação através da medição de atributos de qualidade usando casos de testes. Ao contrário deste trabalho relacionado, a solução de seleção proposta por este trabalho aproveita o uso da plataforma *DSOA* (apresentada na Seção 2.1.5) para realizar a medição dos atributos de qualidade dos serviços, através do monitoramento dos serviços em tempo de execução. Desta forma, é considerado o real funcionamento do serviço para adquirir seus valores de atributos de qualidade. Além disso, na solução proposta é possível alterar dinamicamente a forma como os serviços são classificados (algoritmos de ranqueamento).

Reiff-Marganiec *et. al.* [34] propõe um método completo para a seleção automática dos serviços mais relevantes para uma composição com base em atributos de qualidade e no contexto do usuário.

O método proposto apresenta um modelo de qualidade onde os serviços são agrupados em categorias com funcionalidades semelhantes. Em seguida, estes serviços são filtrados usando suas funcionalidades, e, após a filtragem, os serviços são ranqueados com base em seus atributos de qualidade.

Para ranquear os serviços é utilizado um método que leva em conta alguns atributos de qualidade essenciais. São atributos de qualidade essenciais os que são definidos pelo consumidor. Os serviços que não atendem a tais atributos de qualidade requeridos pelo consumidor são descartados.

Este método foi construído com base nas limitações apresentadas em [9] e é bastante

semelhante ao apresentado neste trabalho. No entanto, o método proposto não apresenta nenhuma técnica de verificação para saber se o serviço realmente atende aos atributos de qualidade requeridos pelo consumidor. Na realidade, são considerados os atributos de qualidade estabelecidos pelo prestador de serviços, ou seja, não há nenhuma técnica para o monitoramento dos atributos de qualidade em tempo de execução.

Rejandran [22] propõe um mecanismo para a seleção dinâmica de serviços com base nos requisitos funcionais e nos atributos de qualidade requeridos pelo consumidor.

O mecanismo foi desenvolvido também como um serviço que faz a interação entre o consumidor e o registro de serviços, fornecendo as operações de gestão de atributos de qualidade. Para realizar tais operações, o *UDDI* foi estendido com a possibilidade de adicionar informações referentes a *QoS*. Isto foi feito através do conceito de modelos técnicos (*tModel*) [35] que permite especificar, padronizar e reutilizar os conceitos de *QoS* no *UDDI*. Além disso, o *tModel* define uma maneira fácil dos consumidores expressarem a qualidade desejada.

O *tModel* consiste em arquivo *XML* contendo uma chave, um nome, uma descrição de qualidade, e uma *URL* que aponta para onde os serviços podem ser encontrados. Quando um serviço é registrado no *UDDI*, um *tModel* é criado e contém uma lista de atributos de qualidade combinando nome e valor. Deste modo, quando o consumidor deseja selecionar um serviço ele utiliza o *tModel* para selecioná-lo com base nos atributos de qualidade.

Além disso, para realizar esta seleção o mecanismo possui um algoritmo de seleção particular. No entanto, diferente do trabalho desta dissertação, não é levado em conta a preferência dinâmica dos consumidores, bem como, não é apresentado detalhadamente como é feito o ranqueamento dos serviços.

Xu *et. al.* [36] propõe um modelo de descoberta de serviços que combina um registro *UDDI* estendido para publicar as informações de *QoS*, um sistema gerenciador de reputação para construir e manter a reputação dos serviços com base no *feedback* do consumidor e um agente de descoberta de serviços, que facilita a descoberta de serviços.

Para inserir as informações de QoS no *UDDI* são usadas estruturas de dados conhecidas como *tModel*. O *tModel* é criado juntamente com a publicação do serviço no registro e cada atributo de qualidade é representado por um nome e contém um valor.

O diferencial do modelo proposto é o uso de *feedbacks* dos consumidores no processo de descoberta de serviços. O gerente de reputação é o responsável por coletar o *feedback* de *QoS* dos serviços, a partir dos seus consumidores, calcular a pontuação de reputação desses serviços e atualizar estes valores em uma base de dados de reputação. O *feedback* indica a satisfação do consumidor com um serviço após o seu uso e nada mais é que um

número que varia de um a dez. A reputação de um serviço é a média de todos os *feedbacks* dos consumidores.

O agente de descoberta de serviços recebe as solicitações de serviços contendo as especificações funcionais e os atributos de qualidade e de reputação do consumidor, encontra os serviços que atendem aos requisitos funcionais e retorna uma lista de serviços para o consumidor. Caso os requisitos de *QoS* tenham sidos especificados é executada uma verificação a partir da primeira lista e retornado um subconjunto de serviços que atendem aos requisitos de *QoS*, caso apenas um serviço satisfaça os critérios de seleção, é retornado este serviço ao consumidor. Caso haja requisitos de reputação, as pontuações de *QoS* são calculadas e uma nova lista de serviços é retornada com uma classificação decrescente com base na reputação de *QoS*.

Embora pareça uma boa estratégia, o uso de *feedback* do consumidor não garante que a qualidade do serviço seja atendida, portanto, é necessária uma maneira de verificar os atributos de qualidade como apresentado pela solução deste trabalho. Além disso, o modelo apresenta um algoritmo de ranqueamento particular para determinar qual serviço deve ser selecionado. No entanto, não fica claro como é a estratégia de ranqueamento dos serviços.

Al-Masri *et. al.* [23] e Zeng *et. al.* [32] apresentam, cada um, soluções para a seleção de serviços que consideram diversos atributos de qualidade (por exemplo, custo, tempo de resposta, disponibilidade) e também consideram as preferências definidas pelos consumidores.

Al-Masri introduz uma estratégia chamada *Web Service Relevancy Function* (WsRF) que é responsável por medir o *ranking* de relevância de um serviço em particular, com base nas preferências do consumidor e nos atributos de *QoS*. Enquanto Zeng utiliza técnicas de programação linear para o *ranking*, para selecionar os serviços ideais para um serviço composto.

Embora apresentem nomenclaturas distintas, ambas as técnicas de ranqueamento utilizam a mestra estratégia, conhecida como *Simple Additive Weighting* [37][38]. Esta estratégia é bastante adotada, por isso, será uma das estratégias empregadas pela solução de seleção de serviços proposta por este trabalho.

Taher *et. al.* [6] propõe um *framework* genérico para o mecanismo de seleção baseado em *QoS* para *Web services*. O *framework* provê três funcionalidades principais: um mecanismo de seleção de serviços baseado em *QoS*, um gerenciador de mudanças dinâmicas dos atributos de *QoS* e um mecanismo de notificação que informa ao consumidor quando valores dos atributos de *QoS* são alterados.

O *framework* é representado por dois modelos, o modelo de dados e o modelo computacional. O modelo de dados é responsável por estabelecer uma ontologia que fornece um entendimento comum do *framework*, dos atributos de *QoS* e de sua semântica. Enquanto que o modelo computacional é responsável pela gerência dos atributos de *QoS* e pela seleção dos serviços com base nesses atributos de *QoS*.

Para gerenciar os atributos de *QoS* o modelo computacional possui um componente chamado *Update Manager* que é responsável por tratar as solicitações enviadas a partir dos provedores de serviços, tais como atualizar no registro de serviços as mudanças de atributos de *QoS* dos seus serviços publicados e consequentemente notificar aos consumidores dos novos valores dos atributos de *QoS* desses serviços. No entanto, diferente da solução de seleção proposta pelo presente trabalho, que considera também um mecanismo que monitora os valores dos atributos de qualidade quando os serviços estão em execução, no *framework* apenas os valores dos provedores são considerados e os provedores que informam quando os valores devem ser atualizados. Consequentemente, não é garantido que o serviço possua a qualidade que o provedor diz ter.

Para selecionar os serviços com base atributos de *QoS* o *framework* possui um algoritmo de *matchmaking* que determina quais serviços são selecionados com base nas especificações de *QoS* do consumidor. Semelhante ao modelo apresentado por Bruno [39] o *framework* utiliza a estratégia de *Euclidean Distance* para ranquear e selecionar os serviços. Por se tratar de uma estratégia de ranqueamento bastante utilizada na seleção de serviços, a *Euclidean Distance* também será outro algoritmo adotado pela solução de seleção proposta por este trabalho.

Yu em [40][24][33] propõe uma arquitetura para facilitar a seleção e coordenação de serviços baseados em *QoS* para uma composição. Para selecionar os serviços, a arquitetura implementa um algoritmo de ranqueamento que seleciona os serviços individuais para uma composição de serviços ideal. No entanto, para que os serviços sejam selecionados, é necessário que estes atendam as restrições de *QoS* do consumidor.

O objetivo do algoritmo implementado é maximizar o valor de uma função de utilidade definida, enquanto atende as restrições de *QoS* do consumidor. Esta função de utilidade é definida por uma soma ponderada dos atributos de qualidade. Além disso, neste algoritmo, cada atributo de qualidade é ponderado pelo seu peso de importância, definido pelo próprio consumidor, além de serem normalizados por suas médias e desvio-padrão. Desta forma, a função de utilidade definida não é influenciada por qualquer atributo com um valor grande.

Os autores deste trabalho se preocuparam em apresentar sua arquitetura, no entanto,

não deixam claro como são feitas declarações de qualidade providas pelos serviços e como são feitas as requisições de atributos de qualidade por parte do consumidor. Além disso, também não é apresentada nenhuma maneira de como os atributos de qualidade são adquiridos ou calculados.

Por fim, o algoritmo de ranqueamento proposto por esta arquitetura, por apresentar aspectos diferentes no ranqueamento dos serviços, tais como, a ponderação dos atributos de qualidade e a normalização destes atributos com base em dados estatísticos (média e desvio padrão), também será utilizado pela solução proposta pelo presente trabalho.

Considerando os trabalhos apresentados, a maioria das abordagens de seleção de serviços apresenta a falta de aspectos mais dinâmicos, por exemplo, não permitem determinar quantos e quais atributos de qualidade eles suportam, nem a maneira de como estes atributos são avaliados, ora são considerados casos de testes com os serviços, ora são considerados feedbacks dos consumidores. Em comparação com estas abordagens existentes, a solução proposta nesse trabalho trata a maioria dos requisitos fundamentais apresentados pelas abordagens de seleção de serviços e considerar alguns aspectos poucos explorados, tais como, a possibilidade de as aplicações diferenciarem os atributos de qualidade requeridos de forma dinâmica, bem como utilizar um mecanismo de monitoração de qualidade dos serviços.

Outro aspecto pouco abordado pelas soluções de seleção de serviços existentes é o processo de ranqueamento dos serviços com base em seus atributos de qualidade e decidir qual serviço deve ser selecionado. Em geral, as abordagens existentes definem um algoritmo de ranqueamento estaticamente, de modo que este, depois que definido, não é alterado em tempo de execução independe da situação de seleção de serviços apresentada. Neste contexto, a solução do presente trabalho deverá considerar diferentes algoritmos de ranqueamento e considerar a possibilidade de substituir estes em tempo execução.

Alguns dos algoritmos de ranqueamento já mencionados, tais como *Simple Additive Weighting*, *Euclidean Distance* e *Utility Function* são amplamente adotados pela literatura e, por isso, serão também adotados pela solução de seleção proposta e, portanto, serão apresentados no Capítulo 3.

Em comparação com as abordagens de seleção de serviços apresentadas, acredita-se que a solução proposta neste trabalho poderá atender aos requisitos fundamentais apresentados, além disso, irá apresentar aspectos originais, por tratar questões pouco exploradas, como a questão do dinamismo no ranqueamento dos serviços através dos atributos de qualidade.

2.3 Considerações Finais

Este capítulo apresentou os conceitos essenciais ao entendimento da proposta deste trabalho. Foi apresentado o paradigma de computação orientada a serviços, o conceito de qualidade de serviço, a plataforma *OSGi*, o conceito de algoritmo de ranquemamento e a plataforma *DSOA*. Por fim, foram apresentados alguns trabalhos relacionados à seleção de serviços com base em atributos de qualidade.

3

Seleção de Serviços Ciente de Qualidade

"...Vencer é nunca desistir."

—ALBERT EINSTEIN

Este capítulo apresenta a solução de seleção de serviços proposta por este trabalho. Inicialmente é apresentada uma visão geral da solução seguida pela descrição dos seus requisitos para implementação. Após os requisitos, a arquitetura da solução é apresentada. Em seguida, todo o processo para a seleção de serviços da solução é apresentado, descrevendo em detalhes os componentes da solução juntamente com as especificações da sua implementação.

3.1 Visão Geral

Este trabalho propõe uma solução para seleção de serviços, chamada de ***QSS***, do inglês *Quality-Aware Service Selector*, que considera não apenas a funcionalidade dos serviços, mas também os atributos de qualidade requeridos pelo consumidor. Para tal propósito, possui como diferencial a possibilidade de ser reconfigurada em tempo de execução, ou seja, sem a necessidade de ser parada e/ou recompilada. Em particular, o processo de reconfiguração permite uma substituição completa da estratégia de seleção.

É importante destacar que estas estratégias de seleção são constituídas de duas características. A primeira, é o processo de certificação dos serviços candidatos, que permite ao *QSS* (1) verificar se os serviços de fato atendem aos nível de qualidade exigido pelo consumidor ou (2) utilizar valores customizados dos atributos de qualidade de cada serviço na seleção.

A segunda e mais importante característica é o processo de classificação, onde os serviços são ranqueados com base nos valores dos atributos de qualidade requeridos pelo consumidor. Em particular, atributos de qualidade ligados a desempenho. Para isso, é utilizado um algoritmo de ranqueamento para determinar o serviço a ser selecionado. O *QSS* suporta vários algoritmos de ranqueamento e permite a substituição destes de forma dinâmica, sem interromper as aplicações em execução.

Em ambos os processos as características marcantes da solução são o dinamismo e a extensibilidade; o *QSS* suporta vários modos de certificação, classificação e permite a substituição desses de forma dinâmica, para que possa ser escolhida a melhor estratégia de seleção que se adapte à situação apresentada pelo consumidor, sem interromper as aplicações em execução. Dessa forma, sempre que necessário, poderão ser desenvolvidos novos serviços de certificação ou novos algoritmos de ranqueamento e estes poderão ser integrados ao *QSS*. Assim, poderá haver uma seleção de serviços mais eficiente através da possibilidade de adotar uma estratégia mais apropriada, dada as diversas situações e aspectos encontrados na seleção.

Numa visão de mais alto nível, o *QSS* visa estender o modelo conceitual comum de *SOA* (apresentado na Seção 2.1.1), dando suporte ao consumidor sobre a seleção dos serviços e estendendo a capacidade da seleção com a possibilidade de utilizar atributos de qualidade para selecionar serviços. A Figura 3.1 apresenta esta extensão de suporte à seleção.

O tipo de ligação entre o consumidor e o *QSS* apresentado na Figura 3.1, indica que o *QSS* também é um serviço, que é consumido por um consumidor com o objetivo de selecionar os serviços desejados com base na funcionalidade e nos atributos de qualidade.

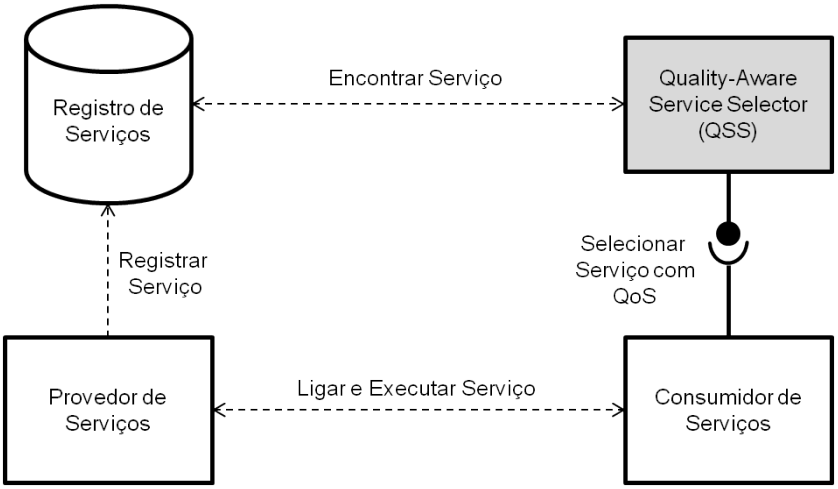

Figura 3.1: Arquitetura *SOA* Estendida com o *QSS*

Diferente do modelo tradicional, onde os consumidores acessam diretamente o registro para encontrar os serviços que irão compor a aplicação, este modelo é estendido com o *QSS* que é responsável por encontrar os serviços para o consumidor com base não só na sua funcionalidade, mas também em atributos de qualidade. Neste contexto, ao invés de acessar diretamente o registro, o consumidor faz a requisição do serviço ao *QSS*, este último faz a busca no registro para encontrar e selecionar os serviços adequados para o consumidor. Desta forma, o consumidor pode adicionar os atributos de qualidade na seleção dos serviços, diferente do modo usual que considera apenas a interface funcional do serviço.

O *QSS* foi idealizado a partir de dois elementos fundamentais: o referencial teórico levantado em toda pesquisa (apresentado na Seção 2.2), que ajudou a construir uma solução de seleção original, abordando aspectos pouco explorados na literatura, e a necessidade da plataforma *DSOA* (apresentada na Seção 2.1.5) em selecionar os serviços com base em seus atributos de qualidade para compor aplicações cientes de qualidade.

Como já foi mencionado (ver Seção 2.1.5), a plataforma *DSOA* provê um ambiente para a composição adaptativa e dinâmica de aplicações orientadas a serviços, com base em atributos de qualidade. Para isso, torna-se clara a necessidade de uma solução que forneça suporte à plataforma na seleção dos serviços, com base nesses atributos, para realizar tais composições.

Portanto, neste trabalho foi desenvolvido o *QSS*, uma solução de seleção de serviços, dinâmica, extensível e automática, que visa selecionar serviços com base nos seus requisitos funcionais e nos seus atributos de qualidade, através de uma estratégia adequada de classificação, que pode ser substituída a qualquer momento, sem afetar a execução das aplicações.

Assim como a plataforma *DSOA*, o *QSS* foi construído usando *OSGi* e utilizado como um componente pela plataforma. O *OSGi* foi utilizado porque tanto a plataforma como o *QSS* foram construídos como um conjunto de componentes colaborativos baseados em serviços. Além disso, o *OSGi* proporciona as características de dinamicidade e extensibilidade pretendidas pelo *QSS*. Para mais detalhes ver Seção 2.1.3.

A seleção de um serviço com base em seus atributos de qualidade é um processo em que várias etapas devem consideradas até que se chegue ao serviço desejado. No *QSS*, o processo de seleção é constituído por três etapas: *Descoberta*, *Verificação* e *Classificação*; cada etapa é responsável por um passo importante no processo de seleção de serviços ciente de qualidade.

3.1.1 Descoberta dos Serviços

A etapa de descoberta dos serviços consiste em encontrar ou localizar as implementações dos serviços que atendam as condições especificadas pelo consumidor. Esta etapa normalmente acontece na busca por serviços em um registro de serviços [14][22], por exemplo, o registro de serviços do *OSGi*. Nesta etapa o *QSS* identifica os requisitos funcionais, através da interface do serviço, os atributos de qualidade exigidos pelo consumidor e acessa um registro para encontrar os serviços que atendam a tais exigências. Os serviços descobertos nessa etapa são denominados serviços candidatos.

3.1.2 Verificação dos Serviços

A etapa de verificação de serviços é responsável por assegurar a qualidade dos serviços candidatos. Ela tem como objetivo certificar se a qualidade que o provedor do serviço alega fornecer de fato é a qualidade esperada pelo consumidor [22][41][42]. Para esta etapa existe um registro que guarda os dados de qualidade dos serviços que já foram selecionados. O *QSS* acessa o registro de qualidade e faz uma comparação com os dados fornecidos pelo provedor do serviço. Caso os atributos de qualidade do consumidor sejam atendidos o serviço é certificado.

Além disso, nesta etapa de verificação é possível utilizar valores personalizados dos atributos de qualidade que foram monitorados e armazenados no registro de qualidade. Ou seja, além dos valores dos atributos de qualidade informados pelo próprio provedor do serviço e dos valores monitorados, é possível personalizar os valores já monitorados para realizar novas verificações dos serviços. Desta forma, além de utilizar só valores históricos monitorados, é possível determinar, por exemplo, a média dos dez últimos registros dos atributos de qualidade de um serviço e utilizar este valor na classificação do serviço. Tão logo os serviços sejam certificados, eles prosseguem para a etapa de classificação.

3.1.3 Classificação dos Serviços

A classificação dos serviços consiste em ordenar os serviços com base em uma métrica quantitativa formada pela agregação dos valores dos atributos de qualidade de cada serviço. Com esta agregação obtém-se uma pontuação final que mostra o índice de qualidade do serviço e determina-se qual serviço deve ser selecionado [6][7]. No *QSS*, a etapa de classificação possibilita utilizar diferentes algoritmos de ranqueamento para classificar os serviços, onde cada algoritmo possui sua estratégia para agregar os valores

dos atributos de qualidade. Então, a partir do algoritmo de ranqueamento utilizado os serviços são ranqueados, depois um ou mais serviços são selecionados e entregues ao consumidor.

3.1.4 Funcionamento do QSS

A partir do que já foi apresentado e para um melhor entendimento do objetivo do *QSS*, foi descrito um típico cenário de funcionamento:

1. O provedor publica o serviço no registro de serviços. A publicação consiste no nome da interface, a implementação do serviço e um conjunto de atributos de qualidade atendidos pelo serviço.

2. Inicialmente, o consumidor invoca o *QSS* para encontrar o serviço desejado. Na invocação do *QSS* é especificada a interface do serviço desejado e as restrições dos atributos de qualidade que os serviços devem atender.

3. O *QSS* interpreta a requisição e cria um filtro contendo os atributos de qualidade especificados. Este filtro é usado como consultas para encontrar os serviços.

4. Com o filtro criado, o *QSS* acessa o registro que faz uma operação de busca para encontrar os serviços.

5. Uma lista de serviços que atendem as especificações é devolvida como resultado da descoberta do *QSS*. Os serviços retornados são chamados de serviços candidatos.

6. A lista de serviços candidatos pode apresentar três situações:

   6.1. Caso apenas um serviço seja retornado na lista, o item 7 é realizado.

   6.2. Caso nenhum serviço seja retornado na lista, o *QSS* fica aguardando até ser notificado pelo registro de serviços que um serviço que atende as especificações do consumidor foi publicado. Após a notificação de publicação desse serviço, o item 7 é imediatamente realizado.

   6.3. Caso múltiplos serviços sejam retornados o *QSS* faz a verificação dos serviços candidatos para assegurar que a qualidade destes serviços de fato atende as exigências do consumidor. Após a verificação, o *QSS* classifica e ordena os serviços em função dos atributos de qualidade especificados pelo consumidor e segue para o item 7.

7. O *QSS* entrega um serviço ou a lista de serviços ao consumidor do serviço e, enquanto houver requisições de serviços, os passos acima se repetem.

Vale ressaltar que tanto a implementação do *QSS* quanto o seu funcionamento serão descritos de forma mais detalhada nas próximas seções deste capítulo.

3.2 Requisitos do QSS

Para construir o *QSS* é necessário considerar alguns requisitos básicos:

- **Modelo de Qualidade de Serviços**

  A fim de representar os atributos de qualidade publicados pelos provedores de serviços, bem como os atributos de qualidade requeridos por um consumidor na requisição de um serviço, um requisito necessário para construir o *QSS* será utilizar um modelo de qualidade existente ou definir um novo modelo de qualidade para representar os atributos de qualidade tanto para dar suporte ao provedor na especificação dos atributos dos serviços providos, bem como para expressar os atributos de qualidade requisitados pelo consumidor.

- **Preferência dos Consumidores**

  Como não é possível propor um modelo unificado de qualidade, é necessário permitir que cada aplicação (consumidor) possa definir o seus próprios atributos de qualidade. Desta forma, é um requisito do *QSS* que este deva utilizar ou propor uma linguagem de modelagem que dê suporte as aplicações para definir atributos de qualidade desejados.

- **Avaliação dos Atributos de Qualidade**

  Outro aspecto relevante é como os valores dos atributos de qualidade são avaliados e coletados. Em geral, algumas soluções existentes consideram os valores anunciados pelo próprio provedor dos serviços, enquanto outras realizam casos de testes para coletar os valores de qualidade. No entanto, nem sempre os valores anunciados são verdadeiros, bem como, os testados estão em conformidade com a realidade do ambiente. Neste contexto, outro requisito do *QSS* é que deve adotar ou implementar um mecanismo de monitoramento de atributos de qualidade.

- **Diversidade de Algoritmos de Ranqueamento**

Após avaliar todos os atributos de qualidade desejados, o próximo passo importante é a agregação desses para obter uma pontuação final para o serviço e proceder a ordenação dos serviços. Nesta etapa um algoritmo de ranqueamento adequado deve ser usado. No entanto, não há um algoritmo que atenda a todas as situações que já foram mencionadas. Portanto, é importante que o algoritmo seja definido e substituído de forma dinâmica para melhor atender a situação apresentada. O *QSS* deverá fornecer suporte ao uso de vários algoritmos de ranqueamento. Além disso, deverá permitir a substituição destes algoritmos em tempo de execução. Desta forma, o consumidor tem a possibilidade de usar o melhor algoritmo dada a situação de seleção apresentada.

- **Linguagem e Paradigma de Programação**

  Visando atender as características obrigatórias, tais como dinamismo e extensibilidade, além de outras, tais como reusabilidade e flexibilidade, a construção do *QSS* deverá utilizar uma linguagem de programação que suporte a implementação de tais características. Outro aspecto que deve ser considerado é a escolha do paradigma utilizado para implementar o *QSS*. Deve-se utilizar um paradigma que dê suporte a implementação do *QSS* como um conjunto de componentes reutilizáveis.

- **Escalabilidade e Eficácia**

  Por fim, o *QSS* deve ser escalável e preciso. Neste contexto, escalabilidade está relacionada à capacidade do *QSS* ser acrescido de vários atributos de qualidade, vários algoritmos de ranqueamento e assim por diante. Ou seja, ser expandido mantendo seu desempenho e sem precisar ser reiniciado. A eficácia está relacionada à qualidade do resultado da seleção dos serviços. Neste aspecto, o consumidor pode determinar o algoritmo de ranqueamento que lhe oferece o melhor resultado.

Estes são os requisitos para a implementação do *QSS*. A partir destes requisitos, foi definida uma arquitetura para o *QSS*. Esta arquitetura foi apresentada em [43] e será descrita com detalhes na próxima seção.

3.3 Arquitetura do QSS

A Figura 3.2 apresenta uma visão geral da arquitetura do *QSS* destacando seus componentes. Como pode ser observada o *QSS* é composto por três componentes principais: o **Selecionador**, o **Certificador** e o **Classificador**. Para cada componente foi definido

um conjunto de atividades para cumprir o objetivo do consumidor de selecionar o melhor serviço que satisfaça suas necessidades.

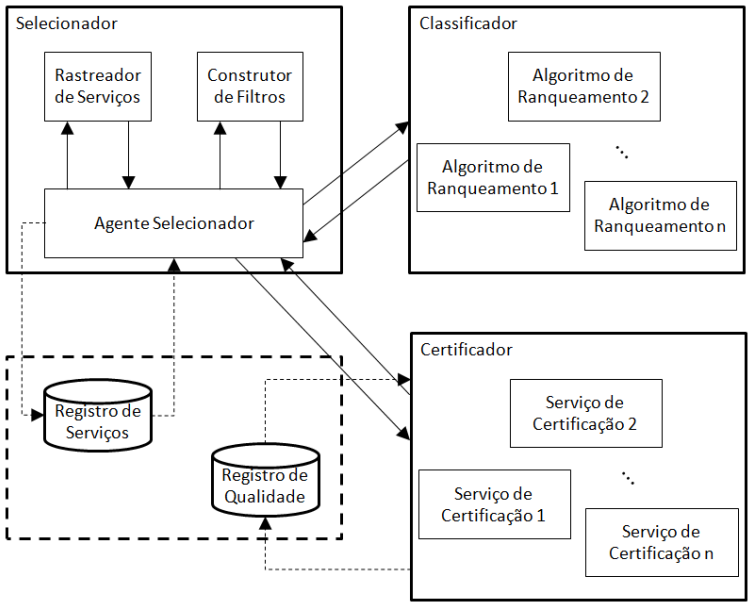

Figura 3.2: Arquitetura do *QSS*

O **Selecionador** é o principal componente do *QSS*. Ele consiste na fachada do *QSS* com o mundo externo, sendo responsável por receber a requisição dos consumidores dos serviços e implementar as operações relacionadas com o tratamento dos valores dos atributos de qualidade recebidos pelo consumidor e pelos serviços providos. Além disso, ele é o componente controlador, responsável por delegar os processos de certificação e classificação dos serviços.

Na prática, o **Selecionador** é responsável por receber a requisição de um serviço pelo consumidor, interpretar essa requisição, identificando a interface funcional dos serviços e os atributos de qualidade exigidos, encontrar os serviços candidatos no registro de serviços, delegar que seja realizada a certificação dos serviços, através do componente **Certificador**, e a classificação dos serviços, através do componente **Classificador**. Finalmente, ele entrega os serviços adequados aos consumidores.

O componente **Certificador** contém serviços responsáveis pela certificação dos atributos de qualidade fornecidos (*Serviços de Certificação*). Através destes é possível verificar se os serviços candidatos encontrados no registro atendem ou não aos atributos de qualidade requisitados pelo consumidor. Adicionalmente, os serviços de certificação podem ser usados para obter diferentes valores de qualidade na classificação dos serviços, por exemplo, usar os valores mais recentes monitorados dos serviços ou atribuir pesos

para os dados monitorados de acordo suas marcas de tempo. Para isso, são utilizados dados de qualidade dos serviço que foram monitorados e que são armazenados em um registro de qualidade.

O componente **Classificador** implementa os algoritmos de ranqueamento responsáveis por determinar quais serviços melhor se adaptam aos atributos de qualidade requisitados por um consumidor. No *QSS*, é possível usar vários algoritmos de ranqueamento, estes são utilizados para agregar os valores e obter um resultado final para cada serviço, classificar os serviços e selecioná-los com base neste *ranking*.

Os outros componentes apresentados na Figura 3.2 são externos ao *QSS*, mas que também contribuem com o processo de seleção. O **Registro de Serviços** consiste no repositório de serviços da plataforma *OSGi*. O **Registro de Qualidade** é a base de dados onde são armazenados os dados históricos dos atributos de qualidade dos serviços monitorados. Este monitoramento é realizado pelos sensores dentro da plataforma *DSOA* (ver Seção 2.1.5).

Conforme descrito anteriormente na Seção 3.2, para implementar o *QSS* é necessário o uso de uma linguagem de programação que suporte as características que o *QSS* possui, tais como dinamismo e extensibilidade, portanto, a linguagem principal utilizada para o desenvolvimento foi a Java. Entretanto, outras linguagens podem ser usadas quando houver necessidades técnicas.

Outra característica do *QSS* é que ele é composto por diversos componentes e que alguns desses, como os algoritmos de ranqueamento, possam ser substituídos em tempo de execução. Portanto, como paradigma de programação foi adotado o modelo de programação orientada a serviços [1][2][44]. Este paradigma permite a construção de aplicações extensíveis e dinâmicas através da composição de serviços. Desta forma, o *QSS* permite todas as facilidades providas pelo paradigma, tais como, o reuso de componentes (serviços), substituição de componentes em tempo de execução, além de permitir ser extensível, ou seja, os desenvolvedores podem implementar seus próprios componentes (algortimos de ranqueamento e serviços de certificação) para a solução.

A tecnologia utilizada para implementar o *QSS* foi o *OSGi*. Como pode ser visto na Seção 2.1.3, o *OSGi* fornece suporte ao desenvolvimento de aplicações a partir de módulos reutilizáveis que requerem e fornecem serviços. Além disso, permite que estes módulos possam ser substituídos em tempo de execução. Portanto, a partir do *OSGi* é possível promover a dinamicidade pretendida. Os módulos do *QSS* foram implementados como módulos dinâmicos *OSGi*, que requerem e fornecem serviços. Neste contexto, todos os componentes definidos como dinâmicos, tais como os módulos do *QSS* e os

algoritmos de ranqueamento, foram desenvolvidos como serviços *OSGi* que podem ser iniciados, trocados e parados individualmente, a qualquer momento.

Nas próximas seções a implementação do *QSS* é descrita em detalhes, incluindo as etapas do processo de seleção dos serviços previamente descritas na Seção 3.1, seus projetos e os módulos que compõe o *QSS*. Estas informações podem ser obtidas de maneira mais sucinta em [45].

3.4 Seleção de Serviços

Conforme descrito anteriormente (ver Seção 3.3), o **Selecionador** é o componente principal do *QSS*. Na prática, é o componente que interage com o meio externo, interceptando as requisições de serviços por parte dos consumidores e buscando os serviços com base nessa requisição. Ele também interage com o meio interno através das operações de processamento das requisições, por exemplo, interpretando as necessidades funcionais e dos atributos de qualidade requeridos pelo consumidor. Além disso, ele controla os demais componentes, delegando a certificação e a classificação dos serviços.

Por fim, ele também é o responsável pela entrega dos serviços com atributos de qualidade mais adequados aos exigidos pelo consumidor. A Figura 3.3 apresenta o **Selecionador** em detalhes e os módulos de software que estão presentes no componente, bem como suas relações.

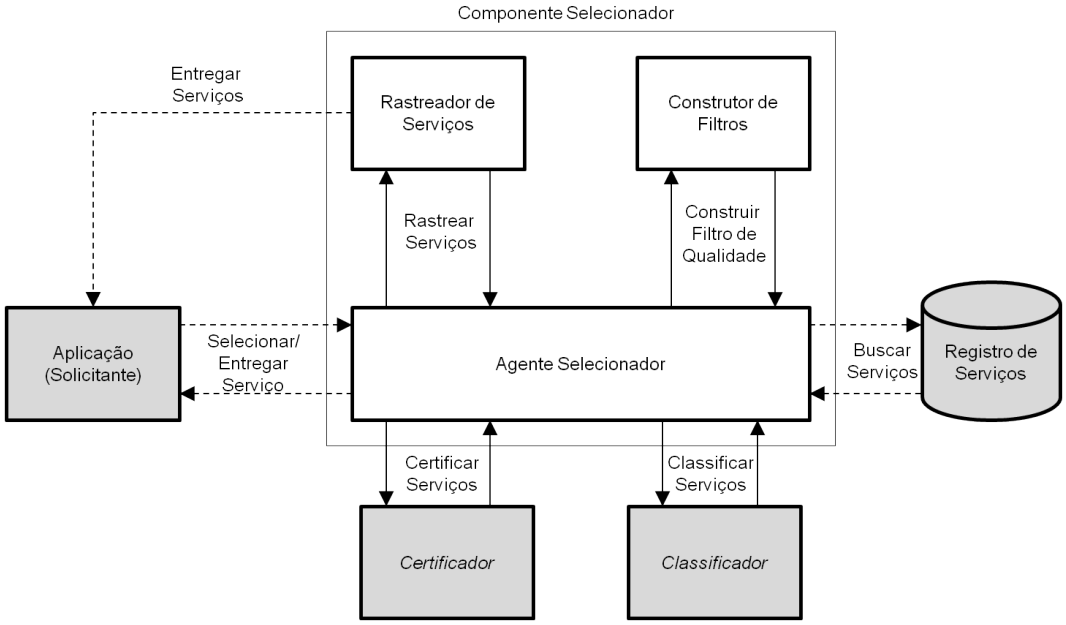

Figura 3.3: Selecionador

Cada módulo presente no componente **Selecionador** será descrito nas próximas subseções.

3.4.1 Agente Selecionador

O *Agente Selecionador* é o elemento principal do componente **Selecionador**. Ele é responsável por receber, interpretar e tratar a requisição de serviço com base em atributos de qualidade do consumidor. Além disso, o *Agente Selecionador* também implementa as operações de controle dos outros dois componentes da solução, o **Certificador** e o **Classificador**. Assim que o *Agente Selecionador* recebe uma requisição o *QSS* inicia o processo de seleção.

Inicialmente, o *Agente Selecionador* serve como a interface para o consumidor de serviços. O consumidor invoca o QSS através deste elemento com uma operação para encontrar os melhores serviços com base na sua funcionalidade e nos atributos de qualidade por ele exigidos. Esta requisição de serviços contém a identificação do consumidor da requisição, as interfaces dos serviços desejados, que fornecem as funcionalidades de cada serviço, e os atributos de qualidade que o consumidor deseja que os serviços atendam.

A partir dos modelos de qualidade e de serviços foi definida uma linguagem de modelagem que representa os conceitos de qualidade relacionados. Além disso, esta linguagem também permite a ligação de tais conceitos para os serviços e as operações. Desta forma, é possível definir, as interfaces funcionais dos serviços, bem como, os atributos de qualidade providos e os requeridos (ver Figura 3.4).

A Figura 3.4 ilustra a descritor de uma requisição de serviço por parte de um consumidor. Para construir este descritor, foi utilizada a linguagem de modelagem definida pelos modelos de qualidade e serviços, propostos pela plataforma *DSOA*.

O campo *requires* indica que trata-se de uma requisição de serviços, enquanto que o item *interface* é referente a interface funcional do serviço requisitado. O campo *constraint*, seguido do item *attribute*, informa quais são os atributos de qualidade requeridos pelo consumidor. Neste mesmo *constraint*, como pode ser observado, é possível adicionar outros itens, por exemplo, *operation* que indica a operação do serviço que o atributo de qualidade está relacionado, *expression* que informa a direção do atributos de qualidade (LE - *Lower Equal* menor ou igual, GE - *Greater Equal* maior ou igual e LT - *Lower True* se for menor) para valores de referências fornecidos pelo consumidor que é indicado pelo item *threshold*. Finalmente, o item *weight* indica o peso da importância do atributo de qualidade para o consumidor para a seleção do serviço.

Ao receber a requisição, o *Agente Selecionador* interpreta e processa as informações

```
<qos:requires interface="homebroker">
<constraint attribute="qos.AvgResponseTime"
    operation="priceAlert"
    expression="LE"
    threshold="200"
    weight="0.2" />

<constraint attribute="qos.Availability"
    expression="GE"
    threshold="75"
    weight="0.3" />

<constraint attribute="qos.BusinessException"
    operation="priceAlert"
    expression="LT"
    threshold="1"
    weight="0.5" />
</qos:requires>
```

Figura 3.4: Descritor de Requisição de Serviço

incluídas na requisição através de uma operação de *parser*. Neste ponto, para implementar a operação de parser foi utilizada uma *API* Java chamada *SAX* (*Simple Api for XML*).

A operação de *parser* consiste na extração das informações desejadas, tais como as interfaces funcionais dos serviços solicitados e os atributos de qualidade requeridos para cada serviço a partir do arquivo *XML* do descritor da requisição.

Os dados coletados na operação de parser são encaminhados para o elemento *Construtor de Filtros*.

3.4.2 Construtor de Filtros

O *Construtor de Filtros* é responsável pela construção automática de filtros contendo as especificações funcionais e de qualidade enviadas pelo consumidor, onde esses filtros são utilizados para restringir a descoberta de serviços no registro.

O elemento *Construtor de Filtros*, através do uso de uma operação para a criação, constrói um filtro que descreve de forma concisa a restrição imposta pelo consumidor para o serviço desejado. Este filtro é uma sequência de expressões simples com sintaxe baseada na representação de filtros de pesquisa *LDAP* [1], conforme definido em [18][19][20]. Um exemplo de filtro é apresentado na Figura 3.5. Ele descreve as restrições impostas pela

[1] http://www.ietf.org/rfc/rfc1960.txt

requisição apresentada na Figura 3.4.

```
(&(ServiceInterface=homebroker)
(homebroker.priceAlert.qos.AvgResponseTime<=200)
(homebroker.qos.Availability>=75)
(homebroker.priceAlert.qos.BusinessException<1))
```

Figura 3.5: Filtro *LDAP* Reference à requisição apresentada em 3.4

Após a construção do filtro, este é passado ao *Agente Selecionador*. Diferente das abordagens existentes que usualmente fazem a busca pelos serviços na forma clássica, olhando unicamente para as implementações com base nas interfaces funcionais, o QSS refina a busca pelos serviços utilizando o filtro criado.

O filtro é um espelho da requisição feita pelo consumidor e mostra os requisitos funcionais que o serviço deve conter, através da interface, bem como os atributos de qualidade que ele deve possuir com base nos atributos requisitados pelo consumidor.

Então, ao receber o filtro criado, o *Agente Selecionador* faz uma operação de busca (*Find*) no *Registro de Serviços*. A operação de busca tem o objetivo de procurar uma coleção de referências dos serviços que satisfaça ao filtro (requisição). Neste contexto, um serviço é considerado adequado se satisfazer os requisitos funcionais, declarados através da interface do serviço, e os atributos de qualidade requeridos pelo consumidor.

O *Registro de Serviços* compreende um elemento externo do *QSS*, ou seja, ele não foi implementado em conjunto com a solução. Na verdade, ele consiste no registro de serviços compartilhados da plataforma *OSGi* [20].

O *Registro de Serviços* [18][20] é um componente crucial da plataforma *OSGi*. Ele permite que os provedores publiquem/compartilhem seus serviços através de um conjunto de interfaces e seus respectivos objetos implementados. Além disso, permite que cada serviço tenha associado a ele um conjunto de propriedades (*metadados*). Estas propriedades tem o objetivo de armazenar informações como pares de *chave->valor* referente ao serviço publicado. Esta característica de associação de propriedades é utilizada para permitir que os provedores de serviços registrassem seus serviços contendo também os atributos de qualidade que o serviço pode prover. A fim de anunciar um serviço, um provedor deve construir uma oferta de qualidade de serviço. Para isso, é usado o modelo de qualidade e serviços para definir uma linguagem para a modelagem da qualidade oferecidas pelos provedores dos serviços. A Figura 3.6 ilustra o descritor que o provedor utiliza para publicar um serviço.

O campo *provides* indica que trata-se de um descritor de um provedor de serviços,

```
<provides>
 <service id="22" interface="homebroker">
       <dsoa-provided:sla>
         <sla>
         <slo attribute="AvgResponseTime"
           threshold="100"
           expression="LE"
           operation="priceAlert" />

         <slo attribute="BusinessException"
            threshold="0"
            expression="LE"
            operation="priceAlert" />

         <slo attribute="Availability"
            threshold="95"
            expression="GE" />
          </sla>
       </dsoa-provided:sla>
 </service>
</provides>
```

Figura 3.6: Descritor de Provedor de Serviços

enquanto que o campo *service* faz referência a um serviço provido e contém os itens *id*, que fazem referência ao identificador do serviço, e *interface* que informa a interface do serviço. Os demais campos, *sla, slo*, fazem referência à qualidade do serviço fornecida pelo provedor. Assim como no descritor dos consumidores, o item *attribute* descreve os atributos de qualidade, no entanto, neste caso são os atributos que o serviço provê, enquanto que o item *threshold* é o valor provido pelo serviço para o atributo de qualidade. O item *operation*, semelhante a requisição, também define que determinado atributo está ligado à uma operação do serviço. O item *expression* indica se o atributo deve ser maximizado ou minimizado, por exemplo, o tempo de resposta desejado é sempre o menor, então tal atributo deve ser minimizado, enquanto a disponibilidade deve ser maximizada.

Quando um serviço é publicado, é realizado um *parser* do descritor e os dados coletados são publicados nas propriedades deste serviço. A Figura 3.7 apresenta uma lista de propriedades de um serviço publicado. Como mencionado anteriormente, um módulo *OSGi*, além de prover os serviços, possui um arquivo manifesto que contém as propriedades do serviço provido. Estas propriedades são armazenadas como pares *chave->valor*. Várias propriedades são cadastradas quando um serviço é publicado, por exemplo,

as propriedades referentes ao acesso remoto a estes serviços, tais como *id*, *ipaddress* e provedor. No entanto, além das propriedades citadas, os atributos de qualidade providos por este serviço também são publicados neste manifesto. Na Figura 3.7 são apresentados apenas o que interessa no momento que é a interface do serviço, representada pelo campo *objectClass* e os atributos de qualidade que o serviço suporta.

```
dsoa-qos-provider (22) provides services:
----------------------------------------
operation.qos.AvgResponseTime.priceAlert.LE = 100.0
operation.qos.BusinessException.priceAlert.LE = 0.0
service.qos.Availability.GE = 95.0
objectClass = Homebroker
```

Figura 3.7: Lista de Propriedades de um Serviço Publicado no Registro de Serviços

Portanto, com o filtro criado pelo *Agente Selecionador* a operação de busca por serviços é realizada no *Registro de Serviços*. Esta operação analisa as propriedades dos serviços (ver Figura 3.7) com os atributos de qualidade requisitados pelo consumidor, expressos no filtro criado. Os serviços que atendem aos atributos de qualidade nesta análise são retornados como serviços candidatos.

No entanto, a natureza dinâmica e distribuída dos serviços retira a certeza de que sempre haverá um serviço a ser selecionado. Por exemplo, devido ao fato de não haver conectividade com a Internet no momento da busca por um serviço, ou a ausência de serviços publicados no registro, ou até mesmo não haver serviço adequado com os atributos de qualidade exigidos pelo consumidor, pode ser que nenhum serviço candidato seja retornado. Neste caso, o *QSS* espera até que um serviço seja publicado para selecioná-lo e enviá-lo ao consumidor, o módulo responsável por isso é o *Rastreador de Serviços*.

3.4.3 Rastreador de Serviços

O *Rastreador de Serviços* é o último elemento interno do **Selecionador**. O *Rastreador de Serviços* só é ativado caso nenhum serviço tenha sido retornado na operação de busca do *Agente Selecionador*.

Quando ativado, ele salva o filtro previamente criado e se registra no *Registro de Serviços*, a fim de receber as notificações de quando novos serviços que atendam a este filtro sejam publicados e estejam disponíveis. Isto porque, seu objetivo é esperar até a publicação de um serviço adequado no registro e imediatamente informar consumidor da publicação deste serviço.

Uma vez que o *Rastreador de Serviços* é notificado, ele verifica se o serviço satisfaz o filtro que foi previamente criado. Se este for o caso, ele utiliza uma operação de retorno para informar imediatamente ao consumidor sobre o novo serviço, que imediatamente se liga ao serviço. Para implementar este módulo foi utilizada uma técnica chamada *Service Tracker* [18][46] proposta pela própria plataforma *OSGi*.

Nos casos em que mais de um serviço é encontrado, o *Agente Selecionador* delega aos componentes **Certificador** e **Classificador**, que exerçam suas operações para selecionar um dos serviços. Após as operações, finalmente, o *Agente Selecionador* é responsável por entregar o serviço ao consumidor. A Figura 3.8 apresenta todo o processo de seleção que foi descrito anteriormente.

O processo apresentado na Figura 3.8 descreve todo o funcionamento da solução de seleção. No entanto, os processos de certificação e classificação foram considerados caixas pretas e serão apresentados nas próximas seções.

3.5 Certificação de Serviços

O **Certificador** é o componente do *QSS* responsável por verificar se os serviços candidatos retornados a partir do *Registro de Serviços* atendem aos atributos de qualidade requisitados pelo consumidor ou não.

A natureza dinâmica de *SOA* geralmente causa impacto na qualidade dos serviços. Como consequência, o nível de qualidade efetivamente provido pode não estar em conformidade com o que foi anunciado pelo provedor.

Um cenário de exemplo é o seguinte: um provedor pode informar que o serviço tem um taxa de 95% de disponibilidade para seu acesso, mas, devido às condições do ambiente, a sua disponibilidade real é de 80%. Nestas situações, usar os dados de atributos de qualidade, tal como publicado pelo provedor, pode levar a uma escolha errônea do ponto de vista do consumidor.

Por essa razão, no *QSS* foi definido o **Certificador** que tem o objetivo de constatar que a qualidade desejada para o serviço seja realmente atendida. A Figura 3.9 apresenta o componente **Certificador**.

Na realidade, o **Certificador** é um componente lógico e extensível que representa um conjunto de *serviços de certificação* que são responsáveis por certificar a qualidade dos serviços candidatos.

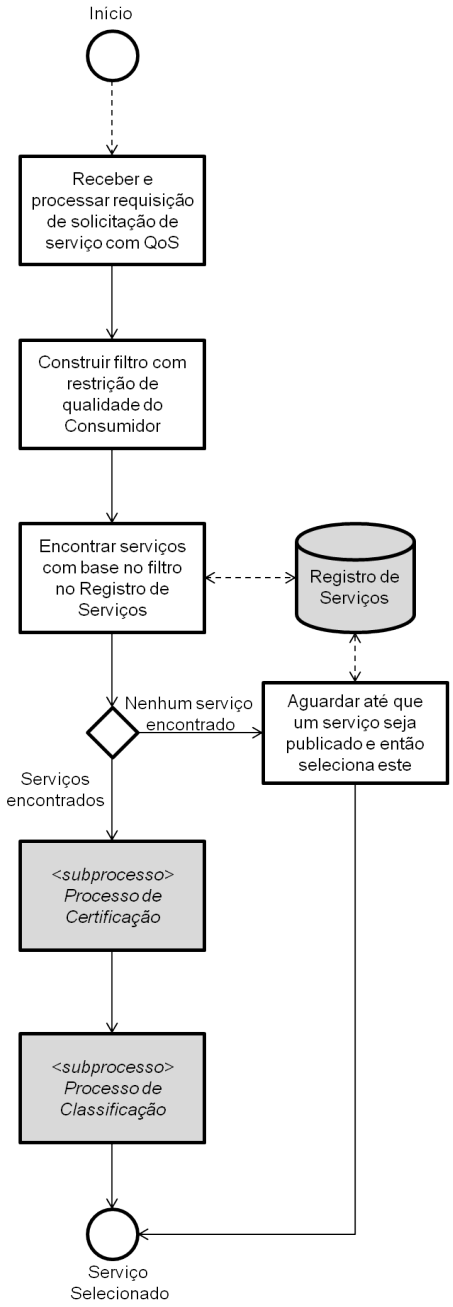

Figura 3.8: Processo de Seleção de Serviços

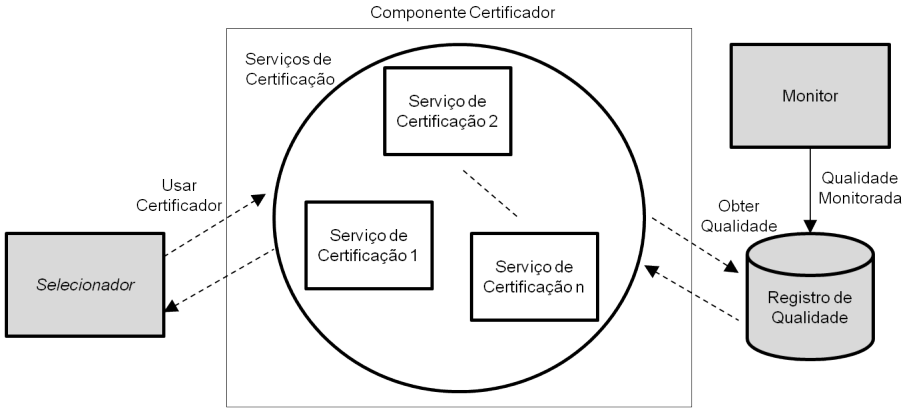

Figura 3.9: Certificador

3.5.1 Serviços de Certificação

A estratégia de utilizar *serviços de certificação* para constatar a qualidade dos serviços é uma característica importante do *QSS*, ele é dinamicamente extensível, o que significa que o processo de certificação não é "*hard coded*". Na verdade, o ponto central do processo de certificação é encapsulado sempre em um *Serviço de Certificação*. O componente pode possuir alguns *Serviços de Certificação*, mas também suporta o desenvolvimento de novos, através de uma interface bem definida dos *Serviços de Certificação*.

Quando o consumidor não utilizar ou implementar algum *serviço de certificação* personalizado, o serviço padrão disponibilizado pelo componente **Certificador** é utilizado. O *serviço de certificação* padrão obtém os valores dos atributos de qualidade monitorados dos serviços, quando estes dados existirem no *registro de qualidade*, neste contexto, os primeiros valores monitorados são considerados, caso não existam valores, o *serviço de certificação* padrão considera os valores dos atributos de qualidade oferecidos pelo provedor do serviço.

E, a partir dos valores de referência especificados pelo consumidor para cada atributo de qualidade, o *serviço de certificação* padrão faz um comparativo com os valores dos atributos de todos os serviços, se este coincidirem com os atributos de qualidade requeridos pelo consumidor, este serviço é certificado e torna-se apto a ser selecionado. Por exemplo, supondo que um consumidor em sua requisição determine que um serviço deva conter o "tempo de resposta" inferior a 50 ms e a disponibilidade superior a 80%, então o *serviço de certificação* padrão irá checar todos os serviços candidatos e certificar aqueles que estejam dentro dos valores de atributos de qualidade requeridos. Portanto, todos os serviços que satisfazem esta restrição são certificados e tornam-se aptos a serem selecionados.

Além disso, a solução permite aos consumidores utilizarem valores personalizados dos atributos de qualidade para certificar os serviços. Para isso, os consumidores podem implementar seus próprios *Serviços de Certificação*. Como o próprio nome sugere, os *Serviços de Certificação* são serviços implementados através de uma interface definida pela solução, que possui uma única operação de certificação e como retorno tem a definição de que o serviço está certificado ou não.

Neste contexto, o consumidor pode implementar seu próprio procedimento de certificação, dentro da operação de certificação disponibilizada pela interface funcional e disponibiliza-lo como um serviço de certificação para ser usado (consumido) pelo *QSS*. Por exemplo, caso um consumidor queira considerar um procedimento que use dados de qualidade históricos armazenados, ou considere uma janela de tempo de dados de qualidade monitorados, ou funções matemáticas para considerar uma certificação mais formal, ele pode implementar seu procedimento dentro da operação de certificação e publicar este como *Serviço de Certificação*.

Para lidar com essa possibilidade de utilizar diferentes valores dos atributos de qualidade, é considerado um mecanismo de monitoramento de atributos de qualidade em tempo de execução (ver Figura 3.9). Na realidade, assim como o *QSS*, este mecanismo de monitoração compreende um componente interno da plataforma *DSOA*.

3.5.2 Monitor

O *Monitor* é um mecanismo externo ao *QSS* responsável por monitorar os atributos de qualidade dos serviços utilizados a partir das aplicações que ele suporta.

O monitoramento dos serviços é realizado enquanto eles estão em execução. Tal característica é importante, pois diferente das abordagens que geralmente fazem casos de testes, esta solução usa o real funcionamento dos serviços para monitorar. Assim, é possível adquirir os valores dos atributos de qualidade mais adequados no momento.

O *Monitor* é componente da plataforma *DSOA* e foi implementado como um módulo dinâmico OSGi. Além disso, ele foi desenvolvido considerando o paradigma baseado em eventos e a tecnologia utilizada foi o *ESPER* [2]. Mais detalhes do monitor podem ser encontrados em [47].

[2] hhttp://esper.codehaus.org/

Os valores monitorados são armazenados no *Registro de Qualidade*.

3.5.3 Registro de Qualidade

Outro componente que faz parte do processo de certificação é o *Registro de Qualidade*. Ele também é externo ao *QSS*. No entanto, nada mais é do que o banco de dados que armazena os valores dos atributos de qualidade que foram monitorados. A tecnologia utilizada para implementar o *Registro de Qualidade* foi o *Apache Derby* [3] *DB*.

A Figura 3.10 ilustra o processo de certificação dos serviços. Quando o componente **Selecionador** usa o **Certificador**, um *Serviço de Certificação* é utilizado para executar a verificação de qualidade dos serviços candidatos. Tal serviço engloba as atividades necessárias para utilizar os dados de atributos de qualidade monitorados, afim de validar o que foi anunciado através dos serviços oferecidos.

No *QSS*, há também a possibilidade de escolher diferentes *Serviços de Certificação* a serem adotados, a fim de obter o mais adequado para empregar os melhores valores dos atributos de qualidade na classificação dos serviços. Por exemplo, enquanto um *Serviço de Certificação* pode utilizar os mais recentes valores de qualidade monitorados do serviço, outro pode atribuir pesos para os dados monitorados, de acordo com seu "*timestamp*".

Após certificar os serviços candidatos vindos do *Registro de Serviços*, uma nova lista de serviços é retornada, pois os serviços não certificados são descartados. Esta lista é passada para o **Selecionador** que aciona o componente **Classificador** para ranquear os serviços e selecionar o que considera mais adequado ao consumidor.

3.6 Classificação de Serviços

O **Classificador**, como o próprio nome sugere, é o componente do *QSS* responsável por determinar quais serviços candidatos melhor se adaptam aos atributos de qualidade requeridos por um consumidor de serviços. Em outras palavras, é responsável por ordenar os serviços candidatos e indicar qual deles deve ser selecionado. Para isso, ele utiliza um *Algoritmo (Estratégia) de Ranqueamento*, que é responsável por ranquear os serviços com base em seus atributos de qualidade. A Figura 3.11 apresenta o **Classificador**.

[3] db.apache.org/derby/

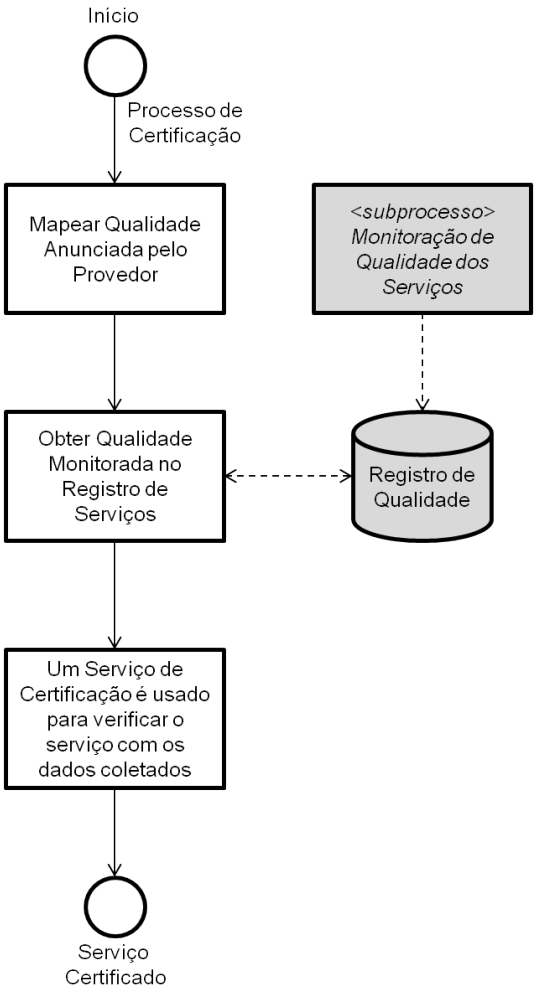

Figura 3.10: Processo de Certificação

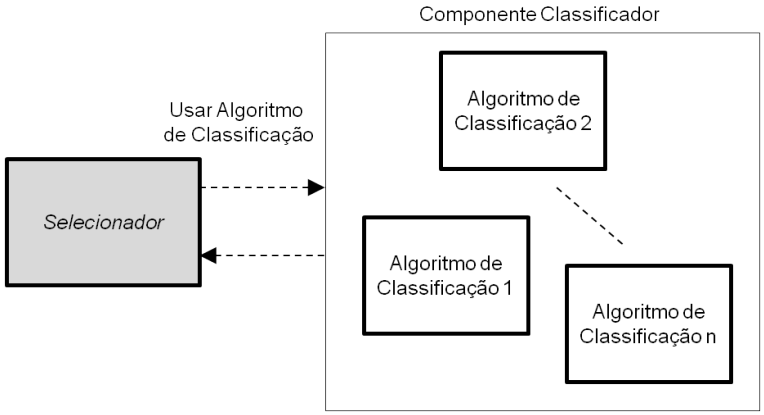

Figura 3.11: Classificador

Como já foi mencionado, o ranqueamento provê um valor quantitativo que mostra a importância relativa de um serviço, no contexto da atual seleção, e define quais serviços devem ser selecionados. Para ranquear os diferentes serviços e alcançar um melhor resultado de desempenho e eficiência, muitos *algoritmos de ranqueamento* foram propostos na literatura [7]. Apesar dessa grande variedade de algoritmos, não há um algoritmo considerado melhor que atenda a todos os fatores e situações anteriormente mencionadas para uma seleção.

Por esta razão, neste trabalho foi proposto o *QSS*, uma solução de seleção de serviços com um diferencial das abordagens atuais que, possuem um único algoritmo de classificação. O *QSS* foi construído com a possibilidade de permitir o uso de vários *Algoritmos de Ranqueamento* e estes podem ser alterados pelo desenvolvedor, em tempo de execução para melhor atender a necessidade da aplicação.

Assim como o **Certificador**, o **Classificador** também é extensível. Cada *algoritmo de ranqueamento* é um serviço interno que pode ser ligado e desligado a qualquer momento, sem afetar o ambiente de execução.

Portanto, assumindo que haja uma coleção de serviços que satisfaz tanto os requisitos funcionais quanto as restrições de atributos de qualidade especificadas pelo consumidor, é necessário determinar o serviço a ser selecionado, isto é, aquele que deve ser utilizado pelo consumidor. Para isso, o *Classificador* é acionado. A Figura 3.12 apresenta o processo de ranqueamento dos serviços.

Como pode ser visto na Figura 3.12, o processo de ranqueamento é composto por uma sequência de três etapas: *normalização, agregação e ordenação*.

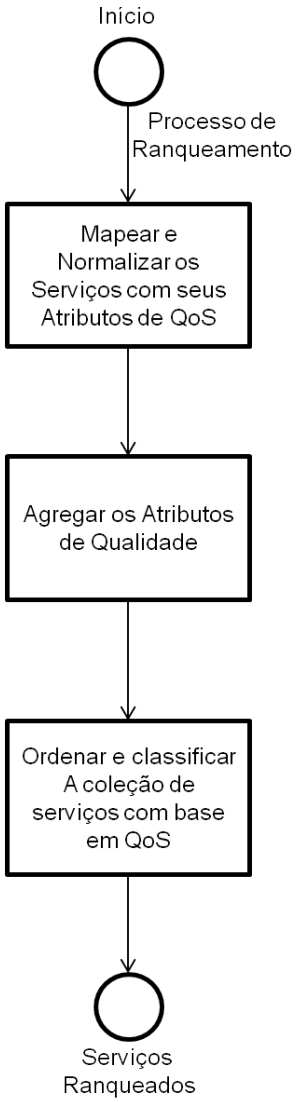

Figura 3.12: Processo de Ranqueamento

3.6.1 Normalização

O primeiro passo para a escolha do serviço é a *normalização* dos atributos de qualidade, de modo que eles possam ser comparados de forma adequada. Na verdade, a *normalização* visa compensar as diferentes escalas de medidas dos atributos, ajustando os valores para o intervalo entre [0,1]. O objetivo é evitar avaliações imprecisas devido à influência de atributos de qualidade com valores grandes.

Para aplicar a *normalização* é necessário considerar a "direção inerente" de cada atributo, que indica se os maiores ou menores valores são os melhores. Se os maiores valores são melhores do que os menores, como a "disponibilidade", deve-se usar a equação 3.1, a fim de calcular o valor do atributo de qualidade normalizado. Se este não for o caso, por exemplo o "tempo de resposta", que quanto menor melhor, a equação 3.2 deve ser usada. Estas equações de normalização foram adotadas, pois são utilizadas também por outros trabalhos relacionados [6][48].

$$Q_{normalizado} = \begin{cases} \frac{Q_{max} - Q_{value}}{Q_{max} - Q_{min}}, \text{se } Q_{max} - Q_{min} \neq 0 \\ 1, \text{se } Q_{max} - Q_{min} = 0 \end{cases} \quad (3.1)$$

$$Q_{normalizado} = \begin{cases} \frac{Q_{value} - Q_{min}}{Q_{max} - Q_{min}}, \text{se } Q_{max} - Q_{min} \neq 0 \\ 1, \text{se } Q_{max} - Q_{min} = 0 \end{cases} \quad (3.2)$$

Onde,

Q designa um atributo de qualidade,

Q_{value}, cada serviço candidato tem um Q_{value} que representa o seu valor para cada atributo de qualidade Q,

Q_{min}, Q_{max} são os valores mínimos e dos atributos de QoS entre todos os serviços candidatos,

$Q_{normalizado}$ representa o valor normalizado do Q_{value}.

Uma vez que os atributos de qualidade foram normalizados, o próximo passo é a agregação dos atributos de qualidade.

3.6.2 Agregação

A *agregação* é responsável por calcular um valor único que resume o nível de qualidade oferecido pelo serviço. Este valor pode ser utilizado para ordenar a coleção de serviços candidatos e assim definir qual serviço deve ser selecionado.

Para agregar os valores de qualidade e obter uma pontuação final para cada serviço, um *algoritmo de ranqueamento* é usado. Cada algoritmo propõe sua maneira para agregar os atributos de qualidade e associar este ao serviço. Por exemplo, um algoritmo pode simplesmente realizar um somatório de todos os atributos de qualidade e selecionar o serviço que possua o maior somatório. Outro pode utilizar uma medida de semelhança e realizar uma comparação entre os atributos de qualidade, requeridos pelo consumidor e oferecidos pelos serviços, e selecionar o serviço que possua a maior semelhança, enquanto que outro pode determinar um *ranking* matemático e selecionar o serviço com base nesse *ranking*.

Portanto, ao receber os serviços certificados por parte do **Certificador**, eles são passados para o **Classificador** que então invoca um *algoritmo de ranqueamento* que ordena os serviços com base em um método de agregação.

Como mencionado anteriormente, o *QSS* é estensível e permite o uso de vários *algoritmos de ranqueamento* e ainda que estes possam ser substituídos para selecionar o serviços mais adequado para o consumidor. Caso queira, o consumidor do *QSS* pode implementar seus próprios *algoritmos de ranqueamento*, através de uma interface bem definida. Como descrito anteriormente, no *QSS* os algoritmos de ranqueamento foram implementados como serviços e a interface definida para este serviço possui uma única operação, que realiza o ranqueamento e retorna uma lista ordenada de serviços. Neste contexto, o desenvolvedor pode implementar o ranqueamento da sua maneira nesta interface e disponbilizar como um serviço para ser consumido pelo *QSS*.

Atualmente o *QSS* incorpora os seguintes algoritmos de ranqueamento:

- ***Simple Additive Weight (SAW)***

  O *SAW* [37][38] é a estratégia mais simples de tomada de decisão de múltiplos atributos e também a mais utilizada. Ela é baseada nas operações básicas de multiplicação e adição para calcular uma média ponderada. Na verdade, uma pontuação é calculada para cada alternativa de serviço candidato através da multiplicação de seus atributos de qualidade com os pesos de importância fornecido pelo consumidor, para cada atributo. Em seguida, é realizado um somatório de todos os atributos. Algumas vantagens do *SAW* é que trata-se de um método simples e

bastante utilizado, além de apresentar bons resultados. Outra vantagem é que são usados pesos, ou seja, os resultados são diretamente afetados pelo consumidor do serviço. Os atributos de qualidade são agregados da seguinte forma:

$$Score(s) = \sum_{i=1}^{n} \varphi i * \omega i \tag{3.3}$$

Onde,

s é o serviço candidato,

n é o número de atributos de qualidade associados ao serviço candidato,

φi é o valor normalizado do i-th atributo de qualidade,

ωi é o peso associado para o i-th atributo de qualidade,

O *Score* é calculado para cada candidato e aquele com a maior pontuação é selecionado.

- ***Euclidean Distance***

  A *Euclidean Distance* [6][39], considerando o contexto de tomada de decisão, tem como objetivo determinar o grau de similaridade entre duas listas de objetos (por exemplo, vetores). Neste contexto, esta estratégia é usada para avaliar a similaridade entre os atributos de qualidade prestados pelos serviços candidatos e os atributos de qualidade requeridos pelo consumidor, e então determinar qual o serviço provê os valores de atributos de qualidade mais semelhantes aos requeridos pelo consumidor para então selecioná-lo.

  Por exemplo, supondo que um consumidor em sua requisição determine que um serviço deva conter o *tempo de resposta* inferior a 30 *ms* e que no registro haja dois serviços, um que apresenta o *tempo de resposta* de 25 *ms* e outro com o *tempo de resposta* de 15 *ms*, caso o algoritmo de *Euclidean Distance* seja utilizado o serviço selecionado será o que possui 25 *ms*, pois mesmo que o outro serviço possua um *tempo de resposta* menor, o serviço com valor de 25 *ms* é o que mais se assemelha ao valor requisitado, que neste caso foi 30 *ms*.

  Neste cenário, os atributos de qualidade dos serviços e dos consumidores são como vetores e é determinada a distância através da raiz quadrada da soma dos quadrados das diferenças entre os elementos correspondentes dos dois vetores:

$$Distance(s) = \sqrt{\sum_{i=1}^{n} \varphi i - \Re i} \tag{3.4}$$

Onde,

s é o serviço candidato,

n é o número de atributos de qualidade associados ao serviço candidato,

φi é o valor normalizado do i-th atributo de qualidade,

$\Re$i é o valor do i-th atributo de qualidade como requerido pelo consumidor,

A *Distance* é calculada para cada serviço candidato e aquele com a menor distância (ou seja, o que é mais similar) é selecionado.

- ***Utility Function***

  No caso da *Utility Function* [24][33][40], o objetivo é utilizar uma função para calcular um valor associado a cada serviço com base em seus atributos de qualidade. No caso da *Utility Function*, é possível adicionar os atributos de qualidade e ainda usar pesos para definir sua importância. O serviço candidato associado ao maior valor deve ser selecionado. A estratégia para agregar os atributos de qualidade representa outra soma ponderada destes, mas neste caso considera a média e o desvio-padrão entre todos os candidatos:

$$Score(s) = \sum_{i=1}^{n} \omega\alpha * \left(\frac{\varphi^{\alpha} - \mu^{\alpha}}{\sigma^{\alpha}} \right) + \sum_{i=1}^{y} \omega\beta * \left(1 - \frac{\varphi^{\beta} - \mu^{\beta}}{\sigma^{\beta}} \right) \tag{3.5}$$

Onde,

s é o serviço candidato,

n é o número de atributos de qualidade que devem ser *maximizado* (*Disponibilidade*),

y é o número de atributos de qualidade que devem ser *minimizados* (*Tempo de Resposta*),

φ_s^{α} é o valor do α-th atributo de qualidade oferecido pelo candidato s,

φ_s^{β} é o valor do β-th atributo de qualidade oferecido pelo candidato s,

μ^{α} é a média do α-th atributo de qualidade entre todos os serviços candidatos,

μ^{β} é a média do β-th atributo de qualidade entre todos os serviços candidatos,

σ^{α} é o desvio padrão do α-th atributo de qualidade entre todos os serviços candidatos,

σ^{β} é o desvio padrão do β-th atributo de qualidade entre todos os serviços candidatos,

ω^{α} é o peso associado para o α-th atributo de qualidade,

ω^{β} é o peso associado para o β-th atributo de qualidade,

O *Score* é calculado para cada serviço candidato e aquele com maior valor é selecionado.

- ***Entropy Function***

  A *Entropy Function* [49][50] é uma estratégia que determina pesos de importância aos critérios utilizados em um processo de decisão. Quanto mais dispersas estão as alternativas, maior a importância do critério na decisão, já que ele possui uma maior capacidade de distinção entre as alternativas. Neste contexto, na solução de seleção de serviços, cada atributo de qualidade é visto como um critério de seleção. Portanto, a idéia é usar a *Entropy Function*, a fim de determinar o peso de cada atributo no processo de seleção de serviços. Depois de obter os pesos para cada atributo de qualidade, eles são somados para obter o serviço a ser selecionado. Uma característica dessa função é a neutralidade na decisão, pois não há influência do consumidor.

  Diferente dos outros *algoritmos de ranqueamento* que foram considerados, o *Entropy Function* foi o único nunca abordado para seleção de serviços. A *Entropy Function* agrega os dados como se segue:

$$\varepsilon(i) = \sum_{i=1}^{n} -\kappa * \varphi i * \log \varphi i \tag{3.6}$$

$$\kappa = \frac{1}{\log i} \tag{3.7}$$

$$D(i) = 1 - \varepsilon(i) \tag{3.8}$$

$$W(i) = \frac{D(i)}{\sum D(i)} \tag{3.9}$$

$$Score(s) = \sum_{i=1}^{n} Wi \tag{3.10}$$

Onde,

s é o serviço candidato,

i é o atributo de qualidade,

n é o número de atributos de qualidade associados ao serviço,

φ_i é o valor do i-th atributo de qualidade,

κ é um valor constante,

ε_i é o valor de *entropia* calculada para o i-th atributo de qualidade,

D_i é o valor de dispersão calculado para o i-th atributo de qualidade,

W_i é o peso de importância calculado para o i-th atributo de qualidade associado a cada serviço candidato,

O *Score* é calculado para cada candidato e aquele com uma maior pontuação é selecionado.

Por fim, após agregar os valores os serviços são ordenados.

3.6.3 Ordenação

A fase de ordenação, como o nome sugere, ordena os serviços com base nos resultados da agregação. Após esta fase, uma lista de serviços ordenada é passada para o *Agente Selecionador* que se encarrega de entregar o serviço adequado ao consumidor.

3.7 Considerações Finais

Este capítulo apresentou a solução para a seleção de serviços baseada em seus requisitos funcionais e nos atributos de qualidade exigidos pelo consumidor, denominada de *QSS*. Em particular, o *QSS* é dinâmico, extensível e automático para selecionar serviços através de uma estratégia de seleção adequada.

4

Avaliação Experimental e Resultados

"O que prevemos raramente ocorre; o que menos esperamos geralmente acontece."

—BENJAMIN DISRAELI

Este capítulo apresenta a avaliação experimental do *QSS*. Inicialmente são apresentados uma visão geral de como foi realizada a avaliação e o ambiente de execução utilizado. Em seguida, os experimentos realizados são apresentados.

4.1 Visão Geral

Para avaliar o *QSS* e as estratégias de ranqueamento por ele adotado foram realizados alguns experimentos. A avaliação experimental foi realizada de acordo com Jain [51] e os experimentos foram realizados no mesmo cenário controlado apresentado em [43], que consiste em uma aplicação orientada a serviços (consumidor), o *QSS* e o registro contendo um conjunto de serviços a serem selecionados pela aplicação.

A aplicação construída foi um sistema de *Homebroker* [52] utilizado para negociação eletrônica de ações que permite a investidores realizarem operações de compra e venda de ações através do site de uma corretora.

Neste trabalho, a aplicação *Homebroker* foi projetada como uma composição de serviços e, para funcionar corretamente, ela precisa consumir outros dois serviços o *Provedor de Informações* e o *Broker*.

O *Provedor de Informações* é o serviço responsável pelo monitoramento contínuo das ações, coletando quantidade de ações e seus preços. E o *Broker* é o serviço responsável por realizar as compras das ações quando estas atingirem a faixa de preço esperada pelo

investidor. A Figura 4.1 apresenta os serviços *Provedor de Informações* e *Broker* sendo consumidos para compor a aplicação *Homebroker* como mencioando.

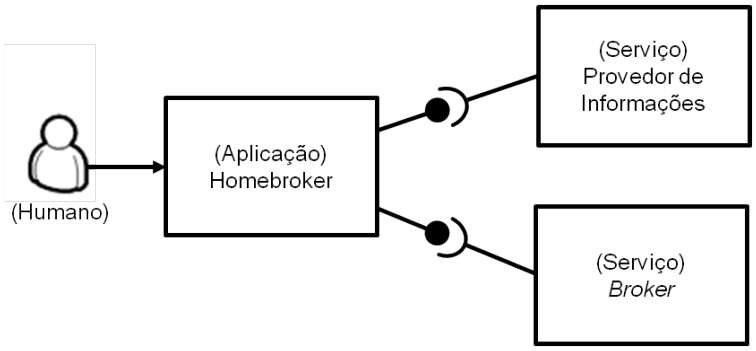

Figura 4.1: Os Serviços Provedor de Informações e *Broker* Consumidos para Compor a Aplicação *Homebroker*.

A aplicação *Homebroker* e os serviços foram desenvolvidos na plataforma *DSOA* (ver Seção 2.1.5). Diante da natureza dinâmica do mercado de ações, onde os preços das ações estão em constantes mudanças, e o número de investidores é cada vez maior, é fundamental que os serviços usados para compor o *Homebroker* tenham um nível de qualidade adequado para atender as demandas dos investidores.

Neste contexto, a aplicação *Homeboker* não seleciona diretamente os serviços. Embora seja possível fazer isto, através da busca padrão do *OSGi*, ela invoca o *QSS*, que deve selecionar os serviços com base nos atributos de qualidade por ele esperados. Por exemplo, que os serviços tenham uma disponibilidade alta e um tempo de resposta baixo.

Para validar o *QSS*, ele será utilizado pela a aplicação *Homebroker* para selecionar os serviços de *Provedor de Informações*. Porque, de fato, selecionar um serviço que provê informações de preço de ações, e este último não possuir uma boa qualidade, por exemplo, demorar a responder ou não estar operacional no momento da sua solicitação, pode acontecer dos preços das ações terem os valores alterados, já que o mercado de ações é bastante dinâmico e os preços das ações mudam constantemente. Neste contexto, o consumidor pode ter prejuízo na compra de ações, devido a má qualidade do serviço *Provedor de Informações* utilizado que não informou ou demorou para informar os preços das ações. Por isso, espera-se que estes serviços possua uma boa qualidade, ou seja, estejam sempre disponíveis e consigam responder às requisições em tempo desejável, com a rapidez necessária para que o *Homebroker* possa realizar a compra antes de qualquer alteração.

Portanto, é necessário que os serviços responsáveis por prover informações de preços de ações contenham um nível adequado de qualidade. Na verdade, estes serviços devem

possuir valores adequados dos seus atributos de qualidade, principalmente os ligados a desempenho. Assim, os consumidores (aplicações) poderão selecionar o serviço *Provedor de Informações* com base nos seus atributos de qualidade, tais como um serviço que possua uma taxa de disponibilidade alta e um tempo de resposta baixo.

4.1.1 Ambiente de Execução

Para realizar os experimentos foi montado um ambiente de execução que consiste na aplicação (*Homebroker*), o *QSS* e o registro de serviços contendo os serviços a serem selecionados.

Os experimentos foram realizados em um Computador PC Intel Core i7 de 2.30 GHz, com 8GB de memória RAM, rodando o Windows 7 Professional de 64 Bits com Service Pack 1. O Java JRE versão 1.6.0_31, é utilizado para implantar o ambiente *OSGi Apache Felix* [1].

Como já mencionado, o *QSS* permite o uso de diferentes algoritmos (estratégias) de ranqueamento para determinar qual serviço deve ser selecionado. Nesta avaliação são considerados quatro algoritmos de ranqueamento: *Simple Additive Weighting*, *Euclidean Distance* e *Utility Function*, e o *Entropy Function*. Os três primeiros algoritmos foram utilizados, pois são amplamente adotados nos trabalhos relacionados (ver Seção 2.2) e o algoritmo *Entropy Function* é uma adoção original na seleção de serviços e representa uma contribuição deste trabalho. Além disso, o algoritmo de seleção padrão do *OSGi*, em que um serviço é selecionado aleatoriamente, também é analisado para que uma comparação seja realizada.

Neste contexto, a partir dos experimentos é possível observar o funcionamento do *QSS* utilizando diferentes algoritmos de ranqueamento para selecionar o serviço para a aplicação. Desta forma, podemos verificar a eficácia dos algoritmos, observando quais serviços são selecionados, bem como o desempenho da solução na seleção dos serviços, mesmo considerando diferentes algoritmos.

Para um melhor entendimento dos experimentos, estes foram decompostos em quatro partes distintas: o objetivo, a descrição do cenário de uso, a execução e os resultados obtidos. O objetivo é uma descrição mais clara de cada experimento e um melhor entendimento dos experimentos realizados. Neste contexto, para analisar o desempenho do *QSS*, bem como as estratégias de classificação por ela utilizada. São consideradas duas métricas: a eficácia dos algoritmos de ranquemamento na seleção dos serviços e o desempenho do *QSS* como um todo.

[1] http://felix.apache.org

A métrica de eficácia foi considerada, pois representa a relação entre o resultado obtido na seleção dos serviços e o objetivo pretendido, neste caso, se os serviços selecionados são de fato mais adequados para o algoritmo utilizado. E a métrica de desempenho foi considerada com o objetivo de verificar o rendimento do *QSS* para selecionar os serviços, mesmo considerando e substituindo diferentes algoritmos de ranqueamento. Deste modo, é possível determinar se um algoritmo causa mais impacto que outro na solução de seleção dos serviços, se o *QSS* tem um bom desempenho e assim por diante.

4.2 Avaliação da Eficácia dos Algoritmos de Ranqueamento

4.2.1 Objetivos

O primeiro experimento tem como objetivo avaliar o *QSS* considerando a métrica de eficácia. O foco neste experimento é verificar se os algoritmos de ranqueamento utilizados pelo *QSS* são capazes de selecionar o serviço adequado ao consumidor.

Na prática, é necessário analisar a eficácia observando quais são os serviços selecionados pelos diferentes algoritmos de ranqueamento utilizados, levando em conta os atributos de qualidade especificados pelo *Consumidor*.

Espera-se que os algoritmos atendam ao seu propósito de selecionar o melhor ou o serviço mais adequado ao consumidor, dependendo da estratégia de ranqueamento utilizada e dos valores dos atributos de qualidade dos serviços candidatos. Este experimento é importante pois, desta forma, é possível determinar dentre os algoritmos implementados aqueles que são eficazes, e, portanto, adequados a serem utilizados.

4.2.2 Cenário

A ilustração do cenário para a avaliação da eficácia é apresentada na Figura 4.2. Como já foi dito, o foco dos experimentos será sempre na ligação entre o *Homebroker* (aplicação), o QSS e o serviço *Provedor de Informações*. Neste caso, o *Homebroker* invoca o *QSS* para selecionar os serviços em um registro. Todos os serviços apresentam a mesma funcionalidade (*Provedor de Informações*), no entanto, possuem diferentes valores dos atributos de qualidade.

Como o objetivo é determinar a eficácia dos algoritmos de ranqueamento analisando se eles selecionam os serviços adequados para o consumidor, os parâmetros de carga

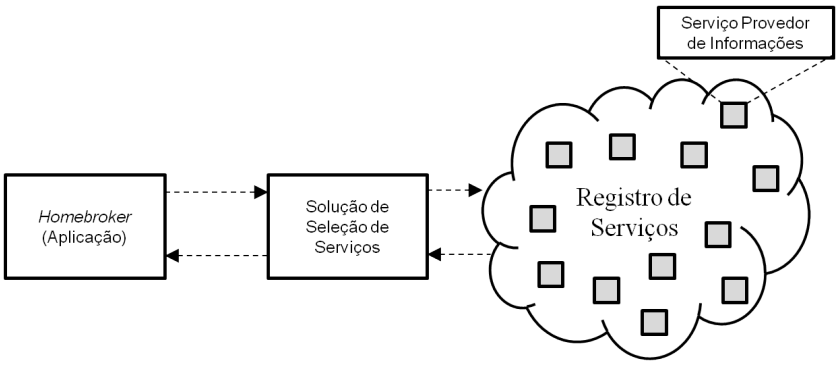

Figura 4.2: Cenário para Avaliação da Eficácia dos Algoritmos de Ranqueamento.

de trabalho que foram utilizados para este experimento foram o número de serviços candidatos, limitados em dez e o número de atributos de qualidade, que foram variados entre um e cinco.

Neste contexto, parâmetros de carga de trabalho são aspectos representativos no uso real de uma aplicação, ou seja, são aspectos comuns a aplicação, que causam impactos significativos sobre os resultados do seu funcionamento.

Estes parâmetros foram utilizados, pois a ideia é saber apenas quais serviços são selecionados e, se estes de fato são os serviços mais adequados dados os atributos de qualidade dos serviços e o algoritmo de ranqueamento adotado.

Por isso, para avaliar a eficácia é necessário que haja um conjunto de serviços que apresente a mesma funcionalidade e que estes serviços possuam os atributos de qualidade requeridos pelo consumidor, mas que os valores desses atributos sejam diferentes. Desta forma, é possível determinar se os serviços selecionados são os apropriados para a requisição do consumidor.

Para implementar os serviços foi utilizado um simulador, que é responsável por implementar serviços e que permite especificar diferentes atributos de qualidade, estes últimos especificados pelo desenvolvedor. Através deste simulador foram criados dez serviços candidatos que implementam a mesma funcionalidade (*Provedor de Informações*), possuem também os mesmos atributos de qualidade, mas apresentam valores diferentes para estes atributos.

A quantidade de serviços criados limitou-se a dez, pois, para este primeiro experimento, a quantidade de serviços não influencia no resultado, já que o objetivo é a qualidade do serviço selecionado com base nos atributos especificados.

Para realizar a execução, o número de atributos de qualidade foram variados de um a cinco, como adotado em [23][34][53]. O objetivo de utilizar vários atributos de qualidade é permitir que um serviço possua um atributo de qualidade melhor que outro serviço e vice-versa. Com isso, será observado qual serviço é selecionado considerando os diferentes algoritmos de ranqueamento.

4.2.3 Execução

Inicialmente, o consumidor (aplicação *Homebroker*) especifica um conjunto de cinco atributos de qualidade que ele espera que serviço *Provedor de Informações* possua: *Tempo de Resposta*, *Vazão*, *Precisão*, *Disponibilidade* e *Custo*. Da mesma forma, são especificados os seus respectivos valores de referência que cada serviço deve atender para se tornar um serviço candidato. A Tabela 4.1 mostra os atributos de qualidade especificados pelo *Homebroker* e os valores de referência esperados para cada atributo de qualidade. Cada coluna da tabela identifica um atributo de qualidade.

Atributos de Qualidade					
Consumidor	**Tempo de Resposta (*ms*)**	**Vazão (*req/s*)**	**Precisão (%)**	**Disponibilidade (%)**	**Custo ($)**
Valores	999	10	85	80	5

Tabela 4.1: Atributos de Qualidade Espeficidados pela Aplicação *Homebroker*.

O *QSS*, como já foi mostrado na Seção 3.4, inicialmente constrói um filtro com os atributos de qualidade especificados pela aplicação e acessa o registro de serviços para encontrar os serviços que atendem a tal especificação. Os serviços que atendem são considerados serviços candidatos.

Os dez serviços candidatos, que foram implementados através do simulador, atendem aos atributos de qualidade requeridos pela aplicação e estes estão presentes no registro de serviços, juntamente com seus atributos de qualidade.

A Tabela 4.2 apresenta os serviços e seus respectivos atributos de qualidade. Nesta tabela, cada coluna representa um atributo de qualidade e cada linha representa um serviço candidato.

Atributos de Qualidade					
Consumidor	**Tempo de Resposta (*ms*)**	**Vazão (*req/s*)**	**Precisão (%)**	**Disponibilidade (%)**	**Custo ($)**
Serviço 1	700	20	89	85	1
Serviço 2	900	10	92	95	1.2
Serviço 3	915	28	90	80	0.5
Serviço 4	910	12	96	98	2.5
Serviço 5	390	28	99	99	5.0
Serviço 6	950	10	95	100	0.9
Serviço 7	600	25	90	95	2.0
Serviço 8	500	25	98	93	1.75
Serviço 9	998	15	85	83	0.0
Serviço 10	545	22	100	90	0.3

Tabela 4.2: Os Serviços Candidatos e Seus Respectivos Atributos de Qualidade.

A partir dos dados apresentados nas Tabelas 4.1 e 4.2 e considerando o cenário descrito, o primeiro experimento foi executado vinte e cinco vezes. Este experimento teve esta quantidade de vezes executada, pois é a quantidade necessária para avaliar os cinco algoritmos de ranqueamento utilizados e os cinco atributos de qualidade considerados.

Desta forma, os cinco algoritmos de ranqueamento foram executados cinco vezes, onde foi considerada a adição de um novo atributo de qualidade a cada execução. O número de atributos de qualidade utilizados no total foram cinco, neste contexto, foram utilizados os mais comuns na área de seleção de serviços, além disso, considerou-se este número de atributos de qualidade, pois diferencia das abordagens atuais que geralmente consideram a variação de três atributos de qualidade.

Deste modo, considerando que foram utilizados cinco algoritmos de ranqueamento e que cada algoritmo foi executado cinco vezes para considerar o uso dos atributos de qualidade.

Neste contexto, foram utilizados os quatro algoritmos de ranqueamento implementados pela solução (ver Seção 4.1.1), além do algoritmo *Default* do *OSGi*.

A execução do experimento foi da seguinte forma: Inicialmente um algoritmo, por exemplo, o *Simple Additive Weighting* foi executado considerando apenas um atributo de qualidade (*Tempo de Resposta*) e foi observado qual serviço foi selecionado, depois o mesmo algoritmo, *Simple Additive Weighting* foi executado considerando dois

atributos de qualidade (*Tempo de Resposta* e *Disponibilidade*) e foi observado qual serviço foi selecionado, após dois atributos, foram considerados três atributos de qualidade (*Tempo de Resposta*, *Disponibilidade* e *Vazão*) e também foi analisado qual o serviço foi selecionado. Após três, foram considerados quatro atributos de qualidade (*Tempo de Resposta*, *Disponibilidade*, *Vazão* e *Precisão*) e novamente foi verificado qual serviço foi selecionado, por fim, foram considerados cinco atributos (*Tempo de Resposta*, *Disponibilidade*, *Vazão*, *Precisão* e *Custo*) e finalmente foi analisado o serviço selecionado.

Deste mesmo modo os demais algoritmos (*Euclidean Distance*, *Utility Function* e *Entropy Function*) foram executados, inclusive o *Default* e foram observados quais serviços foram sendo selecionados.

Na próxima seção são apresentados os resultados da avaliação da eficácia dos algoritmos de ranqueamento. Para um melhor entendimento dos resultados é importante ressaltar que a ordem de inclusão dos atributos de qualidade apresentada na execução deve ser considerada.

4.2.4 Resultados

A Tabela 4.3 apresenta os resultados do experimento de avaliação da eficácia dos algoritmos de ranquamento. O objetivo é analisar quais os serviços são selecionados quando a solução de seleção está ativada e usando diferentes algoritmos de ranqueamento.

	Número de Atributos de Qualidade				
Algoritmos de Ranqueamento	**1**	**2**	**3**	**4**	**5**
Default	Serviço 4	Serviço 1	Serviço 3	Serviço 5	Serviço 1
SimpleAdditiveWeight	Serviço 5	Serviço 5	Serviço 5	Serviço 5	Serviço 5
Distancia Euclidiana	Serviço 9	Serviço 3	Serviço 9	Serviço 9	Serviço 9
Função de Utilidade	Serviço 9	Serviço 6	Serviço 3	Serviço 5	Serviço 5
Função de Entropia	-	Serviço 5	Serviço 5	Serviço 5	Serviço 5

Tabela 4.3: Resultado da Avaliação da Eficácia dos Algoritmos de Ranqueamento

Quando o *QSS* está desabilitada, o serviço é selecionado aleatoriamente e o melhor serviço nem sempre é o escolhido. A partir da combinação de valores da Tabela 4.2 é possível inferir, por observação, que em geral, o serviço que apresenta os melhores valores para os atributos de qualidade requeridos pelo *Homebroker* é o *Serviço 5*. E este último foi selecionado apenas uma vez. Além disso, o *Serviço 3* foi selecionado, mesmo ele apresentando atributos de qualidade com valores muito inferiores aos demais, como

tempo de resposta mais elevado e uma disponibilidade muito baixa (ver linha *Default*).

Esta situação acontece porque, sem o *QSS*, os atributos de qualidade dos serviços não são considerados. Na realidade, neste caso, o que acontece é que o consumidor (*Homebroker*) desconsidera o *QSS* e usa o algorimo de busca do próprio *OSGi* (*Default*) que seleciona os serviços de forma aleatória.

Considerando o uso do *QSS* e para uma melhor explanação dos resultados, a quantidade de atributos de qualidade durante a execução, bem como, a ordem de qual atributo de qualidade é incluído na execução deve ser considerado.

Para a execução com um atributo de qualidade, foi considerado o atributo de *Tempo de Resposta*, neste caso o melhor serviço é o *Serviço 5*, que apresenta o menor tempo de resposta, e o serviço que apresenta o pior valor é o *Serviço 9*, no entanto, é o valor mais próximo ao requerido pelo *Homebroker* (ver Tabelas 4.1 e 4.2).

Com dois atributos de qualidade considerados na execução, foram adotados os atributos de *Tempo de Resposta* e *Disponibilidade*. Neste caso, o *Serviço 5* continua sendo o melhor serviço, no entanto, a combinação dos atributos mostra que o serviço que mais se assemelha ao valores requisitados pelo consumidor agora é o *Serviço 3*.

Quando são considerados três atributos de qualidade na execução, os atributos adotados são o *Tempo de Resposta*, a *Disponibilidade* e a *Vazão*. Neste caso, o *Serviço 5* se manteve como a melhor opção e o *Serviço 9* voltou a ser o mais semelhante ao requisitado pelo consumidor. Além disso, nesta combinação o *Serviço 3* passou a ser o serviço com a menor qualidade.

Para considerar quatro atributos de qualidade na execução, mateve-se os três anteriores (*Tempo de Resposta*, *Disponibilidade* e *Vazão*) e foi adicionado o atributo de *Precisão*. O *Serviço 5* continua sendo a melhor opção, o *Serviço 9* continua sendo o mais semelhante ao consumidor, enquanto que o *Serviço 3* continua a ser o serviço que apresenta a pior qualidade dentre todos.

Por fim, para considerar cinco atributos de qualidade na execução, além dos quantro já considerados, foi adicionado o atributo de *Custo*. Neste caso, foi mantida as opções de escolhas dos serviços. O *Serviço 5* segue como a melhor opção, o *Serviço 9* continua sendo o mais semelhante ao consumidor, e o *Serviço 3* continua a ser o serviço com a pior qualidade.

A partir deste contexto, é possível observar a Tabela 4.3 e inferir alguns resultados. Pordemos observar, por exemplo, que o algoritmo *SimpleAdditiveWeight* pode ser considerado sempre a melhor opção, porque é mais eficaz ao selecionar sempre o melhor serviço (*Serviço 5*).

O algoritmo de *Euclidean Distance* também mostrou ser eficaz, diferente dos demais que tentam selecionar sempre o melhor serviço, ele seleciona o serviço com os valores de atributos de qualidade mais semelhantes aos valores de referência dos atributos de qualidade requeridos pelo consumidor. Quando foi considerado dois atributos de qualidade ele selecionou corretamente o *Serviço 3* e posteriormente continuou eficaz selecionando o *Serviço 9*.

O algoritmo *Utility Function* mostrou-se ineficaz, pois selecionou quatro serviços diferentes no decorrer da sua execução como poder ser observado na Tabela 4.3. Inclusive quando foram considerados três atributos de qualidade distintos o serviço selecionado foi o *Serviço 3*, que neste caso representa um serviço ruim, com valores de atributos de qualidade inadequados para a sua seleção.

Já o algoritmo *Entropy Function*, proposto por este trabalho, mostrou-se bastante eficaz, selecionando sempre o *Serviço 5*, considerado o melhor serviço entre os candidatos. Este resultado é bastante satisfatório, pois são resultados semelhantes aos apresentados pelo algoritmo *SimpleAdditiveWeight*, que é amplamente adotado como anteriormente dito.

No entanto, o *Entropy Function* apresenta uma restrição, que é a incapacidade de ser aplicado em cenários onde é considerado apenas um atributo de qualidade na seleção. Isto porque como apresentado na Seção 3.6.2, ele determina pesos de importância para cada atributo de qualidade e partir da soma destes pesos ele determina qual serviço deve ser selecionado. Como é considerado apenas um atributo, um único peso é determinado para todos os serviços, por isso, não é possível diferenciar qual serviço deve ser selecionado.

Por fim, apesar de apresentar bons resultados na seleção de serviços, o *Entropy Function*, por ser um novo algoritmo para a área de seleção de serviços, precisa ser melhor analisado, considerando novos cenários de seleção, com mais atributos de qualidade e composição de aplicações com mais serviços.

4.3 Avaliação de Desempenho do QSS

4.3.1 Objetivos

O objetivo do segundo experimento é avaliar o desempenho do QSS, para isso, a métrica adotada foi o *Tempo de Seleção de Serviços*. Este tempo, representa o tempo decorrido entre a requisição de um serviço feita pela aplicação *Homeboker* e o fornecimento do serviço mais adequado pela solução de seleção proposta (ver Figura 4.3).

A métrica *Tempo de Seleção de Serviços* foi escolhida para este experimento, porque ela representa todo o período de execução do QSS. Ela abrange todas as etapas de funcionamento da solução proposta: inicia ao receber a requisição do consumidor para selecionar um serviço, passando pelas etapas de seleção, certificação e classificação dos serviços e finaliza na entrega do serviço adequado ao consumidor.

4.3.2 Cenário

O cenário para o segundo experimento é ilustrado na Figura 4.3. Semelhante ao primeiro, o segundo experimento consiste no *Homebroker* invocando a solução de seleção para selecionar os serviços (*Provedor de Informações*) em um registro de serviços.

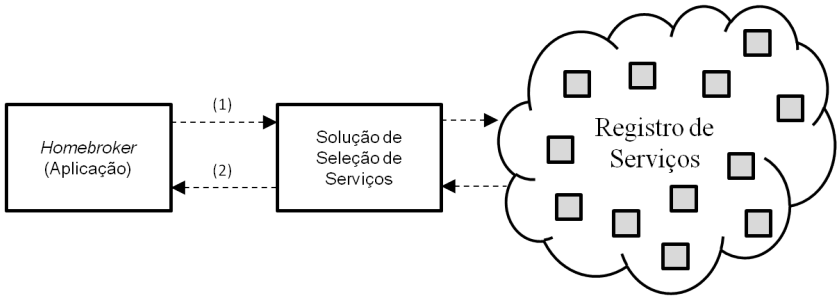

Figura 4.3: Cenário para Avaliação de Desempenho do QSS.

Para avaliar o desempenho da solução proposta foi necessário utilizar parâmetros que são comuns ao funcionamento da solução e que causam impactos significativos no seu desempenho. Por isso, os parâmetros adotados foram os algoritmos de ranqueamento implementados pela solução, o número de atributos de qualidade e a quantidade de serviços candidatos.

Com estes parâmetros é possível medir os aspectos ligados ao desempenho da solução proposta. Semelhante ao primeiro experimento, o número de atributos de qualidade foi variado entre um e cinco. Além disso, o número de serviços candidatos (*Provedor de Informações*) foi fixado em quinhentos. Os valores destes parâmetros foram definidos de tal forma que extrapolassem os cenários atuais [6][23][32][33] que geralmente consideram até três atributos de qualidade e até dez serviços candidatos. Além disso, estes valores foram adotados para causar impacto significativo no desempenho da solução proposta no experimento.

Assim, com a variação dos atributos de qualidade, uma quantidade significativa de serviços candidatos e a substituição dos algoritmos de ranqueamento em tempo de execução pode-se verificar o desempenho do *QSS*, considerando o tempo de seleção de

serviços com cada algoritmo de ranqueamento implementado.

Além disso, foi considerado o cenário em que o *QSS* estava desativado e, portanto, o algoritmo *Default* da plataforma OSGi é usado. Neste contexto, é possível comparar o tempo gasto pelo *QSS* para selecionar um serviço com base em atributos de qualidade e com diferentes algoritmos de ranqueamento e o tempo gasto para selecionar o serviço sem o *QSS*, onde os serviços são selecionados aleatoriamente.

4.3.3 Execução

Na Figura 4.3 são mostrados o começo e o fim do tempo de seleção de um serviço através do *QSS*. O tempo de execução utilizando o *QSS* começa a ser medido a partir da sua invocação pela aplicação consumidora de serviços (*Homebroker*) (1), e continua através dos procedimentos de descoberta, certificação, e classificação dos serviços, que são executados pelo *QSS*. Quando um serviço (ou conjunto de serviços) é fornecido para o consumidor, a medição é parada (2).

Neste segundo experimento foram realizadas mil medições para cada variação de parâmetro de carga de trabalho (atributo de qualidade e algoritmos de ranqueamento). Por exemplo, usando o algoritmo de *Euclidean Distance* e um atributo de qualidade, o tempo de seleção foi medido mil vezes, depois para dois atributos de qualidade mais mil vezes, e assim sucessivamente até terminar os cinco atributos de qualidade. Depois foi substituído o algoritmo de ranquamento e as mesmas medições foram executadas até finalizar todos os algoritmos implementados.

Segundo Jain [51] a determinação do tamanho do intervalo de confiança para o tempo médio real depende do tamanho da amostra considerada. E, quanto maior a amostra, menor o intervalo de confiança e mais preciso o resultado. Neste contexto, com a amostra de mil medições e, através da formulação matemática apresentada em [51] foi determinado que para estimar a média populacional com precisão com uma margem de erro de 5% e um nível de confiança de 95% o número de observações requeridas é 297. No entanto, mesmo que a quantidade observações para amostra tenha sido definida, bem como, que amostras grandes requeiram maior esforço e quantidade de recursos, foi considerado toda a amostra de mil medições para obter resultados mais precisos.

Após realizar todas as execuções, foi considerado o tempo médio de execução das mil medições para cada amostra. Por exemplo, foram coletadas as mil medições do tempo de seleção para o algoritmo de *Euclidean Distance* com um atributo de qualidade, fez-se a média, guardou-se o resultado, depois foram coletadas as mil medições do algoritmo de *Euclidean Distance* com dois atributos de qualidade, foi calculada a média e assim

sucessivamente até completar todos os atributos de qualidade considerados e todos os algoritmos de ranqueamento implementados.

Os resultados são apresentados na próxima subseção.

4.3.4 Resultados

A Figura 4.4 mostra o *Tempo de Seleção de Serviços* quando a solução de seleção de serviços proposta está ativada e desativada (*Default*). Apesar de uma mínima diferença entre os algoritmos de ranqueamento, é possivel perceber que a introdução de um processo de ranqueamento incrementou o tempo médio em aproximadamente 40 *ms* para selecionar um serviço.

Isto significa que a introdução de uma solução para selecionar os serviços e o uso de uma boa estratégia de ranqueamento é muito interessante, uma vez que a aplicação será capaz de selecionar um serviço adequado, que atenda aos seus atributos de qualidade, sem muito impacto para o seu próprio desempenho. Por exemplo, passar algum tempo a fim de selecionar o serviço que oferece o tempo de execução mais curto será certamente melhor do que utilizar um serviço que foi selecionado aleatoriamente e que pode conter um tempo de execução elevado.

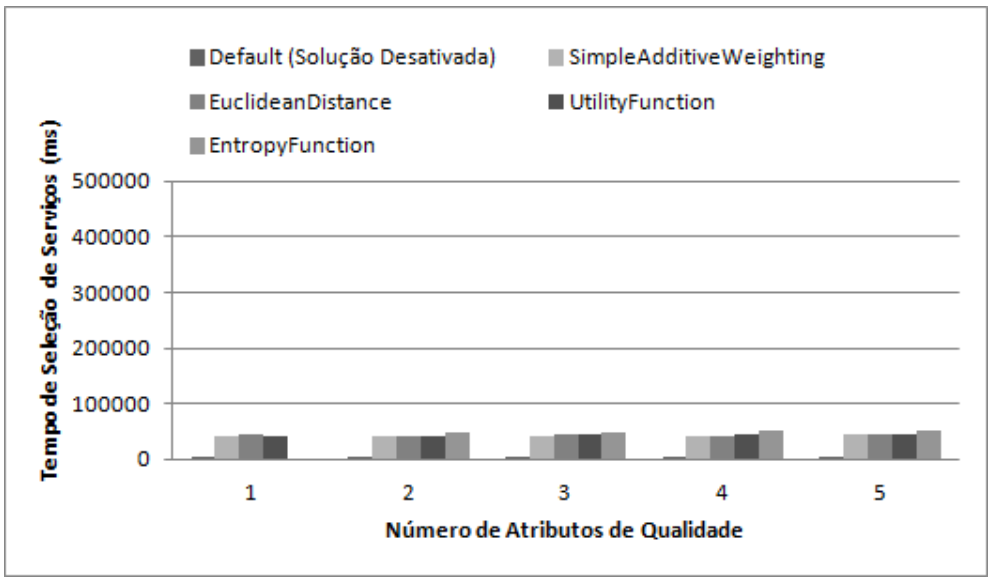

Figura 4.4: Tempo de Seleção de Serviços x Número de Atributos de Qualidade com e sem o QSS

Uma vez ativada a solução de seleção, os serviços são ranqueados com base em seus atributos de qualidade e os serviços mais adequados são selecionados com base em seu *ranking*.

Os *Tempos de Seleção de Serviços* são muito semelhantes e crescem linearmente com a carga de trabalho, independentemente do algoritmo de ranqueamento utilizado, como mostra a Figura 4.4. Esta é uma indicação de que o algoritmo de ranqueamento de serviços não tem impacto significativo sobre o desempenho da seleção de serviços, permitindo que diferentes algoritmos sejam utilizados para selecionar um serviço adequado. Ou seja, independente do algoritmo de ranqueamento utilizado pelo QSS, este ultimo manteve o seu desempenho constante.

De acordo com a avaliação experimental, além dos resultados apresentados ao longo do capítulo, é possível também observar algumas informações relevantes a respeito do QSS, por exemplo, mesmo que o QSS passe mais tempo para classificar e selecionar um serviço, selecionar este com base em seus atributos de qualidade pode ser uma vantagem em termos de qualidade para a aplicação como um todo. Por exemplo, supondo que o tempo gasto a fim de selecionar um serviço que tenha um tempo de execução baixo seja melhor do que o tempo necessário para executar o serviço que tenha sido selecionado aleatoriamente. Neste caso, é melhor utilizar o QSS. A razão é que o serviço selecionado aleatoriamente poderia ter um tempo de execução superior.

4.4 Considerações Finais

Neste capítulo o QSS foi avaliado. Para isso, foram avaliadas as métricas de eficácia e desempenho. Com a eficácia foi avaliado quais serviços são selecionados pelo QSS e seus algoritmos de ranqueamento e com o desempenho é observado seu impacto no desempenho das aplicações. Como carga de trabalho para a avaliação foram selecionados as métricas de quantidade de atributos de qualidade e o número de serviços candidatos. Além disso, foram utilizados quatro algoritmos de ranqueamento de serviços e o *default* que seleciona os serviços aleatoriamente.

5

Conclusão e Trabalhos Futuros

"Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar. Mas o mar seria menor se lhe faltasse uma gota."

—MADRE TERESA DE CALCUTA

Este capítulo apresenta as conclusões e as principais contribuições do presente trabalho, além dos seus aspectos mais relevantes. Ademais, são destacadas as limitações existentes e os trabalhos futuros a serem realizados dentro desta linha de pesquisa.

5.1 Conclusão

Devido ao surgimento de uma grande quantidade de serviços que fornecem a mesma funcionalidade, a escolha de um serviço para fazer parte de uma aplicação tornou-se uma tarefa muito difícil. Por isso, o processo de seleção de serviços que utiliza outros fatores além da funcionalidade é um tema atual e bastante abordado na academia e na indústria.

Geralmente, o fator utilizado na seleção dos serviços além de sua funcionalidade são os atributos de qualidade desses serviços. Ou seja, para refinar as abordagens tradicionais, que normalmente só consideram a funcionalidade do serviço na seleção, outros fatores, tais como os atributos de qualidade, são necessários para determinar quais serviços devem ser selecionados.

Diante desse contexto, o presente trabalho propõe, projeta e implementa o *QSS*, uma solução para a seleção de serviços que leva em conta não só a funcionalidade dos serviços, mas também seus atributos de qualidade. O objetivo foi selecionar o melhor, ou mais adequado, candidato quando serviços funcionalmente similares são publicados em um registro de serviços.

Em particular, o *QSS* é extensível, dinamicamente adaptável e pode ser reconfigurada em tempo de execução sem afetar a execução de outros serviços.

5.2 Contribuições

As principais contribuições realizadas com o desenvolvimento deste trabalho são apresentadas a seguir:

- **Seleção de serviços ciente de qualidade para aplicações orientadas a serviços**

  O *QSS* permite ao consumidor selecionar os serviços com base não apenas na sua funcionalidade, mas também com base nos atributos de qualidade desse serviços. Ela permite a substituição dinâmica (em tempo de execução) dos algoritmos para ranquear os serviços usando atributos de qualidade.

  A substituição dinâmica desses algoritmos é uma originalidade do *QSS* e foi motivada por alguns fatores, incluindo: a diversidade dos atributos de qualidade, a necessidade de dar prioridade a um atributo particular, mudanças nas preferências do consumidor, e assim por diante. Desta forma, a partir da necessidade do consumidor, um algoritmo adequado pode ser escolhido.

- **Algoritmos para a seleção de serviços com base em atributos de qualidade**

  Outra contribuição relevante obtida com o desenvolvimento deste trabalho foi a implementação e comparação de diferentes algoritmos para ranquear e selecionar os serviços com base em seus atributos de qualidade. Devido a capacidade do *QSS* de substituí-los dinamicamente, foram implementados quatro algoritmos: *SimpleAdditiveWeight*, *Euclidean Distance*, *Utility Function* e *Entropy Function*. Sendo os três primeiros algoritmos amplamentes adotados na literatura e a adoção do quarto algoritmo, que é uma contribuição original deste trabalho.

  O *SimpleAdditiveWeight* e o *Euclidean Distance* se mostraram interessantes para seleção de serviços. Já que apresentaram resultados bons, selecionando os melhores ou mais adequados serviços com base nos atributos de qualidade requeridos pelo consumidor.

  O *Utility Function* apresentou resultados muito variáveis, certas vezes seleciou serviços com atributos de qualidade ruins, se mostrando um algoritmo não muito adequado.

Finalmente, o *Entropy Function* apresentou ser bastante adequado para a seleção de serviços, pois selecionou os serviços mais adequados com base nos atributos de qualidade requeridos pelo consumidor. No entanto, considerando que é um novo algoritmo, apresentou algumas restrições, tais como um maior tempo na seleção dos serviços, e a impossibilidade de selecionar, quando apenas um atributo de qualidade é considerado. Por estas razões, mais experimentos devem ser realizados, para mais testes e uma melhor análise do mesmo.

- **Suporte à utilização de atributos de qualidade na publicação e requisição de serviços**

  Usar atributos de qualidade na seleção de serviços também pode ser considerada uma contribuição do trabalho proposto. Embora haja algumas soluções que fazem o mesmo, para propor uma solução que utilize atributos de qualidade na seleção elaborou-se junto com a plataforma *DSOA*, modelos de qualidade e serviços que permitiram especificar uma linguagem para da suporte aos provedores e consumidores especificarem seus atributos de qualidade.

5.3 Limitações

Embora o *QSS* atenda a importantes requisitos para a seleção de serviços com base em atributos de qualidade, é necessário destacar algumas limitações:

- **Adaptação dinâmica e automática do *QSS***

  Atualmente, a substituição dinâmica das estratégias de classificação é realizada de forma semi-automática, isto porque, ela é realizada em tempo de execução, mas a entidade que determina quando o algoritmo deve ser trocado é um ser humano (desenvolvedor).

  Para tornar automática é necessário que um componente de software seja o responsável por essa substituição (adaptação). No entanto, este componente precisa estar ciente de onde e quando deve modificar o programa, ou seja, ele deve entender todo o funcionamento e infraestrutura d *QSS* (por exemplo, o funcionamento da solução e possíveis falhas que possam levar a seleção errônea de solução), e possuir um tomador de decisão para realizar a adaptação a partir destes entendimentos [54]. A falta deste componente é uma limitação deste trabalho.

- **Algoritmos para a seleção de serviços com base em atributos de qualidade**

  Outra limitação são os algoritmos de ranqueamento utilizados. Foram considerados quatro algoritmos e todos são baseados em estratégias que utilizam funções matemáticas. Para um melhor estudo comparativo são necessário outros algoritmos e com diversidades de técnicas, tais como, orientação a contexto, algoritmo genético e algoritmos evolutivos semelhante aos apresentados em [25][55].

5.4 Trabalhos Futuros

A partir da análise das limitações apresentadas, diversas melhorias podem ser elaboradas, e algumas destacadas aqui como trabalhos futuros:

- **Estudos de ténicas de adaptação e aspectos que influenciam no funcionamento da seleção de serviços com base em qualidade.**

  Para propor um mecanismo capaz de tomar a decisão de substituir o algoritmo de ranqueamento e permitir a adaptação automática é necessário um estudo das técnicas de adaptação como as apresentadas em [54].

  Sendo assim, é necessário que se trabalhe na implementação de um mecanismo que se adapte com base na quantidade de quebras de contrato entre o consumidor e o provedor de serviços.

- **Algoritmos para a seleção de serviços com base em atributos de qualidade.**

  Outra possivel extensão para o *QSS*, seria a implementação de algoritmos de ranqueamento adicionais, incluindo idéias relacionadas à orientação a contexto, algoritmo genético e algoritmos evolutivos, como os apresentados em [25] [55]. Entretanto, é também necessário avaliar o impacto do *QSS* e dos algoritmos existentes em diferentes cenários.

- **Expansão do *QSS*.**

  Esta sendo planejado a expansão do *QSS* utilizando algum motor de orquestração, por exemplo, *Apache ODE* [1], que adota uma forma particular de composição de serviços.

[1] http://ode.apache.org/

Referências

[1] PAPAZOGLOU, M. P. Service-oriented computing: Concepts, characteristics and directions. *4th International Conference on Web Information Systems Engineering (WISE'03)*, pages 3–12, 2003.

[2] PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008.

[3] GEORGAKOPOULOS, D.; PAPAZOGLOU, M. P. Service-oriented computing. *Massachusetts Institute of Technology (MIT)*, 2009.

[4] PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[5] CERVANTES, H.; HALL, R.S. Technical concepts of service orientation. *Service-oriented software system engineering: challenges and practices*, pages 1–12, 2005.

[6] TAHER, L.; EL KHATIB, H.; BASHAR, R. A framework and qos matchmaking algorithm for dynamic web services selection. *2th International Conference on Innovations in Information Technology (IIT'05)*, 2005.

[7] SATHYA, M.; SWARNAMUGI, M.; DHAVACHELVAN, P.; SURESHKUMAR, G. Evaluation of qos based web-service selection techniques for service composition. *International Journal of Software Engineering*, pages 73–90, 2010.

[8] SOUZA, A. P. *Um modelo de descoberta dinâmica de serviços de software baseado no contexto de processos de negócios e em qualidade de serviço*. PhD thesis, Universidade Federal de Santa Catarina, 2011.

[9] YU, H. Q.; REIFF-MARGANIEC, S. Non-functional property based service selection: A survey and classification of approaches. *6th IEEE European Conference on Web Services*, pages 12–14, 2008.

[10] ERL, T. Safari Tech Books Online (Online service). *Soa: principles of service design*. Prentice Hall, 2008.

[11] PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: A reseach roadmap. *International Journal of Cooperative Information Systems*, pages 223–255, 2008.

[12] PAPAZOGLOU, M. P.; VAN DEN HEUVEL, W. Service-oriented design and development methodology. *International Journal of Web Engineering and technology*, pages 412–442, 2006.

[13] SHUPING, RAN. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.

[14] MAXIMILIEN, M. E.; SINGH, M. P. Toward autonomic web services trust and selection. *2nd International Conference on Service Oriented Computing (ICSOC)*, pages 212–221, 2004.

[15] OASIS Committee Specification 01. Web services quality factors version 1.0, 2011. `http://docs.oasis-open.org/wsqm/WS-Quality-Factors/v1.0/cs01/WS-Quality-Factors-v1.0-cs01.html`. Último acesso em Maio/2014.

[16] MICHLMAYR, A.; ROSENBERG, F.; LEITNER, P.; DUSTDAR, S. Comprehensive qos monitoring of web services and event-based sla violation detection. *Technical University Vienna*, pages 1–6, 2009.

[17] MENASCÉ, D. A. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[18] HALL, R. S.; SAVAGE, D. *Osgi in action*. Manning Publications Co, 2010.

[19] BARTLETT, N. *Osgi in Practice*. 2009.

[20] O.S.G. Alliance. Osgi service platform, core specification, release 5, version 2.0. *OSGi Specification*, 2012.

[21] QUSAY, H. M. Osgi alliance. `http://www.osgi.org/About/Technology`. Último acesso em Maio/2014.

[22] RAJENDRAN, T.; BALASUBRAMANIE, P.; CHERIAN, R. An efficient ws-qos based architecture for web services selection. *International Journal of Computer Applications*, pages 79–84, 2010.

[23] AL-MASRI, E.; MAHMOUD, Q. H. Qos-based discovery and ranking of web services. *16th International Conference on Computer Communications and Networks*, pages 529–534, 2007.

[24] YU, T.; LIN, K. Service selection algorithms for web services with end-to-end qos constraints. pages 103–126, 2005.

[25] RAJESWARY, C. A survey on efficient evolutionary algorithm for web service selection. *International Journal of Management, IT and Engineering*, pages 177–191, 2012.

[26] ETZION, O.; NIBLETT, P. *Event Processing in Action*. Manning Publications Company, 2010.

[27] ESPER. Java event stream processor. `http://esper.codehaus.org/esper-1.0.5/doc/reference/en/html/index.html`. Último acesso em Dezembro/2011.

[28] iPOJO. A flexible and extensible service component model. `http://felix.apache.org/site/apache-felix-ipojo.html`. Último acesso em Dezembro/2011.

[29] ESCOFFIER, C.; HALL, R. S.; LALANDA, P. ipojo: an extensible service-oriented component framework. *International Conference on Services Computing (SCC)*, pages 474–481, 2007.

[30] NEELAVATHI, S.; VIVEKANANDAN, K. An innovative quality of service (qos) based service selection for service orchrestration in soa. *International Journal of Scientific & Engineering Research*, pages 1–8, 2011.

[31] SERHANI, M. A.; DSSOULI, R.; HAFID, A.; SAHRAOUI, H. A qos broker based architecture for efficient web services selection. *IEEE International Conference on Web Services*, pages 113–120, 2005.

[32] ZENG, L.; BENATALLAH, B.; DUMAS, M.; KALAGNANAM, J.; SHENG, Q. Z. Quality driven web services composition. *12th International Conference World Wide Web*, pages 411–421, 2003.

[33] YU, T.; ZANG, Y.; LIN, K. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web*, pages 1–26, 2007.

[34] REIFF-MARGANIEC, H. S.; YU, H. Q.; TILLY, M. Service selection based on non-functional properties. *Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, pages 1–14, 2007.

[35] BLUM, A. Uddi as an extended web services registry: Versioning, quality of service, and more. *SOA World magazine*, 4, 2004. `soa.sys-con.com/node/45102`. Último acesso em Maio/2014.

[36] XU, Z.; MARTIN, P.; POWLEY, W.; ZULKERNINE, F. Reputation-enhanced qos-based web service discovery. *International Conference on Web Services*, pages 249–256, 2007.

[37] ABDULLAH, L.; ADAWIYAH, C. W. R. Simple additive weighting methods of multi criteria decision making and applications: A decade review. *International Journal of Information Processing and Management*, pages 39–49, 2014.

[38] AFSHARI, A.; MOJAHED, M.; YUSUFF, R. M. Simple additive weighting approach to personal selection problem. *International Journal of Innovation, Management and Technology*, pages 511–515, 2010.

[39] KUEHNE, B. T. *Modelos e algoritmos para composição de Web Services com qualidade de serviço*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, 2009.

[40] YU, T.; LIN, K. Service selection algorithms for web services with end-to-end qos constraints*. 2004.

[41] RAJENDRAN, T.; BALASUBRAMANIE, P. An optimal broker-based architecture for web service discovery with qos characteristics. *International Journal of Web Services Practices*, pages 32–40, 2010.

[42] RATHORE, M.; SUMAN, U. A quality of service broker based process model for dynamic web service composition. *Journal of Computer Science*, pages 1267–1274, 2011.

[43] CAVALCANTI, D. J. M.; SOUZA, F. N.; ROSA, N. S. Adaptive and dynamic quality-aware service selection. *21st Euromicro International Conference on Parallel, Distributed, and Network-based Processing (PDP)*, pages 323–327, 2013.

[44] MICHLMAYR, A.; ROSENBERG, F.; LEITNER, P.; DUSTDAR, S. End-to-end support for qos-aware service selection, invocation and mediation in vresco. *Technical University Vienna, Tech. Rep. TUV-184-2009-03, Jun*, 2009.

[45] CAVALCANTI, D. J. M.; SOUZA, F. N.; SILVA, T. C.; ROSA, N. S. Ranking strategies for quality-aware service selection. *11th International Conference on Services Computing (SCC)*, 2014.

[46] O.S.G. Alliance. Osgi compendium, release 5, version 2.0. *OSGi Specification*, 2013.

[47] SOUZA, F. N.; LOPES, D.; GAMA, K.; ROSA, N. S.; LIMA, R. M. F. Dynamic event-based monitoring in a soa environment. *International Symposium on Distributed Objects and Applications*, pages 498–506, 2011.

[48] MAKHLUGHIAN, M.; HASHEMI, S. M.; RASTEGARI, Y.; PEJMAN, E. Web service selection based on ranking of qos using associative classification. *International Journal of Web Service Computing (IJWSC)*, 3(01), 2012.

[49] OLIVEIRA, L. S. M.; CORREIA, T. C. V. D.; MELLO, J. C. C. B. S. Métodos multicritério de auxílio à decisão aplicados a avaliação e aquisição de imóveis. 2008. Relatórios de Pesquisa em Engenharia de Produção.

[50] HEIN, N.; KROENKE, A.; WILLHELM, V. E. Análise multicritério de materiais de construção - sobre o uso da entropia como medida de importância. 2013. `www.scientiaplena.org.br`.

[51] JAIN, R. *Art of Computer Systems Performance Analysis: Techniques for Experimental Design Measurements Simulation and Modeling*. 1991.

[52] PINHEIRO, M. M.; GOMES, C. F. S. Evolução do mercado acionário: Home broker – estudo de caso hsbc. 2008.

[53] YU, T.; LIN, K. Service selection algorithms for web services with end-to-end qos constraints. *IEEE International Conference on eCommerce Technology*, pages 129–136.

[54] MCKINLEY, P. K.; SADJADI, S. M.; KASTEN, E. P.; CHENG, B. H. C. Departament of Computer Science and Engineering.

[55] PALANIKKUMAR, D.; KOUSALYA, G. An evolutionary algorithmic approach based optimal web service selection for composition with quality of service. *Journal of Computer Science*, pages 573–578, 2012.