Instituto de Computação
Universidade Estadual de Campinas

Incorporação de Qualidade de Serviço no Modelo de Serviços *Web*

Diego Zuquim Guimarães Garcia

05 de Março de 2007

Banca Examinadora:

- Profa. Dra. Maria Beatriz Felgar de Toledo (Orientadora)
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Dr. Eleri Cardozo
  Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Dr. Edmundo Roberto Mauro Madeira (Suplente)
  Instituto de Computação, Universidade Estadual de Campinas

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 5094

Garcia, Diego Zuquim Guimarães
G165i Incorporação de qualidade de serviço no modelo de serviços Web/Diego Zuquim Guimarães Garcia -- Campinas, [S.P. :s.n.], 2007.

Orientador : Maria Beatriz Felgar de Toledo
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Serviços Web – Desenvolvimento. 2. Software – Qualidade. 3. Middleware. 4. World Wide Web (Sistema de recuperação da informação). I. Toledo, Maria Beatriz Felgar de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(mca/imecc)

Título em inglês: Inclusion of quality of service into the web service model

Palavras-chave em inglês (Keywords): 1. Web services – Development. 2. Software – Quality. 3. Middleware. 4. World Wide Web (Information retrieval systems)

Área de concentração: Sistemas distribuídos

Titulação: Mestre em Ciência da computação

Banca examinadora: Profa. Dra. Maria Beatriz Felgar de Toledo (IC-Unicamp)
Prof. Dr. Eleri Cardozo (FEEC-Unicamp)
Prof. Dr. Célio Cardoso Guimarães (IC-Unicamp)

Data da defesa: 05/03/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 05 de março de 2007, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Eleri Cardozo
FEEC – UNICAMP.

Prof. Dr. Célio Cardoso Guimarães
IC – UNICAMP.

Profa. Dra. Maria Beatriz Felgar de Toledo
IC – UNICAMP.

Incorporação de Qualidade de Serviço no Modelo de Serviços *Web*

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Diego Zuquim Guimarães Garcia e aprovada pela Banca Examinadora.

Campinas, 05 de Março de 2007.

Dra. Maria Beatriz Felgar de Toledo
IC/UNICAMP – Universidade Estadual de Campinas (Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

A tecnologia de serviços *Web* possui algumas propriedades importantes para o desenvolvimento e a execução de aplicações distribuídas. Entretanto, ela ainda não oferece apoio para tratar as características não-funcionais dos serviços. Os consumidores de serviços *Web* podem requerer serviços com parâmetros de qualidade específicos e esperar garantias de níveis de qualidade. O objetivo desta dissertação é estender o modelo de serviços *Web* para apoiar a gerência de características não-funcionais para serviços *Web*. O modelo proposto inclui mediadores para auxiliar na descoberta de serviços de acordo com os requisitos funcionais e não-funcionais dos consumidores e monitores para verificar os atributos de qualidade. As principais contribuições desta dissertação são: a utilização do padrão *Web Services Policy Framework* (*WS-Policy*) para complementar as descrições de serviços *Web Services Description Language* (WSDL) com políticas para atributos de qualidade; uma extensão para o padrão *Universal Description Discovery & Integration* (UDDI) para a publicação e a descoberta de serviços *Web* incluindo características não-funcionais; e o monitoramento e a atualização de características não-funcionais para refletir os atributos reais dos serviços.

Abstract

Although the Web service technology allows the development and execution of distributed applications, it still lacks facilities to deal with Quality of Service (QoS). Consumers may require services with particular non-functional characteristics and expect quality level guarantees. The goal of this thesis is to propose an extended Web service architecture supporting QoS management for Web services. It includes brokers to facilitate service selection according to functional and non-functional requirements and monitors to verify QoS attributes. The main contributions of this approach are: the use of the Web Services Policy Framework (WS-Policy) standard to complement Web Services Description Language (WSDL) specifications with QoS policies; an extension to the Universal Description Discovery & Integration (UDDI) standard for QoS-enriched Web service publication and discovery; and QoS updating to reflect actual service attributes.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O interesse crescente na tecnologia de serviços *Web* e o número crescente de provedores de serviços *Web* produzem novas demandas em especificação, publicação e descoberta de serviços. Os consumidores de serviços precisam de ferramentas para localizar serviços apropriados disponíveis na *Web*. Isso impõe desafios não somente em mecanismos de descoberta, mas também na garantia de informações suficientes sobre os serviços publicados [28]. Para que os consumidores possam selecionar os serviços adequados, os provedores devem produzir especificações que ofereçam as informações necessárias e as informações devem ser publicadas em repositórios de serviços.

O padrão *Universal Description Discovery & Integration* (UDDI) [17] é um passo para satisfazer essas demandas. Os repositórios UDDI tornam possível a publicação e descoberta de serviços baseadas em aspectos funcionais. Entretanto, as demandas dos consumidores podem incluir não apenas aspectos funcionais de serviços, mas também aspectos não-funcionais. Assim, a descoberta de serviços depende das habilidades de descrever atributos de qualidade e verificar a compatibilidade entre capacidades e requisitos não-funcionais [54] [2] [39].

Existem diversos atributos de qualidade, tais como desempenho e segurança, os quais são importantes para os consumidores. Conforme observado em alguns trabalhos [42] [48] [59], políticas podem ser definidas para especificar características não-funcionais. O padrão *Web Services Policy Framework* (*WS-Policy*) [4] oferece um modelo *Extensible Markup Language* (XML) [12] para especificar políticas para serviços *Web*. Conseqüentemente, ele é um candidato para descrever atributos de qualidade para serviços *Web*.

No modelo de serviços *Web*, o padrão *Web Services Description Language* (WSDL) [20] é utilizado para descrever alguns aspectos funcionais de serviços. A especificação de políticas para atributos de qualidade provê um modo para distinguir serviços *Web* com funcionalidade equivalente e para localizar serviços capazes de satisfazer as demandas dos consumidores de serviços de forma mais abrangente, incluindo os aspectos funcionais e não-funcionais.

Portanto, o padrão *WS-Policy* pode ser utilizado para complementar as descrições WSDL. Além disso, o padrão UDDI pode ser estendido para incluir as políticas para atributos de qualidade.

A seguir são apresentados o objetivo e a estrutura da dissertação.

1.1 Objetivo

O objetivo desta dissertação é propor uma extensão para o modelo de serviços *Web* para apoiar a gerência de atributos de qualidade. Na abordagem proposta, as informações extraídas de uma política para atributos de qualidade baseada no padrão *WS-Policy* são encapsuladas na estrutura *qosPolicy* incluída no padrão UDDI. As estruturas *qosPolicy* são associadas com estruturas *bindingTemplate*, que representam serviços *Web*. Cada elemento da estrutura *qosPolicy* pode estar associado com uma estrutura *tModel*, que permite a padronização de conceitos de qualidade (Seção 5.1.1). Além disso, a extensão possibilita o emprego de componentes mediadores para descobrir serviços utilizando as políticas dos consumidores com demandas de atributos de qualidade e as políticas dos provedores com ofertas de características não-funcionais, e componentes monitores para atualizar as políticas para os serviços.

Com o intuito de avaliar a eficácia e eficiência da abordagem, o modelo proposto foi parcialmente implementado e alguns experimentos foram desenvolvidos.

As principais contribuições desta dissertação são:

- A utilização do padrão *WS-Policy* para a especificação de características não-funcionais de serviços *Web*;
- A extensão do padrão UDDI para a publicação de atributos de qualidade de serviço e a descoberta de serviços considerando parâmetros de qualidade;
- O emprego de monitores para a verificação de características não-funcionais com o objetivo de garantir níveis de qualidade.

Inúmeros trabalhos, descritos no Capítulo 4, oferecem diversas abordagens para a gerência de características não-funcionais em serviços *Web*. A extensão proposta utiliza e adapta algumas abordagens para apoiar a especificação, publicação, descoberta e verificação de características não-funcionais de serviços *Web*. As abordagens utilizadas são selecionadas e combinadas com o propósito de incluir a extensão no modelo de serviços *Web* atual.

1.2 Estrutura da Dissertação

Esta dissertação está estruturada como segue:

- Capítulo 2: discute Sistema de Gerência de Processos de Negócio como uma motivação para a inclusão de características não-funcionais no modelo de serviços *Web*;
- Capítulo 3: apresenta alguns conceitos básicos, incluindo o modelo de serviços *Web* básico, o padrão *WS-Policy* e características não-funcionais em serviços *Web*;
- Capítulo 4: apresenta alguns trabalhos relacionados nas áreas de especificação, publicação, descoberta e verificação de serviços *Web*;
- Capítulo 5: discute algumas extensões para os padrões *WS-Policy* e UDDI para a gerência de características não-funcionais e descreve um modelo de serviços *Web* baseado nas extensões propostas;
- Capítulo 6: descreve alguns aspectos de uma implementação e avaliação do modelo estendido;
- Capítulo 7: finaliza a dissertação apresentando as conclusões, as contribuições da dissertação e alguns trabalhos futuros.

Capítulo 2

Sistemas de Gerência de Processos de Negócio Baseados em Serviços *Web*

Este capítulo discute processos de negócio como um exemplo de situação em que é necessária a consideração de características não-funcionais em serviços *Web*.

2.1 Gerência de Processos de Negócio

A Gerência de Processos de Negócio (GPN)[1] [32] foca a automatização de processos de negócio. Um processo de negócio é um conjunto de procedimentos ou atividades relacionados com o intuito de realizar um objetivo de negócio, normalmente no contexto de uma estrutura organizacional [31]. Um processo pode ser visto como uma unidade de trabalho iniciada por um evento de negócio. Ele é dirigido por regras de negócio, que iniciam atividades. Atividades são atribuídas a recursos, que são unidades organizacionais capazes de desempenhar papéis específicos em processos. Um processo é definido por meio da descrição de regras, atividades e recursos, e executado por meio da invocação de serviços de negócio existentes, os quais podem residir em qualquer lugar.

A aplicação de GPN em organizações tem sido motivada devido à capacidade que ela proporciona de controle de processos de negócio. Os sistemas utilizados com tal intuito focaram, inicialmente, processos intra-organizacionais, apoiando apenas a execução independente de processos. Entretanto, a interação entre processos é necessária tanto dentro de uma organização quanto entre organizações.

Algumas tendências de mercado aumentaram a necessidade de cooperação entre processos executados em diferentes organizações. O crescimento do comércio eletrônico e da terceirização de funções conduz à necessidade de integrar e automatizar processos interorganizacionais [66] [14] [22].

[1] Do inglês *Business Process Management* (BPM).

2.2 Processos de Negócio Interorganizacionais

Atualmente, torna-se essencial o apoio não apenas para processos internos a organizações individuais, mas também processos cruzando fronteiras organizacionais, marcados pela heterogeneidade dos ambientes.

Para garantir vantagens competitivas, as organizações são pressionadas no sentido de formar organizações virtuais, caracterizadas pelo fato de um processo de negócio ser composto de processos de negócio cooperativos gerenciados pelas diversas organizações envolvidas na associação. Tipicamente, uma organização enfatiza as tarefas que fazem parte de sua linha principal de atuação, transferindo as demais tarefas a organizações especialistas [14] [41].

O modelo de processo interorganizacional caracteriza-se por uma definição de processo baseada em um protocolo de interação. Além disso, a execução de processo é realizada por múltiplas máquinas de *workflow* de modo cooperativo [22].

Três características principais diferenciam processos interorganizacionais de processos intra-organizacionais [9].

Primeiramente, são necessários acordos sobre as interações entre as organizações envolvidas para garantir uma compreensão comum dos dados permutados. Interações entre organizações são normalmente mais complexas em relação a interações entre processos de uma organização, devido à heterogeneidade dos ambientes: diferentes sistemas de gerência e políticas de negócio [14].

Além disso, a autonomia das organizações participantes deve ser considerada. Apesar da necessidade de interação, as organizações podem não desejar expor detalhes de seus processos, além de desejarem ser capazes de modificá-los sem afetar os processos cooperativos [14].

Por fim, a abertura do ambiente gera necessidades relacionadas a segurança, confiabilidade e legalidade não presentes no caso de processos intra-organizacionais [5].

2.3 Sistemas de Gerência de Processos de Negócio

Considerando o modelo de processo interorganizacional, um sistema chamado de Sistema de Gerência de Processos de Negócio (SGPN)[2] foi introduzido com o objetivo de apoiar a execução de processos de negócio [25].

Os SGPNs enfatizam a gerência de processos que cruzam fronteiras organizacionais [21] [22] [40] [9] [31] e o aspecto da dinamicidade de processos, que apresenta maior intensidade nesse novo modelo [40] [57] [67] [31]. Essas perspectivas tornaram a gerência de processos mais complexa.

---

[2] Do inglês *Business Process Management System* (BPMS).

De modo geral, o ciclo de vida de automatização de processo (Figura 2.1) inicia com a definição do processo (Projeto). Em seguida, a definição do processo é registrada em um SGPN (Configuração). Para executar um processo, o sistema cria uma instância de processo e, então, coordena e registra sua execução (Execução). O registro da execução do processo pode ser analisado, podendo gerar uma definição aprimorada do processo de negócio (Diagnóstico) [22] [31].

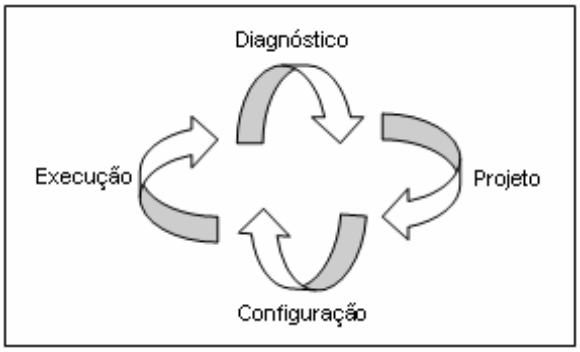

Figura 2.1: Ciclo de vida de gerência de processo de negócio.

2.4 Requisitos de Gerência de Processos de Negócio

Alguns requisitos impostos pelo modelo de processo de negócio interorganizacional são apresentados a seguir:

- Cooperação interorganizacional: em um ambiente de organização virtual, a função de gerência de processo deve ser executada como uma cooperação entre múltiplos sistemas de gerência [22] [54] [72];
- Flexibilidade de processo: a dinamicidade do mercado demanda a capacidade de gerenciar alterações de processos de negócio [18] [57] [72] [49];
- Descoberta dinâmica: uma organização não precisa conhecer previamente as organizações com as quais interagirá. É necessário o apoio de um mecanismo de descoberta [18] [14] [49];
- Características não-funcionais: uma determinada funcionalidade pode ser oferecida por diversas organizações. Portanto, é importante o apoio à seleção de serviços para um processo baseada em parâmetros de qualidade [22] [54];
- Confiabilidade de processo: essencialmente, deve-se garantir a execução em conformidade com a definição do processo [22] [54];

- Escalabilidade: deve ser possível prover apoio para processos adicionais de modo ágil e multiplicar instâncias de processos existentes de modo significativo [22] [57] [72];
- Gerência de contratos: deve existir um mecanismo para possibilitar que as organizações estabeleçam contratos eletrônicos relacionados aos serviços a serem fornecidos [22] [49];
- Inteligência de processo: organizações automatizam processos com o objetivo de otimizá-los. A área de Inteligência de Negócio[3] foca a análise de processos de negócio [22];
- Propriedades transacionais: processos acessam recursos compartilhados e, portanto, propriedades transacionais são necessárias. É importante considerar as características específicas de processos de negócio [22].

2.5 Processos de Negócio Baseados em Serviços *Web*

Com o crescimento da *Web* como a principal plataforma para disponibilizar dados e serviços, a dinamicidade tornou-se preponderante, sendo necessário lidar com relacionamentos de negócio não pré-definidos [57].

Inicialmente, foram propostas soluções utilizando tecnologias de *middleware*, que, no entanto, se apresentaram inapropriadas para a interação na *Web* [22] [2]. Atualmente, serviços *Web* estão emergindo para prover uma estrutura para interações construída sobre protocolos *Web* existentes e baseada em padrões XML abertos [13] [15] [54].

A tecnologia de serviços *Web*, baseada nas especificações W3C (*World Wide Web Consortium*) e OASIS (*Organization for the Advancement of Structured Information Standards*), garante uma solução que possui potencial para apoiar a automatização de negócio em uma área ampla [67] [2]. Com o objetivo de dominar a complexidade de integração de aplicações, serviços *Web* representam um esforço de padronização. Atualmente, serviços *Web* oferecem soluções para alguns dos problemas de interoperabilidade presentes em integração de aplicações.

Serviços *Web*, fundamentados na Arquitetura Orientada a Serviços[4] [54] [11], provêem uma infra-estrutura para acessar e integrar serviços, apresentando algumas vantagens. Diferentemente de outras abordagens de *middleware*, tais como, CORBA (*Common Object Request Broker Architecture*) [51] e *Java* RMI (*Remote Method Invocation*) [63], os padrões de serviços *Web* são apoiados pelos principais fornecedores de *software* [29] [40] [52]. Além disso, os protocolos e linguagens são simples. Já padrões complexos são de difícil compreensão e implementação, e normalmente não alcançam sucesso [13].

---

[3] Do inglês *Business Intelligence*.

[4] Do inglês *Service-Oriented Architecture* (SOA).

A capacidade de apoiar a gerência de processos em relacionamentos interorganizacionais dinâmicos, efetuados em um ambiente heterogêneo, e caracterizados pela necessidade de otimização de operações das organizações, pode ser proporcionada através do emprego de serviços *Web*.

Na abordagem de processos de negócio baseados em serviços *Web*, modelos de processo incluem atividades que representam serviços providos por parceiros de negócio. Durante a execução de processos de negócio, tais atividades são realizadas por serviços *Web* selecionados em repositórios de serviços [70].

Os SGPNs podem apoiar a interação que ocorre quando uma organização incorpora serviços de outra organização em seus próprios processos [44]. Esses SGPNs baseados em serviços gerenciam processos que utilizam serviços *Web*. Entretanto, as organizações de negócio raramente desejam procurar por serviços considerando requisitos funcionais e invocá-los sem conhecer suas capacidades não-funcionais [55].

Portanto, para que a tecnologia de serviços *Web* possa oferecer a gerência adequada de processos de negócio, são necessárias habilidades adicionais para contemplar os requisitos de GPN. O modelo atual de serviços *Web* precisa ser estendido considerando os múltiplos canais de comunicação presentes em transações, incluindo as pesquisas para satisfazer os critérios de qualidade exigidos pelas organizações [49].

Conseqüentemente, esforços de pesquisa estão sendo aplicados atualmente no sentido de proporcionar as vantagens de serviços *Web* à GPN [70]. Um dos requisitos para alcançar esse objetivo é a inclusão da gerência de características não-funcionais no modelo de serviços *Web*. Um caso de uso do mundo real em *Virtual Internet Service Provider* (VISP) motiva o desenvolvimento de soluções para esse problema.

O modelo de negócio VISP [68] é baseado em processos que utilizam serviços de parceiros e constitui um caso típico de utilização de SGPNs baseados em serviços *Web*. No modelo de negócio VISP, *Internet Service Providers* (ISPs) descrevem seus serviços como serviços eletrônicos, incluindo características tais como disponibilidade, tempo de resposta, taxa de serviço, dentre outras. Uma organização interessada em tornar-se um VISP pode procurar por serviços desejados considerando seus requisitos não-funcionais e integrá-los em um produto virtual, o qual pode então ser vendido no mercado. Como muitos ISPs podem fornecer os mesmos serviços básicos em diferentes níveis de qualidade, mecanismos de descoberta de serviços deveriam considerar não somente a adequação funcional de serviços, mas também a qualidade oferecida.

Diversos outros casos de uso são apresentados na literatura [48]. A seguir são brevemente descritos cenários nas áreas de B2C (*Business-to-Consumer*) e B2B (*Business-to-Business*):

- Um consumidor pode desejar procurar por produtos e serviços na *Web*. A aplicação de busca emprega motores de busca para executar a consulta do consumidor e escolhe o mais apropriado de acordo com as preferências do consumidor informadas junto com os critérios da busca. A aplicação de busca descobre e acessa dinamicamente um motor de busca para executar a consulta;

- Um agente de negociação de automóveis provê financiamento aos seus consumidores potenciais. O agente pode desejar oferecer diversas opções de financiamento. Assim, ele não possui relacionamentos fixos com quaisquer instituições financeiras. Ele seleciona os serviços de financiamento considerando os parâmetros dos serviços disponíveis. A aplicação de financiamento do agente utiliza um serviço, o qual é instanciado para cada interação de negócio. O agente seleciona os serviços apropriados utilizando as políticas dos serviços e as preferências dos consumidores e apresenta as alternativas de financiamento aos consumidores.

Para as organizações, a capacidade de caracterizar os processos de negócio com base em suas características não-funcionais proporciona uma vantagem direta: as organizações podem traduzir seus objetivos em seus processos de modo mais eficiente, pois os processos de negócio podem ser projetados de acordo com métricas de qualidade, e os serviços parceiros podem ser selecionados de forma a melhor satisfazer as expectativas dos consumidores.

Capítulo 3

Serviços *Web*

Neste capítulo são introduzidos conceitos fundamentais relacionados a serviços *Web* como Arquitetura Orientada a Serviços na Seção 3.1 e padrões de serviços *Web* na Seção 3.2. Além disso, características não-funcionais gerais no contexto de serviços *Web* são apresentadas na Seção 3.3.

3.1 Arquitetura Orientada a Serviços

A Computação Orientada a Serviços é o paradigma computacional que emprega serviços como os elementos fundamentais para o desenvolvimento de aplicações [11].

Serviços são definidos como componentes de *software* autodescritivos que apóiam a composição ágil de aplicações [40].

A Computação Orientada a Serviços envolve a Arquitetura Orientada a Serviços, que compreende três camadas de serviços: básicos, compostos e gerenciados [54]. Essas camadas são descritas a seguir:

- Serviços simples e operações básicas, incluindo publicação, descoberta e ligação, constituem a camada fundamental da Arquitetura Orientada a Serviços;
- A camada de serviços compostos abrange funcionalidades para a consolidação de diversos serviços simples em um único serviço composto;
- A camada de serviços gerenciados é provida para apoiar soluções críticas e mercados de serviços. As funcionalidades de gerência de operações apóiam soluções críticas que demandam a gerência das aplicações por parte das organizações. A camada de serviços gerenciados também provê apoio para mercados de serviços abertos, que geram oportunidades para que provedores e consumidores de serviços se encontrem. Ela inclui funcionalidades de gerência de mercados, tal como negociação de transação de negócio, dentre outras.

A Arquitetura Orientada a Serviços envolve três aspectos principais [40]:

- A localização de serviços, incluindo a procura por serviços que satisfaçam determinados critérios de negócio;
- A organização de serviços, de forma que um requisitante de serviços possa compreender facilmente o que um serviço oferece;
- A especificação de serviços, incluindo os protocolos para que os serviços possam ser invocados de modo apropriado.

Os papéis envolvidos na Arquitetura Orientada a Serviços são apresentados a seguir [40] [54] [11]:

- Provedores de serviços oferecem serviços. Eles utilizam descrições de serviços para informar a funcionalidade e a qualidade dos serviços providos;
- Consumidores de serviços são usuários finais de serviços. Eles utilizam serviços oferecidos por provedores para alcançar seus objetivos;
- Compositores de serviços são organizações que consolidam múltiplos serviços em um novo serviço;
- Repositórios de serviços são utilizados por requisitantes de serviços para localizar serviços que satisfaçam suas demandas. Um repositório inclui, além de classificações para auxiliar nas procuras, informações fornecidas pelos provedores. Ao encontrar um serviço apropriado, o requisitante conecta-se ao provedor por meio de informações de conexão, também mantidas no repositório, incluindo a especificação do protocolo necessário para utilizar o serviço.

A Arquitetura Orientada a Serviços aplicada como base para a tecnologia de GPN apóia a separação do modelo de processo de negócio das implementações de atividades individuais. Essa propriedade garante maior velocidade no oferecimento de novos produtos, que é um requisito essencial em determinados setores de negócio [18] [31] [72].

3.2 Tecnologia de Serviços *Web*

Inicialmente, a *Web* foi utilizada principalmente para publicar informações. Então, servidores de aplicações foram utilizados para oferecer serviços para clientes humanos. O tempo presente é marcado pelo desenvolvimento da tecnologia de serviços *Web*, em que os serviços podem ser acessados de modo programático [29].

Serviços *Web* surgiram do desejo de realizar transações automatizadas na *Web*, ao invés de utilizar processamento manual. Serviços *Web* representam uma tentativa de criar um conjunto de padrões abertos para a interoperabilidade entre sistemas [5].

O objetivo primário da tecnologia de serviços *Web* é automatizar. Então, a idéia essencial de serviços *Web* é estruturar a interação com serviços eletrônicos de forma que uma máquina, ao invés de um indivíduo, possa controlá-la. Para tanto, a tecnologia de serviços *Web* substitui [49]:

- A combinação pessoa/navegador *Web* por um programa;
- A linguagem HTML pela XML, mais estruturada;
- O servidor *Web* por uma máquina SOAP (inicialmente um acrônimo para *Simple Object Access Protocol*), que traduz mensagens XML e as transmite para as aplicações que fazem os trabalhos requisitados.

A aplicação da Arquitetura Orientada a Serviços na *Web* é realizada pela tecnologia de serviços *Web*. Um serviço *Web* é um tipo específico de serviço eletrônico [21], identificado por um URI (*Uniform Resource Identifier*), que expõe funcionalidades accessíveis através da *Web* e cuja descrição e interação utilizam padrões XML [13] [15] [54] [53].

Diversos modelos de serviços *Web* são apresentados na literatura. Este trabalho é baseado no modelo do W3C. A definição de serviço *Web* incluída no documento que especifica o modelo do W3C consiste de [8]:

- Sistema de *software* projetado para apoiar interações máquina-a-máquina em uma rede;
- Interface descrita em um formato processável por máquina (especificamente WSDL);
- Outros sistemas que interagem com um serviço conforme sua descrição utilizando mensagens SOAP. Essas mensagens são conduzidas utilizando tipicamente o protocolo *HyperText Transfer Protocol* (HTTP), com uma serialização XML em conjunto com outros padrões relacionados.

A Figura 3.1 ilustra o modelo de serviços *Web* básico. Os provedores de serviços publicam seus serviços nos repositórios de serviços (Passo 1 na Figura 3.1). Os consumidores de serviços procuram por serviços nos repositórios em tempo de desenvolvimento ou em tempo de execução (Passo 2), ligando-se aos serviços estática ou dinamicamente. Então, os consumidores podem interagir com esses serviços (Passo 3).

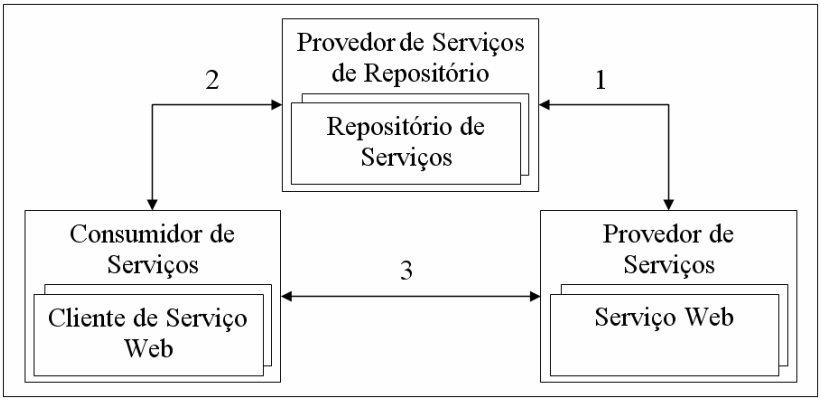

Figura 3.1: Modelo de serviços *Web* básico.

A estrutura da tecnologia de serviços *Web* compreende, entre outras, três áreas básicas: protocolo de mensagens, descrição de serviços e descoberta de serviços. Padrões estão sendo desenvolvidos para cada uma delas. A estrutura de serviços *Web*, apresentada na Figura 3.2, é dividida em camadas [8]. Como a estrutura é modular, apenas as partes necessárias podem ser utilizadas [15]:

- A camada de comunicações inclui protocolos tais como, HTTP, *Simple Mail Transport Protocol* (SMTP), *File Transfer Protocol* (FTP), *Java Message Service* (JMS), entre outros. Conforme a definição de serviços *Web* do W3C apresentada, mensagens SOAP são conduzidas tipicamente sobre HTTP, entretanto, outros protocolos podem ser empregados;
- A camada de mensagens inclui além do protocolo padrão de mensagens SOAP, extensões para o protocolo SOAP para tratar confiabilidade, transações, entre outras questões importantes em serviços *Web*;
- A camada de descrições inclui a linguagem padrão de descrição de serviços *Web* WSDL. Além disso, extensões e novas linguagens estão sendo definidas para tratar de questões dessa camada não abrangidas por WSDL, como, por exemplo, representação da semântica de serviços;
- A camada de processos inclui o padrão de repositório de serviços *Web* UDDI e padrões de composição de serviços [9], incluindo orquestração (por exemplo, *Web Services Business Process Execution Language* – WS-BPEL [1]) e coreografia (por exemplo, *Web Services Choreography Description Language* – WS-CDL [33]);
- As camadas de segurança e gerência atravessam as demais. Padrões estão sendo definidos para a proteção de transmissão de mensagem em serviços *Web*, assim como padrões para a gerência de serviços *Web* e para a gerência de diversos recursos utilizando serviços *Web*;
- Tecnologias básicas, incluindo XML e XML *Schema Definition* (XSD), envolvem a pilha de padrões de serviços *Web*.

Figura 3.2: Estrutura da tecnologia de serviços *Web*.

Os padrões básicos empregados no âmbito da tecnologia de serviços *Web* são descritos nas próximas seções: WSDL (Seção 3.2.1), UDDI (Seção 3.2.2) e SOAP (Seção 3.2.3). Provedores e consumidores de serviços empregam os padrões de serviços *Web* para executar as operações básicas da Arquitetura Orientada a Serviços.

Uma sintaxe comum é requerida para os padrões. Em serviços *Web*, tal sintaxe é provida por XML. Portanto, todos os padrões de serviços *Web* são baseados em XML, com formatos e estruturas de dados descritos em XML [2]. XML é uma linguagem de marcação de documentos que pode ser estendida [12].

3.2.1 WSDL

Web Services Description Language (WSDL) [20] é um formato XML que proporciona uma descrição de serviço *Web* processável por máquina. A linguagem WSDL é apoiada pelo W3C.

Um documento WSDL descreve a funcionalidade de um serviço *Web* e provê um ponto de acesso para consumidores de serviços. Uma descrição WSDL completa inclui dois tipos de informação:

- Uma interface de serviço;
- Detalhes concretos de uma descrição de serviço, que consumidores devem seguir para acessar o serviço.

Em WSDL, uma interface compreende um conjunto de operações realizadas por um serviço. Mensagem é o elemento fundamental de uma descrição de serviço. Cada mensagem trocada consiste de itens de dados com tipos definidos. A descrição de uma mensagem é inicialmente abstrata. Em seção posterior do documento WSDL, a mensagem é associada a um protocolo de comunicações. É a partir das mensagens recebidas/enviadas por um serviço *Web* que o serviço é descrito em WSDL.

A parte abstrata de uma descrição de serviço WSDL compreende os seguintes elementos:

- *portType*: é uma coleção de operações;
- *operation*: define uma troca de mensagens;
- *message*: representa os dados enviados em uma transmissão de mensagem;
- *types*: são os tipos de dados empregados pelo serviço.

Os elementos WSDL que compõem a parte concreta de uma descrição de serviço são apresentados a seguir:

- *binding*: especifica o estilo de interação e o protocolo de comunicações para um elemento *portType*;
- *port*: combina informações de ligação com um ponto de acesso, especificado por um URI, através do qual uma implementação de um *portType* determinado pode ser acessada;
- *service*: é uma coleção de elementos *port*.

A Figura 3.3 ilustra a utilização desses elementos em uma descrição de serviço *Web*. No exemplo de documento WSDL, um elemento *portType* (*TipoPortaAquisicao*) define uma operação chamada *pedirProdutos*. Essa operação inclui uma mensagem de entrada chamada *MensagemPedido*. Os componentes da mensagem são definidos utilizando o elemento *message*. Na parte concreta da descrição de serviço, um elemento *binding* é especificado para o *portType* definido na parte abstrata. O elemento *binding LigacaoAquisicao* determina, por exemplo, a utilização de HTTP como protocolo de comunicações. Finalmente, um serviço (*ServicoAquisicao*) define um elemento *port* chamado *PortaAquisicao*, que combina o elemento *binding* com o ponto de acesso de uma implementação do elemento *portType* definido no documento.

```
<definitions name="Aquisicao"
  xmlns:tns="http://exemplo.com/aquisicao/definicoes"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Parte Abstrata

message

```
<message name="MensagemPedido">
  <part name="NomeProduto" type="xs:string"/>
  <part name="Quantidade" type="xs:integer"/>
</message>
```

operation* e *portType

```
<portType name="TipoPortaAquisicao">
  <operation name="pedirProdutos">
    <input message="MensagemPedido"/>
  </operation>
</portType>
```

Parte Concreta

binding

```
<binding name="LigacaoAquisicao" type="tns:TipoPortaAquisicao">
  <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="pedirProdutos">
      <soap:operation soapAction="http://exemplo.com/pedirProdutos"/>
        <input>
          <soap:body use="literal"/>
        </input>
    </operation>
</binding>
```

port* e *service

```
<service name="ServicoAquisicao">
  <port name="PortaAquisicao" binding="tns:LigacaoAquisicao">
    <soap:address location="http://exemplo.com/aquisicao"/>
  </port>
</service>
```

```
...
</definitions>
```

Figura 3.3: Uma descrição de serviço *Web* WSDL.

3.2.2 UDDI

Universal Description Discovery & Integration (UDDI) [17] provê um repositório de descrições de serviços *Web*.

UDDI é apoiado pelo OASIS, um corpo de padronização que inclui as principais empresas da indústria de *software*.

O padrão UDDI oferece aos consumidores de serviços *Web* um mecanismo para localizar provedores, os serviços disponibilizados por eles e as informações necessárias para acessar os serviços.

Baseado em XML, UDDI provê uma infra-estrutura para integrar informação em ambientes de serviços *Web* tanto para serviços disponíveis publicamente quanto para serviços expostos internamente em organizações.

Apesar de desempenhar função semelhante, um repositório UDDI possui as seguintes diferenças em relação a outros serviços de diretório [13]:

- Independência: entradas podem ser postadas por qualquer organização e repositórios disponibilizados em diferentes tipos de plataforma;
- Flexibilidade: com UDDI é possível utilizar documentos com estrutura livre para a publicação de serviços.

Um repositório UDDI compreende informações sobre os seguintes itens principais: provedor de serviços, especificação de serviço e implementação de serviço.

A Tabela 3.1 mostra o mapeamento entre elementos WSDL e elementos UDDI para a publicação de serviços *Web*.

Elemento WSDL	**Elemento UDDI**
service	*businessService*
port e *binding*	*bindingTemplate*
portType	*tModel*
binding	*tModel*

Tabela 3.1: Mapeamento WSDL-UDDI

A Figura 3.4 apresenta um exemplo da estrutura utilizada para representar um serviço *Web* em um repositório UDDI. O elemento *businessService* contém o nome do serviço especificado. Além disso, ele inclui o identificador do provedor do serviço e um elemento *bindingTemplate*, o qual contém informações sobre uma implementação do serviço. Por exemplo, o elemento *bindingTemplate* inclui o ponto de acesso da implementação e referências para as especificações dos elementos WSDL *binding* e *portType* associados à implementação. Essas informações são definidas pelos elementos *accessPoint* e *tModelInstanceDetails*, respectivamente.

businessService

```
<businessService
  serviceKey="sek1234"
  businessKey="buk1234">
  <name>
    ServicoAquisicao
  </name>
  <bindingTemplates>
```

bindingTemplate

```
    <bindingTemplate
      bindingKey="bik1234"
      serviceKey="sek1234">
      <accessPoint URLType="http">
        http://exemplo.com/aquisicao
      </accessPoint>
```

tModelInstanceDetails

```
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="uuid:b1234">
          <description xml:lang="en">
            binding
          </description>
        </tModelInstanceInfo>
        <tModelInstanceInfo tModelKey="uuid:pt1234">
          <description xml:lang="en">
            portType
          </description>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
```

```
    </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    <keyedReference
      tModelKey="uuid:sn1234"
      keyName="service namespace"
      keyValue="http://exemplo.com/aquisicao/definicoes"/>
  </categoryBag>
...
</businessService>
```

Figura 3.4: Uma publicação de serviço *Web* UDDI.

O padrão UDDI compreende duas especificações que definem a estrutura e operação de um repositório de serviços:

- Uma definição da informação a ser provida sobre cada serviço;
- Uma interface de inserção e consulta para o repositório, que define como inserir informações em um repositório e como as informações armazenadas podem ser consultadas.

Inicialmente, o padrão UDDI foi projetado para apoiar um modelo conhecido como *Universal Business Registry* (UBR). No modelo UBR, o conteúdo de um repositório deve ser mantido consistente com o conteúdo dos outros repositórios e, portanto, alterações em publicações devem ser propagadas aos outros repositórios.

Atualmente, o padrão UDDI também apóia repositórios privados. Os seguintes tipos de repositório são permitidos [2]:

- Repositório público: repositório que provê acesso aberto aos dados registrados. Um repositório disponível como um *site* na *Web* publicado e conhecido é um exemplo de repositório público;
- Repositório privado: repositório interno, isolado por um *firewall* de uma rede privada. Organizações podem utilizar repositórios privados para apoiar a integração de suas próprias aplicações internas;
- Repositório compartilhado: repositório instalado em um ambiente controlado, sendo compartilhado com um grupo limitado de organizações. Por exemplo, uma rede de parceiros de negócio pode utilizar um repositório compartilhado para apoiar a integração de aplicações entre os parceiros.

3.2.3 SOAP

SOAP [46] é um protocolo que possibilita a comunicação entre serviços *Web*. O protocolo SOAP é apoiado pelo W3C. Além da estrutura de mensagem básica, o padrão descreve como receptores devem processar mensagens SOAP.

SOAP permite a troca de informação em um ambiente distribuído e descentralizado, tal como um ambiente típico de serviços *Web*. O mecanismo oferecido por SOAP para a troca de informação utiliza a linguagem XML.

Ao invés de definir um novo protocolo de comunicações, SOAP trabalha sobre protocolos existentes. Uma vantagem de empregar SOAP sobre HTTP é a possibilidade de atravessar *firewalls* sem a necessidade de configurações adicionais.

Uma mensagem SOAP inclui os seguintes componentes:

- Um elemento obrigatório para a identificação de mensagens SOAP;
- Um cabeçalho opcional que contém informação que pode ser processada por nós intermediários;
- Um corpo obrigatório que representa a mensagem sendo transmitida;
- Um elemento opcional com informação sobre falhas de processamento de mensagens.

O cabeçalho e o corpo de uma mensagem SOAP podem possuir múltiplas partes na forma de blocos. Um bloco é qualquer sub-elemento de primeiro nível do elemento *Header* ou *Body* de uma mensagem.

Para cada mensagem SOAP, além dos papéis de emissor e receptor final da mensagem, pode existir um número arbitrário de nós para o processamento e roteamento da mensagem. A informação central que um emissor deseja transmitir a um receptor deve ser incluída no corpo da mensagem. Informações adicionais necessárias para o processamento intermediário da mensagem, como, por exemplo, interação transacional, são inseridas no cabeçalho.

A estrutura de uma mensagem SOAP é influenciada principalmente pelo estilo de interação. SOAP pode ser utilizado em dois estilos de interação: documento e chamada de procedimento remoto (*Remote Procedure Call* – RPC). A seguir, esses estilos são descritos:

- *document-style*: as partes envolvidas na interação estabelecem um acordo sobre a estrutura dos documentos a serem trocados. Mensagens SOAP são empregadas para transportar os documentos entre as partes;
- *RPC-style*: nesse estilo de interação, uma mensagem SOAP encapsula uma requisição enquanto outra mensagem encapsula a resposta para a requisição. SOAP define uma convenção para representar requisições e respostas. O corpo de uma mensagem de requisição inclui uma chamada de procedimento, especificando o nome do procedimento sendo invocado e os parâmetros de entrada. O corpo de uma mensagem de resposta inclui o resultado e os parâmetros de saída.

A Figura 3.5 apresenta um exemplo de uma mensagem SOAP. A mensagem inclui uma solicitação de uma operação chamada *pedirProdutos*. Valores para os dois parâmetros da operação (*itemProduto* e *quantidade*) são definidos. O cabeçalho da mensagem inclui um bloco com um identificador de transação.

envelope

```
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
```

header

```
<env:Header>
  <t:transactionID
    xmlns:t="http://exemplo.com/transacao">
    123
  </t:transactionID>
</env:Header>
```

body

```
<env:Body>
  <m:pedirProdutos
    env:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
    xmlns:m="http://exemplo.com/aquisicao">
    <m:itemProduto>
      Produto
    </m:itemProduto>
    <m:quantidade>
      1
    </m:quantidade>
  </m:pedirProdutos>
</env:Body>
```

```
</env:Envelope>
```

Figura 3.5: Uma mensagem de requisição SOAP.

3.2.4 *WS-Policy*

Empresas desenvolvedoras de *software* e consórcios estão desenvolvendo uma coleção de padrões adicionais para serviços *Web*. Um exemplo é o padrão *Web Services Policy Framework* (*WS-Policy*) [4].

Embora WSDL seja o padrão para a descrição de funcionalidade de serviços *Web*, esforços de pesquisa recentes estão focando linguagens que possam aprimorar tal descrição. Esse aprimoramento inclui a consideração de aspectos que não são diretamente relacionados com o que um serviço faz ou como invocar um serviço *Web*.

WS-Policy representa um desses esforços e, devido à sua flexibilidade e capacidade de extensão, é um candidato para tornar-se um futuro padrão para a especificação de políticas para serviços *Web* [7].

O padrão *WS-Policy* é apoiado por um consórcio formado por diversas empresas, como, por exemplo, BEA, IBM, *Microsoft*, SAP e *VeriSign*. Esse consórcio foi responsável pela submissão de especificações nas áreas de transações, confiabilidade de mensagens, dentre outras, a corpos de padronização de serviços *Web*.

WS-Policy compreende um modelo para definir propriedades de serviços *Web* e uma sintaxe correspondente para expressar essas definições como políticas. Referências para políticas podem ser incluídas em documentos XML [3]. Políticas podem ser aplicadas em diferentes níveis, tais como, mensagem, operação e serviço.

Em *WS-Policy*, uma política é uma coleção de alternativas. Cada alternativa de política é uma coleção de asserções. Tipicamente, asserções de política especificam características que são importantes para a seleção e utilização apropriadas de serviços *Web*, por exemplo, características não-funcionais [4].

O padrão *WS-Policy* define um formato normalizado para especificar políticas com o objetivo de facilitar sua manipulação. A forma normal de política *WS-Policy* é uma disjunção das alternativas de uma política e uma conjunção das asserções em cada uma das alternativas da política.

3.3 Características Não-Funcionais em Serviços *Web*

A consideração de características não-funcionais em serviços *Web* proporciona diversos benefícios. A publicação de características não-funcionais de serviços *Web* auxilia na seleção entre serviços com a mesma funcionalidade. As descrições de características não-funcionais publicadas possibilitam a composição de serviços *Web* baseada em critérios de qualidade. Elas também tornam possível a avaliação de caminhos de execução alternativos para a adaptação de processos.

Além disso, o monitoramento de atributos de qualidade ajuda a alcançar os níveis de qualidade esperados por consumidores. Ele também permite que provedores detectem problemas e acomodem mudanças [65].

A Tabela 3.2 descreve características não-funcionais [45] [55] [42] [48].

Característica	**Descrição**
Tempo de resposta	O intervalo de tempo demandado para que um serviço complete sua tarefa.
Taxa de serviço	A taxa de processamento de requisição oferecida por um serviço.
Escalabilidade	A taxa de crescimento da taxa de serviço em um determinado intervalo de tempo.
Capacidade	O número de requisições concorrentes permitido por um serviço.
Disponibilidade	A porcentagem de tempo em que um serviço está operante.
Confiabilidade	O intervalo de tempo para a continuidade de serviço correto e para a transição para estado correto.
Custo	A medida do custo envolvido na utilização de um serviço.
Segurança	Define se um serviço oferece mecanismos de confidencialidade, integridade, autenticação e autorização.
Transmissão de mensagem confiável	Determina se um serviço oferece mecanismos para garantir entrega de mensagem confiável.
Integridade transacional	Define se um serviço oferece propriedades transacionais.
Interoperabilidade	Determina se um serviço é compatível com perfis de interoperabilidade.

Tabela 3.2: Características não-funcionais de serviços *Web*

Capítulo 4

Trabalhos Relacionados

A qualidade de serviço no contexto de sistemas distribuídos gerou um trabalho amplo. No contexto de serviços *Web*, alguns problemas específicos surgiram e, recentemente, atividades de pesquisa estão sendo conduzidas para tratá-los. Neste capítulo são apresentados alguns trabalhos envolvendo a gerência de características não-funcionais em serviços *Web*.

A necessidade de incluir características não-funcionais em padrões de serviços *Web* foi indicada em trabalhos prévios na academia [45] e na indústria [40]. Além disso, Leymann et al [40] discutem a importância da gerência de características não-funcionais na gerência de processos de negócio, incluindo a necessidade de modelar processos de acordo com características não-funcionais para selecionar suas partes componentes. Alguns pontos para serem resolvidos incluem:

- Especificação de características não-funcionais de serviços *Web*;
- Publicação de atributos de qualidade de serviço e descoberta de serviços considerando parâmetros de qualidade;
- Verificação de características não-funcionais com o objetivo de garantir níveis de qualidade.

Essas questões incluem desafios relacionados a atributos de qualidade [55] e modelos [60]. Em [56], uma classificação inicial de atributos de qualidade para sistemas distribuídos é apresentada. O W3C [38] lista alguns requisitos para a gerência de características não-funcionais em serviços *Web*. Em [55], alguns atributos de qualidade para serviços *Web* são discutidos. A abordagem de verificar características não-funcionais de serviços *Web* em cada execução de serviço é baseada na proposta de Kalepu et al [34]. Em [34], os autores definem um atributo para capturar a conformidade de serviços. Tais estudos oferecem uma contribuição inicial para a inclusão de atributos de qualidade em serviços *Web*.

Os trabalhos correlatos apresentados a seguir focam uma ou várias das questões listadas. Cada uma das seções reúne alguns trabalhos que apresentam abordagens para tratar de aspectos da gerência de características não-funcionais.

4.1 Especificação de Características Não-Funcionais

Na literatura foram apresentadas diversas abordagens com o objetivo de especificar características não-funcionais de serviços *Web*.

4.1.1 *Web Service Offerings Language*

Web Service Offerings Language (WSOL) [64] é uma notação XML para a especificação de múltiplas ofertas de serviço para um mesmo serviço *Web*. Uma oferta de serviço é uma representação de uma classe de serviço para um serviço *Web*. Uma classe de serviço é uma variação da funcionalidade e da qualidade oferecidas por um serviço.

Classes de serviço podem ser diferentes com relação a privilégios de utilização, prioridades de serviço, tempos de resposta garantidos aos consumidores, dentre outros fatores. Diferentes classes de serviço demandam diferentes modos de utilizar os recursos de *hardware* e *software*, e possuem diferentes custos.

A sintaxe da linguagem WSOL é definida utilizando o padrão XSD. A linguagem WSOL complementa o padrão WSDL com alguns componentes necessários para a implementação do mecanismo de ofertas de serviço. Ela permite a especificação de atributos funcionais e não-funcionais, entidades responsáveis pelo monitoramento de restrições em ofertas de serviço e relacionamentos entre ofertas. As características não-funcionais incluem tempo de resposta, confiabilidade e disponibilidade.

Na Figura 4.1, um exemplo de oferta de serviço WSOL é mostrado.

```
<wsol:offeringType name="buyStockO1" ...>
  <wsol:QoSconstraintList ...>
    <wsol:QoSconstraint name="MaxResponseTime" ...>
      <wsol:QoSname qName="QoSns:responsetime"/>
      <wsol:QoStype typeName="QoSns:max"/>
      <wsol:qValue>50</wsol:qValue>
    </wsol:QoSconstraint>
  </wsol:QoSconstraintList>
</wsol:offeringType>
```

Figura 4.1: Exemplo de descrição de atributos de qualidade na linguagem WSOL.

A oferta de serviço (Figura 4.1) especifica o tempo de resposta. A descrição define o limite máximo do tempo de resposta oferecido pela oferta de serviço.

A proposta da linguagem WSOL inclui uma abordagem baseada em classe de serviço. Ela é compatível com o padrão WSDL, porém não se integra no padrão UDDI.

4.1.2 *Web Service Level Agreement*

Keller e Ludwig [35] descrevem um modelo para especificar e monitorar SLAs (*Service Level Agreements*) para serviços *Web*. Um aspecto importante de um contrato para serviços de tecnologia da informação é o conjunto de garantias da qualidade dos serviços, referido como SLA.

O modelo *Web Service Level Agreement* (WSLA) inclui uma linguagem e uma arquitetura de tempo de execução. A linguagem WSLA é baseada no padrão XSD. Ela permite a definição de garantias de qualidade para serviços *Web*. A arquitetura provê mecanismos para verificar atributos de serviços de acordo com uma especificação WSLA.

Um documento WSLA referencia um documento WSDL e complementa a definição de serviço com informações para a gerência de SLAs. Tipicamente, um documento WSLA define diversos parâmetros, tais como, disponibilidade e tempo de resposta.

A Figura 4.2 apresenta uma visão geral do ciclo de vida da gerência de SLAs.

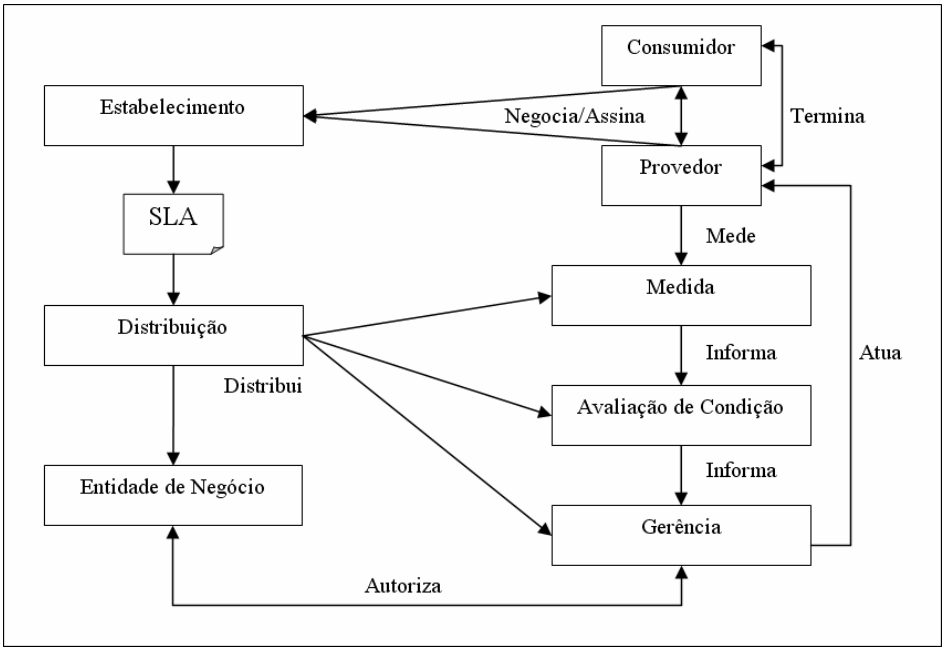

Figura 4.2: Modelo WSLA.

As fases do ciclo ilustrado na Figura 4.2 são descritas a seguir:

- Estabelecimento: o acordo SLA é negociado e assinado pelas partes. Um consumidor utiliza os atributos oferecidos por um provedor para a definição dos parâmetros do SLA;

- Distribuição: as cláusulas apropriadas do SLA são distribuídas às partes envolvidas. Além das partes signatárias de um SLA, partes de apoio podem estar envolvidas;
- Medida de nível de serviço: a arquitetura verifica os parâmetros do SLA, recuperando as medidas diretamente dos serviços, testando os serviços ou interceptando as invocações dos consumidores. Ela compara os atributos medidos com os limites definidos no SLA;
- Ações corretivas: quando a violação de um termo do SLA é detectada, ações corretivas podem ser executadas. A entidade de negócio é consultada para verificar se as ações propostas são permitidas. As ações corretivas são limitadas e tipicamente representam o envio de um evento ao sistema de gerência do provedor;
- Término: o SLA pode conter as condições necessárias para a sua conclusão e as penalidades que uma parte ficará sujeita se transgredir cláusulas do SLA. Negociações podem ser efetuadas entre as partes para finalizar o SLA. Além disso, uma data de expiração pode ser especificada no SLA.

A abordagem explorada no modelo WSLA é baseada em contrato. O modelo oferece apoio para acordos bilaterais entre um provedor e um consumidor para a utilização de serviços *Web*. Ele auxilia na garantia de níveis de qualidade, mas ambientes abertos dinâmicos não são apoiados.

4.1.3 *GlueQoS*

Wohlstadter et al [69] apresentam uma abordagem para gerenciar características não-funcionais de componentes de *software*. A abordagem, chamada *GlueQoS*, foca mudanças dinâmicas em atributos de qualidade.

A abordagem inclui:

- Uma linguagem para especificar atributos de qualidade;
- Um *middleware* que utiliza as especificações para encontrar dinamicamente um conjunto de características não-funcionais que permite a interação entre dois componentes.

A linguagem é uma extensão de uma versão precedente da linguagem *WS-Policy*. A linguagem *GlueQoS* é utilizada para definir políticas que especificam características não-funcionais de componentes e testes de tempo de execução para considerar as condições operacionais.

A Figura 4.3 mostra o *middleware* de mediação proposto.

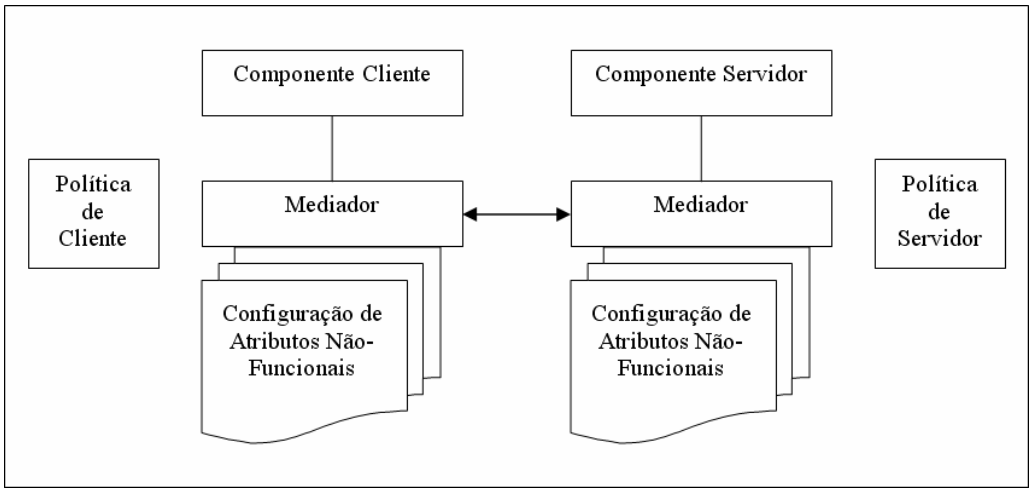

Figura 4.3: *Middleware* de mediação *GlueQoS*.

No modelo *GlueQoS*, um mediador de política é adicionado a cada componente. Os mediadores inspecionam as configurações de atributos não-funcionais nos lados cliente e servidor. Eles trocam informações para selecionar o conjunto de atributos de qualidade adequado. Tal comunicação é realizada utilizando um protocolo de mediação de política.

Os mediadores iniciam uma interação com a avaliação das políticas baseada nas condições de tempo de execução. O objetivo da avaliação preliminar é reduzir as políticas e determinar os conjuntos de asserções aceitáveis. Em seguida, eles trocam informações sobre as políticas. Os mediadores verificam a compatibilidade entre as políticas para definir a possibilidade de interação entre os componentes.

O modelo *GlueQoS* introduz uma abordagem baseada em *middleware*. O modelo se situa na área de componentes de *software*. Apesar de utilizar o padrão *WS-Policy*, ele não considera alguns padrões básicos de serviços *Web*, incluindo WSDL e UDDI. As questões de descoberta de componentes e verificação de atributos de qualidade não são tratadas.

4.1.4 *Web Services Endpoint Language*

Em [61], uma abordagem para considerar aspectos não-funcionais na descoberta de serviços *Web* é descrita. A abordagem aplica a linguagem *Web Services Endpoint Language* (WSEL), cuja necessidade foi prevista na especificação da linguagem *Web Services Flow Language* (WSFL) [37]. WSEL é um formato XML para a descrição de características não-funcionais de serviços.

A abordagem requer que cada serviço publicado no repositório UDDI esteja ligado a uma descrição WSDL. As descrições WSDL devem estar ligadas a arquivos WSEL. Os arquivos WSEL possuem detalhes sobre as características não-funcionais dos serviços definidos nos arquivos WSDL. Os atributos de qualidade considerados incluem tempo de resposta, custo e confiabilidade.

Ao especificar os requisitos para um serviço, o consumidor pode especificar os critérios de qualidade a serem satisfeitos pelo serviço requerido. Os requisitos de qualidade são especificados utilizando um documento XML específico. O documento também inclui os critérios funcionais para a busca de serviços. Ele utiliza um elemento para representar uma consulta de descoberta de serviços padrão do UDDI. A lista de serviços descobertos pode ser utilizada pelo consumidor para selecionar um serviço apropriado.

A Figura 4.4 mostra um exemplo de descrição de atributos de qualidade na linguagem WSEL.

```
<wsel>
  <qos>
    <operation name="delivery">
      <delaytime unit="millisecond">45</delaytime>
      <processtime unit="second">3</processtime> ...
  </qos>
</wsel>
```

Figura 4.4: Exemplo de descrição não-funcional de serviço na linguagem WSEL.

A descrição não-funcional de serviço (Figura 4.4) define dois parâmetros de qualidade de um serviço. A definição especifica os valores dos tempos de retardo e processamento, os quais são componentes do atributo tempo de resposta.

A abordagem proposta por Sivashanmugam et al utiliza a linguagem WSEL. WSEL não é um padrão aberto. Além disso, uma abordagem para a verificação dos atributos de qualidade não é apoiada.

4.2 Publicação e Descoberta Incluindo Características Não-Funcionais

Alguns trabalhos na área de serviços *Web* incluem mecanismos para possibilitar a publicação de características não-funcionais de serviços e a consideração dessas características na descoberta de serviços.

4.2.1 UDDI *eXtension*

Em [19], o sistema UX (UDDI *eXtension*) é proposto para auxiliar os consumidores na descoberta de serviços com atributos de qualidade requeridos, incluindo tempo de resposta, confiabilidade e custo.

No modelo apresentado, a avaliação dos consumidores sobre a qualidade dos serviços invocados é considerada. Por meio do compartilhamento das experiências dos consumidores, o sistema gera sumários que especificam características não-funcionais dos serviços.

Cada domínio de rede inclui um sistema UX. O compartilhamento acontece entre consumidores no mesmo domínio, considerando que, em um domínio, todos os consumidores observam atributos de qualidade similares.

O consumidor deve descrever suas preferências em um perfil. Por exemplo, um consumidor pode preferir serviços com tempo de resposta menor ou com custo menor. Uma aplicação cliente para o servidor UX deve ser utilizada pelo consumidor para fazer as consultas. O sistema gera o resultado de acordo com o perfil do consumidor.

A Figura 4.5 mostra os componentes do sistema UX.

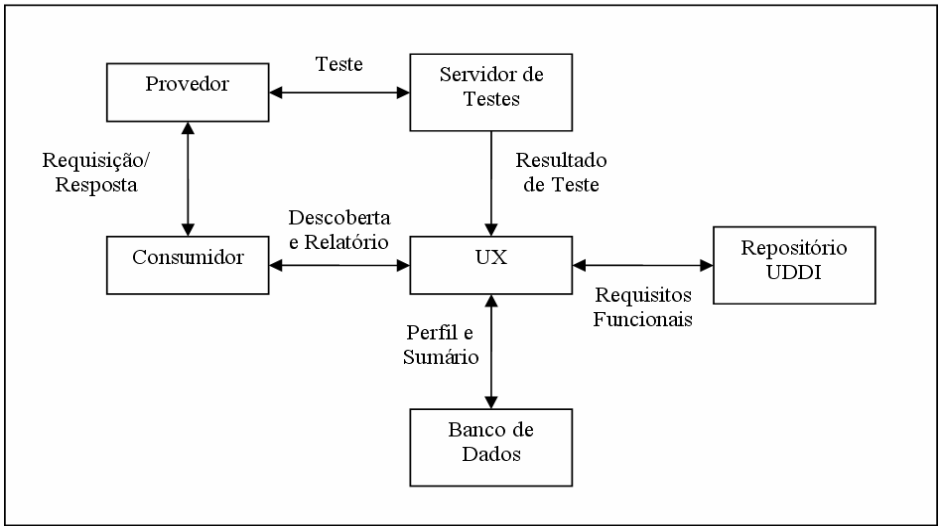

Figura 4.5: Sistema UX.

O consumidor consulta o servidor UX para descobrir serviços. Então, o consumidor invoca os serviços e mede a qualidade dos mesmos. Em seguida, ele gera relatórios para registrar informações sobre a qualidade e os envia em lote para o servidor. Para permitir a geração automática de relatórios, uma aplicação de relatório para o lado do cliente é provida.

No modelo, um repositório UDDI padrão registra as descrições dos serviços do domínio local. O repositório UDDI é conectado ao servidor UX como um repositório secundário utilizando conexões SOAP.

O servidor UX gera um sumário dos relatórios para cada serviço regularmente. Os sumários são utilizados na descoberta de serviços. O consumidor inclui seu identificador na consulta e o servidor extrai as informações do perfil do consumidor armazenado no banco de dados. O servidor ordena os serviços retornados pelo UDDI de acordo com os sumários e as preferências do consumidor. O consumidor pode escolher entre os serviços mais apropriados na lista retornada.

Quando o sistema não possui relatórios recentes para um serviço, pode não ser possível determinar a qualidade do serviço. Um servidor de testes é projetado para gerar relatórios para os serviços registrados no repositório local.

O sistema UX utiliza uma abordagem baseada na avaliação da qualidade dos serviços realizada pelos consumidores. Os consumidores geram relatórios que são utilizados para gerar sumários sobre algumas características não-funcionais dos serviços. O sistema introduz um repositório complementar para armazenar os sumários. Os consumidores podem influenciar de modo incorreto a seleção de serviços.

4.2.2 *UDDIe*

UDDIe, uma extensão para o padrão UDDI, é apresentada em [62]. A extensão define o conceito de páginas azuis. As páginas azuis permitem ao provedor descrever algumas propriedades não-funcionais dos serviços oferecidos. Além disso, a extensão permite a descoberta de serviços baseada em tais propriedades. As páginas azuis complementam os três tipos de informação mantidos pelo repositório UDDI padrão:

- Informações básicas de identificação e contato sobre organizações (páginas brancas);
- Informações para possibilitar a listagem de organizações baseada em suas categorias industriais (páginas amarelas);
- Detalhes de interface para indicar como um serviço deve ser invocado (páginas verdes).

O apoio para a busca por serviços utilizando atributos de qualidade é oferecido pela inclusão do tipo de estrutura de dados *propertyBag*. O novo tipo estende o tipo de estrutura *businessService* no modelo informacional do padrão UDDI. Além disso, as chamadas de API do UDDI para a publicação e descoberta de serviços são estendidas para a inclusão de atributos de qualidade.

O repositório *UDDIe* é empregado no contexto de computação em grade, para apoiar a gerência de atributos de qualidade. No contexto considerado, um serviço é um código científico ou uma rotina matemática. Ele possui atributos codificados em sua interface, tais como, requisitos de largura de banda, UCP e memória.

A Figura 4.6 mostra um exemplo de um documento WSDL que especifica um serviço de acordo com o modelo proposto.

```
<?xml version="1.0" encoding="UTF-8"?>
  <wsdl:definitions
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" ...
    targetNamespace="http://ServiceInterface">
      <wsdl:message name="printNameResponse">
      </wsdl:message> ...
    <QoS>
      <network_bandwidth>512K</network_bandwidth>
      <memory>64MB</memory> ...
    </QoS>
  </wsdl:definitions>
```

Figura 4.6: Exemplo de descrição de serviço WSDL no modelo *UDDIe*.

No modelo *UDDIe*, o padrão WSDL é estendido para especificar as características não-funcionais diretamente nas interfaces dos serviços. A linguagem WSDL é projetada para descrever a funcionalidade dos serviços *Web*. Tipicamente, os aspectos funcionais são mais fixos que os aspectos não-funcionais. Uma maior flexibilidade é alcançada utilizando um documento separado para descrever os atributos de qualidade. Os atributos de qualidade podem mudar sem alterar o documento WSDL.

Um mediador é responsável por processar as requisições. Ele encaminha as requisições ao repositório *UDDIe*. O mediador seleciona o serviço mais apropriado de acordo com os requisitos do consumidor e retorna o resultado.

A abordagem empregada no repositório *UDDIe* é baseada na extensão da estrutura *businessService* do UDDI para a inclusão de atributos de qualidade no contexto de computação em grade. Uma infra-estrutura utiliza a extensão para gerenciar características não-funcionais em serviços de grade. Diferentes implementações de um mesmo serviço com diferentes características não-funcionais não são apoiadas adequadamente. Além disso, alterações potenciais em atributos de qualidade não são consideradas.

4.2.3 *Advanced* UDDI *Search Engine*

Em [71], a máquina de busca *Advanced* UDDI *Search Engine* (AUSE) provê interfaces para descobrir serviços em repositórios UDDI. Uma linguagem baseada em XML, chamada UDDI *Search Markup Language*, é proposta. Ela é utilizada para especificar requisições de descoberta compreendendo critérios não-funcionais.

A máquina de busca proposta aprimora o desenvolvimento de aplicações de negócio eletrônico.

A Figura 4.7 mostra um modelo para a integração de negócios que utiliza a máquina de busca proposta para repositórios UDDI.

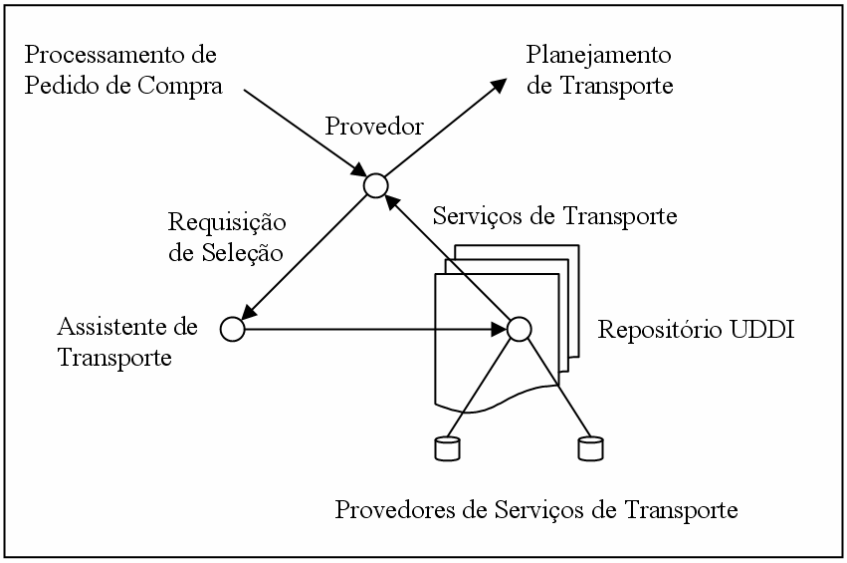

Figura 4.7: Modelo para integrar negócios utilizando a máquina de busca avançada para UDDI.

No modelo apresentado na Figura 4.7, um provedor seleciona um provedor de serviços de transporte para um pedido de compra utilizando um assistente de transporte. O assistente de transporte é projetado para descobrir serviços em repositórios UDDI e filtrar os resultados das buscas utilizando os dados dos pedidos de compra.

Após a recuperação dos serviços de transporte que apresentam as características específicas de transporte definidas no pedido de compra, o provedor escolhe os serviços para os quais ele deseja obter informações sobre custo. Em seguida, após verificar as informações retornadas, ele pode selecionar um serviço de transporte para o pedido de compra específico.

A abordagem proposta na máquina de busca AUSE é baseada na filtragem manual de serviços utilizando critérios não-funcionais. Políticas para atributos de qualidade não são utilizadas e aspectos empíricos não são considerados na seleção de serviços.

4.2.4 *Web Services Agent Framework*

Maximilien e Singh [48] apresentam um modelo de agentes, chamado *Web Services Agent Framework* (WSAF), com uma ontologia para atributos de qualidade de serviço. Os agentes colaboram para determinar a qualidade dos serviços e selecionar os serviços mais adequados para os consumidores.

Um agente de serviço é criado para cada interface de serviço. Ele expõe a interface de serviço, incrementada com a funcionalidade para capturar as preferências de atributos de qualidade dos consumidores e consultar as agências.

Uma ontologia é utilizada pelos agentes para determinar a compatibilidade entre as capacidades não-funcionais dos serviços e os requisitos dos consumidores.

Agências armazenam, agregam e apresentam informações não-funcionais aos agentes. Elas possibilitam o compartilhamento de informações não-funcionais entre os agentes. Cada atributo de qualidade é mantido em uma agência própria.

A Figura 4.8 mostra o modelo proposto.

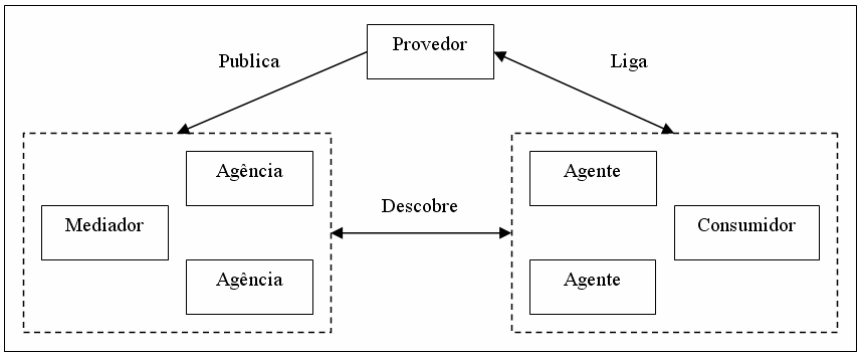

Figura 4.8: Modelo de agentes para a seleção de serviços.

O provedor publica os serviços em repositórios UDDI e agências, nas quais são registradas as informações não-funcionais. O consumidor invoca um agente de serviço com sua política para características não-funcionais. O agente seleciona um serviço utilizando o repositório UDDI por meio de um mediador e as agências. O agente monitora a execução do serviço. Quando o serviço finaliza a execução, o agente armazena as informações obtidas nas agências relevantes.

A abordagem proposta por Maximilien e Singh é baseada em agentes de *software*. As informações não-funcionais sobre os serviços não são mantidas e compartilhadas por meio do padrão UDDI, mas utilizando agências. Uma linguagem para políticas é definida, mas ela não é baseada no padrão *WS-Policy*.

4.2.5 Modelo de Lee et al., 2005

Em [39], um modelo é proposto para apoiar a utilização de serviços *Web* considerando as características não-funcionais dos serviços. Três componentes são adicionados ao modelo de serviços *Web* básico: servidor, mediador e agente para atributos de qualidade.

A Figura 4.9 apresenta as interações entre os componentes do modelo proposto.

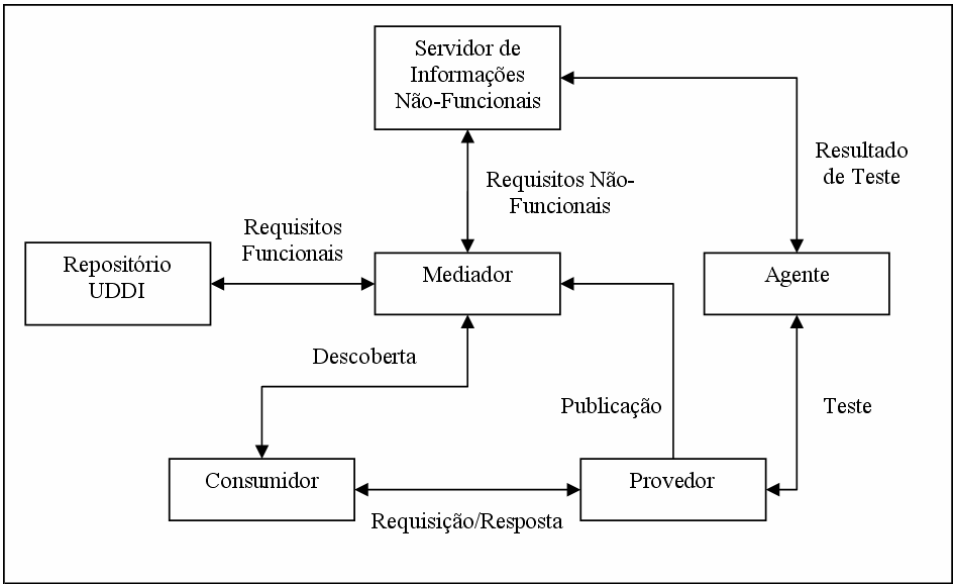

Figura 4.9: Modelo para apoiar a utilização de serviços incluindo características não-funcionais.

Um mediador descobre serviços utilizando as informações incluídas em consultas XML enviadas pelo consumidor. Uma consulta inclui não apenas as informações sobre funcionalidade contidas em uma consulta UDDI padrão, mas também algumas propriedades de qualidade e informações sobre a condição do consumidor. As propriedades não-funcionais consideradas são: reputação, tempo de resposta, disponibilidade e confiabilidade. As informações sobre a condição do consumidor incluem período e área.

Um servidor de informações não-funcionais possui dados históricos sobre cada serviço nele registrado. Um agente testa os serviços registrados em um servidor regularmente, de acordo com as informações enviadas pelo servidor. Ele utiliza dados de teste gerados a partir dos arquivos WSDL dos serviços. Os resultados dos testes são armazenados no servidor.

Para a descoberta de serviços no modelo proposto, o mediador extrai a parte funcional da consulta e a envia ao repositório UDDI. Se algum serviço retornado pelo UDDI não foi registrado no servidor anteriormente, ele é registrado pelo mediador. Em seguida, o servidor calcula os valores dos atributos de qualidade utilizando seus dados históricos e

considerando as condições de busca. Os dados resultantes da avaliação de cada serviço e gráficos com os dados históricos são mostrados para os serviços descobertos.

Após selecionar e utilizar um serviço *Web*, o consumidor pode dar uma nota, a qual é utilizada no atributo reputação.

Na abordagem de Lee et al, a seleção de serviços é baseada em dados históricos sobre execuções de serviços e condições de consumidores. Um repositório separado é utilizado para armazenar tais informações.

4.2.6 Modelo de Kokash, 2005

Kokash [36] propõe um modelo para aprimorar a descoberta de serviços *Web*. O modelo permite a troca de recomendações sobre a qualidade de serviços *Web*. Estatísticas de invocações de serviços podem ser utilizadas para reduzir a probabilidade de falhas.

A Figura 4.10 apresenta o modelo proposto.

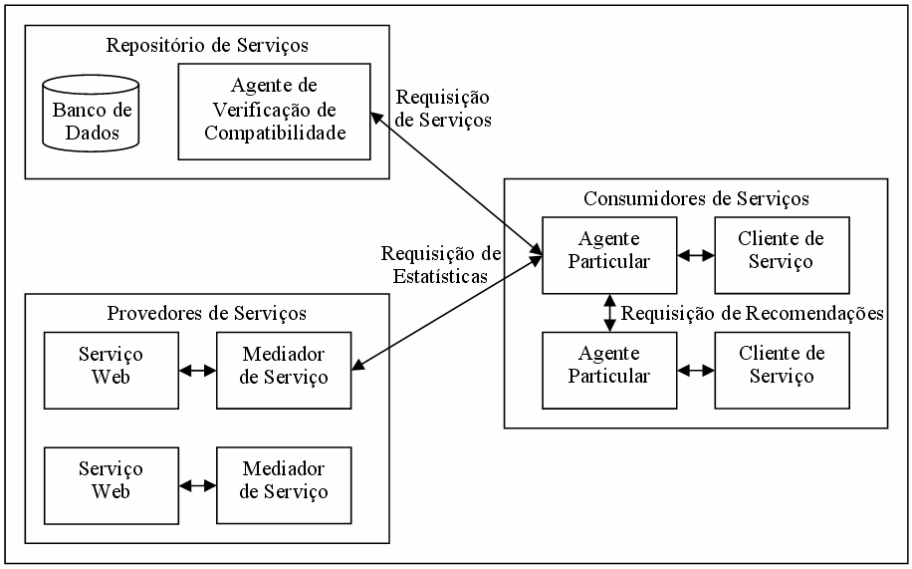

Figura 4.10: Modelo para a descoberta de serviços baseado em um sistema de recomendações.

Cada consumidor possui um agente dedicado. O agente particular recebe as requisições do seu consumidor associado e as redireciona ao agente de verificação de compatibilidade. O agente de verificação gerencia o repositório de serviços. Ele busca por serviços que podem satisfazer a requisição do consumidor. Além disso, ele gera um histórico das requisições. O agente de verificação retorna ao agente particular os serviços relevantes e informa os agentes que procuraram por serviços similares anteriormente. Assim, o agente particular pode solicitar recomendações aos outros agentes.

Dois cenários são possíveis após a solicitação de recomendações:

- Os agentes que invocaram os serviços anteriormente apresentam suas próprias classificações ao agente particular;
- Os agentes apresentam partes dos seus históricos de invocações ao agente particular, deixando para ele a tarefa de processá-las.

Mediadores mantêm estatísticas sobre as invocações atendidas pelos serviços. O agente particular pode comparar os parâmetros providos por agentes e mediadores para obter dados mais confiáveis.

O agente particular classifica os serviços. Finalmente, o serviço mais apropriado é invocado. O agente pergunta ao consumidor se ele deve invocar sempre o serviço utilizado para a mesma consulta. A resposta do consumidor pode influenciar a classificação de serviços dos outros agentes. Se um consumidor continua invocando um determinado serviço *Web*, o agente considera que ele está satisfeito com o serviço. Portanto, a utilização repetida de um serviço influencia positivamente a avaliação da qualidade do serviço.

O modelo proposto por Kokash utiliza uma abordagem baseada na aplicação de um sistema de recomendações para auxiliar na escolha de serviços. Consumidores podem afetar incorretamente a reputação de serviços.

4.3 Verificação de Características Não-Funcionais

A seguir são apresentados alguns trabalhos que propõem mecanismos para verificar características não-funcionais de serviços *Web*.

4.3.1 Modelo de Ran, 2003

Ran [55] indica algumas causas da lenta taxa de adoção da tecnologia de serviços *Web*. Segundo o autor, um fator muito importante é a falta de informações sobre atributos de qualidade dos serviços. Não é possível esperar que organizações de negócio desejem procurar por serviços com base em requisitos funcionais e invocar os serviços sem a garantia de saber antecipadamente se os atributos de qualidade esperados seriam satisfeitos.

Um modelo de serviços *Web* estendido, mostrado na Figura 4.11, é proposto. Ele inclui um novo componente, chamado certificador de atributos de qualidade de serviço. O certificador é responsável por verificar as declarações de atributos de qualidade dos provedores antes da publicação dos serviços.

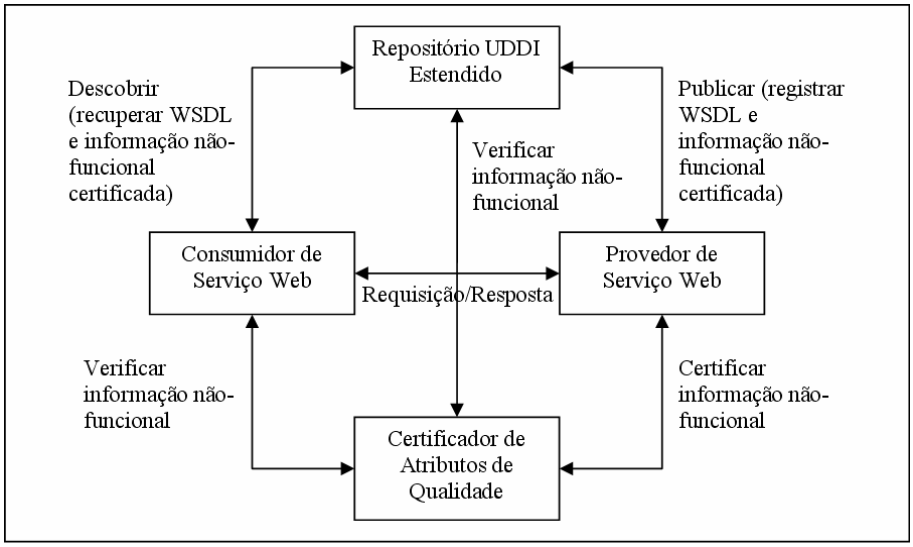

Figura 4.11: Modelo para a certificação de serviços *Web*.

O provedor deve informar as características não-funcionais do serviço ao certificador. O certificador pode ajustar as informações declaradas, caso variações sejam detectadas. O resultado é registrado no repositório do certificador e retornado ao provedor, juntamente com um identificador para a certificação. O repositório UDDI deve verificar a existência do certificado antes de publicar o serviço.

Os consumidores podem verificar a validade de informações sobre características não-funcionais utilizando os identificadores de certificação.

No modelo, o repositório UDDI é estendido para incluir informações sobre atributos de qualidade. Um tipo de estrutura de dados é associado ao tipo *businessService*. O novo tipo se liga a *tModels* como referência para a terminologia de informações não-funcionais que não faz parte do padrão UDDI.

A abordagem proposta por Ran é baseada em certificação. Ela não oferece apoio adequado para o dinamismo típico de serviços *Web*.

4.3.2 Modelo de Singhera, 2004

Em [60], o autor identifica algumas limitações do modelo de serviços *Web*, tais como:

- O papel de um monitor está ausente no modelo de serviços *Web*;
- As informações sobre serviços em repositórios UDDI são estáticas, definidas no momento da publicação dos serviços;
- As consultas feitas pelos consumidores são completamente baseadas em características funcionais dos serviços;

- O modelo existente não impõe uma verificação para motivar os provedores a aprimorarem e manterem a qualidade dos seus serviços.

Considerando as limitações identificadas, um novo modelo é proposto (Figura 4.12).

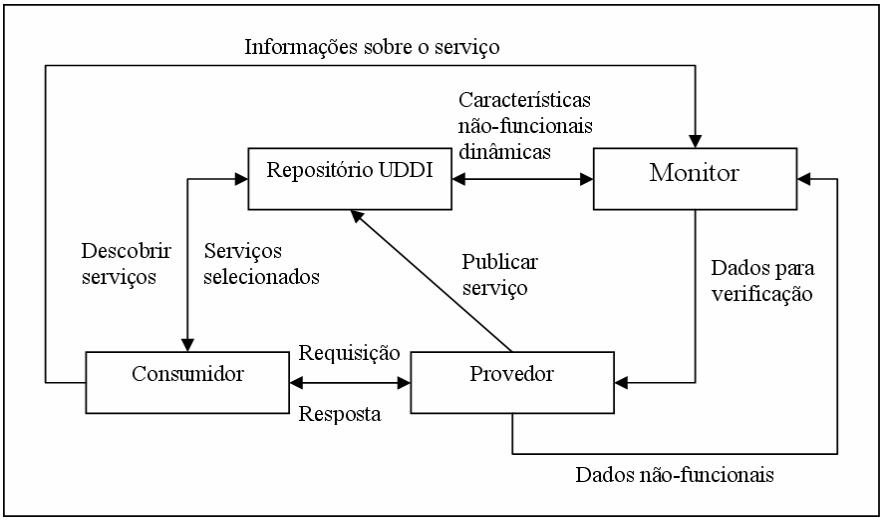

Figura 4.12: Modelo para o monitoramento de serviços.

O provedor especifica o serviço *Web* utilizando uma extensão da linguagem WSDL. A descrição de serviço aprimorada pode incluir características não-funcionais. Ela define se o serviço pode ser considerado em descobertas de serviços envolvendo critérios de qualidade.

Os provedores publicam os serviços nos repositórios UDDI. Os repositórios informam aos monitores a publicação de novos serviços. Os monitores coletam informações não-funcionais e as armazenam em seus repositórios próprios. As características não-funcionais são verificadas freqüentemente. Além dos resultados de monitoramento, informações sobre os serviços providas pelos consumidores são consideradas.

Os consumidores enviam requisições aos repositórios UDDI especificando requisitos funcionais e não-funcionais. Os repositórios são responsáveis por descobrir os serviços que satisfazem os critérios funcionais e não-funcionais estáticos. Eles devem enviar requisições aos monitores para determinar os serviços que satisfazem os critérios não-funcionais dinâmicos. Referências para os serviços selecionados são retornadas aos consumidores.

A abordagem proposta por Singhera lida com a questão do dinamismo de serviços *Web*. Ela é baseada em monitoramento. No modelo, a coleta de informações não-funcionais não é realizada em paralelo com a execução do serviço.

4.3.3 Modelo de Serhani et al., 2005

Em [58], é apresentado um modelo baseado em mediadores para serviços *Web*. O papel de um mediador é garantir o nível de qualidade selecionado na execução de serviços *Web*.

Uma abordagem para a verificação de características não-funcionais é proposta. A verificação é realizada por um mediador em duas fases:

- A primeira fase inclui as análises sintática e semântica da descrição da interface do serviço, incluindo a descrição dos parâmetros de qualidade;
- A segunda fase inclui a certificação das propriedades de qualidade declaradas na interface do serviço. A certificação é executada utilizando casos de teste para medir os parâmetros de qualidade.

O modelo inclui um repositório *UDDIe* [62] para apoiar a publicação e descoberta de serviços *Web* enriquecidos com características não-funcionais. Os seguintes atributos são considerados: tempo de resposta, custo e disponibilidade. As características não-funcionais de um serviço são especificadas na interface WSDL do serviço.

Os componentes do modelo e as interações são apresentados na Figura 4.13.

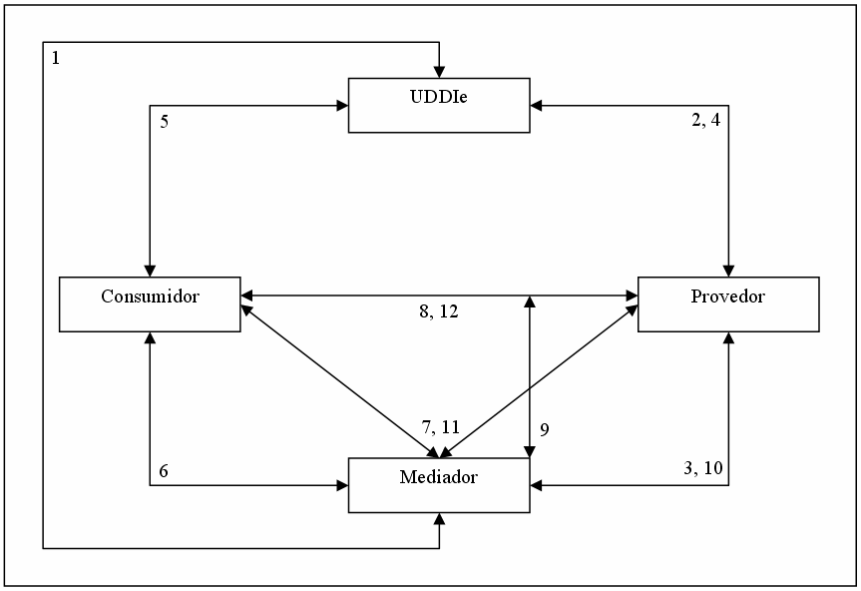

Figura 4.13: Modelo baseado em mediadores para serviços *Web*.

A seguir, os passos mostrados na Figura 4.13 são descritos:

- Passo 1: um mediador publica sua interface em um repositório *UDDIe*;
- Passo 2: um provedor procura pela interface de um mediador em um repositório;
- Passo 3: o provedor requisita ao mediador a certificação de um serviço;
- Passo 4: após a certificação, o provedor publica o serviço em um repositório *UDDIe*;
- Passo 5: um consumidor busca por serviços em um repositório;
- Passo 6: o consumidor pode confirmar com um mediador se as propriedades de um serviço selecionado foram certificadas;
- Passo 7: o mediador arbitra a definição da classe de serviço a ser utilizada;
- Passo 8: o consumidor invoca o serviço *Web* utilizando a classe de serviço selecionada;
- Passo 9: durante a utilização do serviço, o mediador pode monitorar o serviço;
- Passo 10: se o serviço violar a especificação, o mediador notifica o provedor, que inicia uma adaptação dos parâmetros para manter o nível de qualidade selecionado;
- Passo 11: se a adaptação falhar, uma outra classe de serviço deve ser selecionada;
- Passo 12: o processo termina com a liberação dos recursos e a emissão da conta correspondente para o pagamento.

A abordagem proposta por Serhani et al inclui a verificação de atributos de qualidade manipulados utilizando o repositório *UDDIe* [62]. Embora o modelo proposto considere importante uma fase de monitoramento, o trabalho enfatiza os processos de análise e certificação de características não-funcionais.

4.3.4 *Ad-UDDI*

Du et al [23] apresentam um servidor de repositório chamado *Ad-UDDI*. O servidor verifica o estado de serviços e coleta informações sobre os serviços periodicamente.

Depois de ser disparado por um cronômetro, o servidor *Ad-UDDI* inicia a verificação do estado real de serviços. Se o serviço monitorado está atualizado, o servidor executa o processo de atualização de informações.

A Figura 4.14 ilustra o mecanismo de monitoramento proposto.

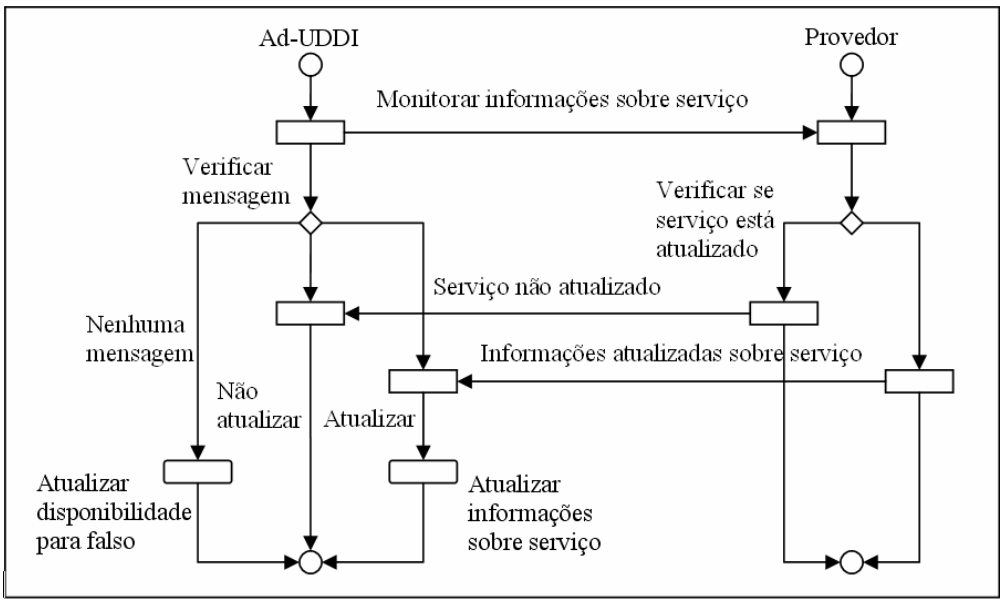

Figura 4.14: Repositório UDDI com um mecanismo de monitoramento.

Para o monitoramento, o servidor *Ad-UDDI* envia mensagens ao provedor periodicamente, contendo o nome, a chave e a versão registrados do serviço. O provedor verifica os itens das mensagens, comparando-os com os dados atuais do serviço, e retorna mensagens para indicar se existe a necessidade de atualizar as informações sobre o serviço contidas no repositório UDDI. Se nenhuma mensagem é retornada em um determinado período de tempo, o serviço é considerado indisponível e o servidor declara a indisponibilidade do serviço.

O servidor inclui a seguinte estratégia de monitoramento:

- As informações sobre um serviço são canceladas após dez minutos de monitoramento sem mensagens retornadas;
- Ao receber uma mensagem de retorno, o servidor atualiza as informações sobre o serviço e determina a disponibilidade do serviço;
- Ao receber uma requisição para a descoberta de serviços, o servidor *Ad-UDDI* busca somente entre serviços disponíveis.

A abordagem utilizada em *Ad-UDDI* é baseada em um servidor para o repositório UDDI com um mecanismo de monitoramento. Entretanto, políticas para atributos de qualidade não são utilizadas.

4.3.5 *Web Service Constraint Language*

Baresi et al [7] apresentam uma abordagem para o monitoramento de serviços. Uma linguagem compatível com o padrão *WS-Policy*, chamada *Web Service Constraint Language* (*WS-CoL*), é proposta para especificar restrições na execução de serviços. Embora a abordagem seja geral, o trabalho se concentra em segurança.

Um monitor é responsável pela verificação de mensagens SOAP. O monitoramento é executado para verificar se as mensagens estão de acordo com as declarações contidas em políticas para serviços *Web*.

As políticas *WS-Policy* são traduzidas em *WS-CoL*. Cada política é traduzida em duas expressões *WS-CoL*: uma para a mensagem de saída e outra para a mensagem de retorno. As expressões são enviadas ao monitor durante a fase de configuração inicial e são armazenadas no repositório de configurações.

O exemplo da Figura 4.15 mostra uma asserção de segurança *WS-Policy* e a expressão *WS-CoL* correspondente para a mensagem de saída.

```
<wsse:Confidentiality>
  <wsse:Algorithm type="wsse:AlgSignature"
  URI="http://www.w3.org/2000/09/xmlenc#3des-cbc"/>
</wsse:Confidentiality>
```

```
<wscol:Expression>
  \returnString(WSDL_XPATH, applyXPATH,
  '\\Envelope\body\EncryptedData\EncryptionMethod\@Algorithm',
  \returnString(WSDL_SOAP_DC, getSOAP,
  'BSPolicy', 'Data'))
  == 'http://www.w3.org/2000/09/xmlenc#3des-cbc';
</wscol:Expression>
```

Figura 4.15: Exemplo de tradução entre *WS-Policy* e *WS-CoL*.

Na expressão do exemplo (Figura 4.15), um coletor de dados SOAP é utilizado para produzir uma mensagem conforme a política declarada no documento *WS-Policy*. Um coletor de dados *XPath* extrai um valor contido no cabeçalho da mensagem. O valor é analisado para verificar se a mensagem foi gerada corretamente. Em seguida, a mensagem pode ser encaminhada ao serviço. A mensagem de retorno também é monitorada.

Ela é processada por um coletor de dados *XPath* e os valores extraídos são analisados. Se a mensagem está de acordo com a política, ela é encaminhada ao consumidor.

De modo geral, o modelo proposto segue a seguinte abordagem: se existe uma fonte de dados capaz de identificar o efeito de uma determinada asserção *WS-Policy* a ser monitorada, uma expressão *WS-CoL* pode ser derivada. Tal expressão especifica como a asserção pode ser monitorada. Ela instrui o monitor a declarar se as propriedades não-funcionais requeridas são satisfeitas. A abordagem não se integra no modelo de serviços *Web* atual. Ela não considera a publicação de políticas e a utilização de políticas na descoberta de serviços.

4.4 Resumo de Trabalhos Relacionados

As tabelas apresentadas a seguir resumem os trabalhos descritos nas seções anteriores: Tabela 4.1 (Seção 4.1), Tabela 4.2 (Seção 4.2) e Tabela 4.3 (Seção 4.3). Nas tabelas, o modo como os trabalhos tratam as questões listadas no início deste capítulo é brevemente descrito.

Trabalho	Especificação	Publicação	Consulta	Descoberta	Verificação
WSOL	Documentos WSOL complementam um documento WSDL e definem ofertas para um serviço, com diferentes propriedades não-funcionais.	-	-	-	Um documento WSOL pode especificar as entidades responsáveis por monitorar as propriedades não-funcionais.
WSLA	Um documento WSLA complementa um documento WSDL com acordos sobre propriedades não-funcionais.	-	-	-	Parâmetros são verifica-dos obtendo medidas diretamente de serviços, testando serviços ou interceptan-do invocações.

Glue QoS	Uma linguagem para políticas é utilizada para especificar atributos de qualidade. Ela é uma extensão de uma versão precedente de *WS-Policy*.	-	A linguagem para políticas também é utilizada para especificar os requisitos de qualidade.	Um *middleware* determina a compatibilidade entre políticas para verificar se componentes podem interagir.	-
WSEL	Um documento WSEL complementa um documento WSDL com informações não-funcionais.	O registro do serviço no UDDI inclui uma referência para o arquivo WSDL. Por sua vez, cada arquivo WSDL é ligado a um arquivo WSEL.	Um formato XML próprio inclui os critérios funcionais, em conformidade com o UDDI, e os não-funcionais.	Para os serviços descobertos utilizando o UDDI, a compatibilidade entre os atributos de qualidade é verificada.	-

Tabela 4.1: Resumo dos trabalhos relacionados citados que focam a especificação

Trabalho	Especificação	Publicação	Consulta	Descoberta	Verificação
UX	Um sumário é gerado a partir das informações não-funcionais contidas nos relatórios produzidos pelos consumidores.	O sistema armazena os relatórios sobre a qualidade dos serviços em um banco de dados. O repositório UDDI padrão é utilizado.	A requisição inclui os critérios funcionais e o identificador do perfil do consumidor.	Serviços com a funcionalidade requerida são obtidos utilizando o UDDI. Os serviços são ordenados de acordo com o perfil.	Consumidores geram relatórios. Um servidor testa os serviços. Os relatórios do servidor correspondem a uma pequena porção do total.

UDDIe	A linguagem WSDL é estendida para incluir atributos de qualidade.	Uma estrutura é incluída no modelo informacional do UDDI, ligada ao *businessService*.	A chamada padrão da API do UDDI é estendida para incluir critérios não-funcionais.	Um mediador seleciona os serviços no UDDI.	-
AUSE	-	Os serviços fornecem as informações e devem ser invocados para obtê-las.	Uma linguagem baseada em XML, chamada UDDI *Search Markup Language*, é utilizada.	Serviços são descobertos utilizando o UDDI e selecionados manualmente considerando atributos não-funcionais.	-
WSAF	Uma linguagem para políticas é utilizada para especificar atributos de qualidade.	Informações não-funcionais são registradas em agências.	Requisitos de qualidade são especificados utilizando políticas.	Agentes selecionam os serviços descobertos no UDDI utilizando informações obtidas nas agências.	Agentes monitoram os serviços e armazenam os resultados nas agências.
Lee et al.	-	Um servidor gera um histórico das informações não-funcionais e sobre as condições de busca. O repositório UDDI padrão é utilizado.	Uma requisição XML incluindo critérios funcionais e não-funcionais e as condições do consumidor é utilizada.	Um mediador separa os requisitos funcionais dos não-funcionais e os envia ao UDDI e ao servidor, respectivamente.	Agentes com condições diferentes testam os serviços regularmente. Informações coletadas são mantidas no servidor.

Kokash	-	Um repositório armazena informações funcionais. Agentes e mediadores mantêm informações não-funcionais.	O consumidor submete a requisição ao seu agente, que a encaminha ao agente que gerencia o repositório.	O repositório retorna serviços e agentes que os procuraram. O agente obtém recomenda-ções de outros agentes.	Mediadores armazenam estatísticas sobre a invocação de serviços, que podem ser compara-das com as informações fornecidas pelos agentes.

Tabela 4.2: Resumo dos trabalhos relacionados citados que focam a publicação e descober-ta

Trabalho	Especificação	Publicação	Consulta	Descoberta	Verificação
Ran	-	Uma estrutura é incluída no UDDI, ligada ao *business-Service*. *tModels* definem informações não-funcionais.	A requisição *find_service* do UDDI é estendida para incluir critérios funcionais e não-funcionais.	Se existirem múltiplos tipos de serviços com atributos funcionais similares, requisitos de qualidade refinam a busca.	Antes da publicação do serviço, um certificador verifica os atributos de qualidade declarados pelo provedor.
Singhera	A linguagem WSDL é estendida para incluir atributos de qualidade.	UDDI inclui atributos funcionais e não-funcionais estáticos. Monitores mantêm atributos não-funcionais dinâmicos.	A chamada padrão da API do UDDI é estendida para incluir critérios não-funcionais.	Serviços são descobertos no UDDI. A busca é refinada utilizando as propriedades dinâmicas armazenadas nos repositórios dos monitores.	Monitores verificam parâmetros não-funcionais de serviços em intervalos pré-definidos.

Serhani et al.	Utiliza a extensão *UDDIe*.	Utiliza a extensão *UDDIe*.	Utiliza a extensão *UDDIe*.	Utiliza a extensão *UDDIe*.	Um mediador executa testes para certificar o serviço antes de sua publicação. Ele pode também monitorar a execução.
Ad-UDDI	-	Repositórios adicionais são utilizados para incluir novas informações.	Chamadas padrões da API do UDDI são utilizadas.	Busca apenas entre os serviços previamente declarados como disponíveis pelo monitor.	O monitor emite mensagens de verificação periódicas.
WS-CoL	Asserções *WS-Policy* são traduzidas em *WS-CoL*. Expressões *WS-CoL* especificam restrições na execução dos serviços.	-	-	-	Uma mensagem é interceptada pelo monitor e processada de acordo com uma expressão *WS-CoL*.

Tabela 4.3: Resumo dos trabalhos relacionados citados que focam a verificação

Capítulo 5

Um Modelo de Serviços *Web* para Tratar Qualidade de Serviço – Extensões para Serviços *Web*

Este capítulo discute algumas extensões para os padrões UDDI, na Seção 5.1, e *WS-Policy*, na Seção 5.2, para possibilitar a gerência de atributos de qualidade de serviço. Além disso, o modelo de serviços *Web* que inclui as extensões propostas [30] é descrito na Seção 5.3.

5.1 UDDI Estendido

A abordagem proposta inclui a utilização de uma extensão para o padrão UDDI para publicar atributos de qualidade de serviço. Na abordagem, os atributos de qualidade são especificados utilizando *WS-Policy*. As informações não-funcionais sobre serviços *Web* registradas em repositórios UDDI estendidos são utilizadas para refinar a descoberta de serviços.

Muitas organizações, incluindo algumas organizações líderes da indústria de *software*, promovem o padrão UDDI. Assim, o número de repositórios UDDI disponíveis na *Web* está crescendo. Fan e Kambhampati [28] apresentam uma lista de repositórios públicos.

Entretanto, o padrão UDDI atual possui alguns problemas. Por exemplo, ele ainda não oferece apoio para a publicação de atributos de qualidade de serviço.

O padrão *WS-Policy* oferece um modelo que pode ser utilizado para especificar atributos de qualidade de serviço para serviços *Web*. Assim, provedores podem especificar seus serviços utilizando os padrões WSDL e *WS-Policy*, e publicar os serviços *Web* providos. Conseqüentemente, o padrão UDDI deve ser estendido para incluir políticas para atributos de qualidade e a descrição de serviços *Web* incrementada pode ser utilizada na descoberta de serviços.

Os aprimoramentos dos mecanismos de publicação e descoberta do padrão UDDI aumentam a flexibilidade de busca. Buscas por serviços *Web* podem ser baseadas em características funcionais de serviços *Web*, com atributos de qualidade requeridos como restrições.

A seguir são descritos o modelo de informação, a API (*Application Programming Interface*) e a utilização de *tModel* no UDDI estendido.

5.1.1 Modelo Informacional

O modelo informacional do padrão UDDI [17] é composto de estruturas de dados representadas em XML. O modelo informacional do UDDI estendido agrega a estrutura de dados *qosPolicy* e seus relacionamentos. A Figura 5.1 ilustra o modelo informacional estendido.

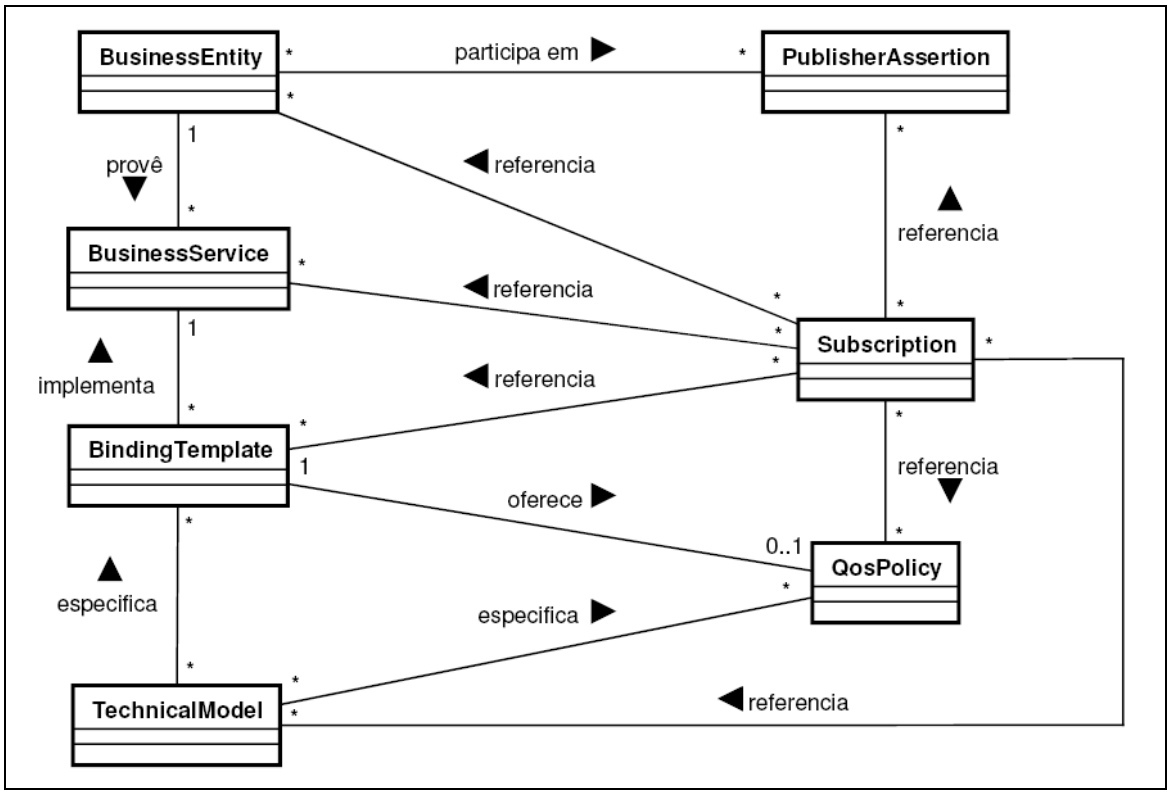

Figura 5.1: Modelo informacional.

A seguir são descritos os tipos de estruturas de dados que compõem o modelo informacional do padrão UDDI, juntamente com o tipo de estrutura incluído para permitir a publicação de políticas para características não-funcionais de serviços *Web*:

- *businessEntity*: tipo de estrutura de dados raiz que contém informações sobre um provedor de serviços, tais como, nome, contato e classificação. Cada *businessEntity* pode prover vários *businessServices*;
- *businessService*: representa um serviço *Web* que pode possuir múltiplas implementações. O tipo *businessService* inclui informações descritivas sobre um serviço *Web*, tais como, nome e categorização;
- *bindingTemplate*: representa uma implementação de serviço e inclui informações necessárias para acessar a implementação. Cada *bindingTemplate* contém informações tais como, ponto de acesso e protocolo de comunicações;
- *tModel*: representa conceitos únicos em UDDI, tais como, protocolos e sistemas de categorias. Exemplos incluem os *tModels* baseados em WSDL e outros documentos que esquematizam e especificam as interfaces de serviços;
- *publisherAssertion*: define uma associação de *businessEntities*. Esse tipo de estrutura é utilizado por organizações associadas para expor seus relacionamentos, por exemplo, companhias subsidiárias e consórcios;
- *subscription*: descreve uma requisição para acompanhar atividades em um repositório UDDI de acordo com as preferências especificadas na requisição. Assinantes podem se registrar para receber informações sobre alterações em qualquer um dos tipos de estruturas de dados do UDDI;
- *qosPolicy*: representa uma política para atributos de qualidade de uma implementação de serviço *Web*. As informações providas descrevem atributos de um serviço *Web* representado pela estrutura *bindingTemplate* associada. Referências para *tModels* podem ser utilizadas para indicar que uma política está em conformidade com especificações particulares.

5.1.2 API

O padrão UDDI define uma API que padroniza a comunicação entre as implementações do padrão. As chamadas de API são agrupadas em alguns conjuntos. A extensão para o padrão UDDI inclui a utilização de três conjuntos de chamadas de API para manipular informações não-funcionais sobre serviços *Web*, incluindo: *Inquiry*, *Publication* e *Subscription*. Além disso, o conjunto *Security* é utilizado durante a execução de algumas operações nos outros conjuntos. A Tabela 5.1 descreve as chamadas de API compreendidas pela extensão para o padrão UDDI.

Chamada de API	**Descrição**
Chamadas de API para *bindingTemplate*	
save_binding	Publica implementações de serviço *Web*, incluindo as políticas para atributos de qualidade associadas.
find_binding	Descobre implementações de serviço *Web* utilizando políticas para atributos de qualidade.
get_bindingDetail	Seleciona implementações de serviço *Web* utilizando chaves.
update_qos	Atualiza políticas para atributos de qualidade.
delete_binding	Exclui implementações de serviço *Web*.
Chamadas de API para *tModel*	
save_tModel	Registra *tModels*.
find_tModel	Encontra informação resumida sobre *tModels*.
get_tModelDetail	Obtém informação completa sobre *tModels*.
delete_tModel	Esconde *tModels* da chamada *find_tModel*.
Chamadas de API para *subscription*	
save_subscription	Registra subscrições.
get_subscriptions	Recupera subscrições de usuários.
get_subscriptionResults	Recupera notificações de determinados períodos relacionadas a subscrições.
delete_subscription	Cancela subscrições.
Chamadas de API para segurança	
get_authToken	Recupera credenciais de autenticação utilizando identificadores de usuário.
discard_authToken	Incapacita credenciais de autenticação.

Tabela 5.1: Chamadas de API

5.1.3 *tModel*

O tipo de estrutura *qosPolicy* é baseado no tipo *tModel* [17]. Os *tModels* para atributos de qualidade possibilitam o compartilhamento de especificações. Por exemplo, consórcios podem criar *tModels* que especificam atributos de qualidade que serviços de domínios específicos podem possuir.

Uma estrutura *qosPolicy* contém referências para estruturas *tModel* para indicar definições de atributos [45] [55] [42] [48].

O tipo *tModel* permite que atributos de qualidade possam ser:

- Definidos para domínios específicos. Exemplo: um atributo taxa de administração pode ser definido especificamente para o domínio de fundos de investimento;
- Redefinidos. Exemplo: o atributo confiabilidade pode considerar a taxa de erro do serviço em um determinado intervalo de tempo;
- Compostos. Exemplo: um atributo razão custo/desempenho pode ser definido por meio da composição dos atributos custo e tempo de resposta;
- Especializados. Exemplo: um atributo tempo de resposta prêmio pode ser definido como uma especialização do atributo tempo de resposta padrão.

As definições de atributos de qualidade podem incluir diversas propriedades. A seguir, as propriedades são discutidas:

- Atributos de qualidade podem ser medidos. Métricas devem ser definidas para atributos;
- Diferentes operações podem ser requeridas para comparar serviços *Web* em termos de atributos de qualidade particulares. Operações de comparação devem ser definidas;
- Diferentes unidades de medida podem ser utilizadas. Unidades de medida podem ser definidas;
- Atributos de qualidade podem estar interconectados. Relacionamentos entre atributos podem ser especificados;
- Condições ambientais podem influenciar atributos. Essas dependências podem ser especificadas.

A Figura 5.2 mostra uma estrutura *tModel*. O exemplo ilustra um *tModel* utilizado para especificar um atributo de qualidade (Linhas 13-15). O atributo especificado é o tempo de resposta (Linhas 02-11). O elemento *overviewURL* (Linhas 08-10) aponta para um arquivo que mantém a definição do atributo. As Linhas 12-22 representam as propriedades do atributo. O *tModel* com a chave indicada na Linha 18 especifica a métrica. A unidade de medida é o milissegundo. Ela é especificada utilizando um *tModel* separado (Linhas 19-21).

```
01  <tModel
02    tModelKey="uddi:uddi.org:qos:attribute:responsetime">
03    <name>uddi-org:qos:attribute:responseTime</name>
04    <description>
05      Response time attribute specification
06    </description>
07    <overviewDoc>
08      <overviewURL …>
09        garnize:8080/tmodel/qos/attribute/responsetime.xml
10      </overviewURL>
11    </overviewDoc>
12    <categoryBag>
13      <keyedReference keyName="QoS attribute specification"
14        keyValue="qosAttributeSpec"
15        tModelKey="uddi:uddi.org:categorization:types"/>
16      <keyedReference keyName="Response time metrics"
17        keyValue="responseTimeMetrics"
18        tModelKey="uddi:uddi.org:qos:metrics:responsetime"/>
19      <keyedReference keyName="Response time unit"
20        keyValue="millisecond"
21        tModelKey="uddi:uddi.org:qos:unit:millisecond"/> …
22    </categoryBag>
23  </tModel>
```

Figura 5.2: Exemplo de *tModel* para atributos de qualidade.

5.2 *WS-Policy* para a Especificação de Características Não-Funcionais

Na abordagem proposta, o padrão *WS-Policy* é utilizado para especificar características não-funcionais de serviços *Web*.

Para utilizar as funcionalidades adicionadas ao UDDI, provedores devem especificar os atributos de qualidade de seus serviços utilizando *WS-Policy*. Tais especificações complementam interfaces de serviços *Web* descritas em WSDL. Assim, consumidores podem selecionar serviços considerando suas demandas de atributos de qualidade, também especificadas em *WS-Policy*.

WS-Policy permite a especificação de políticas para serviços *Web*. Políticas podem ser utilizadas durante as diferentes fases do ciclo de vida de serviços *Web*, conforme descrito a seguir:

- Durante as fases de projeto e instalação, provedores podem criar políticas para serviços *Web*, que especificam as propriedades que os serviços deveriam apresentar. Essas políticas podem combinar políticas independentes de servidor e políticas que definem as características que os serviços instalados em um servidor de aplicações particular deveriam apresentar;
- Durante a fase de execução, consumidores podem criar políticas associadas com os serviços parceiros. Essas políticas especificam as propriedades que deveriam ser oferecidas pelos serviços *Web* requeridos.

O conjunto de construtores de propósito geral provido por *WS-Policy* pode ser estendido para especificar um amplo conjunto de propriedades de serviços *Web*. No caso de políticas para atributos de qualidade, asserções de políticas de consumidores especificam requisitos não-funcionais e asserções de políticas de provedores especificam capacidades não-funcionais de serviços *Web*.

Uma política para características não-funcionais pode incluir:

- O elemento raiz *Policy*. Esse elemento do padrão *WS-Policy* indica uma expressão de política. Uma expressão de política é uma representação XML de uma política para serviço *Web*;
- Dois tipos de identificação para políticas podem ser utilizados, conforme definido por *WS-Policy*. A política pode ser associada com um URI absoluto, utilizando o atributo *Name*, ou a política pode ser associada com uma referência em um documento XML, utilizando o atributo *Id*;
- Cada política contém um atributo *Type*, que é utilizado para indicar se ela é uma política de provedor ou de consumidor;

- O elemento *PolicyReference* do padrão *WS-Policy* permite o reuso de asserções entre políticas. Ele pode ser utilizado para incluir o conteúdo de uma política em uma outra política;
- Uma política de provedor inclui um elemento *Service* para descrever detalhes da implementação de serviço para a qual a política foi especificada. Já uma política de consumidor inclui um elemento *Operation* para descrever o tipo de operação de serviço ao qual a política se aplica;
- Os elementos *ExactlyOne* e *All* são operadores de política do padrão *WS-Policy*. O operador *ExactlyOne* é uma coleção de alternativas de política. O operador *All* agrupa asserções de política em alternativas. Um operador de política pode incluir outro operador.

Para cada atributo de qualidade especificado, uma asserção de política pode incluir:

- Um nome descritivo;
- Uma operação de serviço;
- Uma chave de *tModel*;
- Um valor oferecido (em políticas de provedores) ou requerido (em políticas de consumidores);
- O atributo *Optional*, definido por *WS-Policy*, pode ser utilizado para indicar que uma asserção é opcional;
- Além disso, uma asserção pode conter uma expressão de política.

Asserções para atributos de qualidade contidas em políticas devem ser extraídas. Então, os atributos podem ser utilizados em requisições de descoberta de serviços ou podem ser publicados em repositórios UDDI.

O processamento de políticas é baseado em operações definidas pelo padrão *WS-Policy*. Tais operações permitem a normalização e a decomposição de políticas. A operação de normalização é o processo de converter políticas para um formato padronizado. Ela simplifica a manipulação de políticas. A operação de decomposição é o processo de decompor políticas em alternativas e asserções. Para decompor políticas, inicialmente elas devem ser normalizadas.

O exemplo na Figura 5.3 apresenta uma especificação de política em *WS-Policy*. O exemplo ilustra uma política para atributos de qualidade utilizando uma asserção que define o tempo de resposta oferecido por um serviço *Web* (Linhas 06-10).

```
01  <wsp:Policy …
02    xmlns:wsp="schemas.xmlsoap.org/ws/2004/09/policy"
03    xmlns:qosp="garnize:8080/schema/qospolicy"
04      <wsp:ExactlyOne>
05        <wsp:All>
06          <qosp:ResponseTime
07            xmlns:qosp="garnize:8080/schema/qospolicy"
08            operation="get"
09            specification="uddi:uddi.org:qos:attribute:
10              responsetime">45</qosp:ResponseTime> …
11        </wsp:All>
12      </wsp:ExactlyOne>
13  </wsp:Policy>
```

Figura 5.3: Exemplo de política para atributos de qualidade.

5.3 Modelo Estendido para Qualidade de Serviço

O modelo de serviços *Web* apresentado nesta seção utiliza as extensões propostas para os padrões *WS-Policy* e UDDI. Além dos componentes básicos do modelo de serviços *Web*, que inclui o cliente de serviço, o serviço *Web* e o repositório de serviços, o modelo estendido inclui os componentes mediador e monitor. O modelo inclui também um provedor intermediário de serviços ao conjunto de papéis do modelo de serviços *Web* básico, que compreende o provedor de serviços, o consumidor de serviços e o provedor de repositórios de serviços.

Os principais componentes do modelo de serviços *Web* estendido são:

- Repositórios UDDI estendidos: um repositório UDDI estendido armazena informações sobre serviços *Web*, incluindo informações descritivas, técnicas, de categorização e sobre características não-funcionais;
- Mediadores: um mediador auxilia na descoberta de serviços *Web* utilizando políticas para atributos de qualidade;
- Monitores: um monitor auxilia na atualização de repositórios com informações sobre parâmetros de qualidade adquiridas durante as execuções dos serviços.

Uma visão geral do modelo é apresentada na Figura 5.4. As responsabilidades do mediador e do monitor são discutidas nas próximas seções.

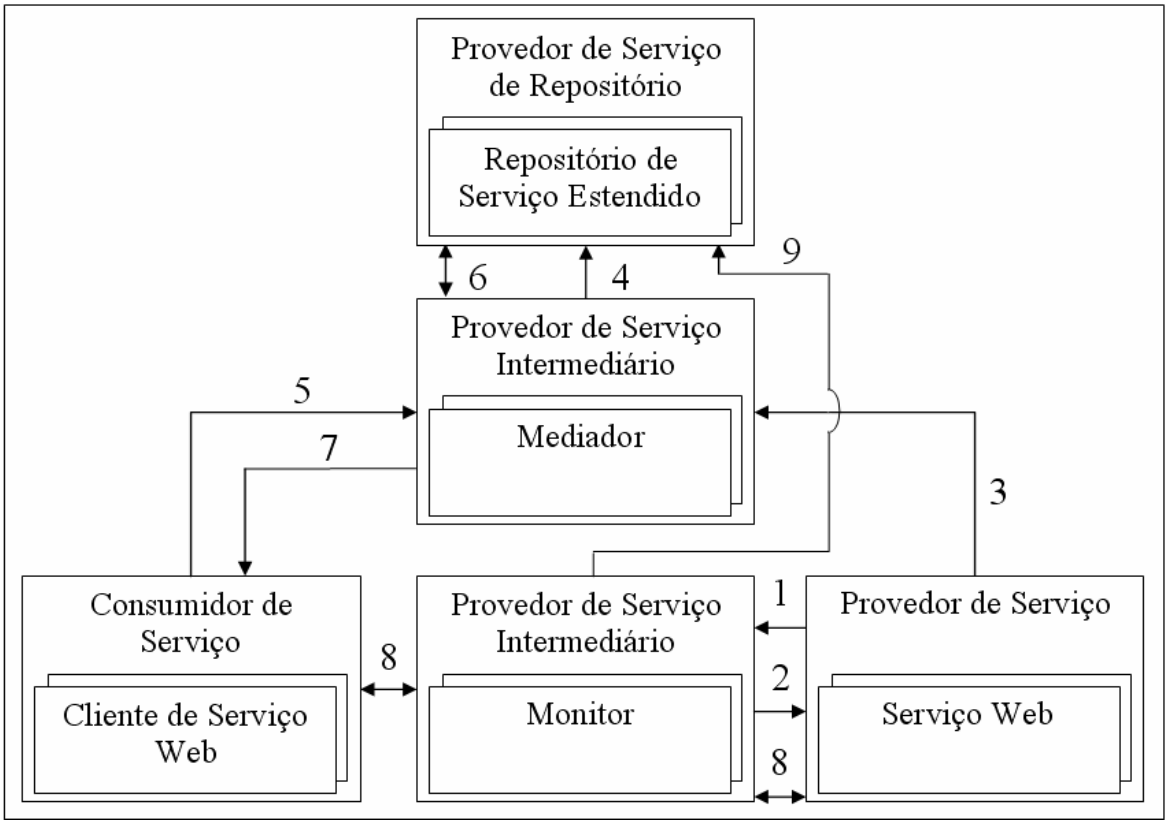

Figura 5.4: Modelo de serviços *Web* estendido.

5.3.1 Mediadores

Serviços de mediação podem ser utilizados para facilitar o acesso a repositórios de serviços [39] [58].

A interação com repositórios UDDI no modelo proposto é realizada através de mediadores. Um mediador provê as operações para a gerência de atributos de qualidade. Ele utiliza a API do UDDI para recuperar dados contidos em repositórios UDDI e para registrar e atualizar dados em repositórios.

O mediador é um serviço *Web*. Isso permite o emprego do modelo em ambientes restritos e abertos, e a utilização de mediadores com características diferentes, de acordo com as necessidades específicas dos consumidores.

A utilização de mediadores gera algumas questões de segurança. Por exemplo, é importante verificar se o mediador empregado é confiável. Mediadores podem ser criados e gerenciados por provedores intermediários de serviços.

Conforme proposto em [16], Intermediários Confiáveis[5] são entidades apropriadas para ambientes abertos de serviços. Assim como as entidades da área de segurança digital chamadas Autoridades Certificadoras, que facilitam a autenticação de organizações, Intermediários Confiáveis oferecem diversos serviços intermediários seguros. Esses serviços intermediários seguros podem ser empregados para apoiar a negociação entre organizações [41]. O modelo estendido proposto neste trabalho inclui provedores intermediários de serviços, os quais são Intermediários Confiáveis. Essas entidades são responsáveis por prover serviços seguros de mediação e monitoramento.

Os mediadores obtêm informações sobre as características não-funcionais de serviços *Web* a partir das políticas dos provedores de serviços. Eles publicam tais informações em repositórios UDDI, juntamente com informações adicionais necessárias para a publicação apropriada e a subseqüente descoberta de serviços *Web*. Os mediadores também ajudam os consumidores na seleção de serviços de acordo com os seus requisitos funcionais e de qualidade.

Os mediadores processam as políticas para atributos de qualidade dos provedores de serviços para permitir o mapeamento das asserções das políticas nas estruturas *qosPolicy* dos repositórios UDDI. A estrutura *qosPolicy* utiliza uma lista de estruturas *keyedReference* [17] do padrão UDDI para apoiar a descrição de características não-funcionais de serviços *Web*.

A Figura 5.5 mostra uma estrutura *businessService*. O exemplo de fragmento de estrutura *businessService* compreende uma estrutura *bindingTemplate* (Linhas 05-15). A estrutura *bindingTemplate* representa o serviço *Web* para o qual a política na Figura 5.3 foi especificada. Essa estrutura compreende uma estrutura *qosPolicy* (Linhas 09-14). O exemplo de estrutura *qosPolicy* contém o atributo tempo de resposta. Esse atributo foi obtido após o processamento da política para atributos de qualidade.

[5] Do inglês *Trusted Intermediaries*.

```
01  <businessService …
02    serviceKey="EB3F1520-C044-11DA-9520-DCE91EDB01F6"
03    <name>Service</name>
04    <bindingTemplates>
05      <bindingTemplate …>
06        <accessPoint …>
07          garnize:8080/axis/Service.jws
08        </accessPoint>
09        <qosPolicy …>
10          <keyedReference
11            tModelKey="uddi:uddi.org:qos:attribute:responsetime"
12            keyName="ResponseTime"
13            keyValue="45"/> …
14        </qosPolicy>
15      </bindingTemplate>
16    </bindingTemplates> …
17  </businessService>
```

Figura 5.5: Exemplo de serviço *Web* publicado.

Os consumidores provêem as suas políticas aos mediadores. As políticas são utilizadas durante as descobertas de serviços.

Para implementar o mecanismo de descoberta de serviços baseado em parâmetros de qualidade, dois passos são executados:

- Os serviços *Web* satisfazendo os requisitos funcionais são descobertos utilizando o mecanismo tradicional de descoberta do UDDI;
- No segundo passo, os serviços descobertos no primeiro passo são selecionados verificando a compatibilidade entre as políticas para atributos de qualidade.

O segundo passo determina a compatibilidade entre políticas discutida em [4]. Compreende dois estágios:

- Primeiramente, o mecanismo de *tModel* é utilizado para determinar se as políticas possuem o mesmo vocabulário. O vocabulário de uma política é o conjunto de todos os tipos de asserções utilizados na política;
- Como a compatibilidade entre as asserções das políticas depende da semântica específica dos atributos de qualidade, o segundo passo envolve processamentos específicos para os atributos.

A Figura 5.6 mostra uma chamada de API pertencente ao conjunto de chamadas de API *Inquiry* do UDDI. Esse exemplo apresenta um fragmento de uma requisição *find_binding*. Ela é uma requisição secundária. Primeiramente, uma requisição *find_service* é emitida para buscar por serviços de acordo com requisitos funcionais. No exemplo, apenas o serviço com a chave indicada na Linha 05 foi descoberto. Então, uma requisição *find_binding* é utilizada para buscar por implementações dos serviços recuperados. Na requisição do exemplo, a busca é refinada utilizando o tempo de resposta requerido (Linhas 06-11), adquirido através da política para atributos de qualidade do consumidor de serviços.

```
01  <soap:Envelope
02    xmlns:soap="schemas.xmlsoap.org/soap/envelope/"...>
03    <soap:Body>
04      <uddi:find_binding xmlns:uddi="urn:uddi-org:api" ...
05        serviceKey="EB3F1520-C044-11DA-9520-DCE91EDB01F6">
06        <uddi:qosPolicy>
07          <uddi:keyedReference
08            keyName="ResponseTime"
09            keyValue="50"
10            tModelKey="uddi:uddi.org:qos:attribute:responsetime"/> ...
11        </uddi:qosPolicy>
12      </uddi:find_binding>
13    </soap:Body>
14  </soap:Envelope>
```

Figura 5.6: Exemplo de requisição para a descoberta de serviços *Web*.

A Figura 5.7 ilustra um exemplo de uma resposta para a requisição SOAP apresentada na Figura 5.6. O fragmento de resposta inclui uma estrutura *bindingDetail* do UDDI com o único *bindingTemplate* (Linhas 05-16) com o parâmetro de tempo de resposta requisitado (Linhas 10-15).

```
01  <soap:Envelope
02    xmlns:soap="schemas.xmlsoap.org/soap/envelope/" ...>
03    <soap:Body>
04      <bindingDetail ...>
05        <bindingTemplate ...
06          serviceKey="EB3F1520-C044-11DA-9520-DCE91EDB01F6">
07          <accessPoint ...>
08            garnize:8080/axis/Service.jws
09          </accessPoint>
10          <qosPolicy>
11            <keyedReference
12              keyName="ResponseTime"
13              keyValue="45"
14              tModelKey="uddi:uddi.org:qos:attribute:responsetime"/> …
15          </qosPolicy>
16        </bindingTemplate>
17      </bindingDetail>
18    </soap:Body>
19  </soap:Envelope>
```

Figura 5.7: Exemplo de resposta com um serviço *Web* descoberto.

As asserções de uma política de um provedor de serviços representam as propriedades de qualidade que devem ser providas pelo serviço ao consumidor. Esse conjunto de asserções é retornado ao consumidor de serviços juntamente com as informações sobre o serviço selecionado.

Após a descoberta, os serviços *Web* devem ser monitorados para verificar se as propriedades declaradas são oferecidas e os atributos de qualidade devem ser atualizados nos repositórios de serviços.

5.3.2 Monitores

Monitores podem ser implementados como serviços *Web* para monitorar os atributos de qualidade de outros serviços *Web* [48].

O monitoramento é realizado durante a execução dos serviços considerando as políticas para atributos de qualidade.

Os monitores atuam como uma camada de interceptação entre os consumidores e os provedores de serviços *Web*.

Existe um monitor para cada serviço *Web*. Os provedores podem registrar os seus serviços com os provedores intermediários de serviços. Nesse caso, essas entidades são responsáveis por criar e gerenciar os monitores para os serviços.

Um monitor é implementado como um Intermediário de Serviço *Web*[6] [16]. Um Intermediário de Serviço *Web* é um serviço *Web* localizado entre um cliente de serviço e um serviço. Ele intercepta as mensagens endereçadas a um serviço, realiza algum processamento adicional e encaminha as mensagens ao seu destinatário.

O mecanismo de Intermediário de Serviço *Web* oferece um modo flexível para prover funcionalidades adicionais, permitindo o monitoramento de atributos de qualidade para serviços *Web* sem modificações no código dos serviços.

Além disso, os Intermediários de Serviços *Web* podem manipular o conteúdo das mensagens SOAP, incluindo os cabeçalhos opcionais que são utilizados para definir as extensões SOAP [46].

Aspectos como segurança e transação são importantes sempre que Intermediários de Serviços *Web* estão envolvidos [16].

Durante a execução dos serviços *Web*, os monitores verificam se os atributos de qualidade oferecidos pelos serviços estão em conformidade com as asserções das políticas para os serviços.

Os monitores atualizam as informações sobre as características não-funcionais de serviços *Web* armazenadas nos repositórios UDDI quando variações nos atributos de qualidade são detectadas. Nesse caso, o mecanismo de autorização do UDDI deve ser utilizado.

---

[6] Do inglês *Web Service Intermediary*.

5.3.3 Interações entre os Componentes

Um cenário de uso típico é ilustrado na Figura 5.4. A seguir, os passos do cenário (mostrados na figura) são descritos considerando um exemplo no qual um consumidor utiliza um serviço de um provedor em sua aplicação:

- **Passo 1.** Inicialmente, o provedor registra o serviço *Web* com um provedor de serviço intermediário. O provedor fornece informações funcionais e não-funcionais sobre o serviço oferecido;
- **Passo 2.** O provedor de serviço intermediário cria um monitor para o serviço *Web*;
- **Passo 3.** O provedor solicita a publicação do serviço. Ele provê a política para atributos de qualidade do serviço juntamente com as informações requeridas para a publicação do serviço;
- **Passo 4.** O mediador publica o serviço *Web* provido no repositório UDDI estendido;
- **Passo 5.** A aplicação do consumidor requisita a descoberta de um serviço. A requisição inclui aspectos funcionais e não-funcionais;
- **Passo 6.** O mediador seleciona um serviço no repositório UDDI de acordo com a funcionalidade requerida e a política para atributos de qualidade da aplicação;
- **Passo 7.** O mediador envia as informações sobre o serviço selecionado para a aplicação;
- **Passo 8.** A aplicação pode então invocar a operação requerida no serviço selecionado através do ponto de acesso recebido. O monitor intercepta as mensagens que são trocadas entre a aplicação e o serviço *Web* para monitorar a execução do serviço;
- **Passo 9.** Finalmente, o monitor atualiza a política para atributos de qualidade do serviço no repositório UDDI de acordo com os resultados do monitoramento.

Capítulo 6

Implementação e Avaliação

Este capítulo apresenta alguns aspectos de uma implementação parcial do modelo descrito no Capítulo 5. O objetivo da implementação foi testar a eficácia e avaliar o desempenho da abordagem proposta.

6.1 Componentes Arquiteturais

A Figura 6.1 mostra o diagrama de pacotes do modelo de serviços *Web* proposto. A seguir, os pacotes são brevemente descritos e algumas classes que compõem os pacotes *repositórioUddi*, *mediador* e *monitor* são apresentadas.

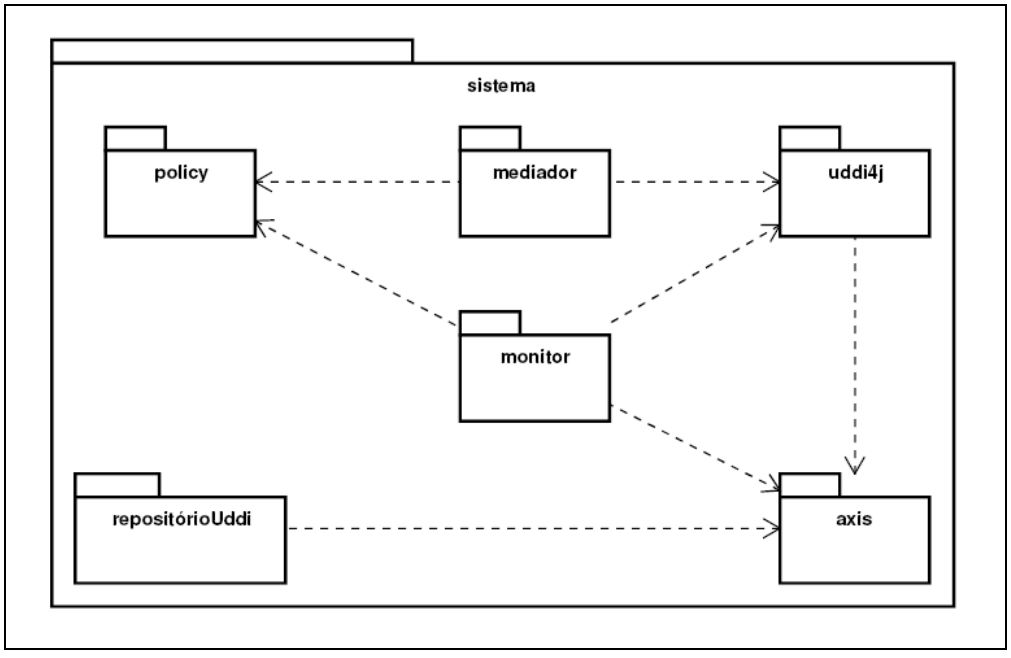

Figura 6.1: Diagrama de pacotes do modelo.

O modelo foi parcialmente implementado utilizando:

- *Sun Java Development Kit Version* 1.5.0_06;
- *Apache Axis Version* 1.3: provê apoio para a linguagem WSDL e uma máquina SOAP;
- *Apache Tomcat Version* 4.1.24: um servidor de aplicações.

Mediadores foram implementados utilizando o padrão *WS-Policy* para manipular políticas para atributos de qualidade. *Apache WS-Commons/Policy Version* 0.90 foi utilizado. Esse *software* oferece uma implementação de *WS-Policy* compatível com a versão 1.2 da especificação do padrão. A implementação provê um mecanismo para processar informações em políticas em tempo de execução.

Um monitor foi implementado com apoio para o monitoramento do atributo tempo de resposta. Para a inclusão de apoio para atributos de qualidade adicionais, as técnicas de medição utilizadas pelos monitores devem ser implementadas de acordo com as métricas associadas com os atributos a serem monitorados.

Assim como o atributo tempo de resposta, os atributos dinâmicos podem ser monitorados utilizando mecanismos de *logging* [16]. Atributos estáticos, como, por exemplo, interoperabilidade, podem ser certificados por meio da análise de mensagens SOAP [10]. Esses atributos são definidos por algumas especificações do OASIS que descrevem extensões para o padrão SOAP, incluindo as seguintes especificações:

- *Web Services Security* [50]: para segurança;
- *Web Services Reliable Messaging* [24]: para transmissão de mensagem segura;
- *Web Services Coordination* [26], *Web Services Atomic Transaction* [43] e *Web Services Business Activity* [27]: para integridade.

O atributo interoperabilidade é definido pelo perfil de interoperabilidade *Basic Profile* [6] da *Web Services Interoperability Organization* (WS-I), uma organização que determina regras para a interoperabilidade em serviços *Web*.

A implementação do componente UDDI envolveu a utilização dos seguintes produtos de *software*:

- *Apache jUDDI Version* 0.9rc4;
- *MySQL AB MySQL Version* 5.0.16;
- *HP/IBM/SAP UDDI4J Version* 2.0.4.

O repositório UDDI foi implementado utilizando o *software Apache jUDDI Version* 0.9rc4, o qual oferece uma implementação do padrão UDDI compatível com a versão 2.0 da especificação do padrão.

MySQL AB MySQL Version 5.0.16 foi utilizado para implementar o banco de dados do UDDI. No banco de dados, a tabela *QOS_POLICY* contém os atributos de qualidade. Essa

tabela liga-se à tabela *BINDING_TEMPLATE*, a qual armazena as instâncias de serviços *Web*. Ela também liga-se à tabela *TMODEL*, a qual armazena os *tModels* para os conceitos relacionados às características não-funcionais.

O repositório UDDI estendido é compatível com o básico e ambos podem coexistir no mesmo ambiente.

Além de utilizar a API do UDDI diretamente para interagir com um repositório UDDI, APIs clientes baseadas em linguagens de programação particulares podem ser utilizadas. Uma API cliente estendida baseada em *Java* foi implementada. A API inclui algumas classes que representam os elementos do UDDI. *UDDI4J Version* 2.0.4, uma biblioteca de classes *Java* apoiada pelas empresas HP, IBM e SAP, foi utilizada para implementar a API cliente. A API possui alguns construtores que geram e analisam as mensagens enviadas para e recebidas de um repositório UDDI.

Os produtos de *software* utilizados para a implementação são produtos de fonte aberta (*open-source*).

6.2 Diagramas de Classes

Para facilitar a visualização, o diagrama de classes do repositório UDDI foi dividido em três figuras. As Figuras 6.2, 6.3 e 6.4 incluem as principais classes envolvidas na publicação de serviços, descoberta de serviços e atualização de políticas, respectivamente.

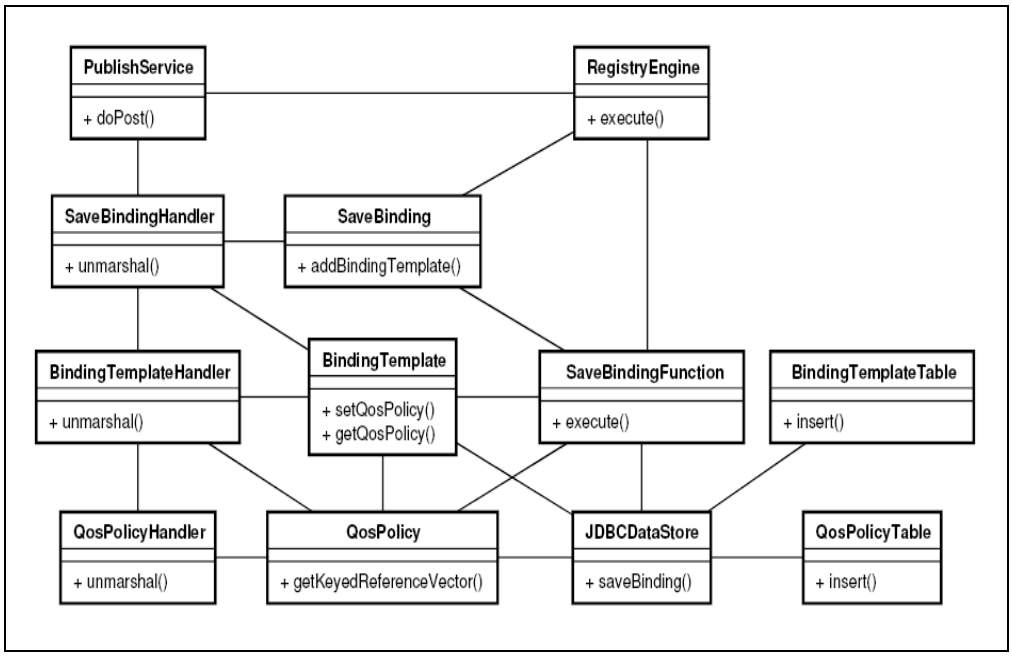

Figura 6.2: Diagrama de classes do repositório UDDI - classes envolvidas na publicação de serviços.

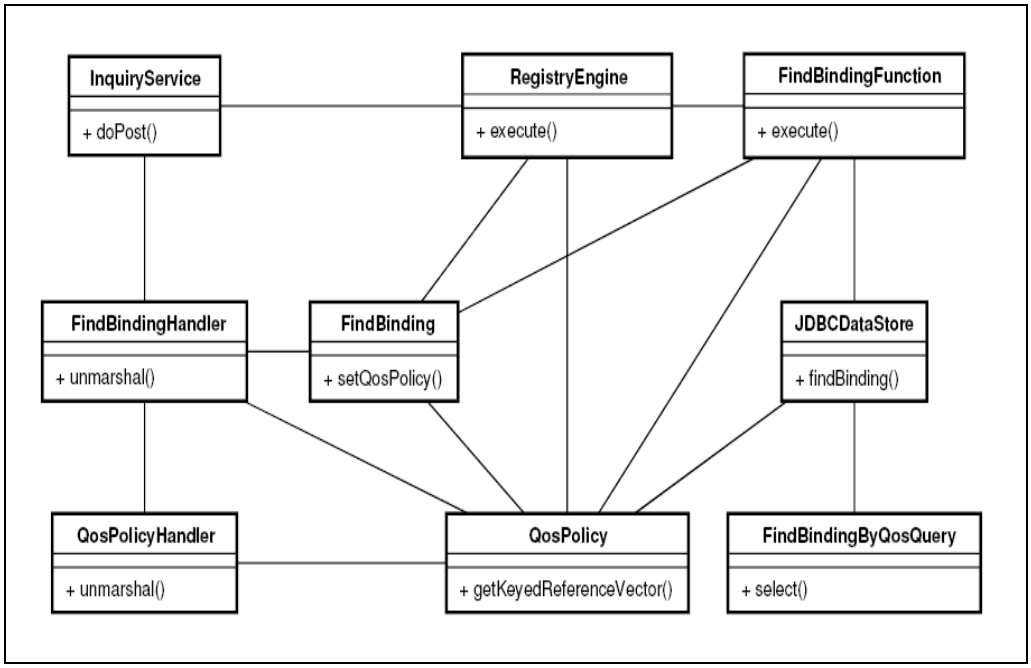

Figura 6.3: Diagrama de classes do repositório UDDI - classes envolvidas na descoberta de serviços.

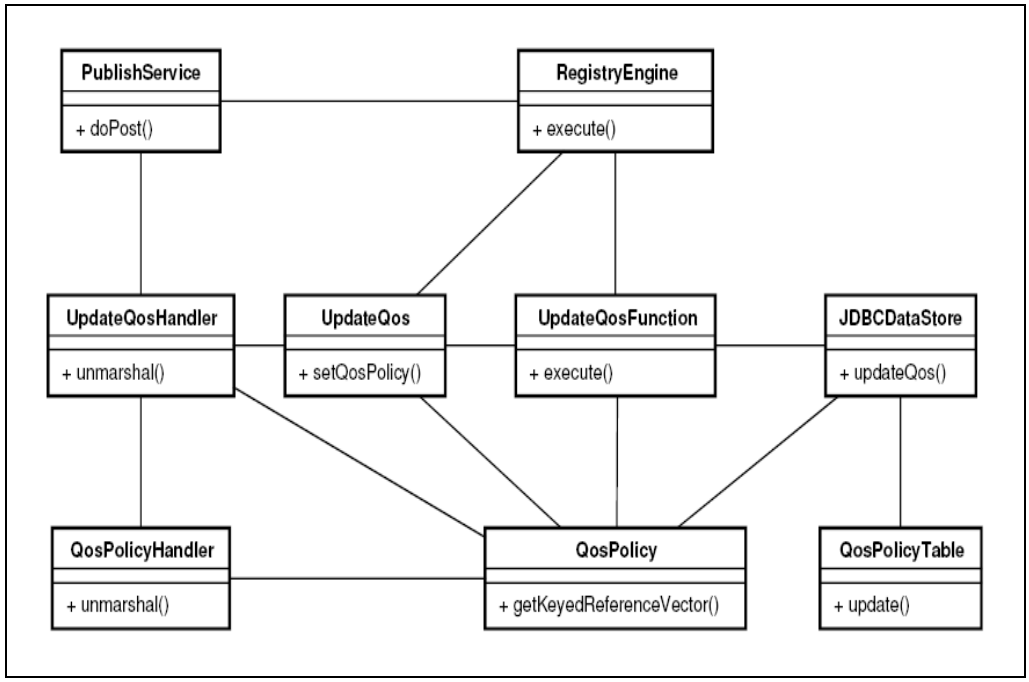

Figura 6.4: Diagrama de classes do repositório UDDI - classes envolvidas na atualização de políticas.

O diagrama inclui as seguintes classes:

- As classes com o sufixo *Service* são pontos de acesso ao repositório. A classe *InquiryService* recebe as requisições que envolvem consulta ao conteúdo do repositório. Já a classe *PublishService* recebe as requisições que envolvem alteração do conteúdo;
- As classes nomeadas com o nome dos tipos de estruturas de dados e das chamadas de API representam os elementos XML do UDDI. A interação com o repositório é realizada utilizando XML. Porém, internamente, o repositório manipula objetos *Java*;
- As classes com o sufixo *Handler* realizam o mapeamento entre as requisições XML do UDDI e os objetos *Java* manipulados pelo repositório;
- A classe *RegistryEngine* representa um repositório UDDI. Ela utiliza as classes com o sufixo *Function*;
- As classes *Function* implementam as ações que podem ser realizadas no repositório UDDI. Elas utilizam a classe *JDBCDataStore*;
- A classe *JDBCDataStore* é uma conexão *Java Database Connectivity* (JDBC) para o banco de dados do UDDI. Ela utiliza as classes com os sufixos *Query* e *Table*;
- As classes *Query* e *Table* realizam a interface entre o sistema e o banco de dados.

A seguir são apresentados os diagramas de classes dos componentes arquiteturais desenvolvidos para a avaliação do modelo. As Figuras 6.5, 6.6 e 6.7 mostram, respectivamente, os diagramas de um mediador para a publicação de serviços, um mediador para a descoberta de serviços e um monitor para a atualização de políticas.

A classe *PBroker* (Figura 6.5) possui um método para a publicação de serviços. O método recebe como parâmetros: as chaves de *tModel* e *businessService*, o endereço, o protocolo e a política do serviço *Web* a ser publicado. Ele retorna a chave de *bindingTemplate* do serviço publicado.

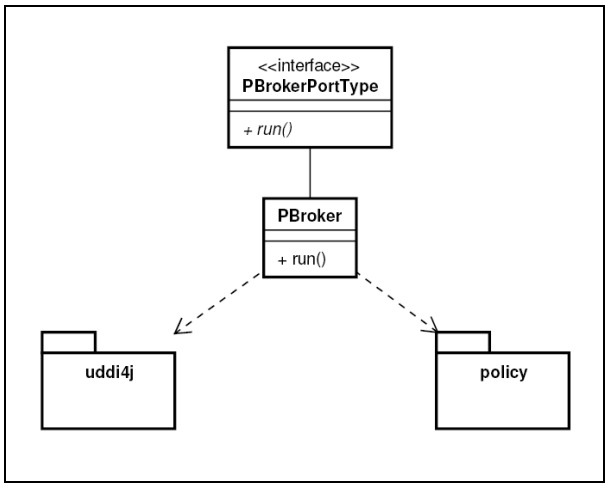

Figura 6.5: Diagrama de classes de um mediador para a publicação de serviços.

A classe *DBroker* (Figura 6.6) possui um método para a descoberta de serviços. O método *run* recebe a chave de *tModel* e a política do consumidor como parâmetros. Ele retorna o ponto de acesso do serviço *Web* selecionado.

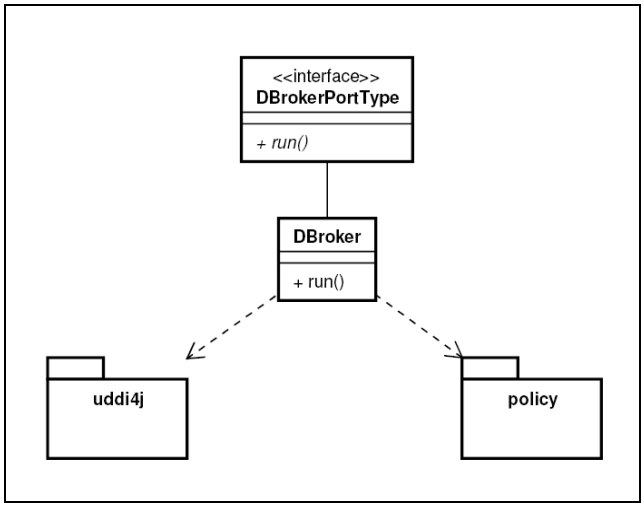

Figura 6.6: Diagrama de classes de um mediador para a descoberta de serviços.

A classe *Monitor* (Figura 6.7) possui um método para a atualização do tempo de resposta contido em uma política publicada. O método recebe uma mensagem. A classe *Monitor* identifica uma mensagem de requisição e a mensagem de resposta associada. Ela armazena o horário de recebimento das mensagens em um arquivo de *log*.

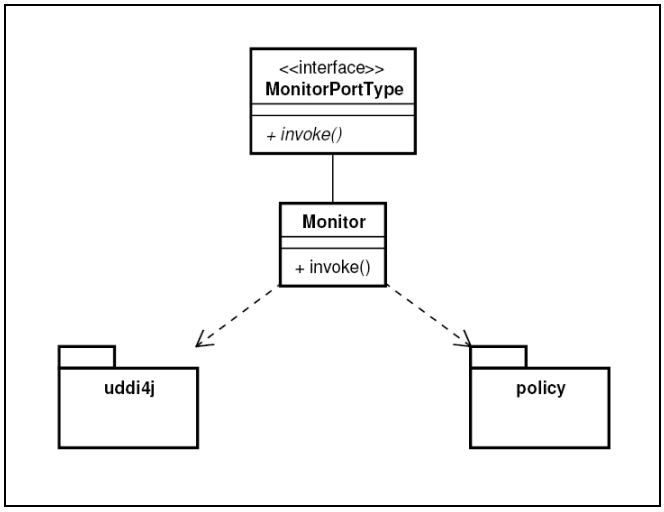

Figura 6.7: Diagrama de classes de um monitor para a atualização de políticas.

6.3 Diagramas de Seqüência

As Figuras 6.8, 6.9 e 6.10 mostram os diagramas de seqüência para a publicação de serviços, descoberta de serviços e atualização de políticas, respectivamente.

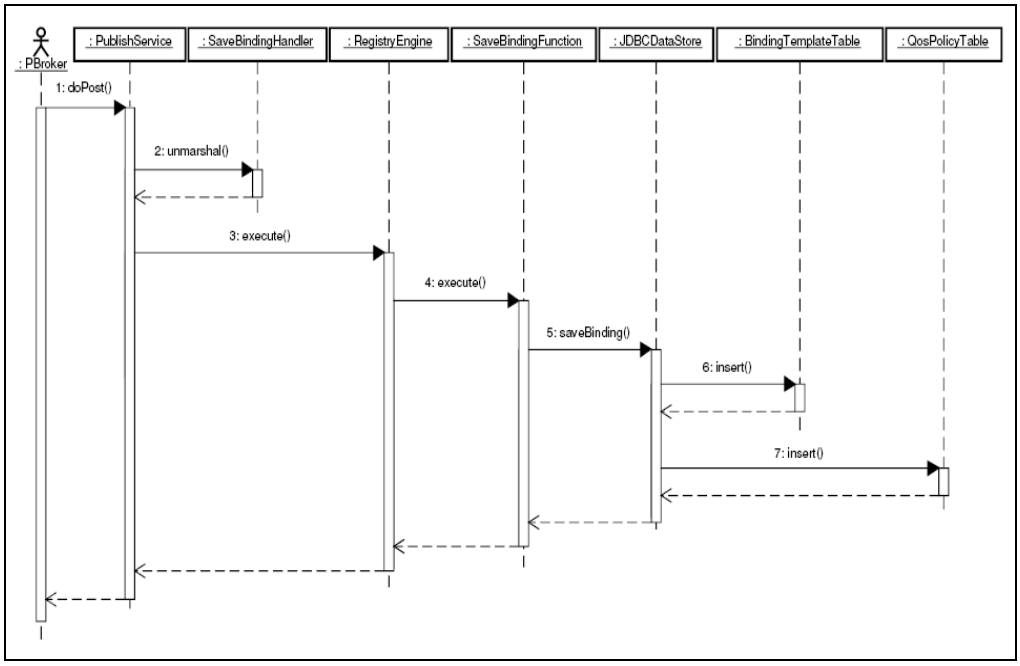

Figura 6.8: Diagrama de seqüência para a publicação de serviços.

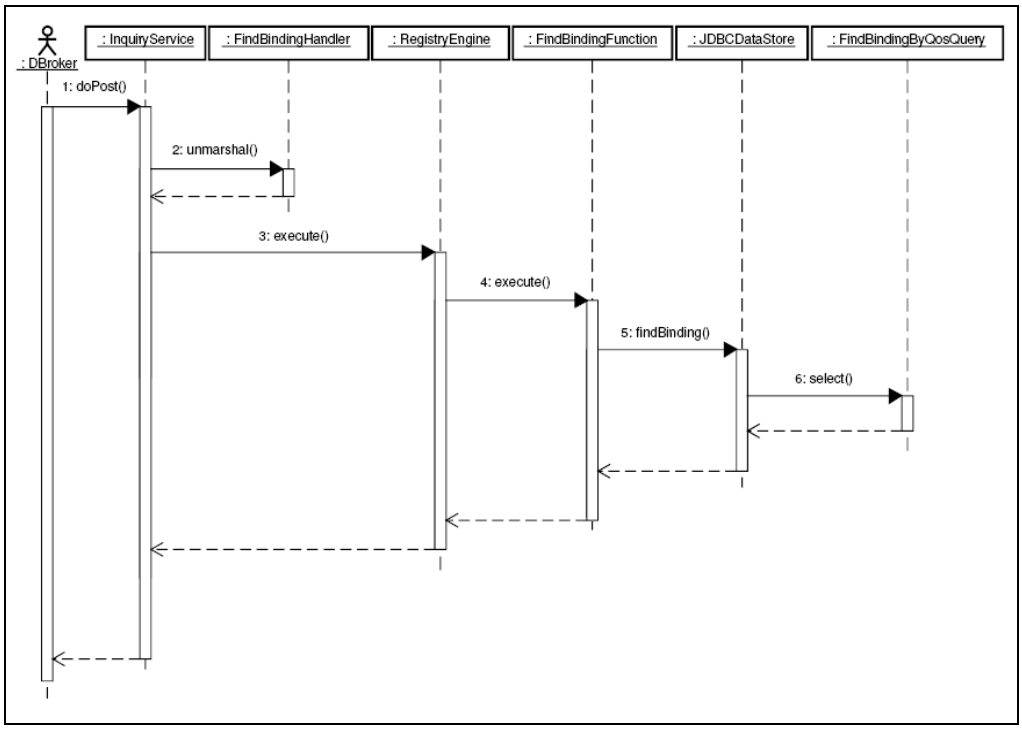

Figura 6.9: Diagrama de seqüência para a descoberta de serviços.

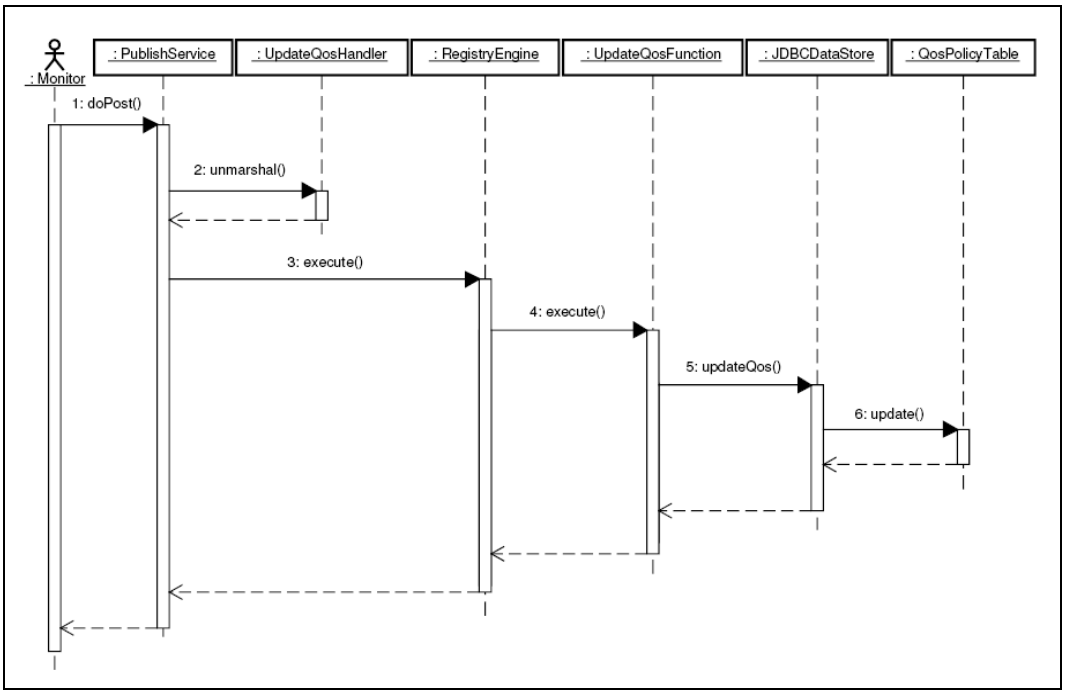

Figura 6.10: Diagrama de seqüência para a atualização de políticas.

As interações apresentadas nos diagramas seguem um padrão comum:

- Para a publicação de serviços, descoberta de serviços e atualização de políticas, a requisição correspondente é submetida ao serviço de publicação ou de consulta, contendo uma estrutura *qosPolicy*, além dos parâmetros padrões nos casos de publicação e descoberta de serviços;
- Por meio do método *unmarshal*, o objeto da classe *Handler* correspondente recebe e traduz a requisição. Ele encaminha a requisição ao objeto *RegistryEngine*;
- O objeto *RegistryEngine* envia um objeto da classe que representa a requisição (*SaveBinding*, *FindBinding* ou *UpdateQos*) ao objeto da classe *Function* correspondente, invocando o método *execute*;
- O objeto *Function* envia um objeto da classe que representa o tipo de estrutura (*BindingTemplate* ou *QosPolicy*) necessário para a publicação de serviços, descoberta de serviços ou atualização de políticas ao objeto *JDBCDataStore*;
- Os dados obtidos a partir do objeto recebido, juntamente com uma conexão com o banco de dados, são encaminhados aos objetos das classes *Table* ou *Query* que incluem os comandos SQL (*Structured Query Language*) necessários para a realização da operação solicitada;
- Os objetos das classes *Table* ou *Query* executam os comandos no banco de dados e retornam os resultados.

6.4 Avaliação de Desempenho

Alguns experimentos simularam buscas por serviços realizadas por consumidores com políticas para atributos de qualidade diferentes.

Para a execução dos experimentos, alguns serviços com funcionalidade equivalente, mas com variadas políticas para atributos de qualidade foram publicados em um repositório UDDI estendido. Um mediador foi utilizado para publicar serviços *Web* no repositório UDDI e outro para descobrir serviços publicados. Além disso, por meio do controle do tempo de resposta dos serviços, alguns casos foram simulados utilizando monitores para testar a atualização de informações sobre atributos de qualidade no repositório.

Os experimentos foram desenvolvidos para avaliar se a gerência de informações sobre características não-funcionais oferecida pelo modelo estendido permite aos consumidores selecionarem serviços *Web* adequados, com respeito às suas políticas para atributos de qualidade além dos seus requisitos funcionais. Os experimentos também tiveram a finalidade de avaliar o desempenho da abordagem proposta. Especificamente, a avaliação foi realizada com os seguintes propósitos:

- Medir o tempo de resposta do modelo e a sobrecarga adicionada pelo modelo;
- Comparar o desempenho do modelo de serviços *Web* estendido com o desempenho do modelo de serviços *Web* atual.

Alguns componentes adicionais foram desenvolvidos para a realização dos experimentos, incluindo consumidores e provedores de serviços com suas políticas para atributos de qualidade. Os consumidores de serviços foram desenvolvidos com módulos para coletar informações sobre desempenho e resultado de seleção. Algumas modificações de parâmetros de serviços foram especificadas para afetar o tempo de resposta dos serviços *Web* em níveis controlados.

Para avaliar o modelo proposto, os testes foram executados em um ambiente computacional com a configuração apresentada na Tabela 6.1.

Sistema operacional	*Linux Fedora Core* 2.6.16-1.2066_FC4
Máquina virtual *Java*	*Sun Java Runtime Environment* 1.5.0_06
Servidor de aplicações	*Apache Tomcat* 4.1.24
Máquina SOAP	*Apache Axis* 1.3
Repositório UDDI	*Apache jUDDI* 0.9rc4
Servidor de banco de dados	*MySQL AB MySQL* 5.0.16
Rede	*Ethernet* / 100 Mbps
Processador / *Chipset*	*Intel Pentium* 4A / 2800 MHz / *Intel Brookdale-G* i845GV
Memória	512 MB / 266 MHz
Placa mãe	*Intel Sea Breeze* D845GVSR
Disco rígido	*Samsung* SP0411N / 40 GB / 7200 RPM

Tabela 6.1: Configuração do sistema

A configuração experimental consistiu de dois cenários de avaliação (CA) com diferentes quantidades de descrições de serviços *Web* publicadas no repositório UDDI, incluindo:

- 250 descrições;
- 500 descrições.

A Tabela 6.2 mostra a distribuição dos serviços *Web* em quatro categorias industriais (CI) e a seletividade de consulta de três requisições de serviços (RS) que foram utilizadas nos experimentos. Por exemplo, 50 serviços foram registrados na categoria de Transporte Aéreo de Carga Programado (CI1) do sistema de classificação *North American Industry Classification System* (NAICS) no cenário 1 (CA1). No cenário 2 (CA2), 100 serviços *Web* foram registrados na mesma categoria industrial. A seletividade de consulta da requisição de serviços 1 (RS1) foi definida para possuir compatibilidade com zero e um serviço *Web* da categoria 1 nos cenários 1 e 2, respectivamente.

Categoria	**Alocação de serviços**		**Seletividade de consulta**					
			RS1		**RS2**		**RS3**	
	CA1	**CA2**	**CA1**	**CA2**	**CA1**	**CA2**	**CA1**	**CA2**
CI1	50	100	0	1	2	3	4	5
CI2	50	100	0	1	2	3	4	5
CI3	75	150	2	4	5	7	8	10
CI4	75	150	3	4	6	7	9	10
Total	250	500	5	10	15	20	25	30

Tabela 6.2: Configuração experimental

O custo de transmissão de descrições de serviços afeta o desempenho do modelo. Esse custo foi avaliado para os casos com e sem a especificação de política para atributos de qualidade. Nas buscas por serviços em que as características não-funcionais não foram consideradas, os resultados incluíram todos os serviços registrados na categoria. A seleção de serviços utilizando políticas para características não-funcionais reduziu o número médio de *bytes* transmitidos ao consumidor, quando comparado com o processamento de requisições sem a gerência de atributos de qualidade. Considerando a requisição 3 (RS3), as reduções foram de aproximadamente:

- 90,0 % no cenário de avaliação 1 (CA1);
- 94,2 % no cenário de avaliação 2 (CA2).

O número de serviços descobertos é diferente para cada situação. Entretanto, a configuração experimental simulou uma situação típica, com um pequeno subconjunto dos serviços registrados na categoria desejada apresentando compatibilidade com as necessidades dos consumidores.

Para avaliar a sobrecarga do modelo, o uso da Unidade Central de Processamento (UCP) foi medido durante a execução dos experimentos.

Conforme mostrado na Tabela 6.3, o modelo gerou uma sobrecarga média adicional de 10,4 % de uso de UCP. Para a descoberta de serviços utilizando o modelo de serviços *Web* básico, o uso de UCP foi de aproximadamente 16,9 %, e aproximadamente 27,3 % para o modelo estendido.

Modelo de serviços *Web*	**Uso de UCP (%)**
Básico	16,9
Estendido	27,3

Tabela 6.3: Sobrecarga do modelo

Para avaliar o tempo de resposta do modelo, o tempo de processamento de requisições de descoberta de serviços foi medido durante a execução dos experimentos. A Figura 6.11 mostra o tempo de resposta médio do modelo para o ambiente de execução descrito nas Tabelas 6.1 e 6.2, considerando o número de serviços publicados e selecionados.

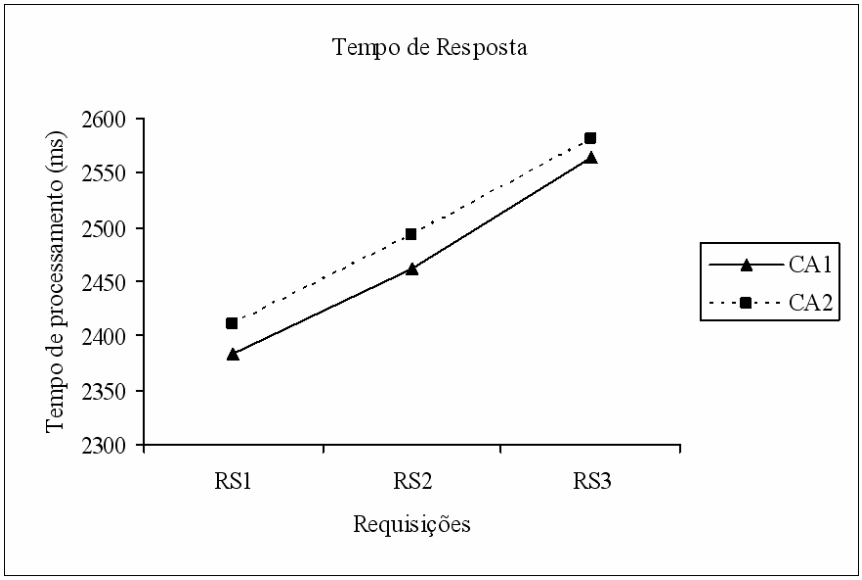

Figura 6.11: Tempo de resposta do modelo de serviços *Web* estendido.

Para medir o impacto em termos do tempo de resposta para a descoberta de serviços, o tempo de processamento de requisições de descoberta considerando características não-funcionais foi comparado com o tempo de processamento para os mesmos serviços *Web* considerando apenas a funcionalidade. O cenário 2 (CA2) foi avaliado para os dois casos. Conforme esperado, os resultados da avaliação, apresentados na Figura 6.12, mostraram um incremento médio no tempo de resposta de aproximadamente 2,1 %.

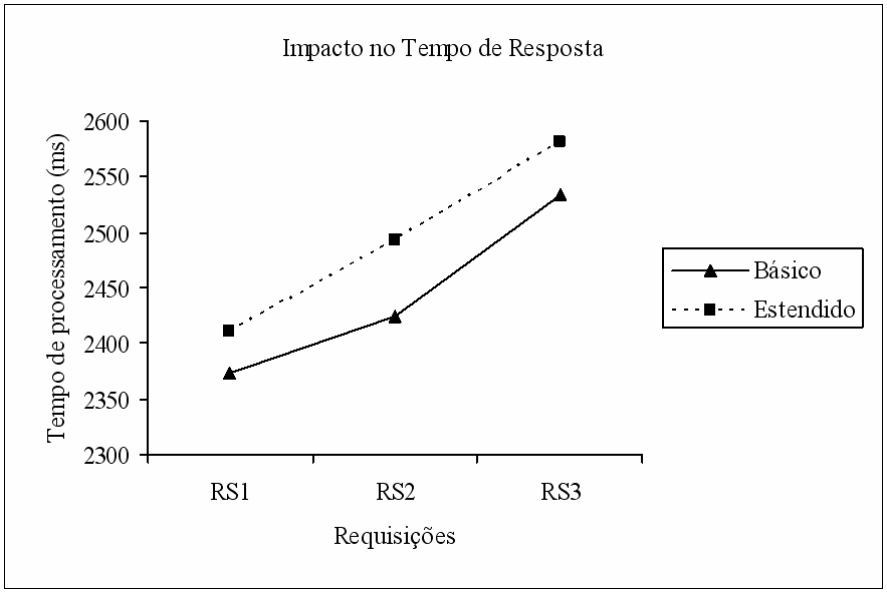

Figura 6.12: Impacto da extensão no tempo de resposta do modelo de serviços *Web*.

Os resultados demonstraram que a abordagem é praticável. O modelo proposto permitiu a descoberta de serviços *Web* satisfazendo as demandas de consumidores de serviços. A análise do comprometimento de desempenho provocado pela extensão do modelo depende do tempo de resposta do serviço. O impacto da extensão no tempo de resposta do modelo de serviços *Web* foi pequeno. Entretanto, para serviços com tempo de resposta pequeno, esse impacto pode ser significativo.

A maior precisão na descoberta de serviços *Web* e a conseqüente redução de dados transmitidos aos consumidores de serviços compensaram o maior tempo de processamento requerido para as requisições de descoberta de serviços.

As políticas para características não-funcionais dos provedores e dos consumidores permitiram uma descoberta de serviços *Web* refinada, pois somente os serviços relevantes para os consumidores foram retornados nas respostas às requisições de descoberta, isto é, os serviços pertencentes às categorias industriais requeridas e com as capacidades de qualidade desejadas. As políticas para características não-funcionais impediram que os serviços *Web* que não se adequavam realmente às necessidades dos consumidores de serviços fossem selecionados.

Por exemplo, no cenário de avaliação 2 (CA2), a busca básica por serviços da categoria industrial de Transporte Aéreo de Carga Programado (CI1) recuperou os 100 serviços *Web* registrados na categoria. Já a busca baseada em políticas para características não-funcionais possibilitou a recuperação do único serviço *Web* adequado aos requisitos completos do consumidor de serviços.

O custo da publicação de políticas para características não-funcionais de serviços *Web* também deve ser considerado, especialmente para grandes quantidades de atributos de qualidade. Porém, tal custo é pago apenas uma vez e, tipicamente, o número de características não-funcionais é limitado pelos atributos aplicáveis à categoria associada.

Outro custo adicional está associado com o monitoramento de características não-funcionais. Entretanto, o uso de monitores permite, por exemplo, que os consumidores de serviços fiquem livres da necessidade de informar os atributos de qualidade reais dos serviços *Web*, além de ajudar a garantir os níveis de qualidade esperados.

Capítulo 7

Conclusões

A tecnologia de serviços *Web* oferece uma base adequada para implementar sistemas distribuídos e executar operações de negócio eletrônico dentro e através de fronteiras organizacionais. Ela ainda está em desenvolvimento. Um progresso foi feito para ampliar a aplicação de serviços *Web*, mas existem alguns problemas que estão impedindo a ampla aceitação da tecnologia. Os padrões WSDL e UDDI são componentes básicos do modelo de serviços *Web* atual. Contudo, eles estão desprovidos de mecanismos para lidar com atributos de qualidade.

Neste trabalho, o modelo de serviços *Web* básico foi estendido para apoiar a gerência de características não-funcionais. Na abordagem, dois novos componentes são adicionados ao modelo de serviços *Web* básico e padrões de serviços *Web* são utilizados e estendidos para tratar alguns aspectos da gerência de características não-funcionais. Foram discutidos: especificação de atributos de qualidade, publicação e descoberta de serviços enriquecidos com atributos de qualidade, e verificação de características não-funcionais.

No modelo, mediadores são utilizados para manipular atributos de qualidade para serviços *Web*. Os mediadores utilizam algumas operações do padrão *WS-Policy* para processar políticas. Eles também utilizam a API do padrão UDDI para interagir com repositórios UDDI. Os mediadores apóiam o estabelecimento de parcerias entre consumidores e provedores de serviços.

A abordagem combina propostas para a inclusão de alguns componentes na linguagem *WS-Policy* para a especificação de requisitos e capacidades não-funcionais, o mapeamento de políticas em repositórios UDDI e a verificação de compatibilidade entre políticas de consumidores e provedores durante a seleção de serviços, e o monitoramento e a atualização de informações sobre características não-funcionais.

O modelo de serviços *Web* proposto foi parcialmente implementado e testado para verificar a aplicabilidade da abordagem. Os resultados demonstraram que a gerência de atributos de qualidade oferece um apoio adequado para aprimorar a descoberta de serviços *Web*.

7.1 Contribuições

Recentemente, devido aos diversos esforços de pesquisa em gerência de atributos de qualidade para serviços *Web*, um progresso substancial foi alcançado. As soluções, entretanto, são tipicamente limitadas por algumas das razões discutidas a seguir:

- A publicação de políticas e o padrão UDDI não são explorados. A publicação de políticas para atributos de qualidade em repositórios de serviços permite que elas sejam utilizadas para refinar a descoberta de serviços. UDDI é o padrão para repositórios de serviços *Web*. Complementar o repositório UDDI com outros repositórios torna a incorporação dos repositórios no modelo de serviços *Web* atual mais difícil. Um modo integrado para lidar com requisitos funcionais e não-funcionais pode ser oferecido estendendo o padrão UDDI;
- O mecanismo de *tModel* não é utilizado para o estabelecimento de uma conceituação de características de qualidade compartilhada. Isso é importante, pois permite, por exemplo, a definição de atributos de qualidade em domínios específicos de serviços. O mecanismo de *tModel* tem outra vantagem: ele é parte do padrão UDDI e atualmente é utilizado para definir um amplo conjunto de conceitos relacionados a serviços *Web*;
- A gerência de atributos de qualidade não considera a possibilidade de mudanças em atributos de qualidade. Então, um mecanismo para a verificação de atributos de qualidade não é empregado para permitir a atualização de descrições de características não-funcionais publicadas;
- Políticas não são utilizadas. Políticas oferecem um modo para definir capacidades e requisitos de propriedades gerais. Conseqüentemente, elas são uma escolha apropriada para especificar características não-funcionais de serviços *Web*. Além disso, a verificação da compatibilidade entre políticas pode ser utilizada para aprimorar a descoberta de serviços;
- Finalmente, o padrão *WS-Policy* não é utilizado. *WS-Policy* oferece um padrão aberto flexível para expressar características não-funcionais de serviços *Web*. O padrão é compatível com os requisitos fundamentais de serviços *Web*: simplicidade e apoio abrangente na indústria [2].

Neste trabalho, as questões citadas são tratadas. As principais contribuições deste trabalho são:

- A utilização do padrão *WS-Policy* para a especificação de políticas para atributos de qualidade. Uma política *WS-Policy* complementa uma descrição WSDL padrão, a qual especifica aspectos funcionais de serviços *Web*. Os consumidores de serviços também utilizam políticas para especificar seus requisitos não-funcionais;
- Uma extensão do padrão UDDI para incluir atributos de qualidade. O modelo informacional e a API do UDDI foram adaptados para permitir a manipulação de políticas para serviços em repositórios UDDI. A extensão inclui a publicação de políticas e a descoberta de serviços utilizando os atributos de qualidade publicados e as

políticas dos consumidores. O mecanismo de *tModel* do UDDI foi utilizado para compartilhar os conceitos de qualidade;
- A verificação de políticas para atributos de qualidade utilizando monitores para refletir os atributos reais. Esse aspecto é importante devido à natureza dinâmica de parte dos atributos de qualidade. A atualização apoiada pelo monitoramento oferece uma garantia dos níveis de qualidade esperados;
- A implementação parcial do modelo como forma de mostrar sua aplicabilidade e avaliar seu desempenho.

A principal vantagem da tecnologia de serviços *Web* é a interoperabilidade alcançada através da padronização. Neste trabalho, alguns padrões de serviços *Web* são utilizados e estendidos com o propósito de incluir a gerência de características não-funcionais no modelo de serviços *Web*.

7.2 Trabalhos Futuros

Alguns possíveis trabalhos futuros são:

- O aprimoramento da implementação e da avaliação desenvolvidas. O modelo pode ser utilizado para a gerência de atributos de qualidade gerais e específicos. Entretanto, a implementação atual é limitada. A avaliação foi baseada na implementação. Uma implementação incluindo o monitoramento de atributos de qualidade adicionais e alguns cenários de avaliação considerando atributos específicos para a descoberta de serviços podem ser desenvolvidos. Tais aprimoramentos possibilitariam uma avaliação mais refinada do modelo;
- A adaptação do modelo para apoiar serviços *Web* móveis. Prover serviços em ambientes móveis inclui desafios adicionais. Por exemplo, é necessário definir atributos de qualidade específicos, tais como, tamanho de resposta de serviço e custo de processamento de resposta de serviço. Essa adaptação é importante, pois o número de esforços de pesquisa em serviços *Web* móveis está crescendo, o que demonstra a tendência de aplicação da tecnologia de serviços *Web* em ambientes móveis;
- A inclusão de um mecanismo de sensibilidade ao contexto[7] no modelo proposto. Além dos atributos de qualidade demandados pelos consumidores, o contexto de cada consumidor pode ser considerado durante a seleção de serviços. Esse mecanismo permitiria uma descoberta de serviços mais refinada;
- A aplicação de uma abordagem semântica para a gerência de características não-funcionais. Uma abordagem semântica pode ser utilizada para solucionar a limitação do padrão *WS-Policy* e do modelo proposto, que permitem apenas a descrição de aspectos sintáticos de propriedades de serviços. Tal abordagem garantiria um conjunto mais amplo de interações possíveis entre participantes.

---

[7] Do inglês *context-awareness*.

Referências Bibliográficas

[1] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, e Alex Yiu. *Web Services Business Process Execution Language Version 2.0 Specification*. OASIS, 23 Agosto 2006. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html, acessado em 12/2006.

[2] Gustavo Alonso, Fabio Casati, Harumi Kuno, e Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2003.

[3] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asir Vedamuthu, Umit Yalcynalp, e David Orchard. *Web Services Policy Attachment (WS-PolicyAttachment) Version 1.2 Specification*. BEA, IBM, Microsoft, SAP, Sonic, e VeriSign, 01 Março 2006. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polatt/ws-polat-2006-03-01.pdf, acessado em 08/2006.

[4] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asir Vedamuthu, Umit Yalcynalp, e David Orchard. *Web Services Policy Framework (WS-Policy) Version 1.2 Specification*. BEA, IBM, Microsoft, SAP, Sonic, e VeriSign, 01 Março 2006. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/WS-Policy-2006-03-01.pdf, acessado em 08/2006.

[5] David Burdett, Qiming Chen, e Meichun Hsu. Conductor: An enabler for Web services-based business collaboration. *IEEE Data Engineering Bulletin*, 25(4):22–26, 2002.

[6] Keith Ballinger, David Ehnebuske, Martin Gudgin, Mark Nottingham, e Prasad Yendluri. *Basic Profile Version 1.1 Specification*. WS-I, 10 Abril 2006. http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html, acessado em 08/2006.

[7] Luciano Baresi, Sam Guinea, e Pierluigi Plebani. WS-Policy for service monitoring. Em Christoph Bussler e Ming-Chien Shan, editores, *TES '05: 6th VLDB Intl. Workshop on Technologies for E-Services*, volume 3811 de *Lecture Notes in Computer Science*, páginas 72–83. Springer, 2006.

[8] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, e David Orchard. *Web Services Architecture*. W3C, 11 Fevereiro 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, acessado em 08/2006.

[9] Martin Bernauer, Gert Kappel, Gerhard Kramler, e Werner Retschitzegger. Specification of interorganizational workflows: A comparison of approaches. Em *SCI '03: Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, páginas 30–36, Orlando, EUA, 2003. IIISCI.

[10] Peter Brittenham e David Lauzon. *Web Services Interoperability Analyzer Tool Functional Specification V. 1.1*. WS-I, 13 Junho 2005. http://ws-i.org/Testing/Specs/AnalyzerFunctionalSpecification_Final_1.1.pdf, acessado em 08/2006.

[11] Boualem Benatallah e H. R. Motahari Nezhad. Service Oriented Computing: Opportunities and challenges. Em Christoph Bussler, Val Tannen, e Irini Fundulaki, editores, *SWDB '04: Second International Workshop on Semantic Web and Databases*, volume 3372 de *Lecture Notes in Computer Science*, páginas 1–8. Springer Berlin / Heidelberg, 2005.

[12] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, e François Yergeau. *Extensible Markup Language (Fourth Edition) Version 1.0 Specification*. W3C, 16 Agosto 2006. http://www.w3.org/TR/2006/REC-xml-20060816/, acessado em 12/2006.

[13] Fabio Casati. A conversation on Web services: What is new, what is true, what is hot. And what is not (invited paper). Em *Proceedings of the ECAI Workshop on Knowledge Transformation and the Semantic Web*, páginas 1–2, Lyon, França, 2002. IOS Press.

[14] Fabio Casati e Angela Discenza. Modeling and managing interactions among business processes. *J. of Systems Integration*, 10(2):100–111, 2001.

[15] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, e Sanjiva Weerawarana. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 2002.

[16] Mike Clark, Peter Fletcher, Jeffrey J. Hanson, Romin Irani, Mark Waterhouse, e Jorgen Thelin. *Web Services Business Strategies and Architectures*. A-Press, 2003.

[17] Luc Clement, Andrew Hately, Claus von Riegen, e Tony Rogers. *Universal Description Discovery & Integration Version 3.0.2 Specification*. OASIS, 19 Outubro 2004. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm, acessado em 08/2006.

[18] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, e Ming-Chien Shan. Adaptive and dynamic service composition in eFlow. Em Benkt Wangler e Lars Bergman, editores, *CAiSE '00: 12th International Conference on Advanced Information Systems Engineering*, volume 1789 de *Lecture Notes in Computer Science*, páginas 13–31. Springer Berlin / Heidelberg, 2000.

[19] Zhou Chen, Chia Liang-Tien, Bilhanan Silverajan, e Lee Bu-Sung. UX: An architecture providing QoS-aware and federated support for UDDI. Em *ICWS '03: Proceedings of the International Conference on Web Services*, páginas 171–176. CSREA Press, 2003.

[20] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, e Sanjiva Weerawarana. *Web Services Description Language Part 1 Version 2.0 Specification*. W3C, 10 Maio 2005. http://www.w3.org/TR/2005/WD-wsdl20-20050510/, acessado em 08/2006.

[21] Fabio Casati, Ming-Chien Shan, e Dimitrios Georgakopoulos. E-services (guest editorial). *VLDB Journal*, 10(1):1, 2001.

[22] Umeshwar Dayal, Meichun Hsu, e Rivka Ladin. Business process coordination: State of the art, trends, and open issues. Em *VLDB '01: Proceedings of the 27th Intl. Conference on Very Large Data Bases*, páginas 3–13, São Francisco, EUA, 2001. Morgan Kaufmann Publishers Inc.

[23] Zongxia Du, Jinpeng Huai, e Yunhao Liu. Ad-UDDI: An active and distributed service registry. Em Christoph Bussler e Ming-Chien Shan, editores, *TES '05: 6th VLDB Intl. Workshop on Technologies for E-Services*, volume 3811 de *Lect. Notes in Comp. Science*, páginas 58–71. Springer, 2006.

[24] Doug Davis, Anish Karmarkar, Gilbert Pilz, Steve Winkler, e Umit Yalçinalp. *Web Services Reliable Messaging Committee Draft 03 Specification*. OASIS, 14 Março 2006. http://docs.oasis-open.org/ws-rx/wsrm/200602/wsrm-1.1-spec-cd-03.pdf, acessado em 08/2006.

[25] Marcelo Fantinato, Maria Beatriz Felgar de Toledo, e Itana Maria de Souza Gimenes. Arquiteturas de Sistemas de Gerenciamento de Processos de Negócio baseados em serviços. Relatório Técnico IC-05-06, Universidade Estadual de Campinas, Campinas, Brasil, Abril 2005.

[26] Max Feingold. *Web Services Coordination Version 1.1 Specification*. OASIS, 15 Março 2006. http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-cd-01.pdf, acessado em 12/2006.

[27] Tom Freund, Alastair Green, John Harby, e Mark Little. *Web Services Business Activity Version 1.1 Specification*. OASIS, 15 Março 2006. http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-cd-01.pdf, acessado em 12/2006.

[28] Jianchun Fan e Subbarao Kambhampati. A snapshot of public Web services. *SIGMOD Record*, 34(1):24–32, 2005.

[29] Dinesh Ganesarajah e Emil Lupu. Workflow-based composition of Web-services: A business model or a programming paradigm?. Em *EDOC '02: Proceedings of the Sixth International Enterprise Distributed Object Computing Conference*, páginas 273–284, Washington, EUA, 2002. IEEE Computer Society.

[30] Diego Zuquim Guimarães Garcia e Maria Beatriz Felgar de Toledo. A Web service architecture providing QoS management. Em *LA-Web '06: Proceedings of the Fourth Latin American Web Congress*, páginas 189–198, Washington, EUA, 2006. IEEE Computer Society.

[31] David Hollingsworth. The workflow reference model 10 years on. Em *Workflow Handbook 2004*. WfMC, Winchester, Reino Unido, 2004.

[32] Dimitris Karagiannis. BPMS: Business Process Management Systems. *SIGOIS Bulletin*, 16(1):10–13, 1995.

[33] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, e Charlton Barreto. *Web Services Choreography Description Language Version 1.0 Specification*. W3C, 09 Novembro 2005. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, acessado em 12/2006.

[34] Sravanthi Kalepu, Shonali Krishnaswamy, e Seng Wai Loke. Verity: A QoS metric for selecting Web services and providers. Em *WISE '03: Proceedings of the International Conference on Web Information Systems Engineering Workshops*, páginas 131–139. Springer, 2004.

[35] Alexander Keller e Heiko Ludwig. The WSLA framework: Specifying and monitoring service level agreements for Web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[36] Natallia Kokash. Web service discovery with implicit QoS filtering. Em Andreas Hanemann, editor, *Proceedings of the IBM PhD Student Symposium at the 3rd International Conference on Service Oriented Computing (ICSOC 2005)*, volume 169 de *IBM Technical Reports*, páginas 61–66. CEUR, 2005.

[37] Frank Leymann. *Web Services Flow Language Version 1.0 Specification*. IBM, Maio 2001. http://www-3.ibm.com/software/solutions/*Web*services /pdf/WSFL.pdf, acessado em 12/2006.

[38] KangChan Lee, JongHong Jeon, WonSeok Lee, SeongHo Jeong, e Sang-Won Park. *QoS for Web Services: Requirements and Possible Approaches*. W3C, 25 Novembro 2003. http://www.w3c.or.kr/kroffice/TR/2003/NOTEwsqos20031125/, acessado em 08/2006.

[39] Eunjoo Lee, Woosung Jung, Wookjin Lee, Young-Joo Park, Byungjeong Lee, Heechern Kim, e Chisu Wu. A framework to support QoS-aware usage of Web services. Em *ICWE '05: Proceedings of the 5th International Conference on Web Engineering*, páginas 318–327. Springer, 2005.

[40] Frank Leymann, Dieter Roller, e Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal, New Developments in Web Services and Electronic Commerce*, 41(2):198–211, 2002.

[41] Amaia Lazcano, Heiko Schuldt, Gustavo Alonso, e Hans Schek. WISE: Process-based E-Commerce. *IEEE Data Engineering Bulletin*, 24(1):01–06, 2001.

[42] Heiko Ludwig. Web services QoS: External SLAs and internal policies or: How do we deliver what we promise?. Em *WISE '03: Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops*, páginas 115–120. Springer, 2003.

[43] Mark Little e Andrew Wilkinson. *Web Services Atomic Transaction Version 1.1 Specification*. OASIS, Março 2006. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-cd-01.pdf, acessado em 08/2006.

[44] Brahim Medjahed, Boualem Benatallah, Athman Bouguettaya, Anne H. H. Ngu, e Ahmed K. Elmagarmid. Business-to-business interactions: Issues and enabling technologies. *VLDB Journal*, 12(1):59–85, 2003.

[45] Daniel A. Menasce. QoS issues in Web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[46] Nilo Mitra. *SOAP Part 0 Primer Version 1.2 Specification*. W3C, 24 Junho 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, acessado em 08/2006.

[47] Nirmal K. Mukhi e Pierluigi Plebani. Supporting policy-driven behaviors in Web services: Experiences and issues. Em *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, páginas 322–328, Nova Iorque, EUA, 2004. ACM Press.

[48] E. Michael Maximilien e M. P. Singh. A framework and ontology for dynamic Web services selection. *IEEE Internet Comp.*, 8(5):84–93, 2004.

[49] Jeffrey V. Nickerson. Logical channels: Using Web services for cross-organizational workflow. *Business Process Management Journal*, 11(3):224–236, 2005.

[50] Anthony Nadalin, Chris Kaler, Ronald Monzillo, e Phillip Hallam-Baker. *Web Services Security: SOAP Message Security Version 1.1 Specification*. OASIS, Fevereiro 2006. http://oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, acessado em 08/2006.

[51] Object Management Group. *Common Object Request Broker Architecture: Core Specification, Version 3.0.3*. OMG, 12 Março 2004. http://www.omg.org/cgi-bin/apps/doc?formal/04-03-12.pdf, acessado em 12/2006.

[52] Paulo F. Pires, Mario R. F. Benevides, e Marta Mattoso. Building reliable Web services compositions. Em Akmal B. Chaudhri, Mario Jeckle, Erhard Rahm, e Rainer Unland, editores, *Web, Web-Services, and Database Systems: NODe 2002, Web- and Database-Related Workshops*, volume 2593 de *Lecture Notes in Computer Science*, páginas 59–72. Springer, 2003.

[53] Michael P. Papazoglou e Jean-Jacques Dubray. A survey of Web service technologies. Relatório Técnico DIT-04-058, Universidade de Trento, Trento, Itália, 2004.

[54] Michael P. Papazoglou e Dimitris Georgakopoulos. Service-Oriented Computing (guest editorial). *Comm. of the ACM*, 46(10):24–28, 2003.

[55] Shuping Ran. A framework for discovering Web services with desired quality of services attributes. Em *ICWS '03: Proceedings of the International Conference on Web Services*, páginas 208–213. CSREA Press, 2003.

[56] Bikash Sabata, Saurav Chatterjee, Michael Davis, Jaroslaw J. Sydir, e Thomas F. Lawrence. Taxonomy for QoS specifications. Em *WORDS '97: Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems*, páginas 100–107, Washington, EUA, 1997. IEEE Computer Society.

[57] Rainer Schmidt. Web services based architectures to support dynamic inter-organizational business processes. Em Mario Jeckle e Liang-Jie Zhang, editores, *ICWS-Europe '03: Proceedings of the International Conference on Web Services*, volume 2853 de *Lecture Notes in Computer Science*, páginas 123–136. Springer, 2003.

[58] M. Adel Serhani, Rachida Dssouli, Abdelhakim Hafid, e Houari Sahraoui. A QoS broker based architecture for efficient Web services selection. Em *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, páginas 113–120, Washington, EUA, 2005. IEEE Computer Society.

[59] Vishal Sikka. Data and metadata management in service-oriented architectures: Some open challenges. Em *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, páginas 849–850, Nova Iorque, EUA, 2005. ACM Press.

[60] Zafar U. Singhera. Extended Web services framework to meet non-functional requirements. Em *SAINT-W '04: Proceedings of the 2004 Symposium on Applications and the Internet Workshops*, páginas 334–340, Washington, EUA, 2004. IEEE Computer Society.

[61] Kaarthik Sivashanmugam, John A. Miller, Amit Sheth, e Kunal Verma. Framework for semantic Web process composition. *International Journal of Electronic Commerce*, 9(2):71–106, 2005.

[62] Ali ShaikhAli, Omer F. Rana, Rashid Al-Ali, e David W. Walker. UDDIe: An extended registry for Web services. Em *SAINT-W '03: Proceedings of the 2003 Symposium on Applications and the Internet Workshops*, páginas 85–89, Washington, EUA, 2003. IEEE Computer Society.

[63] Sun Microsystems. *Java Remote Method Invocation, Java SE 6, Spec*. Sun, 2006. http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html, acessado em 12/2006.

[64] Vladimir Tosic, Kruti Patel, e Bernard Pagurek. WSOL: Web Service Offerings Language. Em Christoph Bussler, Richard Hull, Sheila A. McIlraith, Maria E. Orlowska, Barbara Pernici, e Jian Yang, editores, *WES '02: Proceedings of the CAiSE 2002 International Workshop on Web Services, E-Business, and the Semantic Web*, volume 2512 de *Lecture Notes in Computer Science*, páginas 57–67. Springer Berlin / Heidelberg, 2002.

[65] Vladimir Tosic, Aad van Moorsel, e Raymond Wong. Quality of Service (QoS) middleware for Web services: Achieved results and challenges for the future (panel discussion). Em *MWS '05: Proceedings of the EDOC Middleware for Web Services Workshop*, páginas 19–23, Washington, EUA, 2005. IEEE Computer Society.

[66] Wil M. P. van der Aalst. Process-oriented architectures for electronic commerce and interorganizational workflow. *Information Systems*, 24(9):639–671, 1999.

[67] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, e Mathias Weske. Business Process Management: A survey. Em *BPM '03: Proceedings of the Intl. Conf. on Business Process Management*, páginas 1–12. Springer, 2003.

[68] Le-Hung Vu, Manfred Hauswirth, e Karl Aberer. QoS-based service selection and ranking with trust and reputation management. Em Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Ozalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, e Stefano Spaccapietra, editores, *OTM Conferences '05: Proceedings of the OTM Confederated International Conferences CoopIS, DOA, and ODBASE*, volume 3760 de *Lecture Notes in Computer Science*, páginas 466–483. Springer, 2005.

[69] Eric Wohlstadter, Stefan Tai, Thomas Mikalsen, Isabelle Rouvellou, e Premkumar Devanbu. GlueQoS: Middleware to sweeten quality-of-service policy interactions. Em *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, páginas 189–199, Washington, EUA, 2004. IEEE Computer Society.

[70] J. Leon Zhao e Hsing Kenneth Cheng. Web services and process management: A union of convenience or a new area of research?. *Decision Support Systems*, 40(1):1–8, 2005.

[71] Liang-Jie Zhang, Tian Chao, Henry Chang, e Jen-Yao Chung. XML-based advanced UDDI search mechanism for B2B integration. *Electronic Commerce Research*, 3(1-2):25–42, 2003.

[72] Xiaohui Zhao, Chengfei Liu, e Yun Yang. Web service based architecture for workflow management systems. Em Fernando Galindo, Makoto Takizawa, e Roland Traunmuller, editores, *DEXA '04: Proceedings of the 15th International Conference Database and Expert Systems Applications*, volume 3180 de *Lecture Notes in Computer Science*, páginas 34–43. Springer, 2004.

Apêndice A

Esquema XML para o Padrão UDDI Estendido

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
targetNamespace="urn:uddi-org:api_v2" <!-- OASIS Open Specification -->
xmlns:uddi="urn:uddi-org:api_v2" xmlns:ue="urn:uddi-org:api_ue"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <!-- Attributes -->
      <xsd:simpleType name="bindingKey">
       <xsd:restriction base="string"/>
      </xsd:simpleType>
      <xsd:simpleType name="serviceKey">
       <xsd:restriction base="string"/>
      </xsd:simpleType>
      <xsd:simpleType name="tModelKey">
       <xsd:restriction base="string"/>
      </xsd:simpleType>

      <!-- Types and elements for registry contents -->
      <xsd:element name="accessPoint" type="uddi:accessPoint"/>
      <xsd:complexType name="accessPoint">
       <xsd:simpleContent>
           <xsd:extension base="string">
               <xsd:attribute name="URLType" type="uddi:URLType"
use="required"/>
           </xsd:extension>
       </xsd:simpleContent>
      </xsd:complexType>
      <xsd:element name="authInfo" type="string"/>
      <xsd:element name="bindingKey" type="uddi:bindingKey"/>
      <xsd:element name="bindingTemplate" type="uddi:bindingTemplate"/>
```

```
      <xsd:complexType name="bindingTemplate">
       <xsd:sequence>
           <xsd:element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:choice>
               <xsd:element ref="uddi:accessPoint"/>
               <xsd:element ref="uddi:hostingRedirector"/>
           </xsd:choice>
           <xsd:element ref="uddi:tModelInstanceDetails"/>
           <xsd:element ref="ue:qosPolicy" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="serviceKey" type="uddi:serviceKey"
use="optional"/>
       <xsd:attribute name="bindingKey" type="uddi:bindingKey"
use="required"/>
      </xsd:complexType>
      <xsd:element name="qosPolicy" type="ue:qosPolicy"/>
      <xsd:complexType name="qosPolicy">
       <xsd:sequence>
           <xsd:element ref="uddi:keyedReference" maxOccurs="unbounded"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="businessKey" type="uddi:businessKey"/>
      <xsd:element name="businessService" type="uddi:businessService"/>
      <xsd:complexType name="businessService">
       <xsd:sequence>
           <xsd:element ref="uddi:name" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="uddi:bindingTemplates" minOccurs="0"/>
           <xsd:element ref="uddi:categoryBag" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="serviceKey" type="uddi:serviceKey"
use="required"/>
       <xsd:attribute name="businessKey" type="uddi:businessKey"
use="optional"/>
      </xsd:complexType>
      <xsd:element name="dispositionReport"
type="uddi:dispositionReport"/>
      <xsd:complexType name="dispositionReport">
       <xsd:sequence>
           <xsd:element ref="uddi:result" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
       <xsd:attribute name="operator" type="string" use="required"/>
       <xsd:attribute name="truncated" type="uddi:truncated"
use="optional"/>
      </xsd:complexType>
      <xsd:element name="findQualifier" type="string"/>
      <xsd:element name="findQualifiers" type="uddi:findQualifiers"/>
```

```
      <xsd:complexType name="findQualifiers">
       <xsd:sequence>
           <xsd:element ref="uddi:findQualifier" minOccurs="0"
maxOccurs="unbounded"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="keyedReference" type="uddi:keyedReference"/>
      <xsd:complexType name="keyedReference">
       <xsd:attribute name="tModelKey" type="uddi:tModelKey"
use="optional"/>
       <xsd:attribute name="keyName" type="string" use="optional"/>
       <xsd:attribute name="keyValue" type="string" use="required"/>
      </xsd:complexType>
      <xsd:element name="tModel" type="uddi:tModel"/>
      <xsd:complexType name="tModel">
       <xsd:sequence>
           <xsd:element ref="uddi:name"/>
           <xsd:element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="uddi:overviewDoc" minOccurs="0"/>
           <xsd:element ref="uddi:identifierBag" minOccurs="0"/>
           <xsd:element ref="uddi:categoryBag" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="tModelKey" type="uddi:tModelKey"
use="required"/>
       <xsd:attribute name="operator" type="string" use="optional"/>
       <xsd:attribute name="authorizedName" type="string"
use="optional"/>
      </xsd:complexType>
      <xsd:element name="tModelInfo" type="uddi:tModelInfo"/>
      <xsd:complexType name="tModelInfo">
       <xsd:sequence>
           <xsd:element ref="uddi:name"/>
       </xsd:sequence>
       <xsd:attribute name="tModelKey" type="uddi:tModelKey"
use="required"/>
      </xsd:complexType>
      <xsd:element name="tModelInstanceDetails"
type="uddi:tModelInstanceDetails"/>
      <xsd:complexType name="tModelInstanceDetails">
       <xsd:sequence>
           <xsd:element ref="uddi:tModelInstanceInfo" minOccurs="0"
maxOccurs="unbounded"/>
       </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="tModelInstanceInfo"
type="uddi:tModelInstanceInfo"/>
```

```
      <xsd:complexType name="tModelInstanceInfo">
       <xsd:sequence>
           <xsd:element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="uddi:instanceDetails" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="tModelKey" type="uddi:tModelKey"
use="required"/>
      </xsd:complexType>
      <xsd:element name="tModelKey" type="uddi:tModelKey"/>

      <!-- Types and elements for input messages -->
      <xsd:element name="delete_binding" type="uddi:delete_binding"/>
      <xsd:complexType name="delete_binding">
       <xsd:sequence>
           <xsd:element ref="uddi:authInfo"/>
           <xsd:element ref="uddi:bindingKey" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
      </xsd:complexType>
      <xsd:element name="delete_tModel" type="uddi:delete_tModel"/>
      <xsd:complexType name="delete_tModel">
       <xsd:sequence>
           <xsd:element ref="uddi:authInfo"/>
           <xsd:element ref="uddi:tModelKey" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
      </xsd:complexType>
      <xsd:element name="discard_authToken"
type="uddi:discard_authToken"/>
      <xsd:complexType name="discard_authToken">
       <xsd:sequence>
           <xsd:element ref="uddi:authInfo"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
      </xsd:complexType>
      <xsd:element name="find_binding" type="uddi:find_binding"/>
      <xsd:complexType name="find_binding">
       <xsd:sequence>
           <xsd:element ref="uddi:findQualifiers" minOccurs="0"/>
           <xsd:element ref="uddi:tModelBag"/>
           <xsd:element ref="ue:qosPolicy" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
       <xsd:attribute name="maxRows" type="int" use="optional"/>
       <xsd:attribute name="serviceKey" type="uddi:serviceKey"
use="required"/>
      </xsd:complexType>
      <xsd:element name="update_qos" type="ue:update_qos"/>
```

```
        <xsd:complexType name="update_qos">
         <xsd:sequence>
             <xsd:element ref="uddi:authInfo"/>
             <xsd:element ref="uddi:bindingKey"/>
             <xsd:element ref="ue:qosPolicy"/> </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="find_tModel" type="uddi:find_tModel"/>
        <xsd:complexType name="find_tModel">
         <xsd:sequence>
             <xsd:element ref="uddi:findQualifiers" minOccurs="0"/>
             <xsd:element ref="uddi:name" minOccurs="0"/>
             <xsd:element ref="uddi:identifierBag" minOccurs="0"/>
             <xsd:element ref="uddi:categoryBag" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="generic" type="string" use="required"/>
         <xsd:attribute name="maxRows" type="int" use="optional"/>
        </xsd:complexType>
        <xsd:element name="get_authToken" type="uddi:get_authToken"/>
        <xsd:complexType name="get_authToken">
         <xsd:attribute name="generic" type="string" use="required"/>
         <xsd:attribute name="userID" type="string" use="required"/>
         <xsd:attribute name="cred" type="string" use="required"/>
        </xsd:complexType>
        <xsd:element name="get_bindingDetail"
type="uddi:get_bindingDetail"/>
        <xsd:complexType name="get_bindingDetail">
         <xsd:sequence>
             <xsd:element ref="uddi:bindingKey" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="generic" type="string" use="required"/>
        </xsd:complexType>
        <xsd:element name="get_tModelDetail" type="uddi:get_tModelDetail"/>
        <xsd:complexType name="get_tModelDetail">
         <xsd:sequence>
             <xsd:element ref="uddi:tModelKey" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="generic" type="string" use="required"/>
        </xsd:complexType>
        <xsd:element name="save_binding" type="uddi:save_binding"/>
        <xsd:complexType name="save_binding">
         <xsd:sequence>
             <xsd:element ref="uddi:authInfo"/>
             <xsd:element ref="uddi:bindingTemplate" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="generic" type="string" use="required"/>
        </xsd:complexType>
        <xsd:element name="save_tModel" type="uddi:save_tModel"/>
```

```
      <xsd:complexType name="save_tModel">
       <xsd:sequence>
           <xsd:element ref="uddi:authInfo"/>
           <xsd:element ref="uddi:tModel" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="uddi:uploadRegister" minOccurs="0"
maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
      </xsd:complexType>

      <!-- Types and elements for response messages -->
      <xsd:element name="bindingDetail" type="uddi:bindingDetail"/>
      <xsd:complexType name="bindingDetail">
       <xsd:sequence>
           <xsd:element ref="uddi:bindingTemplate" minOccurs="0"
maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
       <xsd:attribute name="operator" type="string" use="required"/>
       <xsd:attribute name="truncated" type="uddi:truncated"
use="optional"/>
      </xsd:complexType>
      <xsd:element name="tModelDetail" type="uddi:tModelDetail"/>
      <xsd:complexType name="tModelDetail">
       <xsd:sequence>
           <xsd:element ref="uddi:tModel" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="generic" type="string" use="required"/>
       <xsd:attribute name="operator" type="string" use="required"/>
       <xsd:attribute name="truncated" type="uddi:truncated"
use="optional"/>
      </xsd:complexType>

…
</xsd:schema>
```

Apêndice B

Interfaces das Principais Classes da Extensão do *jUDDI*

Este apêndice apresenta as interfaces das principais classes que implementam a extensão do UDDI. As interfaces apresentadas são baseadas na implementação *Java* do padrão UDDI chamada *jUDDI*, desenvolvida pela *Apache Software Foundation*. Este apêndice está estruturado como segue:

- Seção B.1: inclui a classe que representa um repositório UDDI. Inclui também a classe que implementa uma conexão JDBC para o banco de dados do UDDI;
- Seção B.2: apresenta as classes que manipulam o tipo de estrutura de dados *bindingTemplate*;
- Seção B.3: apresenta as classes que manipulam o tipo de estrutura de dados *qosPolicy*;
- Seção B.4: mostra as classes que manipulam a requisição de descoberta de serviços;
- Seção B.5: mostra as classes que manipulam a requisição de publicação de serviços;
- Seção B.6: mostra as classes que manipulam a requisição de atualização de políticas.

B.1. *RegistryEngine* e *JDBCDataStore*

```
/* UDDI registry engine. */
public class RegistryEngine
{
      /* Executes the required function. */
      public RegistryObject execute(RegistryObject registryObject)
       throws RegistryException;
…
}
```

```
/* JDBC connection. */
public class JDBCDataStore
{
      /* Service discovery. */
      public Vector findBinding(String serviceKey, CategoryBag
categoryBag, TModelBag tModelBag, FindQualifiers findQualifiers,
QosPolicy qosPolicy)
       throws RegistryException

      /* Service information retrieval. */
      public BindingTemplate fetchBinding(String bindingKey)
       throws RegistryException

      /* Service publication. */
      public void saveBinding(BindingTemplate bindingTemplate)
       throws RegistryException

      /* Policy update. */
      public void updateQos(String bindingKey, QosPolicy qosPolicy)
       throws RegistryException

      /* Service removal. */
      public void deleteBinding(String bindingKey)
       throws RegistryException
…
}


/* QOS_POLICY table. */
class QosPolicyTable
{
      /* Inserts a row into the QOS_POLICY table (an attribute). */
      public static void insert(String bindingKey, Vector
keyedReferences, Connection connection)
       throws SQLException

      /* Selects all rows from the QOS_POLICY table (attributes) for a
given binding key. */
      public static Vector select(String bindingKey, Connection
connection)
       throws SQLException

      /* Deletes multiple rows from the QOS_POLICY table (attributes)
that are assigned to a given binding key. */
      public static void delete(String bindingKey, Connection connection)
       throws SQLException

      /* Updates a row of the QOS_POLICY table (an attribute). */
      public static void update(String bindingKey, String keyName, String
keyValue, Connection connection)
       throws SQLException
…
}
```

```
/* Service discovery request using QoS policy. */
class FindBindingByQosQuery
{
      /* SQL statement to select binding templates using QoS policy. */
      public static Vector select(String serviceKey, QosPolicy qosPolicy,
Vector keysIn, FindQualifiers findQualifiers, Connection connection)
       throws SQLException

      /* Constructs the WHERE clause. */
      private static void appendWhere(DynamicQuery sql, String
serviceKey, QosPolicy qosPolicy, FindQualifiers findQualifiers)

      /* Constructs the IN clause. */
      private static void appendIn(DynamicQuery sql, Vector keysIn)

      /* Constructs the ORDER BY clause. */
      private static void appendOrderBy(DynamicQuery sql, FindQualifiers
findQualifiers)
…
}
```

B.2. *BindingTemplate*

```
/* bindingTemplate data structure type. */
public class BindingTemplate
{
      /* Inserts a QoS attribute into the QoS policy of the binding
template. */
      public void addQos(KeyedReference keyedReference)

      /* Retrieves the QoS policy of the binding template. */
      public QosPolicy getQosPolicy()

      /* Sets the QoS policy of the binding template. */
      public void setQosPolicy(QosPolicy qosPolicy)
…
}


/* XML handler for the bindingTemplate structure. */
public class BindingTemplateHandler
{
      /* Maps the bindingTemplate XML structure to the Java object. */
      public RegistryObject unmarshal(Element element)

      /* Maps the BindingTemplate Java object to the XML structure. */
      public void marshal(RegistryObject registryObject, Element element)
…
}
```

B.3. *QosPolicy*

```
/* qosPolicy data structure type. */
public class QosPolicy
{
      /* Inserts a QoS attribute. */
      public void addKeyedReference(KeyedReference keyedReference)

      /* Sets the QoS attribute set. */
      public void setKeyedReferenceVector(Vector keyedReferenceVector)

      /* Retrieves the QoS attribute set. */
      public Vector getKeyedReferenceVector()

      /* Returns the number of QoS attributes. */
      public int size()
…
}


/* XML handler for the qosPolicy structure. */
public class QosPolicyHandler
{
      /* Maps the qosPolicy XML structure to the Java object. */
      public RegistryObject unmarshal(Element element)

      /* Maps the qospolicy Java object to the XML structure. */
      public void marshal(RegistryObject registryObject, Element element)
…
}
```

B.4. *FindBinding*

```
/* find_binding request. */
public class FindBinding
{
      /* Adds a QoS attribute to the QoS policy argument. */
      public void addQos(KeyedReference keyedReference)

      /* Retrieves the QoS policy argument. */
      public QosPolicy getQosPolicy()

      /* Sets the QoS policy argument. */
      public void setQosPolicy(QosPolicy qosPolicy)
…
}
```

```
/* Database connection for the find_binding request. */
public class FindBindingFunction
{
      /* Gets a JDBCDataStore instance. Validates the find_binding
request parameters. Invokes the JDBCDataStore findBinding operation.
Creates a BindingDetail instance with the result. */
      public RegistryObject execute(RegistryObject registryObject)
       throws RegistryException
…
}


/* XML handler for the find_binding request. */
public class FindBindingHandler
{
      /* Maps the find_binding XML request to the Java object. */
      public RegistryObject unmarshal(Element element)

      /* Maps the FindBinding Java object to the XML structure. */
      public void marshal(RegistryObject registryObject, Element element)
…
}
```

B.5. *SaveBinding*

```
/* save_binding request. */
public class SaveBinding
{
      /* Adds a binding template to the binding template set argument. */
      public void addBindingTemplate(BindingTemplate bindingTemplate)

      /* Retrieves the binding template set argument. */
      public Vector getBindingTemplateVector()

      /* Sets the binding template set argument. */
      public void setBindingTemplateVector(Vector bindingTemplateVector)
…
}


/* Database connection for the save_binding request. */
public class SaveBindingFunction
{
      /* Gets a JDBCDataStore instance. Validates the save_binding
request parameters. Invokes the JDBCDataStore saveBinding operation.
Creates a BindingDetail instance with the argument. */
      public RegistryObject execute(RegistryObject registryObject)
       throws RegistryException
…
}
```

```
/* XML handler for the save_binding request. */
public class SaveBindingHandler
{
      /* Maps the save_binding XML request to the Java object. */
      public RegistryObject unmarshal(Element element)

      /* Maps the SaveBinding Java object to the XML structure. */
      public void marshal(RegistryObject registryObject, Element element)
…
}
```

B.6. *UpdateQos*

```
/* update_qos request. */
public class UpdateQos
{
      /* Sets the binding key argument. */
      public void setBindingKey(String bindingKey)

      /* Sets the QoS policy argument. */
      public void setQosPolicy(QosPolicy qosPolicy)

      /* Sets the attribute name. */
      public void setKeyName(String keyName)

      /* Sets the attribute value. */
      public void setKeyValue(String keyValue)

      /* Sets the authentication information argument. */
      public void setAuthInfo(AuthInfo authInfo)

      /* Gets the binding key argument. */
      public String getBindingKey()

      /* Gets the QoS policy argument. */
      public QosPolicy getQosPolicy()

      /* Gets the attribute name. */
      public String getKeyName()

      /* Gets the attribute value. */
      public String getKeyValue()

      /* Gets the authentication information argument. */
      public AuthInfo getAuthInfo()
…
}
```

```
/* Database connection for the update_qos request. */
public class UpdateQosFunction
{
      /* Gets a JDBCDataStore instance. Validates the update_qos
parameters. Invokes the JDBCDataStore updateQos operation. Creates a
DispositionReport instance with an indication of the operation success.
*/
      public RegistryObject execute(RegistryObject registryObject)
       throws RegistryException
…
}


/* XML handler for the update_qos request. */
public class UpdateQosHandler
{
      /* Maps the update_qos XML request to the Java object. */
      public RegistryObject unmarshal(Element element)

      /* Maps the UpdateQos Java object to the XML structure. */
      public void marshal(RegistryObject registryObject, Element element)
…
}
```

Apêndice C

Interfaces das Principais Classes da Extensão do *UDDI4J*

Este apêndice apresenta as interfaces das principais classes que implementam a extensão de uma API cliente *Java* para o repositório UDDI. As interfaces apresentadas são baseadas na implementação *UDDI4J* desenvolvida pelas empresas HP, IBM e SAP. Este apêndice está estruturado como segue:

- Seção C.1: inclui a classe *UDDIProxy* que representa as ações que podem ser realizadas no repositório UDDI;
- Seção C.2: apresenta a classe que representa o tipo de estrutura de dados *bindingTemplate*;
- Seção C.3: apresenta a classe que representa o tipo de estrutura de dados *qosPolicy*;
- Seção C.4: mostra a classe que representa a requisição de descoberta de serviços;
- Seção C.5: mostra a classe que representa a requisição de publicação de serviços;
- Seção C.6: mostra a classe que representa a requisição de atualização de políticas.

C.1. *UDDIProxy*

```
/* UDDI registry operations. */
public class UDDIProxy
{
      /* Service discovery. */
      public BindingDetail find_binding(FindQualifiers findQualifiers,
String serviceKey, TModelBag tModelBag, int maxRows, QosPolicy qosPolicy)
       throws UDDIException, TransportException

      /* Service information retrieval. */
      public BindingDetail get_bindingDetail(String bindingKey)
       throws UDDIException, TransportException
```

```
      /* Service publication. */
      public BindingDetail save_binding(String authInfo, Vector
bindingTemplates)
       throws UDDIException, TransportException

      /* Policy update. */
      public DispositionReport update_qos(String authInfo, String
bindingKey, QosPolicy qosPolicy)
       throws UDDIException, TransportException

      /* Service removal. */
      public DispositionReport delete_binding(String authInfo, String
bindingKey)
       throws UDDIException, TransportException
…
}
```

C.2. *BindingTemplate*

```
/* bindingTemplate data structure type. */
public class BindingTemplate
{
      /* Constructs a BindingTemplate object including a QoS policy. */
      public BindingTemplate(String bindingKey, TModelInstanceDetails
tModelInstanceDetails, AccessPoint accessPoint, QosPolicy qosPolicy)

      /* Constructs a BindingTemplate Java object from a received UDDI
message. */
      public BindingTemplate(Element element)
       throws UDDIException

      /* Inserts a QoS attribute into the QoS policy of the binding
template. */
      public void addQos(KeyedReference keyedReference)

      /* Retrieves the QoS policy of the binding template. */
      public QosPolicy getQosPolicy()

      /* Sets the QoS policy of the binding template. */
      public void setQosPolicy(QosPolicy qosPolicy)

      /* Serializes a BindingTemplate Java object to a XML structure to
send a UDDI message. */
      public void saveToXML(Element element)
…
}
```

C.3. *QosPolicy*

```
/* qosPolicy data structure type. */
public class QosPolicy
{
      /* Constructs a QosPolicy Java object from a received UDDI message.
*/
      public QosPolicy(Element element)
       throws UDDIException

      /* Inserts a QoS attribute. */
      public void add(KeyedReference keyedReference)

      /* Removes a QoS attribute. */
      public boolean remove(KeyedReference keyedReference)

      /* Retrieves a QoS attribute. */
      public KeyedReference get(int index)

      /* Sets the QoS attribute set. */
      public void setKeyedReferenceVector(Vector keyedReferenceVector)

      /* Retrieves the QoS attribute set. */
      public Vector getKeyedReferenceVector()

      /* Returns the number of QoS attributes. */
      public int size()

      /* Serializes a QosPolicy Java object to a XML structure to send a
UDDI message. */
      public void saveToXML(Element element)
…
}
```

C.4. *FindBinding*

```
/* find_binding request. */
public class FindBinding
{
      /* Constructs a FindBinding object including a QoS policy. */
      public FindBinding(String serviceKey,
      TModelBag tModelBag, CategoryBag categoryBag, QosPolicy qosPolicy)

      /* Constructs a FindBinding Java object from a received UDDI
message. */
      public FindBinding(Element element)
       throws UDDIException

      /* Retrieves the QoS policy argument. */
      public QosPolicy getQosPolicy()
```

```
      /* Sets the QoS policy argument. */
      public void setQosPolicy(QosPolicy qosPolicy)

      /* Serializes a FindBinding Java object to a XML structure to send
a UDDI message. */
      public void saveToXML(Element element)
…
}
```

C.5. *SaveBinding*

```
/* save_binding request. */
public class SaveBinding
{
      /* Constructs a SaveBinding Java object from a received UDDI
message. */
      public SaveBinding(Element element)
       throws UDDIException

      /* Retrieves the binding template set argument. */
      public Vector getBindingTemplateVector()

      /* Sets the binding template set argument. */
      public void setBindingTemplateVector(Vector bindingTemplateVector)

      /* Serializes a SaveBinding Java object to a XML structure to send
a UDDI message. */
      public void saveToXML(Element element)
…
}
```

C.6. *UpdateQos*

```
/* update_qos request. */
public class UpdateQos
{
      /* Constructs an UpdateQos object with the required fields. */
      public UpdateQos(String authInfo, String bindingKey, QosPolicy
qosPolicy)

      /* Constructs an UpdateQos Java object from a received UDDI
message. */
      public UpdateQos(Element element)
       throws UDDIException

      /* Sets the binding key argument. */
      public void setBindingKey(String bindingKey)

      /* Sets the QoS policy argument. */
      public void setQosPolicy(QosPolicy qosPolicy)
```

```
      /* Sets the attribute name. */
      public void setKeyName(String keyName)

      /* Sets the attribute value. */
      public void setKeyValue(String keyValue)

      /* Sets the authentication information argument. */
      public void setAuthInfo(AuthInfo authInfo)

      /* Gets the binding key argument. */
      public String getBindingKey()

      /* Gets the QoS policy argument. */
      public QosPolicy getQosPolicy()

      /* Gets the attribute name. */
      public String getKeyName()

      /* Gets the attribute value. */
      public String getKeyValue()

      /* Gets the authentication information argument. */
      public AuthInfo getAuthInfo()

      /* Gets a String object with the authentication information
argument. */
      public String getAuthInfo()

      /* Serializes an UpdateQos Java object to a XML structure to send a
UDDI message. */
      public void saveToXML(Element element)
…
}
```

Apêndice D

Script MySQL para a Criação do Banco de Dados para a Extensão do *jUDDI*

```
CREATE TABLE BUSINESS_SERVICE
(
  SERVICE_KEY VARCHAR(41) NOT NULL,
  LAST_UPDATE TIMESTAMP NOT NULL,
  PRIMARY KEY (SERVICE_KEY),
);

CREATE TABLE SERVICE_DESCR
(
  SERVICE_KEY VARCHAR(41) NOT NULL,
  SERVICE_DESCR_ID INT NOT NULL,
  LANG_CODE VARCHAR(5) NULL,
  DESCR VARCHAR(255) NOT NULL,
  PRIMARY KEY (SERVICE_KEY,SERVICE_DESCR_ID),
  FOREIGN KEY (SERVICE_KEY)
    REFERENCES BUSINESS_SERVICE (SERVICE_KEY)
);

CREATE TABLE BINDING_TEMPLATE
(
  SERVICE_KEY VARCHAR(41) NOT NULL,
  BINDING_KEY VARCHAR(41) NOT NULL,
  ACCESS_POINT_TYPE VARCHAR(20) NULL,
  ACCESS_POINT_URL VARCHAR(255) NULL,
  HOSTING_REDIRECTOR VARCHAR(255) NULL,
  LAST_UPDATE TIMESTAMP NOT NULL,
  PRIMARY KEY (BINDING_KEY),
  FOREIGN KEY (SERVICE_KEY)
    REFERENCES BUSINESS_SERVICE (SERVICE_KEY)
);
```

```
CREATE TABLE BINDING_DESCR
(
  BINDING_KEY VARCHAR(41) NOT NULL,
  BINDING_DESCR_ID INT NOT NULL,
  LANG_CODE VARCHAR(5) NULL,
  DESCR VARCHAR(255) NOT NULL,
  PRIMARY KEY (BINDING_KEY,BINDING_DESCR_ID),
  FOREIGN KEY (BINDING_KEY)
    REFERENCES BINDING_TEMPLATE (BINDING_KEY)
);

CREATE TABLE QOS_POLICY
(
  BINDING_KEY VARCHAR(41) NOT NULL,
  QOS_ID INT NOT NULL,
  TMODEL_KEY_REF VARCHAR(41) NOT NULL,
  KEY_NAME VARCHAR(255) NULL,
  KEY_VALUE VARCHAR(255) NOT NULL,
  PRIMARY KEY (BINDING_KEY,QOS_ID),
  FOREIGN KEY (BINDING_KEY)
    REFERENCES BINDING_TEMPLATE (BINDING_KEY)
);

CREATE TABLE TMODEL_INSTANCE_INFO
(
  BINDING_KEY VARCHAR(41) NOT NULL,
  TMODEL_INSTANCE_INFO_ID INT NOT NULL,
  TMODEL_KEY VARCHAR(41) NOT NULL,
  OVERVIEW_URL VARCHAR(255) NULL,
  INSTANCE_PARMS VARCHAR(255) NULL,
  PRIMARY KEY (BINDING_KEY,TMODEL_INSTANCE_INFO_ID),
  FOREIGN KEY (BINDING_KEY)
    REFERENCES BINDING_TEMPLATE (BINDING_KEY)
);

CREATE TABLE TMODEL_INSTANCE_INFO_DESCR
(
  BINDING_KEY VARCHAR(41) NOT NULL,
  TMODEL_INSTANCE_INFO_ID INT NOT NULL,
  TMODEL_INSTANCE_INFO_DESCR_ID INT NOT NULL,
  LANG_CODE VARCHAR(5) NULL,
  DESCR VARCHAR(255) NOT NULL,
  PRIMARY KEY
    (BINDING_KEY,TMODEL_INSTANCE_INFO_ID,TMODEL_INSTANCE_INFO_DESCR_ID),
  FOREIGN KEY (BINDING_KEY,TMODEL_INSTANCE_INFO_ID)
    REFERENCES TMODEL_INSTANCE_INFO (BINDING_KEY,TMODEL_INSTANCE_INFO_ID)
);
```

```
CREATE TABLE TMODEL
(
  TMODEL_KEY VARCHAR(41) NOT NULL,
  AUTHORIZED_NAME VARCHAR(255) NOT NULL,
  PUBLISHER_ID VARCHAR(20) NULL,
  OPERATOR VARCHAR(255) NOT NULL,
  NAME VARCHAR(255) NOT NULL,
  OVERVIEW_URL VARCHAR(255) NULL,
  DELETED VARCHAR(5) NULL,
  LAST_UPDATE TIMESTAMP NOT NULL,
  PRIMARY KEY (TMODEL_KEY)
);

CREATE TABLE TMODEL_DESCR
(
  TMODEL_KEY VARCHAR(41) NOT NULL,
  TMODEL_DESCR_ID INT NOT NULL,
  LANG_CODE VARCHAR(5) NULL,
  DESCR VARCHAR(255) NOT NULL,
  PRIMARY KEY (TMODEL_KEY,TMODEL_DESCR_ID),
  FOREIGN KEY (TMODEL_KEY)
    REFERENCES TMODEL (TMODEL_KEY)
);

CREATE TABLE AUTH_TOKEN
(
  AUTH_TOKEN VARCHAR(51) NOT NULL,
  PUBLISHER_ID VARCHAR(20) NOT NULL,
  PUBLISHER_NAME VARCHAR(255) NOT NULL,
  CREATED TIMESTAMP NOT NULL,
  LAST_USED TIMESTAMP NOT NULL,
  NUMBER_OF_USES INT NOT NULL,
  TOKEN_STATE INT NOT NULL,
  PRIMARY KEY (AUTH_TOKEN)
);

…
```

Apêndice E

Interfaces WSDL dos Mediadores e Monitores

Este apêndice apresenta as interfaces WSDL dos componentes arquiteturais desenvolvidos para a avaliação do modelo proposto. Este apêndice está estruturado como segue:

- Seção E.1: apresenta uma interface de um mediador para a publicação de serviços;
- Seção E.2: mostra uma interface de um mediador para a descoberta de serviços;
- Seção E.3: apresenta uma interface de um monitor para a atualização de asserções de tempo de resposta em políticas.

E.1. *PBroker*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ArchitectureComponents"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="urn:ArchitectureComponents"
    xmlns:intf="urn:ArchitectureComponents"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <wsdl:message name="runResponse">
      <wsdl:part name="runReturn" type="soapenc:string"/>
   </wsdl:message>

   <wsdl:message name="runRequest">
      <wsdl:part name="tModelKey" type="soapenc:string"/>
      <wsdl:part name="pPolicy" type="soapenc:string"/>
      <wsdl:part name="serviceKey" type="soapenc:string"/>
      <wsdl:part name="url" type="soapenc:string"/>
      <wsdl:part name="protocol" type="soapenc:string"/>
   </wsdl:message>
```

```
   <wsdl:portType name="PBroker">
      <wsdl:operation name="run"
         parameterOrder="tModelKey pPolicy serviceKey url protocol">
         <wsdl:input message="impl:runRequest" name="runRequest"/>
         <wsdl:output message="impl:runResponse" name="runResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="PBrokerSoapBinding" type="impl:PBroker">
      <wsdlsoap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="run">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="runRequest">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="runResponse">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="PBrokerService">
      <wsdl:port binding="impl:PBrokerSoapBinding" name="PBroker">
         <wsdlsoap:address location=
    "http://garnize:8080/axis/PBroker.jws"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

E.2. *DBroker*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ArchitectureComponents"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="urn:ArchitectureComponents"
    xmlns:intf="urn:ArchitectureComponents"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <wsdl:message name="runResponse">
      <wsdl:part name="runReturn" type="soapenc:string"/>
   </wsdl:message>
```

```
   <wsdl:message name="runRequest">
      <wsdl:part name="tModelKey" type="soapenc:string"/>
      <wsdl:part name="cPolicy" type="soapenc:string"/>
   </wsdl:message>

   <wsdl:portType name="DBroker">
      <wsdl:operation name="run" parameterOrder="tModelKey cPolicy">
         <wsdl:input message="impl:runRequest" name="runRequest"/>
         <wsdl:output message="impl:runResponse" name="runResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="DBrokerSoapBinding" type="impl:DBroker">
      <wsdlsoap:binding style="rpc" transport=
    "http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="run">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="runRequest">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="runResponse">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="DBrokerService">
      <wsdl:port binding="impl:DBrokerSoapBinding" name="DBroker">
         <wsdlsoap:address location=
    "http://garnize:8080/axis/DBroker.jws"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

E.3. *Monitor*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ArchitectureComponents"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="urn:ArchitectureComponents"
    xmlns:intf="urn:ArchitectureComponents"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://axis.apache.org"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
   <wsdl:message name="invokeResponse" />

   <wsdl:message name="invokeRequest">
      <wsdl:part name="msgContext" type="xsd:anyType"/>
   </wsdl:message>

   <wsdl:portType name="Monitor">
      <wsdl:operation name="invoke" parameterOrder="msgContext">
         <wsdl:input message="impl:invokeRequest" name="invokeRequest"/>
         <wsdl:output message="impl:invokeResponse" name=
    "invokeResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="MonitorSoapBinding" type="impl:Monitor">
      <wsdlsoap:binding style="rpc" transport=
    "http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="invoke">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="invokeRequest">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="invokeResponse">
            <wsdlsoap:body encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" namespace=
    "urn:ArchitectureComponents" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="MonitorService">
      <wsdl:port binding="impl:MonitorSoapBinding" name="Monitor">
         <wsdlsoap:address location=
    "http://garnize:8080/axis/services/Monitor"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```