Uma Abordagem para Viabilização de Serviços e Tarifação de Componentes de Software

Anderson Souza Marinho

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por:

_____
Anderson Souza Marinho

Aprovado por:

_____
Profª Cláudia Maria Lima Werner, D.Sc.
(Presidente)

_____
Prof. Leonardo Gresta Paulino Murta, D.Sc.
(Co-orientador)

_____
Profª Vanessa Braganholo Murta, D.Sc.

_____
Prof. Antonio Carlos Gay Thomé, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2008

Agradecimentos

A Deus, por ter sido sempre generoso comigo em todos os momentos da minha vida. Mesmo nos momentos difíceis, eu sei que você está comigo e está fazendo o melhor por mim. Por isso, estarei sempre agradecido.

Aos meus pais Lydia Maria e Luiz Antônio, por estarem sempre presentes na minha formação com a maior atenção do mundo e amor. Cada um teve um papel fundamental para eu estar aqui hoje. Esta monografia é dedicada a vocês dois.

A toda minha família que me ajudaram direta ou indiretamente para eu estar presente aqui. Em especial as minhas tias Cirene, Cirlei e Dalva e tios Jorge, José Clementino e Roberto, que foram essenciais no começo da minha graduação.

Aos amigos que eu adquiri na faculdade, fico muito feliz de tê-los conhecido. Eu aprendi muito com essas pessoas. Em especial, Eldanae, Rafael, Hua Lin e Cláudio.

Aos meus amigos em geral que tiveram contribuição neste trabalho.

Aos meus orientadores Claudia Werner e Leonardo Murta, que tiveram a maior paciência do mundo para me ensinar e mostrar o melhor caminho. Eu aprendi muito com vocês.

Aos integrantes da minha banca, Vanessa Braganholo e Antonio Carlos Thomé, que, junto com os meus orientadores, são os meus ícones como professores.

Resumo

Uma Abordagem para Viabilização de Serviços e Tarifação de Componentes de Software

Anderson Souza Marinho

Orientadores: Cláudia Maria Lima Werner e Leonardo Gresta Paulino Murta

Biblioteca de componentes é uma ferramenta que dá suporte ao Desenvolvimento Baseado em Componentes, provendo um lugar para a publicação, o armazenamento, a busca e a recuperação de componentes de software.

Nos últimos tempos, serviços vêm se tornando uma nova alternativa para a construção de aplicações, devido a sua característica de interoperabilidade, tornando seu uso possível em qualquer plataforma. Com isso, uma biblioteca de componentes precisa evoluir de um modelo de disponibilização de componentes apenas físicos para um modelo de disponibilização de serviços de componentes também. Junto a isso, outro ponto que precisa ser levado em consideração em uma biblioteca de componentes está relacionado à tarifação. Componentes e serviços precisam ser tarifados para que uma biblioteca possa ser aplicada em um sentido comercial.

Este trabalho apresenta uma infra-estrutura de serviços de componente em uma biblioteca de componentes, com mecanismos de controle de acesso e mecanismos automáticos de geração de serviços a partir de componentes previamente publicados. Este trabalho, também, apresenta um solução para tarifação de componentes, considerando duas possíveis formas de aquisição/utilização: *download* de componente e utilização de serviços de componentes.

Abstract

An Approach to Enabling Software Component Services and Pricing

Anderson Souza Marinho

Advisors: Cláudia Maria Lima Werner and Leonardo Gresta Paulino Murta

Components Library is a tool that supports Component Based Development, providing one place to publish, store, search, and retrieve components of software.

Nowadays, services are becoming a new alternative to construct applications due their interoperability feature, making their use possible in any platform. So, a component library needs to evolve from providing components only physically to provide component services as well. With that, another point of concern in a component library is about pricing, component and services need to be priced to make the library available in a commercial scenario.

This work presents an infrastructure of component services in a component library, with automatic web service generation from previously published components and access control mechanisms. This work also presents a solution to component pricing in two possible acquisition/utilization ways: component download and component services utilization.

Índice

Índice de Figuras

Capítulo 1 - Introdução

1.1 ***Preâmbulo***

A Reutilização de Software é a disciplina responsável pela criação de sistemas de software a partir de software preexistente (KRUEGER, 1992). Diferentemente da reutilização *ad hoc*, que usualmente se concretiza por meio de cópia de trechos de artefatos preexistentes, a disciplina de Reutilização de Software visa sistematizar e difundir práticas de reutilização na organização.

Diversos paradigmas podem ser considerados como iniciativas para sistematizar a reutilização. Um dos mais conhecidos é o de orientação a objeto, que conseguiu, de certa forma, atingir bons resultados. Entretanto, devido a sua unidade de encapsulamento de dados (ou seja, classe) ser muito fina, dificultava na substituição ou reutilização sistematizada de partes dos sistemas (MURTA, 2006). Para resolver este problema, surge o paradigma de Desenvolvimento Baseado em Componentes (DBC), que tenta contornar algumas das deficiências detectadas na orientação a objetos com relação à reutilização de software (PAGE-JONES, 1999).

A unidade de encapsulamento do DBC (ou seja, componente) é mais grossa e isto acarreta um menor acoplamento entre os componentes e uma representação mais coesa do próprio componente (MURTA, 2006). Além disso, esses acoplamentos são feitos através de interfaces e conectores, que é uma forma de isolar aspectos implementacionais do componente ao mundo externo. Ou seja, um componente torna-se uma caixa-preta, facilitando o processo de substituição de partes constituintes de sistemas. Fora os aspectos estruturais, de acordo com JACOBSON *et al*. (1997), o DBC envolve aspectos metodológicos que colocam a reutilização em destaque no processo de desenvolvimento de sistemas.

Com o passar dos tempos, sistemas maiores e mais complexos são especificados e pequenos prazos de desenvolvimento desses sistemas são exigidos. Para atender essa demanda, o número de componentes dentro de uma organização aumenta para realizar a prática de DBC com mais eficiência e agilidade. Entretanto, com o número elevado de componentes em uma organização, a dificuldade de gerenciamento sobre esses componentes aumenta consideravelmente, tornando-se quase inviável gerenciá-los de

uma maneira *ad hoc*. Com isso, se torna necessário uma abordagem sistematizada para o controle destes componentes e o uso de ferramentas de apoio para que se alcance melhores índices de reutilização. Uma dessas ferramentas é uma biblioteca de componentes, que consiste em um local de publicação, armazenamento, busca e recuperação de componentes, de forma que possam ser aplicados no desenvolvimento de novos sistemas (WERNER, 2000).

1.2 Motivação

Serviços web (W3C, 2002) vêm aumentando o seu uso dia após dia e está levando pessoas importantes no mundo da computação como, por exemplo, o presidente da Sun Microsystems, Jonathan Schwartz, a considerarem esta tecnologia como o futuro da computação. Schwartz acredita que o processamento não será mais feito localmente, mas sim comprado através de serviços oferecidos pela Web, como se fosse um serviço de eletricidade ou telefone (SCHWARTZ, 2008).

A Sun acredita tanto nesta idéia que já lançou o serviço Sun Grid (SUN, 2008a), que funciona como supercomputador sob demanda que disponibiliza serviços na internet, que podem ser acessados por um navegador e serem pagos com cartão de crédito. Além do SunGrid, a Sun está investindo na parte de sistemas operacionais, com o lançamento do Solaris 10 (SUN, 2008b), para dar apoio na construção de serviços web seguros. A Google também é outro exemplo de empresa que acredita nessa idéia, disponibilizando diversos tipos de serviços na Web (GOOGLE, 2008).

Neste contexto, componentes de software, que são comumente utilizados localmente, vêm sendo utilizados também a partir de serviços, que são acessados remotamente, sem a necessidade de serem implantados em servidores locais. A partir disso, uma biblioteca de componentes, onde componentes são disponibilizados apenas fisicamente, deve também ser capaz de fornecer os serviços dos componentes.

Além disso, os componentes e seus serviços disponibilizados em uma biblioteca devem ter a opção de serem tarifados. Com isso, é garantido o uso da biblioteca tanto em um sentido não-comercial quanto em um sentido comercial, ou seja, o seu foco de aplicação é ampliado.

Entretanto, para aplicar tarifação sobre os componentes e serviços de uma biblioteca, é necessário que a mesma possua mecanismos de controle de acesso que

restrinja o acesso a usuários autorizados. Com relação aos componentes fornecidos fisicamente pela biblioteca, a aplicação de controle de acesso não é uma tarefa complexa, já que os componentes estão em poder da biblioteca e o acesso a estes só pode ser feito através da biblioteca. No entanto, com relação aos serviços, a aplicação de controle de acesso é uma tarefa mais complexa. Um serviço não é algo concreto que pode ser passado para a biblioteca quando publicado, apenas o seu endereço de acesso é fornecido. Com isso, a biblioteca só participa no processo de divulgação dos serviços. Porém, quando um serviço é requisitado, a biblioteca não tem conhecimento deste evento, já que a requisição do serviço é feita diretamente ao endereço de acesso divulgado, que não é necessariamente um endereço vinculado à biblioteca.

Como nem todos os produtores de componente possuem uma infra-estrutura para prover serviços, uma biblioteca poderia arcar com uma infra-estrutura própria para realizar o provimento dos serviços, que podem ser gerados automaticamente a partir dos componentes armazenados na biblioteca.

1.3 Problema

Um dos problemas tratados neste trabalho está relacionado com a forma de disponibilização dos serviços em uma biblioteca. Um serviço pode ser fornecido pelo produtor do componente, que utiliza a biblioteca como meio de divulgação do seu serviço, ou pela própria biblioteca, facilitando o processo de publicação do serviço de um componente, caso o produtor não possua intra-estrutura para fornecer o serviço. Nestes dois casos, o serviço publicado deve estar adaptado para realizar um controle de acesso sobre os usuários, da mesma forma que um *download* de um componente é restringido somente a um usuário autorizado na biblioteca. Entretanto, em cada caso, problemas diferentes surgem para realizar esta tarefa.

No caso onde o usuário fornece o seu próprio serviço, como o serviço não está em poder da biblioteca, a alteração do mesmo torna-se uma tarefa complexa. Já no caso onde a biblioteca gera o serviço a partir do componente, o componente também não possui mecanismos de controle de acesso, que deverão ser gerados pela biblioteca. Além disso, um problema ainda maior, neste caso, é a própria geração automática de um serviço a partir de um componente. Um componente pode ser escrito em qualquer linguagem de programação. No entanto, de acordo com a linguagem que o componente

for escrito, a geração do serviço é tratada de maneira diferente, ou seja, não existe uma solução genérica para a geração de serviço. Logo, como um componente publicado na biblioteca pode estar escrito em qualquer linguagem, a biblioteca deve estar preparada para gerar serviços a partir de qualquer linguagem requisitada. Mais ainda, a solução adotada para este problema deve ser flexível o bastante para sofrer evoluções na sua estrutura para suportar linguagens de programações que surjam no futuro.

A respeito da aplicação de tarifação na biblioteca, um dos problemas a ser tratado está relacionado à forma de precificação de um componente, ou seja, quais são os fatores que compõem o custo de um componente. Isso deve ser tratado de maneira diferente se estiver relacionado a um *download* de um componente ou se estiver relacionado à utilização de um serviço. Por exemplo, no caso do download de um componente, as licenças que o componente oferece implicam em valores diferentes na aquisição do componente. Por outro lado, no caso da definição do custo de um serviço, deve ser levado em consideração questões como infra-estrutura que está sendo usada para disponibilizar o serviço.

1.4 Objetivo

Este trabalho tem dois objetivos principais: o primeiro é definir uma infra-estrutura de serviços em uma biblioteca de componentes. Isto é, componentes, além de serem disponibilizados fisicamente, devem ser também disponibilizados através de serviços. Esta infra-estrutura deve trabalhar de maneira análoga à disponibilização física de componente, com mecanismos de controle de acesso, monitoramento de utilização, etc. Esta infra-estrutura também deve fornecer mecanismos automatizados de geração de serviços a partir dos componentes fisicamente cadastrados na biblioteca.

O segundo objetivo é a aplicação de tarifação em uma biblioteca de componentes. Isto quer dizer, aplicar tarifas a todos os meios de aquisição de um componente, seja por um download ou por utilização de um serviço de um componente. Com isso, este objetivo depende indiretamente do primeiro objetivo, já que o primeiro objetivo deve propiciar uma disponibilização de serviços de maneira controlada, ou seja, através de mecanismos de controle de acesso.

1.5 Contexto

Este trabalho de pesquisa está contextualizado no Projeto Brechó, que tem como objetivo desenvolver uma biblioteca de componentes e serviços de software que possui mecanismos de armazenamento, busca e recuperação (WERNER *et al*., 2007). O conceito de componente na Brechó é bastante amplo, permitindo uma representação flexível que não fique vinculada apenas ao código binário, mas inclua os diferentes artefatos produzidos durante o processo de desenvolvimento do componente como, por exemplo, especificações, código fonte, manuais, etc.

1.6 Organização

Esta monografia está organizada em mais quatro capítulos, além deste capítulo de introdução.

O Capitulo 2 realiza uma revisão da literatura, contextualizando o cenário onde este trabalho está inserido. Além disso, é feita uma análise sobre trabalhos relacionados ao proposto.

O Capítulo 3 apresenta a abordagem proposta por este trabalho, detalhando os problemas a serem tratados e apresentando as soluções adotadas.

O Capítulo 4 discute os detalhes de implementação desta abordagem e realiza uma demonstração do seu funcionamento através de exemplos.

Fechando esta monografia, o Capítulo 5 apresenta a conclusão deste trabalho, listando as contribuições, mostrando as limitações existentes e destacando as oportunidades de novos trabalhos a partir deste projeto.

Capítulo 2 - Revisão da Literatura

2.1 Introdução

Neste capítulo é feita uma descrição dos conceitos e tecnologias utilizadas neste trabalho, além de apresentar alguns trabalhos relacionados ao tema proposto.

A Seção 2.2 faz uma contextualização sobre o tema reutilização de software, destacando uma de suas técnicas, o Desenvolvimento Baseado em Componentes (DBC), e suas ferramentas de apoio. A Seção 2.3 apresenta uma visão geral sobre serviços, definindo conceitos, padrões e tecnologias relacionadas. A Seção 2.4 descreve os trabalhos relacionados a este projeto, apontando suas características em comum e suas diferenças. Dentre estes trabalhos, propostas acadêmicas e ferramentas comerciais são identificadas.

2.2 Reutilização de software

A Reutilização de Software é a disciplina responsável pela criação de sistemas de software a partir de software preexistente (KRUEGER, 1992). A reutilização de software é considerada uma forma de obter melhores indicadores de qualidade e produtividade na construção de sistemas maiores e mais complexos (WERNER, 2000). Dentre as técnicas mais utilizadas de reutilização de software, destaca-se o Desenvolvimento Baseado em Componentes (DBC), que utiliza técnicas e procedimentos para a construção de componentes de software reutilizáveis.

Em organizações onde o número de componentes é elevado, a dificuldade de gerenciamento desses componentes aumenta consideravelmente, tornando-se quase inviável gerenciá-los de uma maneira *ad hoc*. Com isso, se torna necessário uma abordagem sistematizada para o controle destes componentes e o uso de ferramentas de apoio. Uma dessas ferramentas é uma biblioteca de componentes, que consiste em um local de publicação, armazenamento, busca e recuperação de componentes, de forma que possam ser aplicados no desenvolvimento de novos sistemas. A Seção 2.2.1 definirá e detalhará as principais funcionalidades de uma biblioteca de componentes.

2.2.1 Biblioteca de componentes

Uma biblioteca de componentes consiste em um local de publicação, armazenamento, busca e recuperação de componentes de forma que possam ser aplicados em novos sistemas (WERNER, 2000). Estes componentes podem ser compostos, não apenas por código binário, mas também por qualquer tipo de artefato produzido durante o desenvolvimento de software (especificações, código fonte, manuais, etc.).

No contexto de uma biblioteca, existem os papéis de produtores e consumidores. Os produtores são definidos como os responsáveis pela publicação de componentes. Por outro lado, os consumidores são aqueles que adquirem os componentes disponibilizados na biblioteca. Estes papéis, porém, não são exclusivos. Um usuário da biblioteca pode ser um produtor, publicando seu componente, e também um consumidor, no qual adquiriu um componente para a construção do seu próprio.

Os componentes adquiridos na biblioteca são vinculados a contratos que provêm aos consumidores e produtores papéis e responsabilidades bem definidos. Dentre estas definições contratuais, podem ser incluídas questões de tarifação pela reutilização e suporte dos componentes, de propriedade intelectual dos componentes e adaptações, de garantia de qualidade e de manutenção. Ao funcionar como um mecanismo de vinculação entre fornecedores e consumidores, uma biblioteca de componentes é fundamental no controle da evolução dos componentes e na manutenção dos sistemas que os utilizam (envolvendo muitas vezes o controle de versões, requisição de mudanças, novas demandas e defeitos relatados).

Uma biblioteca de componentes diferencia-se de um repositório relativo a processos de Gerência de Configuração (ESTUBLIER, 2000). Repositórios de controle de versão são ferramentas utilizadas para controlar a evolução dos ativos de software. Além disso, dá suporte ao trabalho cooperativo dentro de uma equipe de desenvolvimento de software. Existem diversas ferramentas disponíveis no mercado, dentre elas, destacam-se o CVS (CEDERQVIST, 2003), ClearCase (IBM, 2008a), Subversion (TIGRIS, 2008) e Team Foundation Version Control (MICROSOFT, 2008).

Uma biblioteca de componentes tem como preocupação disponibilizar versões de produção dos componentes de uma organização e não controlar a evolução dos ativos em desenvolvimento na organização, sendo isto preocupação do repositório de controle de versão.

Mesmo com propósitos diferentes, existe uma relação de dependência entre eles. Como, por exemplo, um componente publicado em uma biblioteca está atrelado a uma liberação do repositório de controle de versão.

2.3 Serviços

Serviços são aplicações que tem como característica fundamental a interoperabilidade (SABBOUH, 2001). Independem em qual plataforma é executada, já que na verdade estas aplicações não rodam no usuário final onde foram solicitadas, mas sim em um ambiente preparado onde estão implantadas.

O paradigma de serviços vem sendo considerado como o futuro da computação por muitas pessoas importantes da área. Um exemplo é o presidente da Sun, Jonathan Schwartz (2008), que considera que, em poucos anos, a maioria das aplicações será baseada em serviços. Até mesmo um simples editor de texto será utilizado a partir de serviços, segundo Schwartz. Um exemplo desta transição pode ser visto com os serviços que a Google disponibiliza para seus usuários, que vai desde visualizadores de imagens, calendários e até editores de textos, dentre outros (GOOGLE, 2008).

Nas próximas Seções, é feito um levantamento de alguns conceitos, especificações e tecnologias usadas em serviços. A Seção 2.3.1 descreve sobre XML (*Extensible Markup Language*) (W3C, 2006), a unidade fundamental de descrição amplamente utilizada em serviços. A Seção 2.3.2 descreve a arquitetura de serviços SOA (*Service Oriented Architecture*) (CAREY, 2008). A Seção 2.3.6 apresenta a principal tecnologia baseada na arquitetura SOA, Web Services (W3C, 2002), formada pelo tripé SOAP (*Simple Object Access Protocol*) (W3C, 2007a), responsável pelo protocolo de comunicação, WSDL (*Web Services Description Language*) (W3C, 2007b), responsável pela descrição do serviço, e UDDI (*Universal Description, Discovery, and Integration*) (OASIS, 2005), com seu papel de publicar e buscar informações dos serviços. Apresentados, respectivamente, nas Seções 2.3.3, 2.3.4 e 2.3.5. A Seção 2.3.7 apresenta a ferramenta AXIS (APACHE, 2008), utilizada para gerar servidores e clientes de serviços web. Por fim, a Seção 2.3.8 apresenta *SOA Governance* (WINDLEY, 2006), que consiste em iniciativas de governança de ambientes fundamentados em SOA.

2.3.1 XML

Extensible Markup Language (XML) é uma linguagem de marcação definida pela *World Wide Web Consortium* (W3C, 2008). Ela surgiu devido à insatisfação do consórcio nos formatos existentes (padronizados ou não) na época, sendo definida a partir de um subconjunto do SGML (*Standard Generalized Markup Language*) (ISO, 1986), o que resultou em uma linguagem mais simples, inspirada pelo HTML (*HyperText Markup Language*).

Os dados de um documento XML são estruturados de forma hierárquica, trazendo uma melhor organização do documento. Existe um conjunto de regras de boa formação que qualificam um documento XML como um documento bem-formado. Além disso, cada documento XML pode ser ligado a um esquema que possui regras específicas que devem ser atendidas. Caso o documento não atenda essas regras, este é considerado um documento inválido.

Atualmente, XML é considerada uma das soluções para integração de aplicações, além de ser bastante utilizada para dar suporte à configuração de aplicações. XML é considerada, também, o padrão para comunicação entre aplicações. Tomando como exemplo a tecnologia orientada a serviços, XML se faz presente na maioria das especificações.

2.3.2 SOA

Service Oriented Architecture (SOA) é uma arquitetura que tem como fundamento o desenvolvimento de sistemas distribuídos nos quais suas funcionalidades sejam providas totalmente por serviços, já que serviços são independentes de plataforma, ou seja, não dependem de programas específicos ou linguagem para serem executados (CAREY, 2008). Isto ocorre porque, quando um serviço é solicitado, ele não é executado na máquina do usuário final que o solicitou, mas sim na máquina do provedor do serviço. O único requisito para o usuário final é seguir o protocolo de comunicação do provedor.

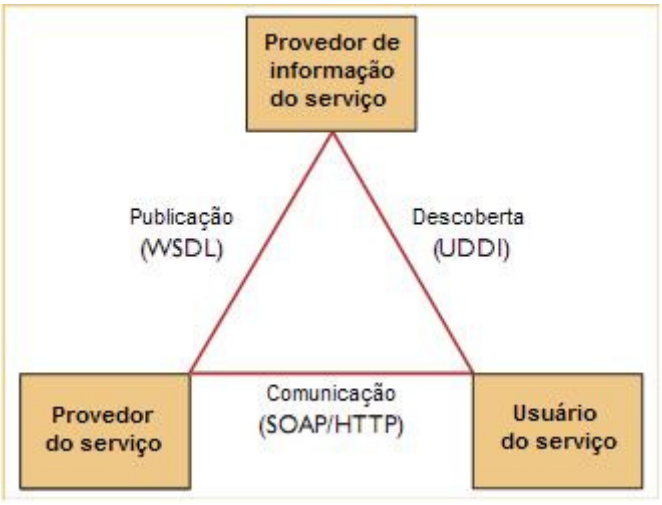


Figura 2.1 - Arquitetura SOA (HUHNS *et al.*, 2005)

A arquitetura SOA, de acordo com a Figura 2.1, é composta de três entidades: O usuário do serviço, o provedor de informação do serviço, e o provedor do serviço. Um provedor de serviço, quando gera um novo serviço, publica o documento de descrição de serviço (geralmente, no formato WSDL) em um provedor de informação do serviço. Este funciona como um intermediador entre o usuário e o provedor do serviço. O usuário, por sua vez, faz consultas para este provedor de informação. A linguagem de consulta mais utilizada é a UDDI. Com a descoberta do serviço, o usuário recupera o documento de descrição do mesmo e realiza a comunicação, geralmente, utilizando o protocolo de mensagem SOAP. Estas mensagens são trocadas entre o usuário e o provedor, geralmente, através de requisições e respostas HTTP (*HyperText Transfer Protocol*) (IETF, 1999). Além do HTTP, um outro protocolo de transporte também utilizado é o SMTP (*Simple Mail Transport Protocol*) (IETF, 2001).

2.3.3 SOAP

SOAP é um protocolo de comunicação e um formato de codificação baseado em XML para comunicação entre aplicações (CURBERA *et al.*, 2002). Foi concebido, originalmente, pela Microsoft e Userland software, e tem passado por várias evoluções. É visto como o principal protocolo de comunicação independente de plataforma e linguagem entre aplicações, adotado como padrão para serviços web. Atualmente, sua especificação está a cargo da W3C.

Como é um protocolo de comunicação entre aplicações, SOAP possui uma estrutura bem definida para transferência de dados. Estes dados são transferidos de uma aplicação para a outra em um formato XML, já que a mensagem SOAP é escrita nesse formato. O protocolo SOAP já tem definido a maioria dos tipos simples que as linguagens de programação usam, tais como: *int*, *float*, *char*, *boolean*, *string*, etc. SOAP também suporta representações de tipos complexos. Um tipo complexo consiste na agregação de tipos simples, como também de outros tipos complexos.

Em trocas de mensagens SOAP entre aplicações, dois processos são bastante conhecidos: o processo de serialização e o processo de deserialização de dados. O processo de serialização, também conhecido como *marshalling*, consiste na transformação dos dados no nível da aplicação para XML quando uma aplicação envia uma mensagem para outra aplicação. Enquanto o processo de deserialização, também conhecido como *unmarshalling*, consiste na transformação dos dados em XML para dados no nível da aplicação quando uma aplicação recebe uma mensagem.

2.3.4 WSDL

Para a execução de Web Services, SOAP oferece a comunicação básica, porém não fornece mecanismos descritivos informando quais mensagens devem ser trocadas para que o serviço seja realizado com sucesso. Esse papel é exercido pelo WSDL (*Web Service Description Language*), um documento baseado no formato XML, desenvolvido pela Microsoft e pela IBM para descrever Web Services como coleções de pontos de comunicação que podem trocar determinadas mensagens (CURBERA *et al*., 2002).

Uma descrição completa de um serviço por um arquivo WSDL é dividida em duas grandes partes: a primeira com uma descrição do serviço no nível de aplicação, e a segunda com os detalhes específicos sobre o protocolo de comunicação que os usuários devem seguir para que consigam acessar o serviço. Esta separação se faz necessária, pois uma mesma aplicação pode ser implantada em diferentes pontos de acesso. Com isso, esta divisão propicia uma melhor representação.

No exemplo da Figura 2.2, são descritas as operações fatorial e seno, que formam um conjunto denominado *ServiçosMatemática*. Na verdade, um serviço, chamado de *portType* no WSDL, é representado como um conjunto de subserviços, chamados de *operations*. Estes subserviços são os serviços reais disponíveis pelo

provedor. Cada *portType* pode ter uma ou mais descrições de protocolos de acesso que são definidas pelo elemento *service* no WSDL. Estas descrições são ligadas com os *portTypes* através de elementos chamados *bindings*. No exemplo, *ServiçosMatemática* possui apenas uma opção de acesso que está definido no elemento *service* com atributo *name* com o valor "MatematicaService". Nele se encontra a descrição do endereço de acesso, que no caso é "http://reuse.cos.ufrj.br/MatematicaServices". Fazendo a ligação entre *ServiçosMatematica* e *MatematicaService*, existe o elemento *binding* com atributo *name* com o valor "MatematicaSoapBinding".

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://reuse.cos.ufrj.br/MatematicaServices"
    xmlns:intf="http://localhost:8080/axis/Matematica.jws"
    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"
    targetNamespace="http://reuse.cos.ufrj.br/MatematicaServices">
  + <wsdl:message name="fatorialRequest">
  + <wsdl:message name="senoRequest">
  + <wsdl:message name="senoResponse">
  + <wsdl:message name="fatorialResponse">
  - <wsdl:portType name="ServiçosMatematica">
    - <wsdl:operation name="fatorial" parameterOrder="n">
        <wsdl:input name="fatorialRequest" message="impl:fatorialRequest" />
        <wsdl:output name="fatorialResponse" message="impl:fatorialResponse" />
      </wsdl:operation>
    - <wsdl:operation name="seno" parameterOrder="i">
        <wsdl:input name="senoRequest" message="impl:senoRequest" />
        <wsdl:output name="senoResponse" message="impl:senoResponse" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="MatematicaSoapBinding" type="impl:ServiçosMatematica">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    + <wsdl:operation name="fatorial">
    + <wsdl:operation name="seno">
    </wsdl:binding>
  - <wsdl:service name="MatematicaService">
    - <wsdl:port name="ServiçosMatematica" binding="impl:MatematicaSoapBinding">
        <wsdlsoap:address location="http://reuse.cos.ufrj.br/MatematicaServices" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Figura 2.2 – Exemplo de documento WSDL

Em uma visão de mais alto nível, este esquema é representado em um diagrama, como está ilustrado na Figura 2.3. Nele estão definidas três entidades principais: a entidadade *portType,* chamada ServiçosMatemática; a entidade *service,* chamada MatemáticaService; e entidade *binding* que realiza a ligação, chamada MatematicaSoapBinding.

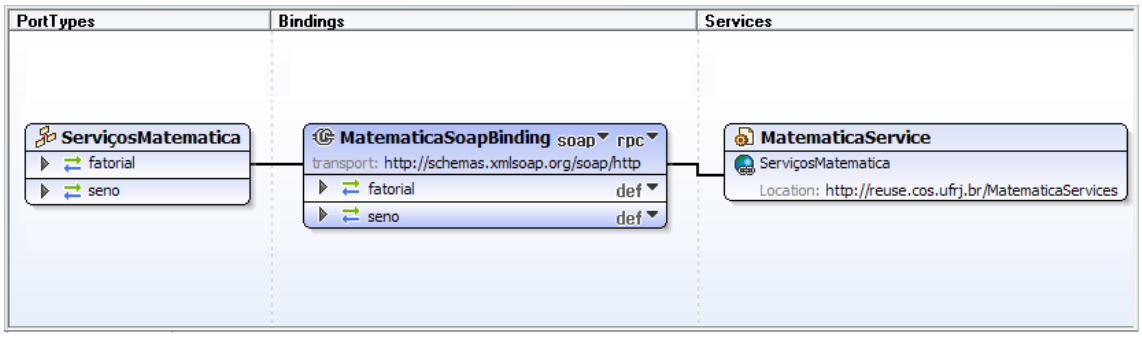


Figura 2.3 - Diagrama do WSDL (ALTOVA, 2008) apresentado na Figura 2.2

2.3.5 UDDI

Com a interface de comunicação definida e um padrão de descrição de como acessar um Web Service, isso seria o suficiente para executá-lo. Porém, para alcançar este estágio, é necessário "descobrir" que este Web Service existe. E é para este propósito que o UDDI foi criado. UDDI é um mecanismo unificado e sistematizado de busca de serviços através de um repositório centralizado de registros de serviços (CURBERA *et al*., 2002).

UDDI provê duas especificações que definem a estrutura e operação de um registro de serviço:

- Uma definição da informação a ser disponibilizada para cada serviço;
- Uma API para consulta e atualização de registros, descrevendo como esta informação pode ser acessada e atualizada.

A estrutura de informação de um registro UDDI, baseada no formato XML, é composta de duas entidades que descrevem o provedor do serviço e os serviços fornecidos. O elemento *businessEntity* possui informações gerais, incluindo identificadores (IDs, números de segurança, etc), informações de contato e uma descrição simples sobre o provedor do serviço. Cada *businessEntity* deve conter um ou mais elementos do tipo *businessService,* que representam os serviços que o provedor fornece. Cada elemento, também, pode adicionar um outro elemento do tipo *categoryBag* para categorizar o provedor ou o serviço. Usando categorização, consultas sobre os registros podem ficar mais apuradas para localizar serviços específicos.

A Figura 2.4 mostra um trecho de um documento UDDI. Na Figura 2.4.a, estão descritas as informações do provedor do serviço, o *Grupo de reutilização*. Na Figura

2.4.b, estão descritas as informações de um serviço, *ServiçoMatemática.* E por último, na Figura 2.4.c, está definida uma categoria, *Matemática*, para categorizar o serviço.

```
<businessEntity businessKey="uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40"
        operator="http://www.reuse.cos.ufrj.br"
        authorizedName="Anderson Marinho">
    <name>Grupo de reutilização</name>
    <description>
        Temos os melhores serviços
    </description>
    <contacts>
        <contact useType="info geral">
            <description>informações gerais</description>
            <personName>Anderson</personName>
            <phone>(21) 2562-8785</phone>
            <email>contatoreuso@cos.ufrj.br</email>
        </contact>
    </contacts>
  <businessServices>
    <businessService serviceKey="uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
        businessKey="uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
     <name>ServiçoMatemática</name>
     <description>Diversas operações matemáticas</description>
     ...
    </businessService>
  </businessServices>
  ...
  <categoryBag>
     <keyedReference
        tModelKey="UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
        name="Matemática"
        value="111336"/>
   </categoryBag>
</businessEntity>
```

a) b) c)

Figura 2.4 - Trecho de um documento UDDI

2.3.6 Web Service

A W3C define Web Service, ou serviço web, como "sistema de software designado para suportar interações entre máquinas com plataformas diferentes (interoperabilidade) em uma rede." (BOOTH *et al*., 2004).

A definição da W3C sobre serviço web é muito abrangente, mas comumente essa definição refere-se a clientes e servidores que se comunicam usando mensagens XML que seguem o padrão SOAP.

Um serviço web consiste em um conjunto de especificações, porém seu núcleo está definido pelo trio SOAP, WSDL e UDDI. A especificação SOAP fica encarregada com o protocolo de comunicação entre requisitante e provedor do serviço. Já a especificação WSDL define o padrão de descrição de serviço. E, por fim, a

especificação UDDI estabelece a forma de publicação e busca de informações sobre os serviços.

Existem várias outras especificações adicionais desenvolvidas, ou em desenvolvimento, que estendem as capacidades de serviços web (CURBERA *et al*., 2002). Dentre as mais conhecidas estão:

- WS-Security: Define como usar assinaturas e dados criptografados em XML no uso de SOAP para troca de mensagens seguras, usando protocolos seguros de transporte como o HTTPS.
- WS-Reliability: Protocolo para troca de mensagens confiáveis entre dois Web Services.
- WS-Addressing: Um meio de descrever o endereço do destinatário e o remetente de uma mensagem, dentro da própria mensagem SOAP.
- WS-Transaction: Uma maneira de utilizar transações em serviços.

2.3.7 AXIS

AXIS é um *framework* da Apache para a construção de processadores SOAP, tais como clientes, servidores, etc. Feito para aplicações Java (GOSLING, 2005), possui funcionalidades de geração automática de serviços web a partir de classes. Definições de classes com declarações de métodos, também, podem ser geradas a partir de descrições de serviços de um arquivo WSDL.

O AXIS disponibiliza um servidor simples *standalone* para prover os serviços, tendo também a possibilidade de ser utilizado no contexto de um servidor externo.

O AXIS possui dois tipos de provedores de serviços: RPC e MSG. O provedor RPC (*Remote Procedure Call*), ao receber uma requisição SOAP, automaticamente deserializa os dados em XML da requisição para objetos Java e invoca o método específico que está determinado na requisição. Esse tipo de provedor trabalha de maneira mais rigorosa com as requisições SOAP recebidas, aceitando apenas as requisições que estão de acordo com a assinatura do método escolhido, caso contrário, uma exceção é gerada. Por outro lado, o provedor MSG trabalha de maneira mais flexível, aceitando qualquer tipo de requisição SOAP. Isto porque este provedor, ao receber a requisição, converte os dados, que estão em XML, em objetos DOM (*Document Object Model*) (W3C, 1998) que são passados como parâmetro na chamada

de um método genérico para processar a requisição. Ou seja, neste caso, o provedor transfere a responsabilidade para o usuário fazer uma possível verificação de dados em cada requisição.

2.3.8 SOA Governance

SOA Governance são iniciativas voltadas às boas práticas do uso de SOA nos sistemas empresarias distribuídos, para garantir maiores índices de desenvolvimento. Seu foco é direcionado no ciclo de vida dos serviços, determinando como os serviços são desenvolvidos, implantados e gerenciados.

SOA Governance defende uma definição clara de papéis das pessoas envolvidas no desenvolvimento de serviços, com permissões para desenvolvedores, implantadores e gerenciadores para cada novo serviço criado.

Para verificar se as políticas adotadas estão, realmente, favorecendo no desenvolvimento em SOA, métricas são usadas para quantificar características relevantes. Estas métricas, também, são usadas para identificar quais são os projetos que mais contribuem com os objetivos do negócio.

2.4 Trabalhos relacionados

Esta Seção faz um levantamento sobre os trabalhos relacionados a este projeto. Os trabalhos analisados possuem uma ou mais das características deste projeto: biblioteca de serviços, geração automática de serviços web e tarifação de componentes em uma biblioteca.

A Seção 2.4.1 detalha trabalhos sobre biblioteca de serviços. A Seção 2.4.2 destaca trabalhos sobre a geração automática de serviços web. A Seção 2.4.3 destaca os trabalhos relacionados à tarifação de componentes em uma biblioteca.

2.4.1 Bibliotecas de serviços

Esta Seção destaca três ferramentas comerciais de biblioteca de serviços. A ferramenta da IBM, *WebSphere Service Registry and Repository* (IBM, 2008b),

apresentada na Seção 2.4.1.1, a ferramenta da LogicLibrary, Logidex (LOGICLIBRARY, 2008), apresentada na Seção 2.4.1.2, e a ferramenta da Digital Assets, *DA Manager eS* (DIGITALASSETS, 2008), apresentada na Seção 2.4.1.3.

2.4.1.1 WebSphere Service Registry and Repository

A ferramenta da IBM, *WebSphere Service Registry and Repository*, provê um conjunto de funcionalidades para o gerenciamento e governança de aplicações baseadas em SOA (IBM, 2008b). Destas funcionalidades, as principais são publicação, recuperação e gerenciamento de informação de serviços, chamada metadado de serviço. Estas funcionalidades facilitam a seleção, utilização, controle e reutilização dos serviços.

Tendo uma preocupação na reutilização, suas ferramentas de publicação e recuperação procuram promover uma alta reutilização de serviços em projetos SOA, através de uma maior visibilidade e acesso para os serviços existentes.

Seus mecanismos de gerenciamento possibilitam o controle de metadados, assim como as interações, dependências e redundâncias dos serviços. Além disso, serviços podem ser classificados baseados em objetivos do negócio, políticas de uso podem ser configuradas e versionamentos e mudanças podem ser visualizados.

A partir destas características, *WebSphere Service Registry and Repository* tenta cobrir os principais requisitos para a prática de *SOA Governance,* como o controle de acesso, a classificação de serviço, a análise de impacto, o ciclo de vida do serviço, etc.

Um componente de software pode ser utilizado em uma perspectiva física ou pode ser usado em uma perspectiva de serviços. A ferramenta da IBM tem como finalidade dar suporte apenas à perspectiva de serviços. Entretanto, para uma organização que disponibiliza seus componentes tanto fisicamente quanto em serviços, a ferramenta da IBM propiciaria uma falta de integração das informações dos componentes, pois seria necessária uma outra biblioteca para armazenar os componentes fisicamente.

Uma outra característica negativa do *WebSphere Service Registry and Repository* é a carência de mecanismos de tarifação sobre os serviços, já que é uma ferramenta focada ao uso dentro das organizações e não voltada ao mercado. Entretanto, mesmo dentro da organização, mecanismos de tarifação poderiam ser interessantes para

fazer uma espécie de controle de utilização dos serviços na organização. Ao invés do uso de dinheiro, poderia ser utilizado o conceito de créditos, onde os membros da organização teriam créditos pra realizar a compra de serviços.

2.4.1.2 Logidex

Logidex é uma plataforma de governança de ciclo de vida de desenvolvimento de serviços, que possibilita a integração de processos de ativos de software dentro do ciclo de vida de serviços SOA (LOGICLIBRARY, 2008). Não se restringindo em apenas serviços, Logidex é uma ferramenta de mapeamento e descoberta de qualquer outro ativo de software de uma organização. A partir disto, Logidex se torna um inventário dos ativos da organização, mostrando seus relacionamentos, processos de negócio e sua infra-estrutura técnica.

O núcleo do Logidex, focado em gerenciamento e governança, garante que equipes desenvolvedoras de serviços produzam serviços de acordo com o escopo de negócio da organização, arquiteturas, boas práticas e políticas SOA. Estas funcionalidades são disponibilizadas durante todo o ciclo de vida do serviço.

Diferente da ferramenta *WebSphere Service Registry and Repository*, a biblioteca Logidex realiza a integração dos ativos de software da organização. Entretanto, devido ao seu enfoque no uso dentro da organização, carece também de mecanismos de tarifação.

2.4.1.3 DA Manager eS

DA Manager eS (*enabling Service*) é uma biblioteca de ativos reutilizáveis responsável pela publicação de serviços SOA, componentes e qualquer tipo de artefatos produzidos durante o desenvolvimento de software. A biblioteca utiliza uma base de metadados como fator integrador de todos os ativos reutilizáveis da organização.

Tendo como foco principal a publicação de serviços, *DA Manger eS* tenta resolver alguns desafios relacionados à reutilização e SOA. Alguns destes desafios são: baixa visibilidade dos serviços existentes, crescente complexidade nas áreas de negócio e nas áreas de TI, dificuldade de gerenciar os impactos das mudanças.

Algumas das funcionalidades providas são: definição de relacionamentos entre ativos como, por exemplo, dependências; diretórios de serviços UDDI que facilita a busca de serviços; mecanismos de *workflow* para realizar o controle e a governança do ciclo de vida dos ativos.

A Figura 2.5 apresenta a tela de cadastro de um ativo no DA *Manager*. Nesse exemplo, o ativo é um serviço para CRM (*Customer Relationship Management*) que está classificado na categoria *CRM*, que faz parte de uma categoria maior *Sistemas*.

Figura 2.5 - Cadastro de serviço no DA Manager eS

Uma das características mais interessantes do *DA Manager eS* é a preocupação de realizar a integração das informações de todos os ativos reutilizáveis de uma organização. Entretanto, a parte de serviços na biblioteca possui algumas deficiências. Uma delas está relacionada ao controle de acesso, já que serviços podem ser utilizados por qualquer pessoa sem nenhum controle. Sem esse controle, um outro fator importante que se perde é o registro de utilização de serviços que poderia ser utilizado para se obter um maior *feedback* dos serviços mais utilizados.

Tendo uma visão de que os ativos reutilizáveis da organização sejam utilizados apenas internamente, o *DA Manager eS* desconsidera questões de tarifação.

2.4.2 Geração dinâmica de serviços web

Esta Seção destaca trabalhos relacionados à geração automática ou semi-automática de serviços web. Dentre os trabalhos encontrados, o trabalho de Lee *et al*. (2004), que trata de conversão automática de componentes de software em serviços, é detalhado na Seção 2.4.2.1. Outro trabalho é o de Smith (2006), que propõe um mecanismo de geração automática de serviços a partir da camada de negócio de uma organização, detalhado na Seção 2.4.2.2.

2.4.2.1 Lee et al. (2004)

Lee *et al.* (2004) têm como abordagem a utilização de serviços para dar suporte ao desenvolvimento baseado em componentes, com a motivação de que as aplicações feitas a partir destes sejam independentes de plataforma.

Esse trabalho foi implementado em uma biblioteca de componentes, que faz parte de um ambiente de desenvolvimento de componentes (LEE *et al*., 2003). Os componentes nessa biblioteca eram publicados e recuperados apenas fisicamente.

Para dar suporte à utilização de serviços na biblioteca, foi criado um módulo, intitulado Web Services, que tem como principais funcionalidades:

- Cadastrar serviços de componentes, e
- Gerar serviços web dinamicamente a partir de componentes do repositório

A parte de cadastro de serviços consiste na publicação de documentos WSDL que representam os serviços associados aos componentes cadastrados na biblioteca. Estes serviços são fornecidos pelos produtores através de suas próprias infra-estruturas. A biblioteca não tem nenhum vínculo com o funcionamento destes, apenas serve como um meio de publicação.

A partir destes serviços cadastrados, consumidores utilizam mecanismos de busca para localizar serviços através de funcionalidades desejadas. Porém, se o serviço não é encontrado, é feita uma nova busca, só que desta vez, esta é realizada sobre a base

de componentes publicados fisicamente na biblioteca. Caso a funcionalidade desejada do serviço case com a de algum componente, o módulo Web Services é acionado para realizar a segunda funcionalidade descrita anteriormente, que é a geração dinâmica de serviço do componente. Este serviço, recém-criado, permanece no ar até o consumidor terminar o acesso.

Nesta abordagem, há a preocupação de disponibilizar serviços dos componentes cadastrados em uma biblioteca, porém a forma de disponibilização destes serviços não leva em consideração a questão de controle de acesso. A biblioteca não tem controle sobre os serviços dos componentes disponibilizados, permitindo o livre acesso a qualquer usuário. Além disso, a biblioteca perde informações importantes que poderiam ser coletadas a partir do controle dos serviços, como por exemplo, a freqüência de utilização ou a lista de usuários de um serviço.

Outra deficiência encontrada está relacionada à extensibilidade da arquitetura. A abordagem não leva em consideração que o módulo responsável pela geração de serviços dinâmicos seja fracamente acoplado à biblioteca, para facilitar possíveis evoluções, já que o mecanismo é disponível apenas para componentes escritos em Java e em C++.

2.4.2.2 Smith (2006)

Smith (2006) propõe um mecanismo de geração dinâmica de serviços web para acessar as funcionalidades da camada de negócio de uma organização. Este mecanismo tem a preocupação de manter os serviços sincronizados com as funcionalidades, já que estes podem sofrer constantes modificações.

O mecanismo proposto por Smith (2006) é, basicamente, um serviço genérico que realiza chamadas para a camada de negócio, de acordo com os argumentos passados. Estes argumentos definem o nome da funcionalidade e os seus próprios argumentos necessários para que esta seja executada. Por exemplo, para executar a funcionalidade *fatorial(int)* na camada de negócio, o serviço genérico seria invocado passando dois argumentos: o primeiro argumento, uma *String,* com o nome da funcionalidade e o segundo, um *int,* com o argumento da funcionalidade fatorial.

Para realizar essa tarefa, o serviço genérico precisa saber acessar qualquer funcionalidade da camada de negócio em tempo de execução, já que, segundo o

problema descrito por Smith, funcionalidades são inseridas e alteradas constantemente na camada de negócio. Desta forma não é possível estabelecer previamente quais seriam todas as funcionalidades da camada de negócio.

Para atingir esse objetivo, a solução adotada foi a utilização de reflexão. Reflexão é uma API, disponível na maioria das linguagens de programação modernas, que possibilita fazer análise em elementos do programa em tempo de execução. No Capítulo 3, reflexão é melhor detalhada. Com a ajuda de reflexão, é possível fazer uma análise na camada de negócio e executar qualquer funcionalidade em tempo de execução.

Em linhas gerais, o mecanismo é prático e fácil de aplicar. Entretanto, uma das principais deficiências deste mecanismo está relacionada à falta de descrição sobre as funcionalidades providas. Como o serviço é genérico, este possui uma interface genérica que permite a passagem de qualquer tipo de dados. Ou seja, o serviço não fornece informações sobre as funcionalidades. Com isso, para acessá-las, o usuário deve ter o conhecimento prévio sobre as mesmas, para que se possa fazer uma requisição de maneira correta. Uma outra questão que não foi levantada pela solução está relacionada à forma de manipulação de dados do serviço genérico. O serviço genérico precisa estar configurado para saber serializar ou deserializar os dados relacionados às funcionalidades. Além disso, essa configuração não pode ser feita de uma só vez durante a criação do serviço, é necessário ser feita dinamicamente a cada nova funcionalidade inserida na camada de negócio, para que o serviço genérico funcione corretamente. Entretanto, esses detalhes não foram tratados pela solução.

2.4.3 Tarifação de componentes

Esta Seção apresenta alguns trabalhos sobre bibliotecas que possuem mecanismos de tarifação de componentes.

O termo mecanismo de tarifação está relacionado às funcionalidades responsáveis pela execução de tarifação em uma biblioteca. Essas funcionalidades vão desde a etapa de precificação até a etapa de aquisição de componentes. Estes mecanismos podem apresentar características peculiares como, por exemplo, o seu modelo de tarifação. Dois modelos de tarifação conhecidos são os modelos pré-pago e pós-pago.

Na web, existem diversas bibliotecas comerciais voltadas não necessariamente a componentes de software, mas, sim, para software em geral. Estas bibliotecas, porém, possuem mecanismos de tarifação semelhantes ao que pode ser aplicado em uma biblioteca de componentes, podendo servir como um comparativo. Dentre estas bibliotecas, pode-se citar: Softchoice (SOFTCHOICE, 2008), Devshop (DEVSHOP, 2008), ProLib Tools (PROLIBTOOLS, 2008).

Dentre as bibliotecas de componentes comercias, as mais conhecidas são ComponentSource (2008) e FlashLine (2008). No entanto, a biblioteca FlashLine foi vendida para o ComponentSource (INFOWORLD, 2003).

A Seção 2.4.3.1 discute sobre bibliotecas de software comerciais genéricas. A Seção 2.4.3.2 detalha a biblioteca de componentes comercial de maior destaque na atualidade, a biblioteca ComponentSource.

2.4.3.1 Bibliotecas de software comerciais genéricas

A maioria das bibliotecas de software comerciais existentes utiliza mecanismos de tarifação básicos semelhantes a sites de compras. Com o conceito de carrinho de compras, produtos de software são adicionados no carrinho até o usuário finalizar a compra e realizar o pagamento. Geralmente, o pagamento é feito através de cartão de crédito.

Um recurso presente nessas bibliotecas é a disponibilização de software demonstrativo como um atrativo aos clientes, onde estes utilizam o demonstrativo e realizam a compra da versão completa do software, caso achem interessante.

A Figura 2.6 apresenta um exemplo de uma dessas bibliotecas de software comercias, a biblioteca Softchoice (SOFTCHOICE, 2008).

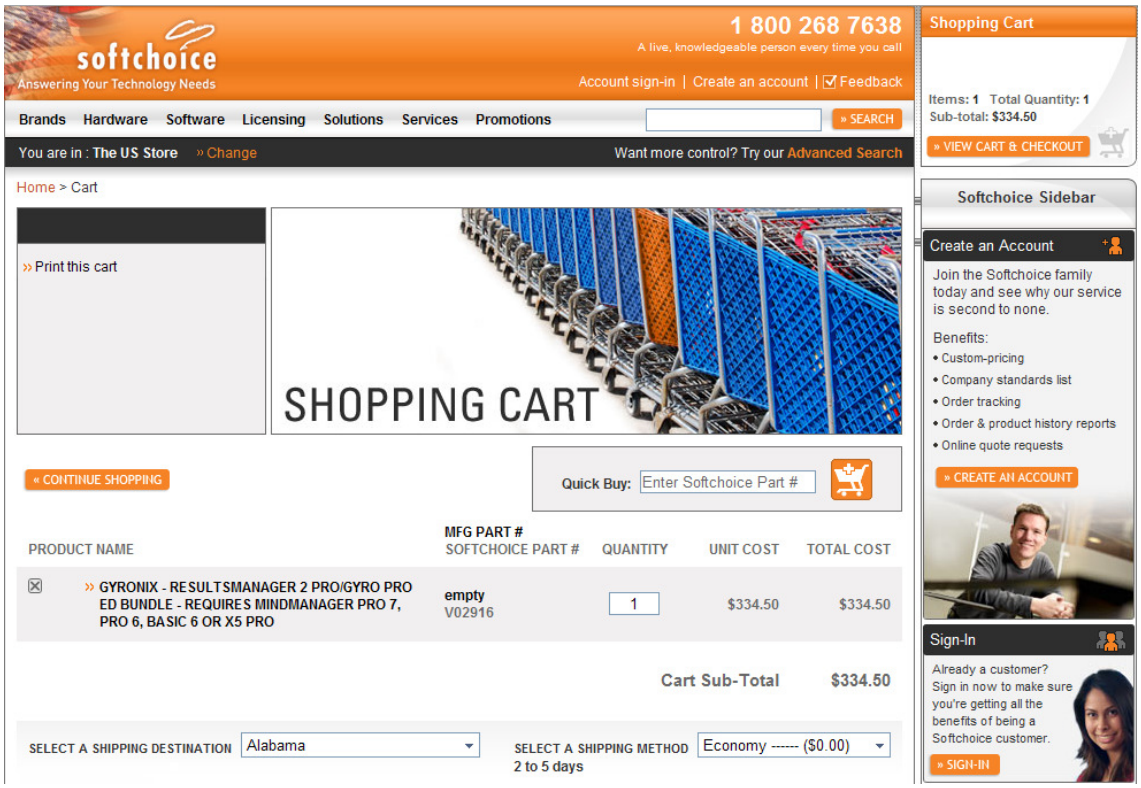


Figura 2.6 - Tela do Softchoice

Estas bibliotecas, conforme dito anteriormente, possuem mecanismos de tarifação básicos, com isso, carecem de alguns aspectos para a realização de tarifação mais abrangente. Uma das deficiências está relacionada à ausência de modelos de tarifação mais atrativos para os usuários, como, por exemplo, modelos pré-pago e pós-pago. Outra deficiência é a falta de informações extras sobre o software à venda como, por exemplo, informações sobre dependências, que indica quais produtos de software são necessários para a execução de uma ou mais funcionalidades de um software específico.

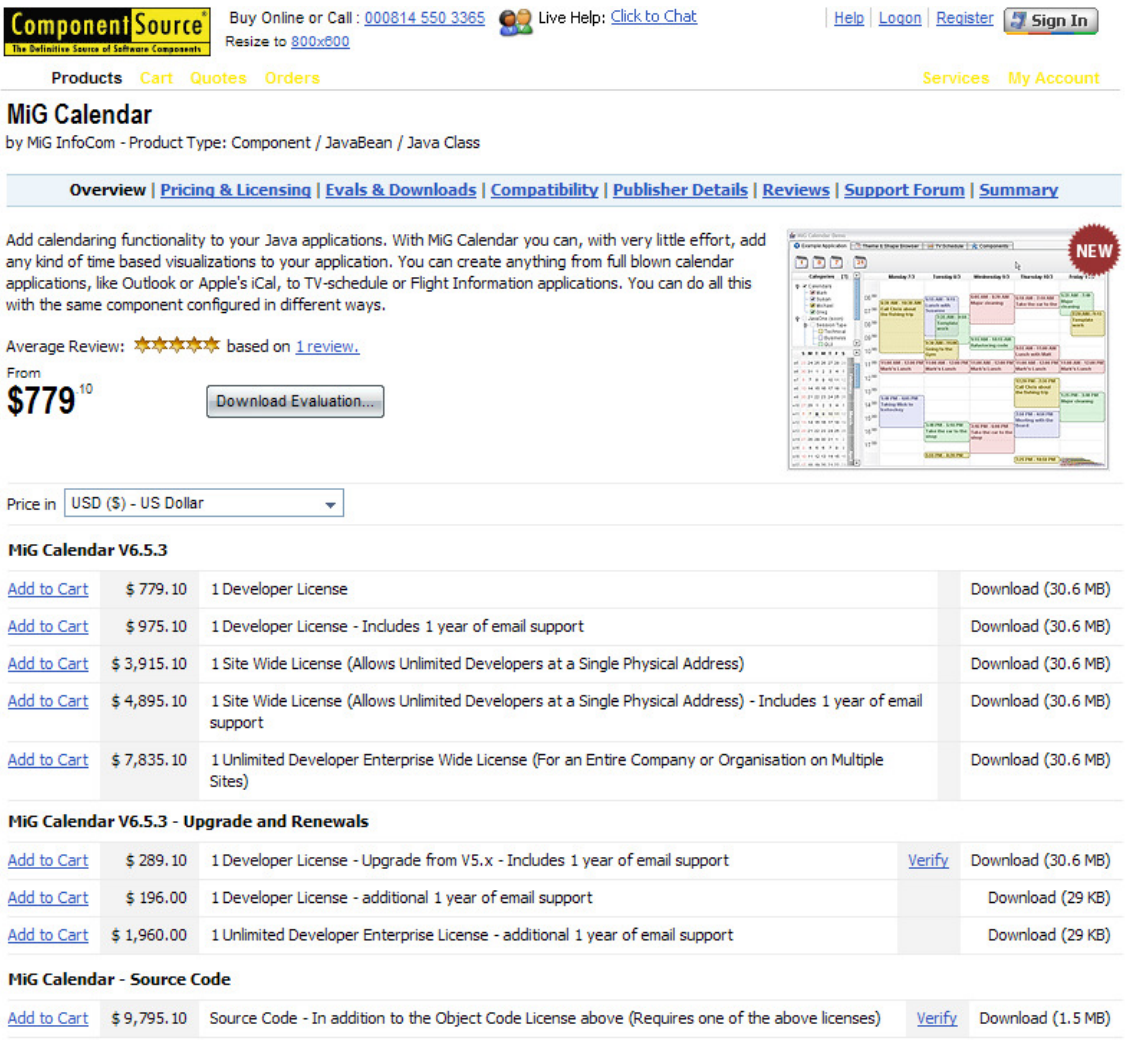
2.4.3.2 ComponentSource

ComponentSource é uma biblioteca de componentes comercial que trabalha com componentes COTS (*Commercial Off-The-Shelf*), disponibilizando-os por meio de uma interface Web.

ComponentSource organiza seus componentes através de categorias intuitivas, facilitando a busca de um componente. Além disso, a busca de um componente pode ser feita através de palavras-chave.

Como é uma biblioteca comercial, o seu ponto forte é a tarifação de componentes. ComponentSource possui mecanismos de tarifação que taxam valores sobre os componentes armazenados, além de taxar valores sobre as suas licenças associadas. De acordo com a licença associada, que viabiliza direitos e deveres sobre o componente, o valor do componente é acrescido.

A Figura 2.7 apresenta um exemplo de tarifação de um componente disponível no ComponentSource. O componente MIG Calendar, que pode ser adquirido a partir do valor $ 779.10, através de uma licença simples de desenvolvedor. No entanto, este também pode ser adquirido através de licenças mais robustas e caras que fornecem o código fonte, além de serviços extras como, por exemplo, um ano de suporte por e-mail.


ComponentSource
The Definitive Source of Software Components

Buy Online or Call : 000814 550 3365 Live Help: Click to Chat
Resize to 800x600

Help | Logon | Register | Sign In

Products Cart Quotes Orders Services My Account

MiG Calendar

by MiG InfoCom - Product Type: Component / JavaBean / Java Class

Overview | Pricing & Licensing | Evals & Downloads | Compatibility | Publisher Details | Reviews | Support Forum | Summary

Add calendaring functionality to your Java applications. With MiG Calendar you can, with very little effort, add any kind of time based visualizations to your application. You can create anything from full blown calendar applications, like Outlook or Apple's iCal, to TV-schedule or Flight Information applications. You can do all this with the same component configured in different ways.

NEW

Average Review: based on 1 review.

From
$779.10

Download Evaluation...

Price in USD ($) - US Dollar

MiG Calendar V6.5.3

Add to Cart	$ 779.10	1 Developer License	Download (30.6 MB)
Add to Cart	$ 975.10	1 Developer License - Includes 1 year of email support	Download (30.6 MB)
Add to Cart	$ 3,915.10	1 Site Wide License (Allows Unlimited Developers at a Single Physical Address)	Download (30.6 MB)
Add to Cart	$ 4,895.10	1 Site Wide License (Allows Unlimited Developers at a Single Physical Address) - Includes 1 year of email support	Download (30.6 MB)
Add to Cart	$ 7,835.10	1 Unlimited Developer Enterprise Wide License (For an Entire Company or Organisation on Multiple Sites)	Download (30.6 MB)

MiG Calendar V6.5.3 - Upgrade and Renewals

Add to Cart	$ 289.10	1 Developer License - Upgrade from V5.x - Includes 1 year of email support	Verify	Download (30.6 MB)
Add to Cart	$ 196.00	1 Developer License - additional 1 year of email support		Download (29 KB)
Add to Cart	$ 1,960.00	1 Unlimited Developer Enterprise License - additional 1 year of email support		Download (29 KB)

MiG Calendar - Source Code

Add to Cart	$ 9,795.10	Source Code - In addition to the Object Code License above (Requires one of the above licenses)	Verify	Download (1.5 MB)

Figura 2.7 - Exemplo de tarifação de um componente no ComponentSource

O ComponentSource, sem dúvida, é um dos melhores exemplos de biblioteca de componentes comercial na atualidade. No entanto, este ainda é deficitário em algumas características referentes à disponibilização de informação ao consumidor durante a aquisição de um componente. Por exemplo, não existem informações sobre dependências de componentes. Este tipo de informação é importante durante a aquisição de um componente, pois informa quais são os outros componentes que o consumidor deve ter para conseguir utilizar o componente de forma correta. Um outro ponto negativo é o não detalhamento sobre todas as versões de um componente, restringindo o consumidor apenas à última versão do componente. Com relação à tarifação, o ComponentSource não fornece modelos de tarifação mais flexíveis como, por exemplo, modelos pré-pago e pós-pago, para facilitar a compra de componentes.

2.5 Considerações finais

Este capítulo apresentou uma visão geral do cenário onde este trabalho está inserido, além de detalhar tecnologias e ferramentas que vão ser usadas nos capítulos de abordagem e implementação do trabalho. Este capítulo também fez uma análise de trabalhos que possuem características semelhantes ao proposto.

Em uma biblioteca de componentes que fornece funcionalidades de serviços e tarifação, algumas características são esperadas: serviços possuam mecanismos de controle de acesso e registro de utilização; dados sobre componentes e seus serviços estejam integrados; tarifação em todas as formas de aquisição/utilização de componentes; modelos de tarifação mais flexíveis; histórico de compras e vendas de componentes.

No entanto, a partir do estudo dos trabalhos relacionados, percebeu-se uma clara deficiência das ferramentas em algumas dessas características citadas anteriormente, indicando que existe a necessidade de uma solução que contemple todos essas características.

O Capítulo 3 apresenta a abordagem proposta para a implantação de infra-estrutura de serviços e tarifação em uma biblioteca de componentes que contempla todas essas características citadas acima e mais algumas outras.

Capítulo 3 - Abordagem

3.1 Introdução

Tendo como base a revisão da literatura, apresentando o contexto onde este trabalho está inserido e, além disso, através do estudo dos trabalhos relacionados, a aplicação de serviços em uma biblioteca de componentes deve atender aos seguintes requisitos:

R1: A biblioteca deve disponibilizar mecanismos automatizados para geração de serviços.

R2: Serviços previamente criados pelos produtores podem ser publicados na biblioteca, porém devem ser adaptados ao contexto da biblioteca.

R3: Serviços devem possuir mecanismos de controle de acesso.

R4: A utilização dos serviços deve ser registrada pela biblioteca.

De forma análoga, a tarifação deve atender aos seguintes requisitos:

R5: A tarifação deve ser aplicada em todos os meios de aquisição (ou utilização) de um componente em uma biblioteca.

R6: Componentes devem possuir formas de precificação diferentes de acordo com sua forma de aquisição.

R7: Os modelos de tarifação aplicados em uma biblioteca devem ser os que melhor se adaptem aos perfis dos usuários.

R8: A biblioteca deve possuir mecanismos para acessar o histórico de vendas e compras de componentes.

A abordagem proposta se fundamenta nesses requisitos. De acordo com as disposições destes requisitos, a abordagem pode ser dividida em duas etapas. A primeira etapa trata da ampliação das perspectivas de reutilização em uma biblioteca de componentes, com a inserção de serviços. Já a segunda etapa tem como foco a aplicação de tarifação em uma biblioteca de componentes. Um fator interessante a salientar é que estas duas etapas estão interligadas, e isso é destacado nas Seções a seguir, mostrando

que dependendo da forma de disponibilização de serviços em uma biblioteca, a tarifação se torna inviável de ser aplicada com êxito total.

O capítulo está organizado da seguinte forma: a Seção 3.2 descreve a arquitetura da abordagem, destacando seus elementos de apoio; nas Seções 3.3 e 3.4, é feito um detalhamento em cada um destes elementos; a Seção 3.3 trata dos elementos que dão apoio à aplicação de serviços, e a Seção 3.4 trata do elemento da arquitetura que dá apoio à tarifação em uma biblioteca de componentes.

3.2 Arquitetura

Segundo a Figura 3.1, existem módulos conectados à biblioteca de componentes para dar suporte na geração e utilização de serviços. Estes módulos podem ser classificados em dois tipos: Módulos que geram serviços web a partir de componentes armazenados na biblioteca, e módulos que recebem serviços web já implantados em outros ambientes e os adaptam para o contexto da biblioteca, denominados módulos Legados.

Cada módulo gerador de serviços web é definido para uma determinada linguagem de programação para viabilizar a sua implementação. Enquanto isso, o módulo Legados é único, já que não existem variâncias na adaptação dos serviços web legados, como é detalhado na Seção 3.3.3.

Outro requisito importante desses módulos é que os serviços gerados ou adaptados por eles possuam mecanismos de controle de acesso (R3). Este requisito é importante para que a biblioteca não perca o controle sobre os seus serviços de componentes disponibilizados, podendo restringir o acesso somente a usuários autorizados.

Um outro elemento presente na arquitetura é o elemento Tarifação. Como o seu próprio nome diz, ele fica responsável pela tarifação do uso dos componentes da biblioteca. Este uso pode estar relacionado à *downloads* ou serviços de componentes publicados na biblioteca. Um fator importante a salientar é que a tarifação sobre os serviços é possível devido à forma de publicação de serviços feita pelos módulos citados anteriormente.

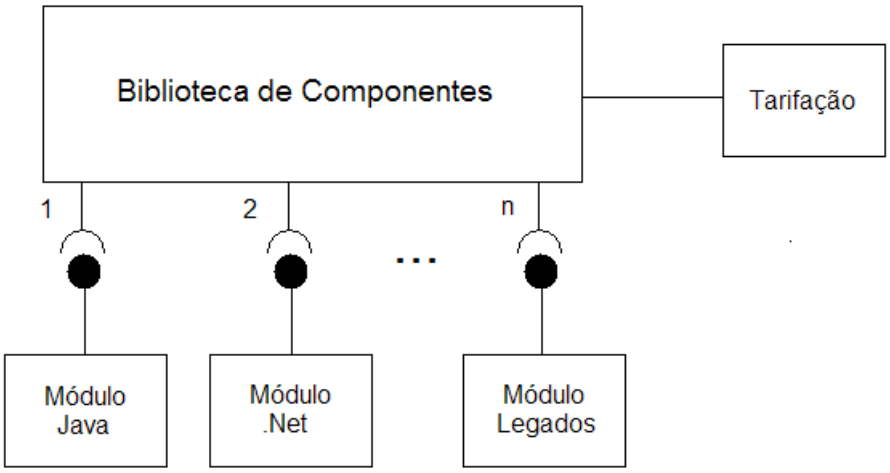


Figura 3.1 - Arquitetura da abordagem proposta

3.3 *Módulos*

De acordo com a arquitetura da Figura 3.1, os módulos são estruturas a parte da biblioteca. Esta separação é devido a fatores como extensibilidade e escalabilidade. Extensibilidade quer dizer a facilidade de evolução do sistema ou aplicação sem afetar os elementos envolvidos. Escalabiblidade é a facilidade do sistema ou aplicação aumentar a sua capacidade de processamento.

Extensibilidade é necessária devido ao problema da criação de serviços web ser específico à linguagem de programação adotada pelo componente. Essa separação facilita a biblioteca a não sofrer constantes modificações toda vez que tiver que atender a uma nova linguagem de programação. Para isso, basta criar mais um módulo para a linguagem em questão e informar a biblioteca de sua criação.

Esta separação também propicia uma melhor escalabilidade, uma vez que módulos e bibliotecas não precisam estar implantados em uma mesma máquina. Além disso, um mesmo módulo, específico de uma linguagem de programação, pode ter várias instâncias instaladas em várias máquinas, caso este módulo possua uma alta demanda de uso.

O objetivo desta Seção é detalhar os módulos que dão suporte a publicação de serviços em uma biblioteca.

Vale ressaltar que, na abordagem proposta, estes serviços podem ser publicados de duas maneiras:

1. Publicação de serviços a partir da criação de serviços web de componentes armazenados na própria biblioteca, e
2. Publicação de serviços a partir de serviços web legados fornecidos pelo produtor do componente.

Na publicação de serviços a partir da criação de serviços web de componentes armazenados na biblioteca, os módulos que darão apoio à biblioteca, denominados módulos específicos de linguagem, devem ser capazes de receber os componentes da biblioteca e gerar serviços web dos mesmos. Este processo é detalhado na Seção 3.3.2. Por outro lado, no caso da publicação de serviços a partir de serviços web legados fornecidos pelo produtor do componente, o módulo específico para esta situação, denominado Legados, deve receber os documentos descritores dos serviços web e adaptá-los ao contexto da biblioteca, inserindo mecanismos de controle de acesso (de acordo com o requisito R3). Este processo é detalhado na Seção 3.3.3.

Outra característica importante destes módulos, além da criação de serviços web, é a comunicação com a biblioteca. Essa comunicação vai desde o pedido da biblioteca ao módulo para a criação de um serviço, até o uso de um serviço. Essa comunicação entre biblioteca e módulo é explicada na Seção 3.3.1.

3.3.1 Comunicação com a biblioteca

Devido à separação entre biblioteca e módulo, a comunicação entre essas duas entidades é importante. Essa comunicação acontece em dois momentos: durante a criação dos serviços e durante a utilização dos serviços.

Durante a criação de serviço, a comunicação entre biblioteca e módulo possui características diferentes de acordo com o tipo do módulo em questão. Em linhas gerais, essa comunicação pode ser generalizada de acordo com a Figura 3.2. A biblioteca solicita a criação de um serviço e o módulo retorna o WSDL do serviço gerado. Uma descrição mais detalhada para cada tipo específico de módulo é apresentada na Seção 3.3.2 e na Seção 3.3.3.

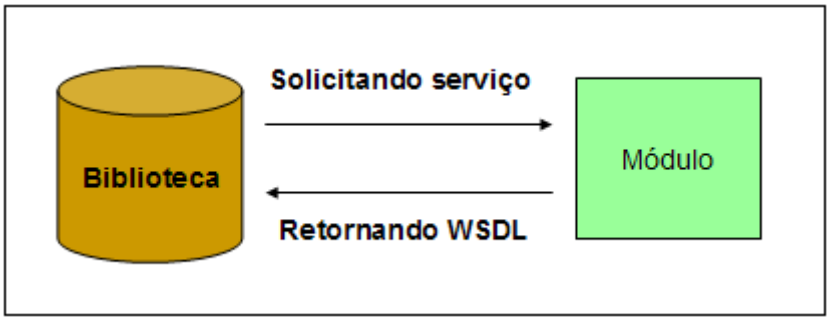


Figura 3.2 - Estrutura básica de comunicação de solicitação de serviço

Durante a utilização dos serviços gerados pelos módulos, a comunicação entre biblioteca e módulo é importante para garantir o controle de acesso sobre os serviços. A cada serviço requisitado, o módulo se comunica com a biblioteca para saber se o solicitante tem permissão para utilizar o serviço. Além disso, o módulo presta serviços de *log* para a biblioteca, enviando informações sobre as utilizações dos serviços (R4). Informando qual serviço foi utilizado e qual usuário o utilizou. Estas informações são importantes para os produtores dos serviços, para que eles possam identificar quais são os seus serviços mais utilizados e para terem contato com os seus consumidores. Estas informações vão ser de suma importância também para a aplicação da tarifação sobre os serviços.

A Figura 3.3 ilustra todo o ciclo da requisição de um serviço. Na primeira ação, o usuário se autentica com a biblioteca e recebe um identificador de sessão. Logo após, o usuário solicita o serviço, onde um de seus parâmetros é o identificador de sessão (aberta na biblioteca pelo usuário). Com o serviço solicitado, o módulo se comunica com a biblioteca para verificar se o usuário tem autorização para utilizar o serviço. Em caso positivo, o serviço é executado e, antes do resultado do serviço ser retornado para o usuário, o módulo se comunica novamente com a biblioteca para registrar a utilização do serviço.

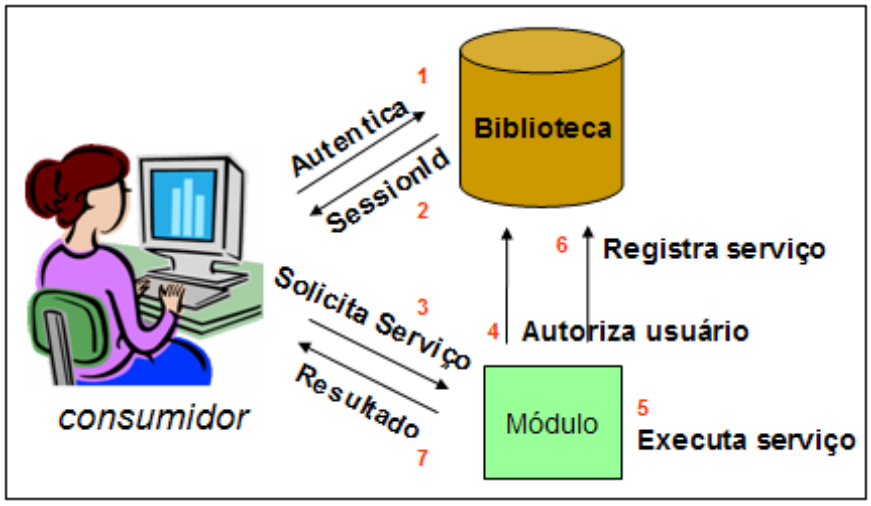


Figura 3.3 - Ciclo de requisição de serviço

O passo 5, apresentado na Figura 3.3, é executado de formas distintas em função do tipo do módulo em questão, como é detalhado na Seção 3.3.2 e na Seção 3.3.3.

3.3.2 Módulos específicos de linguagem

Estes módulos são encarregados de gerar serviços web a partir de um componente de software.

O processo de criação de serviços web deve ser feito da forma mais automática possível (vide requisito R1), ou seja, com o menor número de intervenções do usuário durante o processo. Este processo fica caracterizado em quatro etapas:

1. Analisar o componente e extrair suas possíveis operações.
2. Selecionar as operações desejadas para o serviço web a ser criado.
3. Gerar uma nova aplicação que simule o componente original e que suporte as funcionalidades de comunicação com a biblioteca.
4. Gerar o serviço web a partir dessa nova aplicação.

Esse processo é detalhado em duas Seções. A Seção 3.3.2.1 cobre a primeira etapa do processo, ou seja, a etapa de análise e extração de possíveis operações de um componente. A Seção 3.3.2.2 detalha as três últimas etapas do processo, começando da etapa de seleção das operações até a etapa de geração do serviço web.

3.3.2.1 Análise do componente

Para se construir um serviço web de um componente, é necessário saber o que esse componente "oferece". Quais são seus métodos, funções, classes, bibliotecas, pacotes, etc. Estas informações são importantes para que se consiga mapear as possíveis operações do serviço web a ser construído. Geralmente, as operações de um serviço web são definidas a partir dos métodos (ou procedimentos) de um programa.

Esta primeira etapa, de análise do componente, é muito dependente da linguagem de programação em que este componente foi implementado. Cada linguagem de programação possui sua sintaxe, de acordo com um paradigma. Dependendo da linguagem, a análise pode ser feita a partir do próprio arquivo compilado, enquanto em outras, a análise só pode ser feita a partir do código fonte. Por isso, devido à dificuldade de uma solução genérica, cada módulo deve ser específico para uma linguagem de programação.

Esta análise, dependendo da linguagem de programação, pode ser feita através de APIs de reflexão fornecidas pela própria linguagem. Na Seção 3.3.2.1.1, estas APIs de reflexão são descritas com maior detalhe.

3.3.2.1.1 Reflexão computacional

Reflexão é uma API fornecida por algumas linguagens para analisar elementos de um programa durante sua execução. Tomando como exemplo uma linguagem orientada a objetos, estes elementos podem ser classes, interfaces, métodos e atributos. Reflexão é comumente usada para escrever ferramentas de desenvolvimento como *debuggers*, visualizadores de classes e construtores de interface gráfica.

Tendo como exemplo a API de reflexão Java, esta fornece as seguintes funcionalidades:

- Determinar a classe de um objeto.
- Obter informação a respeito de modificadores, atributos, métodos, construtores de classes.
- Descobrir quais constantes e declarações de métodos pertencem a uma interface.

- Criar uma instância de uma classe na qual seu nome era desconhecido até a execução do programa.
- Capturar e colocar valores em um atributo de um objeto, sendo seu nome desconhecido até a execução do programa.
- Invocar métodos os quais eram também desconhecidos antes da execução do programa.

3.3.2.2 Seleção e geração do serviço web

Após a análise do componente e a extração das possíveis operações para o serviço web a ser gerado, a próxima etapa do processo seria a seleção das operações desejadas. Nesta etapa, o módulo transfere a responsabilidade para o usuário, que fica encarregado de analisar as opções disponíveis e selecionar as mais adequadas.

A proposta do processo de geração de serviços web é que ele seja semi-automático, pois esta segunda etapa do processo é manual, devido à necessidade da resposta do usuário em relação às operações que deverão ser incluídas no serviço web. Se todo serviço web criado contivesse todas as funcionalidades de um componente, esta etapa não se faria mais necessária. Contudo, para dar uma maior flexibilidade para o processo de geração de serviços, optou-se que o usuário (produtor) fizesse a seleção das operações desejadas.

Com as operações selecionadas, a próxima etapa seria a geração do serviço web. Entretanto, um dos requisitos necessários para os serviços publicados em uma biblioteca é que se tenha o controle de acesso (R3). Ou seja, o usuário que queira usar o serviço deve se autenticar na biblioteca. Para fazê-lo, o componente utilizado deve passar por alterações. O resultado esperado é que toda operação (mapeado de um método/função do componente) do serviço tenha um parâmetro a mais de entrada, o parâmetro de autorização *sessionId*.

Por exemplo, considere um componente escrito em Java que possui um método que calcula o fatorial de um número a partir de um parâmetro de entrada n, inteiro. A assinatura do método é definida desta forma:

fatorial(int n)

Ao transformá-lo em serviço, o método, além de possuir o parâmetro de entrada n, irá possuir o parâmetro de autorização *sessionId*. Sua assinatura passaria a ser dessa forma:

fatorial(String *sessionId*, int n)

Além da alteração da assinatura do método, algumas instruções também são adicionadas no corpo do método. Instruções que realizam a comunicação com a biblioteca para autorizar e registrar o uso do serviço.

A forma utilizada para alterar o componente foi criar um componente semelhante ao original, em termos de assinaturas de métodos, com exceção da adição do parâmetro de autorização em cada método. No corpo de cada método, há um redirecionamento para o método do componente original, além de instruções de comunicação com a biblioteca para realizar o controle de acesso e o registro de utilização do serviço. Ou seja, a aplicação criada está trabalhando como um *proxy*.

Proxy é um padrão de projeto estrutural (GAMMA *et al*., 1995). Em termos de orientação a objeto, um *proxy* é, genericamente, uma classe que serve de interface para outra coisa. Como por exemplo, uma conexão de rede, um arquivo ou qualquer outra coisa que seja difícil de duplicar.

Continuando com o exemplo do componente em Java, a Figura 3.4 apresenta a classe *Math* do componente original à esquerda, que possui o método *fatorial*. À direita, temos a classe *proxy* criada, que também possui o mesmo método da classe original. Porém, no corpo do método, existem instruções para realizar o controle de acesso e o registro de utilização do serviço, além da chamada do método da classe original, no qual o seu resultado é retornado.

Classe Original

```
public class Math{
   public static int fatorial(int n){
      int fat = 1;
      for(int i=1;i<=n; i++)
            fat = fat*i;
      return fat;
   }
}
```

Classe Gerada

```
public class Proxy{
   public int fatorial(String sessionId, int n)
    throws UsuarioSemAutorização {
      autorizaUsuario(sessionId);
      int resultado = Math.fatorial(n);
      registraChamada("fatorial");
      return resultado;
   }
}
```

Figura 3.4 - Classe *Proxy*

Com a aplicação *proxy* criada, a última etapa do processo é a implantação do serviço e o documento WSDL é retornado para o solicitante da criação do serviço, neste caso, a biblioteca.

3.3.3 Módulos Legados

Em uma biblioteca, serviços podem ser publicados a partir de serviços web previamente criados pelos produtores. Esta opção de publicação pode ser a melhor escolha se o produtor já possui toda uma infra-estrutura preparada, ou devido à complexidade de se criar um serviço web automaticamente a partir de um componente armazenado na biblioteca.

Entretanto, quando estes serviços web "legados" são publicados, a biblioteca não detém o controle sobre a utilização. Todas as requisições feitas para estes serviços, de acordo com as suas especificações, serão aceitas. Ou seja, a biblioteca participa apenas na descoberta dos serviços, porém, quando são utilizados, a biblioteca não percebe. Isto acontece porque estes serviços web não estão adaptados ao contexto da biblioteca.

Para resolver esse problema, os serviços legados devem ser de alguma maneira adaptados ao contexto da biblioteca (vide requisito R2). Estes devem suportar mecanismos de controle de acesso e de registro de utilização, igual aos suportados pelos serviços gerados pelos módulos específicos de linguagem, descritos na Seção 3.3.2. O módulo responsável por realizar esta adaptação nos serviços web é denominado Módulo Legados.

Para fazer a adaptação no serviço web, é necessário alterar o seu código para modificar as assinaturas das operações (com a adição do parâmetro de entrada para o controle de acesso *sessionId*) e realizar a comunicação com a biblioteca. Mas estes serviços web legados não estão ao alcance da biblioteca, já que pertencem aos produtores. Com isso, uma alternativa para fazer essa adaptação é construir um novo serviço que se passe como o serviço web legado, mas que realize as funcionalidades de controle de acesso e registro de chamadas. Novamente, é usado o padrão de *proxy*, só que desta vez, um *proxy* em nível de serviço.

Este serviço *proxy* vai atuar como um filtro que intercepta as requisições, faz as alterações necessárias na requisição e repassa para o serviço legado. A Figura 3.5 e a

Figura 3.8 mostram o funcionamento do serviço *proxy* durante o processo de requisição e resposta de um serviço legado, respectivamente.

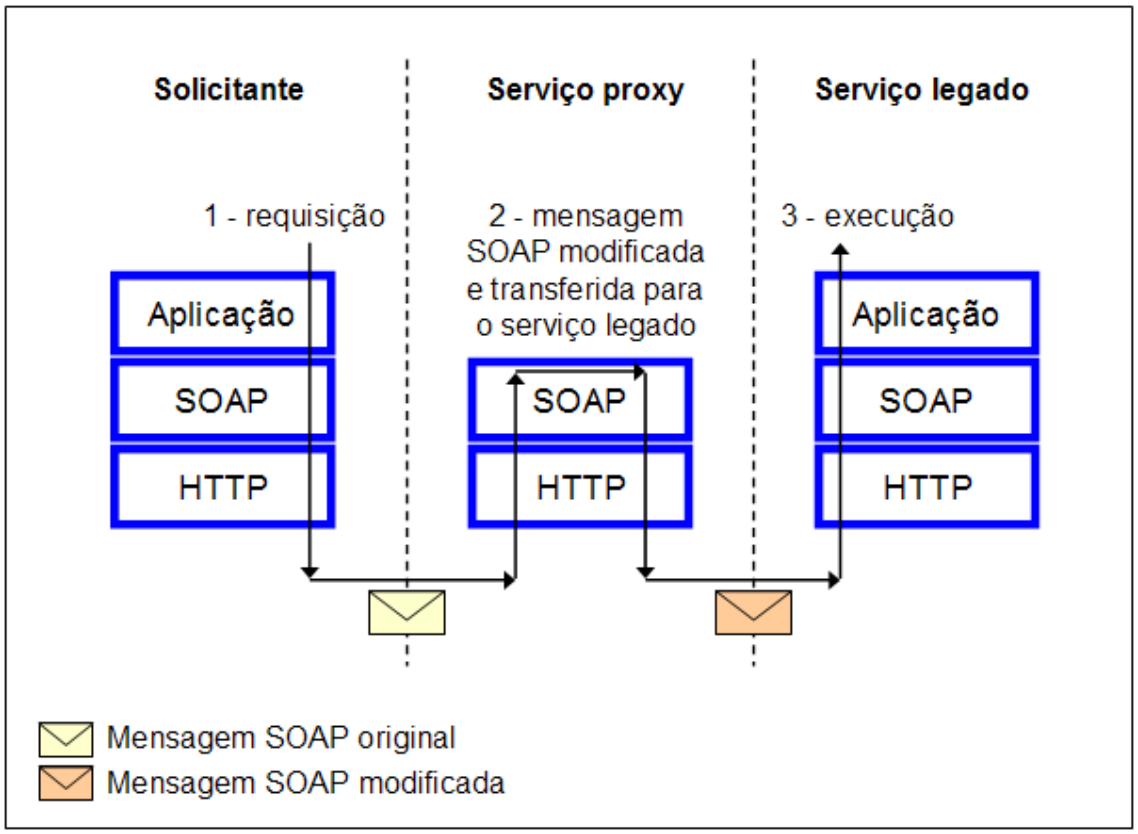


Figura 3.5 – Exemplo de funcionamento do serviço *proxy* durante uma requisição de um serviço

O processo de requisição, mostrado na Figura 3.5, começa com o solicitante fazendo a requisição ao serviço legado. O solicitante gera a mensagem de requisição SOAP, convertendo os dados que estavam no nível de aplicação. A seguir essa mensagem é empacotada dentro de uma requisição HTTP, que é endereçada para o serviço *proxy*, embora o solicitante pense que a requisição esteja endereçada diretamente para o serviço legado.

Ao receber a requisição, o serviço *proxy* extrai a mensagem SOAP da requisição HTTP. Como a mensagem SOAP será repassada para o serviço legado, não há necessidade de converter os dados XML para dados de aplicação, pois eles terão que ser convertidos de volta para dados XML, desperdiçando tempo de processamento, e, além disso, esse processo de (de)serialização não é simples de ser executado, já que cada serviço pode ter seu formato de dados no nível de aplicação. Com isso, o serviço *proxy* trabalha com os dados no nível da mensagem SOAP e realiza a sua modificação, extraindo do seu conteúdo, o parâmetro de controle de acesso *sessionId*. Este parâmetro

é retirado, já que o serviço legado desconhece essa informação. Se a mensagem fosse enviada com esse parâmetro, o serviço legado não iria funcionar de forma esperada.

Um exemplo de requisição SOAP e a sua alteração estão ilustrados na Figura 3.6 e na Figura 3.7. A Figura 3.6 apresenta a mensagem original, onde se encontra o parâmetro *sessionId*.

```
<soapenv:Envelope>
   <soapenv:Body>
      <fatorial>
         <sessionId xsi:type="xsd:string">1184334635609</sessionId>
         <n xsi:type="xsd:int">7</n>
      </fatorial>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 3.6 - Exemplo de mensagem SOAP para o serviço *proxy*

Já a Figura 3.7 apresenta a mensagem modificada sem o parâmetro *sessionId*. Esta mensagem é repassada para o serviço legado.

```
<soapenv:Envelope>
   <soapenv:Body>
      <fatorial>
         <n xsi:type="xsd:int">7</n>
      </fatorial>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 3.7 - Exemplo de mensagem SOAP alterada para o serviço legado

Voltando ao exemplo da Figura 3.5, com a mensagem SOAP modificada, uma nova requisição HTTP é feita para ser remetida para o serviço legado. Entretanto, antes da requisição ser repassada, o serviço *proxy* se comunica com a biblioteca para verificar se o solicitante está autorizado a acessar o serviço. Em caso positivo, a mensagem é repassada para o serviço legado que, por sua vez, converte os dados até o nível de aplicação e executa a operação solicitada. Caso contrário, uma mensagem de acesso negado é retornada para o solicitante.

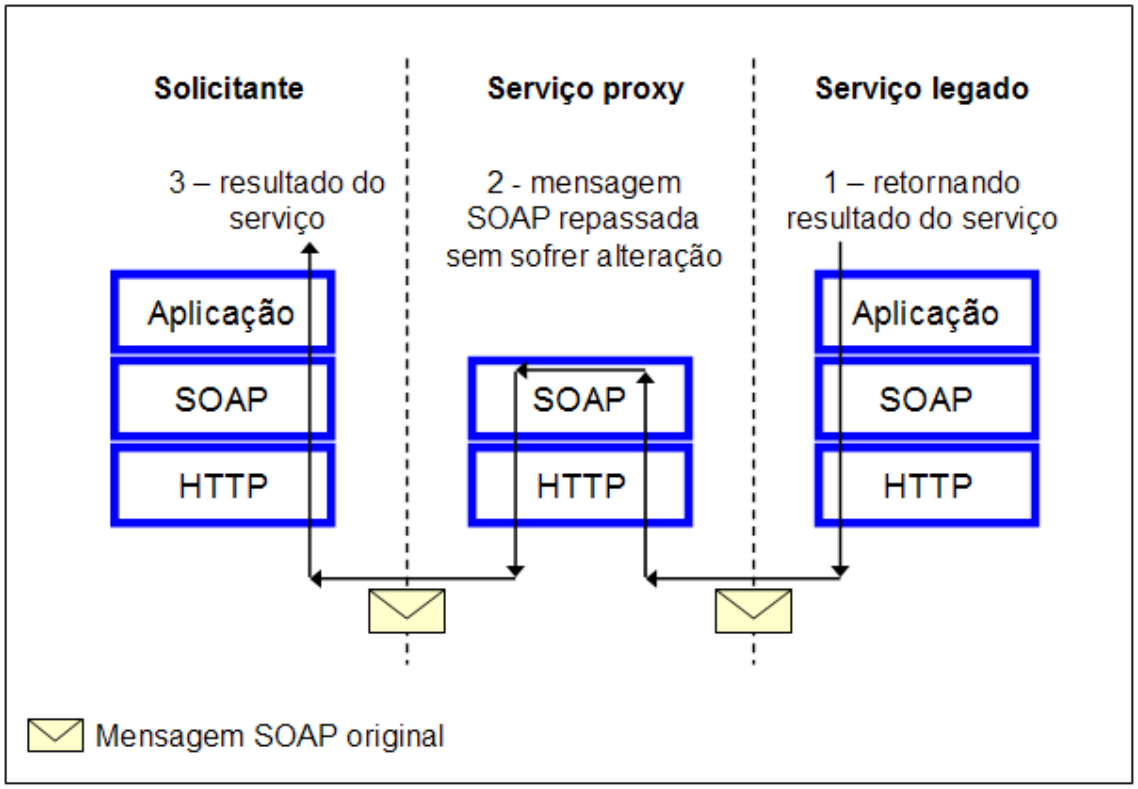


Figura 3.8 - Exemplo de funcionamento do serviço *proxy* durante uma resposta de um serviço

De acordo com a Figura 3.8, após a execução da requisição feita ao serviço legado, o resultado do serviço, que está no nível de aplicação, é convertido em uma mensagem de resposta SOAP e esta é empacotada em uma mensagem de resposta HTTP, no qual é remetida para o serviço *proxy*, já que é o solicitante do serviço, na visão do serviço legado.

Ao receber a resposta, o serviço *proxy* extrai a mensagem SOAP do pacote HTTP e, sem alterar a mensagem SOAP, a remete em uma mensagem de resposta HTTP para o solicitante. Neste momento também, o serviço *proxy* se comunica novamente com a biblioteca para informar sobre a utilização do serviço. Por fim, o solicitante recebe a resposta do serviço.

Resumindo, as operações realizadas por um serviço *proxy* a cada requisição de serviço são:

1. Alterar mensagem SOAP
   1.1. Extrair *sessionId*
2. Realizar controle de acesso
3. Repassar a requisição alterada para o serviço legado
4. Receber resposta do serviço legado
5. Realizar registro de chamada de serviço

6. Repassar a resposta do serviço legado para o solicitante

Uma vez detalhado o funcionamento do serviço *proxy*, a Seção 3.3.3.1 detalha o processo de geração dos serviços *proxy*s para cada serviço legado.

3.3.3.1 Criação do serviço proxy

De acordo com a característica do serviço *proxy* de trabalhar com os dados no nível SOAP, este serviço se torna um serviço genérico para adaptar qualquer tipo de serviço legado. Isto porque o serviço *proxy* não trata dos dados pertinentes ao serviço legado, mas, sim, dos dados relacionados à biblioteca, que é o caso do *sessionId,* para realizar o controle de acesso e o registro de chamada.

A partir disto, a criação de um serviço *proxy* se torna um processo mais simples, pois não há a necessidade de se criar uma aplicação específica para tratar cada serviço legado requisitado. O que deve ser feito é apenas a instanciação do serviço genérico para cada serviço legado que se queira adaptar. Este processo de instanciação, porém, necessita de algumas informações do serviço legado para realizar a configuração do serviço *proxy* a ser instanciado. Estas informações são referentes às operações e aos pontos de acesso que o serviço legado possui.

Como apresentado no Capítulo 2, um serviço web é composto de várias operações. Cada operação pode ter um ou vários pontos de acesso que são usados para acessá-la. Uma operação pode usar mais de um ponto de acesso quando há necessidade de servir a operação em mais de um servidor, ou quando precisa disponibilizar o acesso para mais de um protocolo de comunicação. Por exemplo, uma operação pode ter um ponto de acesso que trabalhe com o protocolo de comunicação HTTP, e outro ponto de acesso que trabalhe com o protocolo SMTP. Como cada operação pode ter seu próprio ponto de acesso, a instância do serviço *proxy* tem que estar configurada para saber direcionar para o ponto de acesso certo de acordo com a operação solicitada.

Na Figura 3.9, é apresentado um exemplo de um serviço legado que possui duas operações: a operação fatorial e a operação seno. Cada operação possui seu ponto de acesso próprio. A operação fatorial, com o ponto de acesso 1 e 2, os quais representam dois servidores servindo a operação, e a operação seno, com o ponto de acesso 3. Se este serviço estiver adaptado com o serviço *proxy*, quando o serviço *proxy* receber uma

requisição para a operação fatorial, o serviço *proxy* deve redirecionar a requisição (depois de alterada) para o ponto de acesso 1 ou para o ponto de acesso 2. Se a requisição for redirecionada para o ponto de acesso 3, o qual está esperando requisições para a operação seno, um resultado não esperado será gerado.

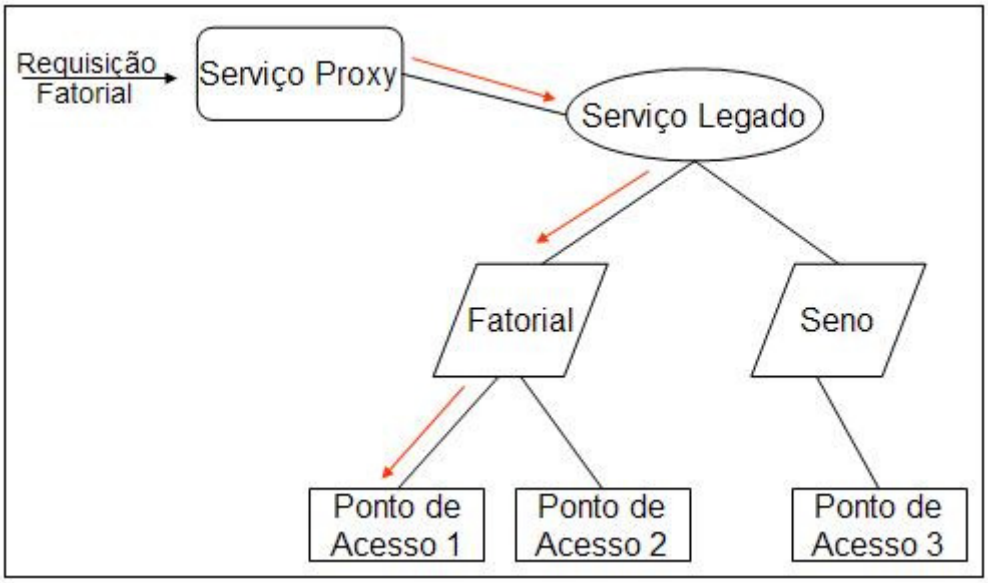


Figura 3.9 – Esquema de serviço legado adaptado com serviço *proxy*

Quando uma operação possui mais de um ponto de acesso disponível, onde cada ponto de acesso representa um servidor, o serviço *proxy* pode trabalhar também com distribuição de carga, repassando uma parcela do fluxo de requisições para cada ponto de acesso. Utilizando o mesmo exemplo da Figura 3.9, o fluxo de requisições para a operação fatorial pode ser distribuída 50% para o ponto de acesso 1 e os outros 50% para o ponto de acesso 2.

Todas estas informações do serviço web estão descritas no documento WSDL. É a partir deste documento que o módulo Legado extrai as informações para criar um serviço *proxy*. Então, o processo de criação do serviço *proxy* inicia-se com o recebimento de um WSDL de um serviço legado e é caracterizado pelas seguintes etapas:

1. Extrair informações do WSDL
2. Instanciar o serviço *proxy*
3. Gerar o WSDL do novo serviço

Uma vez que o serviço web *proxy* esteja criado e implantado, o próximo passo é a geração do WSDL do novo serviço. Esse WSDL pode ser gerado a partir do WSDL do serviço legado, porém algumas alterações devem ser feitas. A primeira alteração está relacionada com as definições de dados das operações, mais especificamente, nas especificações dos parâmetros de entrada das operações. Todas as especificações dos parâmetros de entrada das operações devem possuir um parâmetro a mais, o parâmetro de controle de acesso *sessionId* que vai ser usado pelo serviço *proxy*. A Figura 3.10 mostra um trecho de um documento WSDL de um serviço legado em quem uma das suas operações foi adicionado o parâmetro *sessionId* (destacado em negrito).

```
<wsdl:message name="fatorialRequest">
      <wsdl:part name="sessionId" type="xsd:string"/>
      <wsdl:part name="n" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="fatorialResponse">
      <wsdl:part name="fatorialReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:portType name="ServiçosMatematica">
      <wsdl:operation name="fatorial" parameterOrder="sessionId n">
            <wsdl:input message="impl:fatorialRequest" name="fatorialRequest"/>
            <wsdl:output message="impl:fatorialResponse" name="fatorialResponse"/>
      </wsdl:operation>
</wsdl:portType>
```

Figura 3.10 – Trecho de um documento WSDL

A segunda alteração está relacionada à configuração dos pontos de acesso. No WSDL do serviço legado, os pontos de acesso estão configurados com o(s) endereço(s) do(s) servidor(es) das operações do serviço legado. Estes endereços não devem estar acessíveis ao usuário da biblioteca porque, se estivessem, as requisições feitas para o serviço legado não seriam filtradas pelo serviço *proxy*. Com isso, a alteração necessária é justamente na definição destes endereços dos pontos de acessos, trocando-os pelo endereço do serviço *proxy*. A Figura 3.11 mostra outro trecho de um documento WSDL que contém a configuração de um ponto de acesso. O atributo *location* (destacado em negrito) contém o endereço do ponto de acesso. É este atributo que deve ser modificado para o endereço do serviço *proxy*.

```
<wsdl:service name="legacyService">
    <wsdl:port binding="impl:legacyServiceSoapBinding" name="legacyService">
        <wsdlsoap:address location="http://reuse.cos.ufrj.br/legacyService"/>
    </wsdl:port>
</wsdl:service>
```

Figura 3.11 - Trecho de um documento WSDL

3.4 Tarifação

Tarifação é o outro elemento da arquitetura, apresentada na Figura 3.1, que dá apoio à abordagem proposta. Seu papel é realizar a taxação de preço em todos os meios de aquisição ou utilização dos componentes em uma biblioteca de componentes (R5).

A tarifação de componentes em uma biblioteca não consiste em simplesmente atribuir preços a componentes. Conceitualmente, é necessário analisar formas de precificação de componentes, levando em consideração as diferentes formas de distribuições de um componente. É necessário, também, analisar quais modelos de tarifação que melhor se adaptem aos perfis de usuários da biblioteca (R7). Alguns exemplos são os modelos pós-pago e pré-pago, bastante utilizados no domínio de telefonia celular. Em termos de biblioteca, a biblioteca tem que sofrer adaptações para suportar tal mecanismo. Dentre estas adaptações, as seguintes características devem estar presentes:

1. Conta de usuário adaptada para o mecanismo de tarifação
2. Geração de contratos para vincular produtor, consumidor e componente
3. Mecanismos de visualização de histórico de vendas e compras (R8)

As contas de usuário da biblioteca devem estar preparadas para suportar a tarifação dos componentes. Dependendo do modelo de tarifação aplicado, as contas têm que fornecer informações de crédito e limite de compras de componentes. Estas características são esclarecidas na Seção 3.4.2, onde se discute sobre modos de tarifação.

Com a aquisição do componente pelo usuário, a biblioteca tem que registrar essa transação através de contratos. Nestes contratos, devem estar contidas informações como: o componente consumido; a licença do componente (caso tenha alguma); o consumidor; e o produtor do componente.

Estas informações, registradas no contrato, devem ser externalizadas na biblioteca de diversas maneiras, cada uma atendendo às necessidades de um tipo de usuário da biblioteca. Como, por exemplo, para um consumidor que queira consultar suas últimas transações feitas, ou para um produtor que deseja verificar o número de consumidores de um de seus componentes.

A Seção 3.4.1 analisa a tarifação em dois cenários onde a biblioteca de componentes pode ser aplicada: dentro e fora da organização. Analisando as vantagens e desvantagens encontradas em cada cenário.

Nas Seções seguintes, são esclarecidas questões que foram levantadas sobre a aplicação de tarifação em uma biblioteca de componentes. Dentre essas questões estão: modelos de tarifação, detalhados na Seção 3.4.2, e formas de precificação de componentes, detalhados na Seção 3.4.3.

3.4.1 Tarifação dentro e fora de uma organização

Uma biblioteca de componentes pode ser aplicada dentro de uma organização, armazenando os componentes gerados pela organização e os disponibilizando para o uso da própria organização. Por outro lado, a biblioteca pode ser aplicada fora da organização, com um sentido comercial, disponibilizando componentes a partir de taxas pré-estabelecidas aos consumidores interessados.

Entre os dois casos de aplicação de biblioteca apresentados, o segundo caso é o que mais necessita recursos de tarifação de componentes. Os componentes comercializados têm que ser tarifados de acordo com a forma de aquisição do componente, seja via *download* do componente, utilização de algum serviço, ou qualquer outra forma adicional de aquisição. A tarifação também deve estar de acordo com as licenças disponibilizadas pelos componentes.

Entretanto, o primeiro caso de aplicação de biblioteca (dentro da organização), mesmo aparentando não precisar de mecanismo de tarifação, pode tirar proveito deste mecanismo. Isto porque tarifação não se resume em simplesmente cobrar um componente. Existe todo um mecanismo por trás para chegar a este resultado que uma organização pode utilizar para aplicar em outros fins. Por exemplo, com a utilização monitorada dos componentes, a biblioteca é capaz de guardar no seu histórico cada consumidor que utilizou cada componente. Isso pode ser útil em uma organização,

permitindo que sejam feitos diversos tipos de análise sobre a utilização dos componentes. Como, por exemplo, análise sobre o grau de interação dos grupos da organização, verificando se estes grupos estão utilizando componentes uns dos outros, ou análise sobre o grau de serventia de um componente da organização através de sua freqüência de uso.

3.4.2 Modelos de Tarifação

Em diversos outros produtos e serviços fora do mundo de software, várias formas de tarifação são aplicadas para melhor adaptação ao perfil do consumidor. Em uma biblioteca de componentes de software, isso pode ser tratado da mesma maneira. De acordo com o perfil de usuários de uma biblioteca, um melhor modelo de tarifação pode ser aplicado para viabilizar um maior consumo de componentes.

Dos modelos de tarifação mais conhecidos, os modelos pós-pago e pré-pago (WIKIPEDIA, 2008a; WIKIPEDIA, 2008b) são largamente usados no domínio de telefonia. O modelo pós-pago funciona com a cobrança posterior dos produtos ou serviços consumidos pelo consumidor em um intervalo de tempo. O modelo pré-pago funciona da maneira inversa, onde o consumidor adiciona créditos na sua conta, e o seu consumo se restringe a apenas o valor total dos seus créditos adicionados.

Estes dois modelos bastante populares podem, também, ser aplicados em uma biblioteca de componentes. Viabilizando que consumidores de uma biblioteca, por exemplo, realizem o pagamento posterior de serviços e componentes requisitados (modelo pós-pago), ou então, impondo que estes tenham crédito para poderem consumir na biblioteca (modelo pré-pago).

Para aplicar estes modelos, cada usuário cadastrado na biblioteca deve ter sua conta relacionada, que contabiliza todas as suas transações (utilizações de serviços ou *downloads* de componentes) feitas. Cada conta deve possuir um limite, de forma análoga a uma conta bancária, viabilizando transações mesmo que o usuário não possua crédito. Esse limite, porém, se for aplicado o modelo pré-pago, será zero. Ou seja, o usuário só consome se possui crédito. No caso do modelo pós-pago, o limite é maior do que zero, representando o total de aquisições e serviços que o usuário pode fazer em um intervalo de tempo.

Aplicando o modelo pré-pago, a forma de pagamento deve ser feita antes do consumo, como já foi dito anteriormente. Com isso, a biblioteca deve viabilizar mecanismos para a adição de créditos, via interface gráfica, ou via administrador da biblioteca.

Aplicando o modelo pós-pago, a biblioteca deve emitir de tempos em tempos a fatura do usuário para que o mesmo realize o pagamento.

3.4.3 Formas de precificação de componentes

Precificação de componentes é um assunto complexo, com isso, diversas soluções podem ser adotadas. Estas soluções estão relacionadas com a forma de aquisição do componente (vide requisito R6). Por exemplo, a maneira de atribuir preço a um componente não é a mesma de atribuir preço a um serviço do componente.

Para o caso de componentes que vão ser adquiridos via download, a forma mais básica de se precificar é estipular um preço fixo. Esse preço fixo é estipulado através do cálculo dos gastos durante o desenvolvimento do componente e outros fatores. Em um nível de detalhe maior, um componente pode ser precificado a partir dos artefatos (código fonte, binário, documentação) e licenças que ele possui. Trabalhando nesse nível maior de detalhe, a biblioteca pode ajudar a gerar um valor total do componente.

Tomando como exemplo a Figura 3.12, o componente é composto de dois artefatos: binário (bin) e código fonte (src). Para cada um foi atribuído um valor. Artefato binário com o valor de 10 e artefato código fonte com o valor de 20. Com estes valores, a biblioteca é capaz de estipular um valor total do componente de acordo com a configuração desejada do componente, que no caso é 30. Isso pode depender, também, de como o componente pode ser disponibilizado. Um componente pode possuir várias configurações destes artefatos. Por exemplo, uma configuração pode conter somente o artefato binário. Então, o valor do componente será igual ao valor do artefato binário: 10. Em outra configuração, pode conter apenas código fonte, o que implicaria em valor total igual a 20.

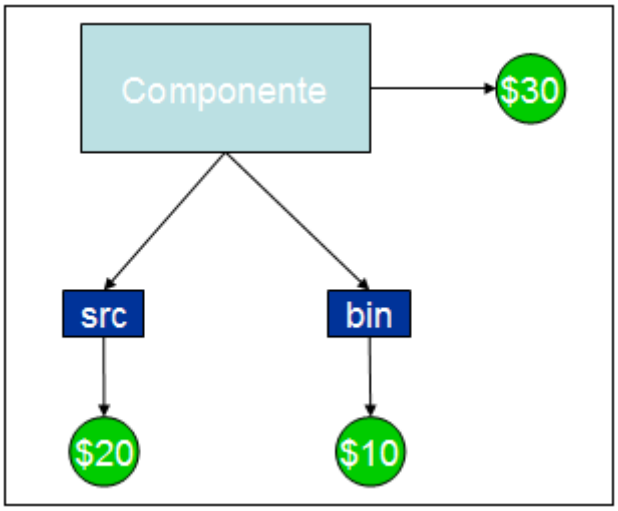


Figura 3.12 – Esquema de automação de precificação

No caso de serviços de componentes, precificá-los se torna um caso ainda mais difícil. A pergunta que surge em questão é a seguinte: Quantas vezes deve se usar um serviço até que atinja o custo total do componente? A partir desta questão, surge uma outra forma de precificação: o valor amortizado. O valor amortizado é um cálculo que pode auxiliar a extração de um valor apropriado para um serviço. A partir do valor do componente, o valor do serviço pode ser calculado usando como base o número de vezes que o serviço deve ser usado para que tenha o mesmo valor que o componente. Além disso, existem outros fatores que influenciam o valor do serviço, como, por exemplo, o custo de manutenção do servidor, o grau de demanda sobre o serviço, etc.

Em suma, esta abordagem não tem a preocupação de estabelecer regras ou fórmulas para estipular valores de componentes e serviços, mas sim especificar mecanismos para precificá-los em uma biblioteca.

3.5 Considerações finais

Este capítulo apresentou uma abordagem para a implantação de uma infra-estrutura que suporte serviços e tarifação de componentes em uma biblioteca de componentes. Com relação aos serviços de componentes, a abordagem possui as seguintes características:

- Geração semi-automática de serviços a partir dos componentes armazenados na biblioteca.

- Serviços previamente criados pelos produtores podem ser publicados na biblioteca. Porém, quando publicados, sofrem alterações para se adaptarem ao contexto da biblioteca.
- Todos os serviços publicados na biblioteca possuem controle de acesso.
- A utilização dos serviços é registrada pela biblioteca.

Com relação à tarifação de componentes, a abordagem possui as seguintes características:

- Componentes possuem formas de precificação diferentes de acordo com a sua forma de aquisição.
- Todos os meios de aquisição de componentes da biblioteca podem ser tarifados.
- Modelos de tarifação são definidos de acordo com os perfis de usuário da biblioteca.
- Visualização de histórico de compras e vendas de componentes.

O Capítulo 4 apresenta a implementação desta abordagem em uma biblioteca de componentes específica, a biblioteca Brechó. Com a implementação desta abordagem nessa biblioteca, outras características importantes inerentes à abordagem surgem, devido a algumas peculiaridades que a Brechó possui.

Capítulo 4 - Implementação

4.1 Introdução

O trabalho proposto é direcionado a bibliotecas de componentes que ainda não possuem uma infra-estrutura de disponibilização de serviços e mecanismos de tarifação de componentes. O trabalho, também, é direcionado para bibliotecas que tratam este problema, mas suas abordagens são deficientes, devido à ausência de alguns requisitos apresentados no Capítulo 3.

A implementação deste trabalho foi aplicada em uma biblioteca que não possuía nenhuma dessas duas características, a biblioteca de componentes Brechó, que é apresentada na Seção 4.2. Posteriormente, trata-se do detalhamento da implementação aplicada na Brechó, estando esta dividida em mais duas Seções. A Seção 4.3 detalha a implementação de serviços, e a Seção 4.4 detalha a implementação da tarifação.

4.2 Biblioteca de Componentes Brechó

Este trabalho foi concebido no contexto do Projeto Brechó (WERNER, 2008a), desenvolvido pelo grupo de reutilização de software da COPPE (WERNER, 2008b). O Projeto Brechó tem como objetivo desenvolver uma biblioteca de componentes de software que provê mecanismos de armazenamento, busca e recuperação de componentes (WERNER *et al*., 2007).

Esta Seção apresenta uma descrição geral dos mecanismos fundamentais da Brechó, na Seção 4.2.1, e um detalhamento da sua organização interna, na Seção 4.2.2.

4.2.1 Mecanismos fundamentais da Brechó

A biblioteca de componentes Brechó, em seu núcleo inicial, é um sistema de informação para Web com uma base de dados de componentes, fornecedores e consumidores. Este núcleo define os mecanismos de documentação, armazenamento, busca, e recuperação de componentes.

Os mecanismos de publicação e documentação consideram um conceito flexível de componente, além do binário, visando uma representação que inclua os possíveis artefatos produzidos durante o desenvolvimento do componente (como especificações, código fonte, manuais, etc.). Desta forma, a Brechó torna-se capaz de permitir a aquisição de diferentes combinações de conjuntos de artefatos empacotados, como por exemplo, um componente composto apenas pelo artefato código fonte, ou então, um componente composto pelos artefatos binário e manual.

A estrutura de documentação é fundamentada em categorias e formulários dinâmicos e configuráveis, ao invés de um formato prefixado. Desta forma, é possível classificar um componente em várias categorias, além da possibilidade de criar novas que podem ser associadas a formulários de documentação que permitem a construção da documentação do componente como um mosaico.

A flexibilidade de publicação dos componentes influencia diretamente os mecanismos de pesquisa e recuperação. A biblioteca permite a busca por componentes via filtros de categorias e palavras-chave (RAPOSO, 2007). A partir dos resultados de pesquisa por componentes, torna-se possível a sua disponibilização (*download*) para os consumidores, de acordo com as políticas de recuperação definidas pelos fornecedores.

4.2.2 Organização Interna da Brechó

A organização interna da Brechó é dividida em níveis, exibidos na Figura 4.1.a, para a representação de um componente. Cada nível representa um corte em um componente, levando em consideração diferentes aspectos, exibidos na Figura 4.1.b. A Figura 4.1.c exibe um exemplo de cada um destes níveis.

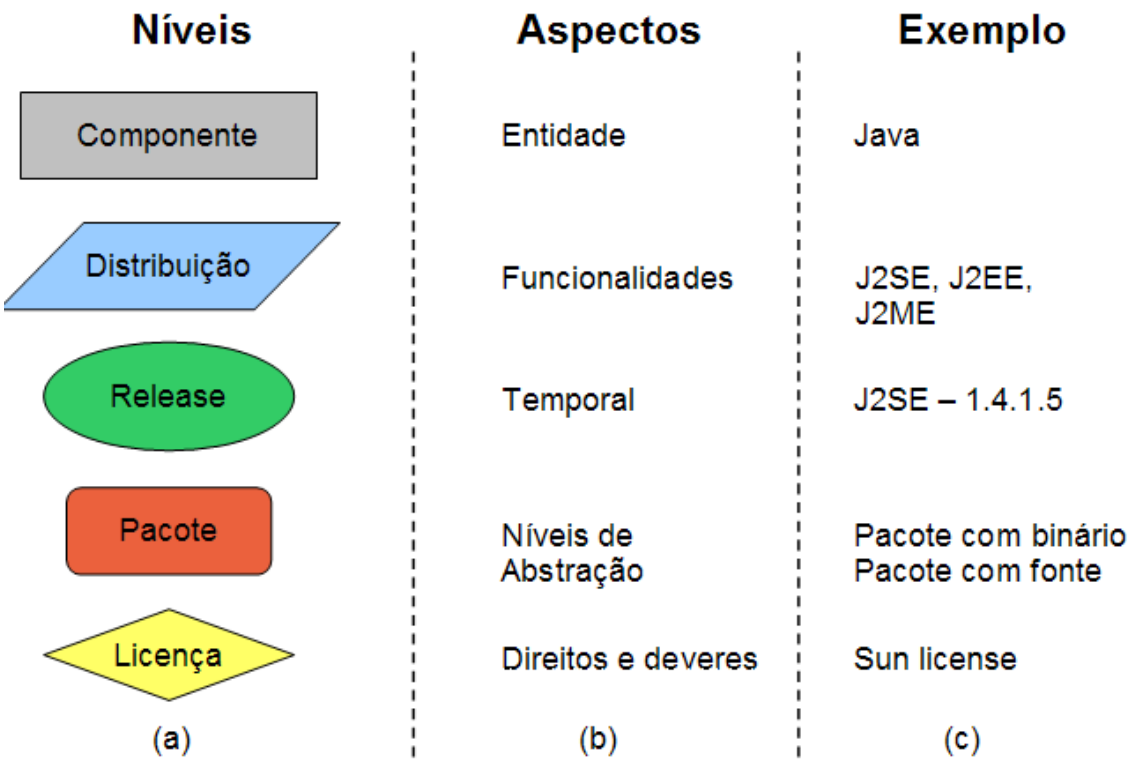


Figura 4.1 – Organização Interna da Brechó

O primeiro nível, **Componente**, representa conceitualmente as entidades armazenadas na Brechó, sem as informações concretas sobre as implementações dessas entidades. A Figura 4.1.c apresenta um exemplo que é o componente Java. O nível **Distribuição** representa um corte funcional sobre as entidades, fornecendo conjuntos de funcionalidades que são desejadas por grupos específicos de usuários. Continuando com o exemplo da Figura 4.1.c, o componente Java possui as distribuições J2SE, J2EE, J2ME. O nível **Release** representa um corte temporal sobre as distribuições, que define versões dos artefatos que implementam as entidades em um determinado instante no tempo. Por exemplo, a distribuição J2SE possui a release J2SE-1.4.1.5, de acordo com a Figura 4.1.c. A partir desse nível, as entidades passam a ter informações concretas sobre suas implementações. Essas implementações concretas das entidades, usualmente, existem em diferentes níveis de abstração (e.g.: documentação do usuário, análise, projeto, código, binário, etc.), e diferentes empacotamentos podem ser definidos para possibilitar a reutilização de parte dos níveis de abstração disponíveis (e.g.: um pacote contendo documentação do usuário e binário). Desta forma, o nível **Pacote** permite que seja feito um corte em níveis de abstração, possibilitando que sejam agrupados artefatos de acordo com o público alvo de reutilização. Seguindo o exemplo da Figura 4.1.c, a

release J2SE-1.4.1.5 possui o pacote binário e o pacote fonte. O nível **Licença** possibilita a definição de direitos e deveres sobre os pacotes. Para cada pacote, podem ser estabelecidas licenças específicas, que garantem direitos e deveres entre os produtores e os consumidores dos componentes. Finalizando o exemplo da Figura 4.1.c, a licença disponível para os pacotes binário e fonte é a licença Sun License.

Além disso, torna-se necessário deixar bem claras as relações entre os componentes dentro de uma biblioteca. A dependência é uma das relações mais importantes. A relação de dependência estabelece quais componentes são requeridos para que um determinado componente funcione adequadamente. Em um nível mais concreto, uma dada *release* de um componente pode depender de uma determinada *release* de outro componente. Tendo esta preocupação, a Brechó provê ferramentas para o estabelecimento e inferência de dependências em diferentes níveis de abstração. Para viabilizar o uso das dependências, a Brechó fornece ferramentas de apoio à recuperação das dependências de um componente.

4.3 Serviços

Com a adição de serviços na Brechó, sua organização interna, explicada anteriormente, sofre alterações. A alteração acontece a partir do nível de release. Uma release passa a possuir dois mecanismos distintos para a reutilização de componentes. O primeiro é o mecanismo de pacotes, que continua sendo como definido na Seção 4.2.2. Este mecanismo permite a reutilização concreta da release, com a aquisição dos seus artefatos. O segundo mecanismo que surge é o de serviços. Este mecanismo, por sua vez, viabiliza a reutilização da release através de serviços.

A Figura 4.2 apresenta a nova organização interna da Brechó. Pacotes e serviços se encontram no mesmo nível, já que os dois representam formas de reutilização de uma release. Logo após, vem o nível de licenças que especificam direitos e deveres na reutilização de pacotes e serviços.

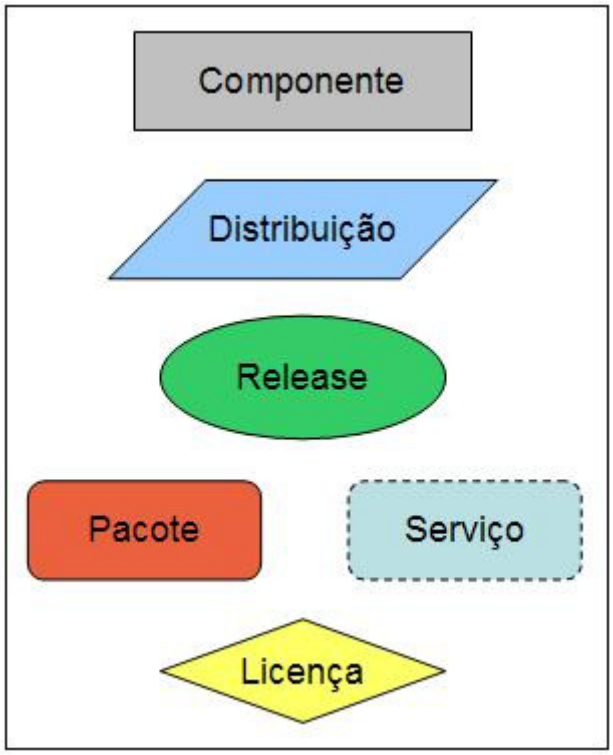


Figura 4.2 – Nova organização interna da Brechó

Nas próximas Seções, é feito um detalhamento das funcionalidades inseridas na Brechó para suportar a estrutura de serviços. As funcionalidades inseridas são: Publicação de serviço (Seção 4.3.2), catalogação dos serviços (Seção 4.3.3), utilização de serviço (Seção 4.3.4), verificação de uso dos serviços (Seção 4.3.5). Entretanto, antes de discutir as funcionalidades inseridas, a Seção 4.3.1 apresenta a implementação dos módulos que dão apoio à publicação de serviços na Brechó.

4.3.1 Módulos

Os dois tipos de módulos que dão apoio à publicação de serviços foram implementados na Brechó: módulos específicos de linguagem e módulo Legados. Entretanto, dentro do conjunto de módulos específicos de linguagem, apenas o módulo específico para linguagem Java foi implementado.

Estes módulos, de acordo com a abordagem proposta, são fracamente acoplados à Brechó para facilitar uma futura extensão da biblioteca como, por exemplo, a adição de mais módulos para tratar de outras linguagens de programação. Com isso, estes módulos seguem uma API de comunicação definida pela Brechó para realizarem as suas funcionalidades. Esta API possui variâncias de acordo com o tipo de módulo em

questão, devido às suas funcionalidades distintas. Com isso, esta API é definida diferentemente para os dois tipos de módulos. A API para o módulo específico de linguagem possui os seguintes serviços:

- *getPossibleServiceOperations*
  - Parâmetros
    - *SessionId*
    - Componente
  - Retorna lista das possíveis operações de um serviço
- *createService*
  - Parâmetros
    - *SessionId*
    - Dependências do componente
    - Lista das operações desejadas
  - Retorna WSDL do serviço
- *undeployService*
  - Parâmetros
    - *SessionId*
    - Nome do serviço
  - Retorna confirmação

Em contrapartida, a API para o módulo Legados possui os seguintes serviços:

- *adaptService*
  - Parâmetros
    - *SessionId*
    - WSDL do serviço legado
  - Retorna WSDL do serviço gerado (adaptado)
- *undeployService*
  - Parâmetros
    - *SessionId*
    - Nome do serviço

- Retorna confirmação

Além da API que os módulos devem seguir, também existe a API de registro dos módulos na Brechó, que é composta de um único serviço:

- *registerModule*
  - Parâmetros
    - Endereço de acesso
    - Tipo do módulo
  - Retorna confirmação

O serviço *registerModule* tem como parâmetros o endereço de acesso do módulo e o tipo do módulo. Os tipos de módulo definem se o módulo é Legado ou específico de linguagem. A Brechó possui uma lista de tipos esperados que podem ser configurados pelo administrador. Um exemplo de lista de tipos esperados pode ser:

1-Legados
2-Java
3-.NET

O módulo Java e o módulo Legados têm como papel implantar serviços que estejam configurados ao contexto da Brechó. Entretanto, estes módulos não necessariamente precisam funcionar, também, como os servidores dos serviços. A principal funcionalidade destes módulos é gerar a aplicação que será usada como serviço e realizar a implantação do serviço em um servidor de serviços web. Com isso, durante a implementação destes módulos, foi preciso definir qual servidor de serviços web utilizar, pois de acordo com o servidor escolhido, a forma de implantação de um serviço pode ser diferente. A ferramenta adotada para implantar serviços web foi o *framework* AXIS, devido ao conjunto de funcionalidades que a ferramenta possui direcionadas para aplicações Java (destacadas no Capítulo 2), além de ser uma ferramenta livre, disponibilizada pela Apache.

Para realizar a implantação do serviço usando o *framework* AXIS, uma das possíveis maneiras segue as seguintes etapas:

1. Adicionar a aplicação em um diretório específico do AXIS
2. Comunicar com o AXIS para fazer a implantação do serviço

Estas duas etapas são consideradas o *deploy* físico, com a adição de arquivos no diretório específico do AXIS, e o *deploy* lógico, com a comunicação com o AXIS, do processo de implantação do serviço.

As próximas Seções apresentam aspectos técnicos sobre estas duas etapas, destacando diferenças da aplicação de cada etapa, de acordo com o tipo de módulo implementado. A Seção 4.3.1.1 detalha a etapa de *deploy* físico e a Seção 4.3.1.2 a etapa de *deploy* lógico.

4.3.1.1 Deploy físico

O AXIS reserva um diretório especial para receber aplicações que serão utilizadas para a implantação de serviços, ou seja, para realizar o *deploy* físico. No contexto dos módulos específicos de linguagem, todo componente enviado pela Brechó para a geração de serviço é armazenado dentro desse diretório do AXIS. Um problema com esta abordagem é que se todos os componentes forem armazenados no diretório principal do AXIS, colisões de nomes entre arquivos de componentes diferentes podem ocorrer. Para resolver este problema e prover uma melhor organização dos componentes armazenados, foi utilizada uma estrutura de subdiretório para os componentes implantados, isolando cada componente dos demais. Essa estrutura facilita também a remoção de um serviço de um componente, já que o módulo não precisa saber os nomes de cada arquivo que o componente possui, e, sim, o endereço do seu subdiretório para fazer a remoção completa.

Além da colisão física dos arquivos, existe também a colisão lógica que ocorre durante a execução dos componentes, quando estes componentes possuem classes com nome e endereço de pacotes iguais, mesmo estando situadas em diretórios diferentes. Em Java, existe uma classe, denominada *ClassLoader* (TAVIS, 2008), que realiza o gerenciamento do carregamento das classes durante a execução de um programa. Esta classe armazena os endereços de todas as classes definidas através de *classpaths* ou diretórios *defaults*, além de armazenar os endereços de arquivos JAR. A classe *ClassLoader* é instanciada quando o JVM (SUN, 2008c) é inicializado para rodar um

programa e, por definição, é herdada de objeto a objeto criado para ser usada durante o carregamento de alguma classe. Como a instância do *ClassLoader*, por definição, é herdada de objeto a objeto criado, quando duas aplicações que possuem classes com nome e endereços de pacotes iguais estão rodando no mesmo JVM, estas irão herdar a mesma instância do *ClassLoader*. Com isso, quando a classe de nome em comum entre as duas aplicações for requisitada, o *ClassLoader* não saberá definir em qual diretório correto carregar a classe.

A Figura 4.3 mostra um exemplo deste conflito no contexto dos componentes que serão transformados em serviços. No exemplo, o servidor AXIS foi inicializado e recebeu uma instância do *ClassLoader, ClassLoader'*. Logo após, o componente 1 é hospedado e inicializado, recebendo a instância do *ClassLoader* do servidor. O componente 1 possui a classe *java.lang.ComponentClass*. A seguir, um segundo componente, componente 2, é hospedado no servidor. Este componente possui uma classe com o mesmo nome e endereço de pacote do componente 1, a classe *java.lang.ComponentClass*. O componente 2, ao ser executado, também recebe a mesma instância do *ClassLoader*. Com isso, quando os dois componentes forem executados e requisitarem a classe *java.lang.ComponentClass*, o *ClassLoader* não saberá definir em qual diretório correto carregar a classe.

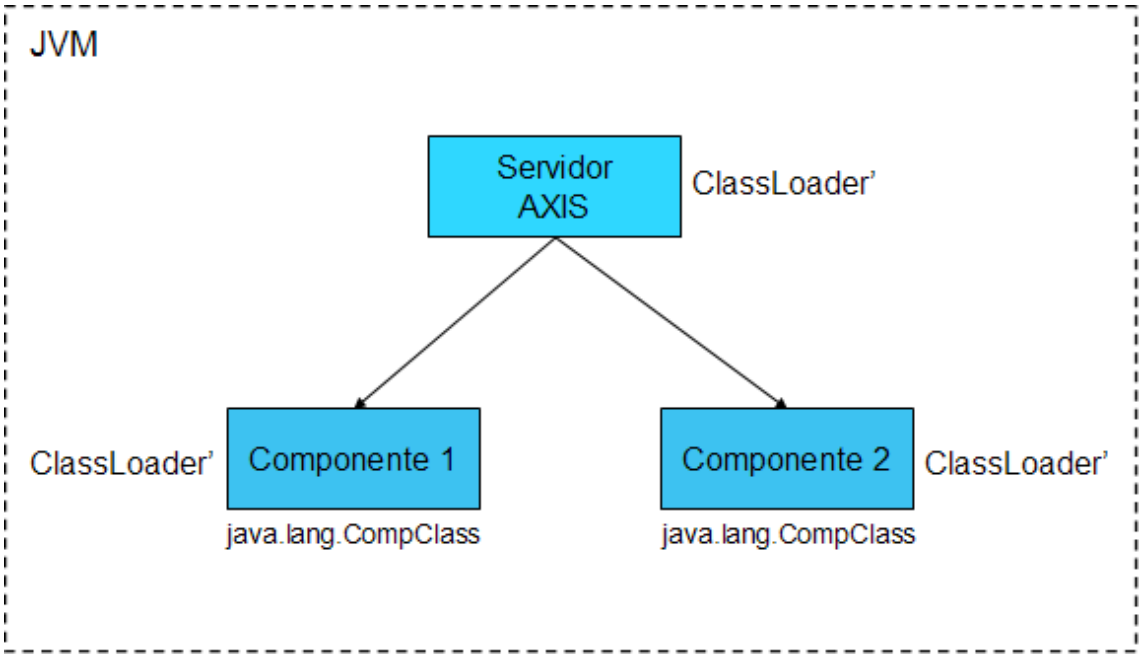


Figura 4.3 - Exemplo de conflito devido a herança de ClassLoader

Para resolver este problema, existe a possibilidade de evitar a herança do *ClassLoader* e instanciar um novo objeto. Com isso, quando os componentes forem executados no servidor, eles instanciarão os seus próprios *ClassLoader*'s, isolando as suas definições dos diretórios das classes e, assim, evitando o conflito de classes com nome e endereço de pacote em comum.

Um *ClassLoader* é propagado por definição quando um objeto é criado através do seu construtor. Quando o construtor é acionado, o objeto criado herda o *ClassLoader* do objeto que o criou. Para evitar a herança, a solução é instanciar um novo *ClassLoader* e criar o objeto a partir dele. Com isso, o novo objeto criado herdará este novo *ClassLoader* e todas as instâncias, geradas posteriormente por este objeto através de chamadas de construtores, também herdarão o novo *ClassLoader*.

No exemplo do servidor e dos componentes da Figura 4.3, a mesma estratégia deve ser feita. Ou seja, ao executar os componentes, um novo *ClassLoader* deve ser gerado, evitando que o componente herde o do servidor. Para realizar essa tarefa, é necessário ter a informação de qual classe dentro do componente seria executada primeiro e que propagaria a instância do *ClassLoader* para os outros objetos. Entretanto, dependendo da funcionalidade requisitada do componente, diversas classes podem iniciar a execução. Além disso, mesmo tendo essa informação, um outro problema ainda maior é fazer o AXIS mudar o seu comportamento para gerar um novo *ClassLoader* na execução de um componente.

Para resolver esse problema, uma solução possível é isolar o acesso do componente para o servidor através de alguma interface de comunicação, interpondo estes dois pontos, ou seja, com a utilização de um *proxy*. No entanto, a existência de um *proxy* de componente para realizar o controle de acesso dos serviços já foi discutido no Capítulo 3. Com isso, este mesmo *proxy* pode ser utilizado para bloquear a propagação do *ClassLoader* do servidor, instanciando um novo *ClassLoader* que será propagado para as classes do componente.

A Figura 4.4 mostra novamente o exemplo do servidor AXIS hospedando os serviços dos componentes. Desta vez, é usada a classe *proxy* para gerar a interface do componente para o servidor. Com isso, o servidor não tem acesso direto ao componente, instanciando apenas a classe *proxy*, que herda o ClassLoader do servidor, ClassLoader'. A classe *proxy*, por sua vez, instancia um novo *ClassLoader* e propaga para as classes dos componentes. No caso do *Proxy 1*, gera o *ClassLoader''* e propaga para o

Componente 1. Já o *Proxy 2*, instancia o *ClassLoader'''* e propaga para o *Componente 2*.

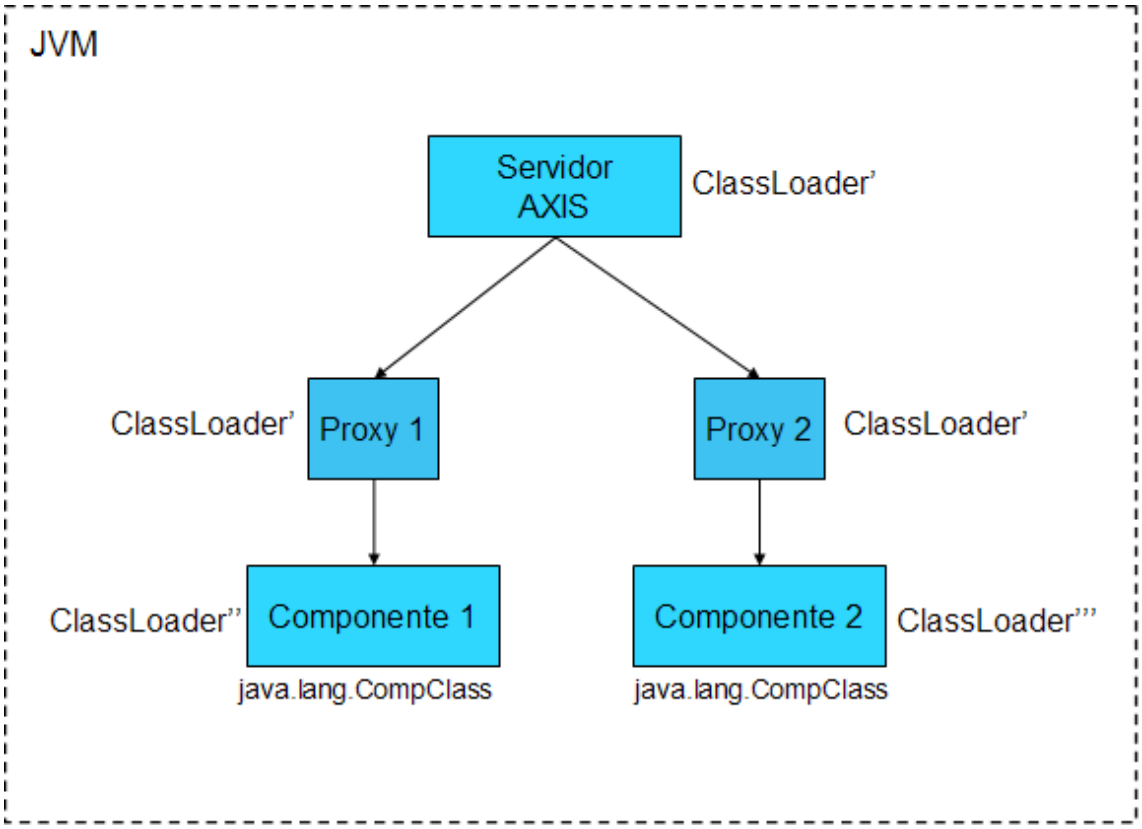


Figura 4.4 - Geração de novo ClassLoader a partir da classe proxy

Com relação ao módulo Legados, o *deploy* físico não se faz necessário para cada solicitação de adaptação de um serviço, já que a aplicação usada para fazer adaptação é uma aplicação genérica que trabalha com os dados no nível da mensagem SOAP, como foi descrito no Capítulo 3. Com isso, essa primeira etapa é realizada apenas uma vez, e a cada solicitação de adaptação de serviço, apenas a segunda etapa é realizada, instanciando um serviço a partir desta aplicação.

4.3.1.2 Deploy lógico

Uma vez que os arquivos necessários estejam armazenados no diretório específico do AXIS, o próximo passo é fazer a comunicação com o AXIS, solicitando a implantação do serviço, ou seja, o *deploy* lógico. Nesta etapa, diversas informações devem ser fornecidas para que o serviço seja implantado com sucesso. Informações gerais, como nome, e parâmetros de configuração do serviço, como estilos de

mensagem, mapeamentos de serialização e deserialização, etc. Essas informações são descritas em um documento XML que segue um esquema específico definido pelo AXIS. O documento utilizado possui a extensão WSDD (Web Service Deploy Descriptor).

Com relação ao módulo específico, essa etapa de implantação é mais delicada, devido ao mapeamento de serialização e deserialização. Como o componente pode ser composto de qualquer tipo de classe, definir o tipo de mapeamento correto para o AXIS é um grande desafio. Esse mesmo problema não ocorre com o módulo Legados, já que este módulo não trabalha com os dados no nível da aplicação, como foi discutido no Capítulo 3.

O mapeamento de (de)serialização serve para instruir o AXIS sobre como realizar a captura dos dados das classes para transformá-los em XML e como preencher as classes com dados, oriundos de um documento XML. Por exemplo, se uma classe seguir um padrão JavaBean (SUN, 2008d), o mapeamento informa que a captura e o preenchimento de um objeto dessa classe é através de métodos GET e SET, respectivamente.

O AXIS já possui mapeamento para as principais classes Java como, por exemplo, *Integer*, *Double*, *String*, *Char*, *Date*, *Array*, dentre outros. Entretanto, um componente pode ser composto de outros tipos de classes não mapeadas pelo AXIS. Com isso, para realizar o mapeamento completo, o módulo Java deve analisar todos os tipos de parâmetros de entrada e saída dos métodos que serão as futuras operações do serviço e verificar se estes parâmetros já são mapeados pelo AXIS. Em caso positivo, não há necessidade de inserir mapeamento no documento WSDD. Porém, em caso negativo, um mapeamento para essa classe é inserido no WSDD. No entanto, como esse mapeamento é feito de forma automática, sem ajuda do produtor do componente, a descoberta do tipo de mapeamento de uma classe se torna uma tarefa difícil. Com isso, assume-se que qualquer classe não mapeada pelo AXIS segue o padrão JavaBean, já que é o padrão mais usado para classes para o armazenamento de dados. Contudo, caso as classes não estejam seguindo esse padrão, o serviço gerado não irá funcionar corretamente.

Por exemplo, considere o método *fatorial*, que possui um parâmetro de entrada do tipo *Integer*, e retorna um valor do tipo *Integer* também, e o método *getTemperatura*, que possui um parâmetro de entrada do tipo *PosiçãoGeografica* e retorna um valor do tipo *Double*. A assinatura dos métodos é definida abaixo:

fatorial(Integer n) : Integer

getTemperatura(PosiçãoGeografica pos) : Double

Ao transformar estes métodos em operações, o Módulo Java verifica se os parâmetros de entrada e saída de cada método já possuem mapeamentos definidos pelo AXIS. No exemplo do método fatorial, a única classe usada é a classe *Integer* que já possui mapeamento definido pelo AXIS. Com isso, não há a necessidade de gerar o mapeamento para essa classe. Já no método *getTemperatura*, a classe *PosiçãoGeografica* não é uma classe mapeada pelo AXIS. Com isso, o módulo Java gera o mapeamento no documento WSDD, inferindo que esta classe segue o padrão JavaBean, embora não se tenha certeza que este seja o mapeamento correto. Ainda no exemplo do método *getTemperatura*, o parâmetro de saída é do tipo *Double*, que já é mapeado pelo AXIS.

4.3.2 Publicação de serviço

De acordo com a nova organização interna da Brechó, um serviço é publicado a partir de uma release pré-existente na Brechó. Vale salientar que, a partir de uma mesma release, pode-se publicar vários serviços.

A Figura 4.5 mostra a tela *Minhas Releases*, que lista as releases de um determinado componente. Cada release possui o botão *Novo Serviço* para cadastrar um novo serviço, além da opção *Serviços* para listar todos os serviços da release.

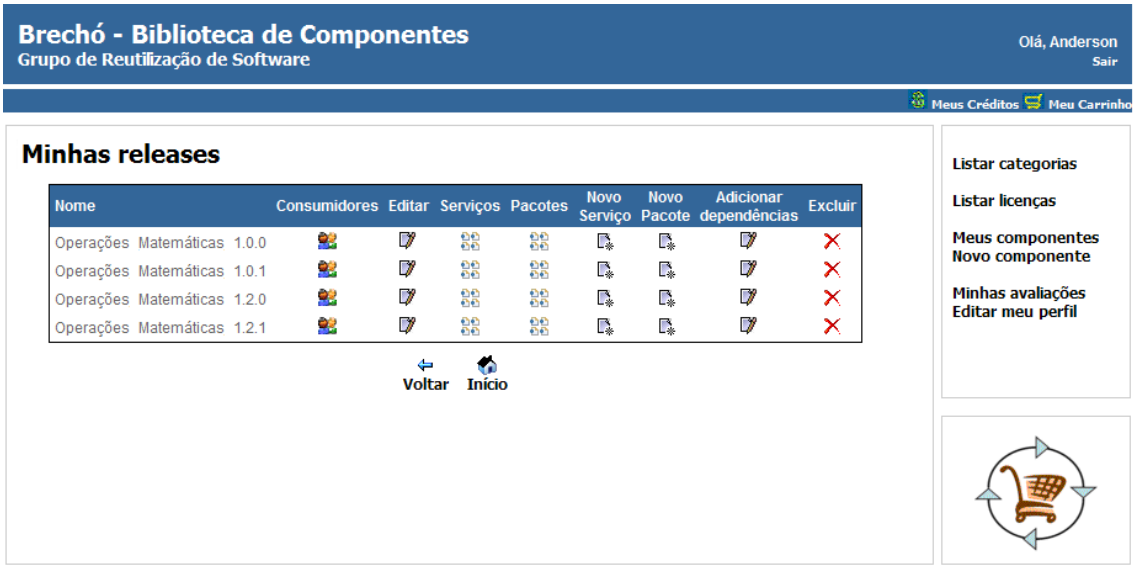


Figura 4.5 – Tela Minhas Releases

A Figura 4.6 mostra a tela de cadastro de serviço. Para se publicar um serviço, as informações necessárias são: nome, descrição, link do documento WSDL do serviço que se queira publicar e preço. Caso o usuário não possua um serviço web para publicar e, conseqüentemente, não possua o WSDL do mesmo, o botão ao lado do campo de WSDL, intitulado *Gerar Web Service,* aciona o mecanismo de geração semi-automática de serviços web.

Figura 4.6 – Tela de cadastro de serviço

Dentre estes dois cenários, onde o usuário possui ou não um serviço web preexistente, entram em ação os módulos, descritos no Capítulo 3, para criar ou adaptar o serviço na biblioteca.

No cenário onde o usuário fornece o serviço web, a etapa de criação terminaria na própria tela de criação de serviço da Figura 4.6, ao se clicar no botão confirmar. Neste momento, de maneira transparente para o usuário, a Brechó se comunica com o módulo Legados, solicitando a adaptação do serviço web Legados fornecido pelo usuário. O módulo retorna o WSDL do novo serviço web adaptado para a Brechó e o serviço é cadastrado.

No cenário onde o usuário não possui o serviço web, seu próximo passo é executar a funcionalidade *Gerar Web Service*, descrita anteriormente. Nesta etapa, a

Brechó irá se comunicar com os módulos específicos de linguagem que coletarão o artefato binário da release e suas dependências e gerarão o serviço web desejado.

Para se comunicar com o módulo correto, a Brechó precisa saber em qual linguagem a release foi implementada. Mais especificamente, em qual linguagem de programação o artefato binário da release foi implementado. Para isso, a Brechó apresenta uma tela com opções de linguagens de programação para que o usuário informe, como mostra a Figura 4.7, a linguagem utilizada. No exemplo, a linguagem selecionada é Java.

Figura 4.7 – Tela de Seleção de linguagem

A partir da informação da linguagem de programação utilizada, a Brechó se comunica com o módulo específico para a linguagem em questão. Neste momento, a Brechó faz uma consulta ao módulo, passando o artefato binário da release como parâmetro e solicitando as possíveis operações de serviços que podem ser geradas a partir desta release. Para o módulo Java, as possíveis operações de serviços são extraídas a partir dos métodos públicos existentes de cada classe pública da aplicação enviada (artefato binário). Com o recebimento da lista das possíveis operações, estas são apresentadas para o usuário pela Brechó, para que este selecione as operações desejadas.

No exemplo utilizado, a Brechó enviou ao módulo Java um artefato binário que possui duas classes públicas: A classe *OperacoesGeometricas* e a classe *OperacoesGerais*. A classe *OperacoesGeometricas* possui os métodos públicos *seno*, *senoh* (seno hiperbólico), *cosseno*, *cossenoh* (cosseno hiperbólico), *tan* (tangente) e *tanh* (tangente hiperbólica). E a classe *OperacoesGerais* possui os métodos públicos *raizQuadrada*, *log*, *min* (mínimo), *max* (máximo) e *fatorial*.

De acordo com a Figura 4.8, as operações são apresentadas em forma de árvore, onde cada nó da árvore representa uma classe e as folhas de cada nó são os seus respectivos métodos. Cada método possui um *checkbox* associado para o usuário fazer a seleção da operação desejada. No exemplo, os nós da árvore são as classes *OperacoesGeometricas* e *OperacoesGerais*, e as folhas são os seus métodos. Os métodos seno e fatorial foram selecionados como as operações do serviço a ser criado.

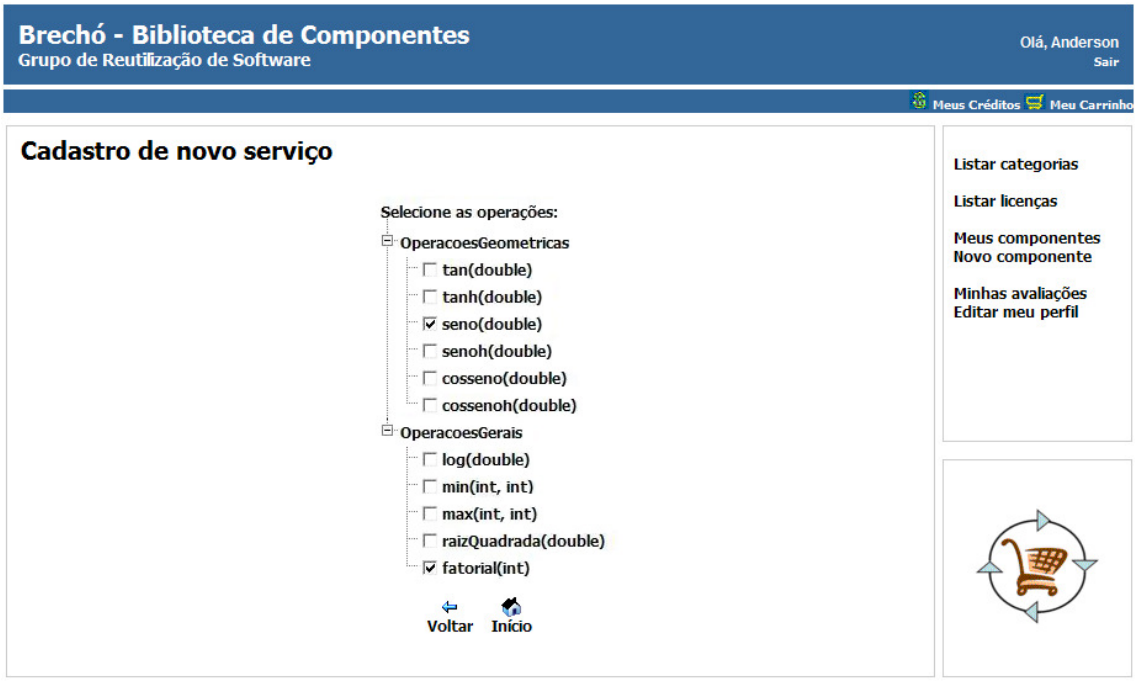


Figura 4.8 – Tela de Seleção de Operações do Serviço

Com as operações selecionadas, a Brechó se comunica novamente com o módulo encarregado para finalizar o processo de criação do serviço web. Dessa vez, a Brechó passa como parâmetro as operações selecionadas pelo usuário e, além disso, passa também as dependências da release que será transformada em serviço web.

Para utilizar um componente como serviço, é necessário saber se este componente possui algum tipo de dependência com outro componente, para garantir o

seu funcionamento normal. Graças ao mecanismo de recuperação de dependências de componente fornecida pela Brechó, apresentado na Seção 4.2, a recuperação das dependências da release que será transformada em serviço se torna possível.

Essa dependência se torna recursiva quando as dependências de uma release também possuem dependências. Por exemplo, de acordo com a Figura 4.9, a release X possui dependências para as releases Y e Z, consideradas dependências diretas, porém Y e Z possuem dependência para a release W. Ou seja, não basta apenas ter as releases Y e Z para que a release X funcione normalmente, já que sem a release W, Y e Z não funcionariam e, conseqüente, a release X também não funcionaria. Com isso, a release W é considerada uma dependência indireta para a release X.

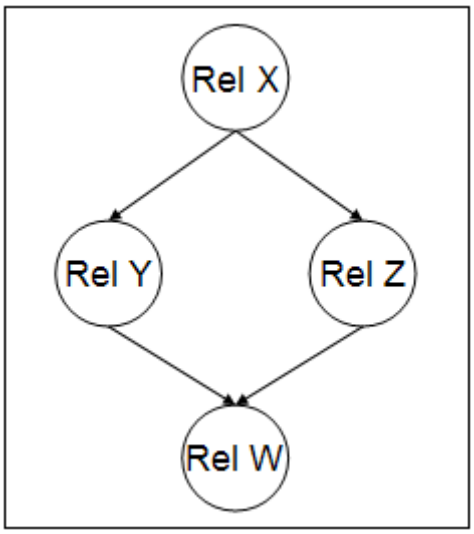


Figura 4.9 – Exemplo de Dependência entre releases

Com as operações desejadas e com todas as dependências da release, diretas e indiretas, o módulo realiza o processo de implantação do serviço no servidor. Um fator interessante a salientar é que o artefato binário da release, fornecido ao módulo pela primeira vez para fazer a consulta das operações, não é novamente fornecido ao módulo. Isto ocorre devido ao mecanismo de geração de sessão que o módulo possui, onde uma sessão é gerada a cada processo de geração de serviço. O artefato binário fica armazenado durante um intervalo de tempo no módulo, esperando a confirmação de geração de serviço feita pela Brechó. Caso não seja recebida nenhuma confirmação de geração de serviço, o artefato é descartado. Com isso, o processo é otimizado, já que evita a troca de informações redundantes entre a Brechó e os módulos.

Após a implantação do serviço, o módulo retorna o documento WSDL do mesmo e a Brechó finaliza o processo de publicação do serviço.

4.3.3 Catalogação dos serviços

Os serviços são catalogados seguindo o padrão de catalogação já usado na Brechó. De acordo com a organização interna na Brechó, com o seus níveis de abstração, a catalogação dos componentes é representada em vários níveis. Por exemplo, quando um usuário acha um componente de interesse na biblioteca, ele navega nos seus níveis de abstração até chegar a um dos seus últimos níveis concretos, ou seja, o nível de pacotes, e, agora, o nível de serviços. Assumindo que o último nível que se queira chegar é o nível de serviço, o usuário no nível de componente lista as distribuições deste componente para escolher a distribuição mais adequada. Dentro da distribuição escolhida, ele lista as releases desta distribuição. Ao escolher uma release, o último passo é listar os serviços desta release. Este percurso é ilustrado iniciando na Figura 4.10.a, onde a partir da lista de componentes são listadas as distribuições do componente *Operações Matemáticas,* que possui apenas uma distribuição, chamada *Default*, ilustrada na Figura 4.10.b. A partir desta distribuição, são listadas as suas releases, que são *1.0.0, 1.0.1, 1.2.0, 1.2.1*, na Figura 4.10.c. Por fim, na Figura 4.10.d, são listados os serviços da release *Operações Matemáticas* 1.2.1, que são: *Operações Diversas* e *Operações Geométricas*.

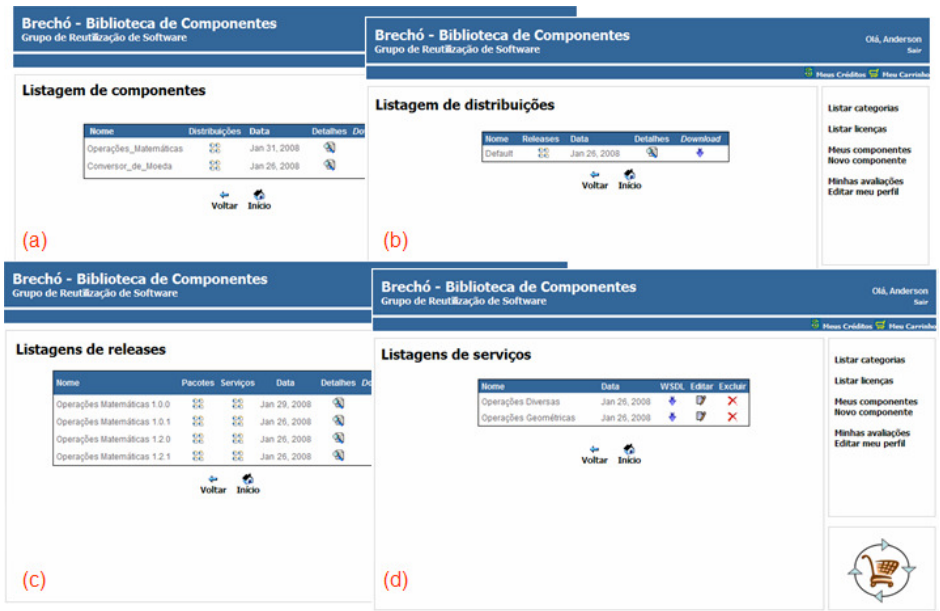


Figura 4.10 - Percurso de seleção de um serviço

Na lista de serviços, ilustrada com maior destaque na Figura 4.11, cada serviço possui um *link* para exibir os seus detalhes e um *link* para o WSDL do serviço que descreve suas operações e as formas de acesso. Além disso, também podem estar presentes os *links editar* e *excluir,* que são exclusivos para o produtor do serviço ou para o administrador da Brechó, onde o primeiro realiza a edição dos dados do serviço e o segundo realiza a remoção do mesmo.

Figura 4.11 - Tela de listagens de serviços

4.3.4 Utilização de serviço

A utilização de um serviço consiste na etapa de consultar o documento WSDL do serviço desejado e realizar o acesso ao mesmo, de acordo com a descrição deste documento. Entretanto, para o usuário fazer o acesso a um serviço publicado na Brechó, é necessário um identificador de sessão. Este identificador de sessão é utilizado para autorizar o uso de um serviço. Cada operação de um serviço publicado na Brechó possui como primeiro argumento o identificador de sessão.

A Brechó utiliza o padrão de controle de sessão *statefull*. Utilizando este padrão, o usuário, ao se autenticar na Brechó, recebe um identificador da sessão estabelecida, denominado *sessionId*. Este *sessionId* pode ser utilizado para acessar vários serviços, sem a necessidade do usuário realizar a autenticação novamente na Brechó. Este

sessionId só se torna inválido quando o tempo de validade da sessão do usuário expira, ou se o próprio usuário finaliza a sessão na Brechó. Outro padrão de controle de sessão é o padrão *stateless*. Este padrão, por sua vez, não trabalha com tempo de vida de sessão, forçando que o usuário se autentique a cada requisição ao servidor.

A Figura 4.12 mostra um exemplo do padrão de controle de sessão *statefull* utilizado na Brechó, onde o usuário realiza a autenticação e recebe o *sessionId* no passo 1. Nos passos 2, 3 e 4, o usuário solicita os serviços a, b e c, respectivamente, passando o *sessionId* como um dos argumentos dos serviços, sem ter a necessidade de se autenticar na Brechó novamente. No último passo, o usuário finaliza a sua sessão com a Brechó.

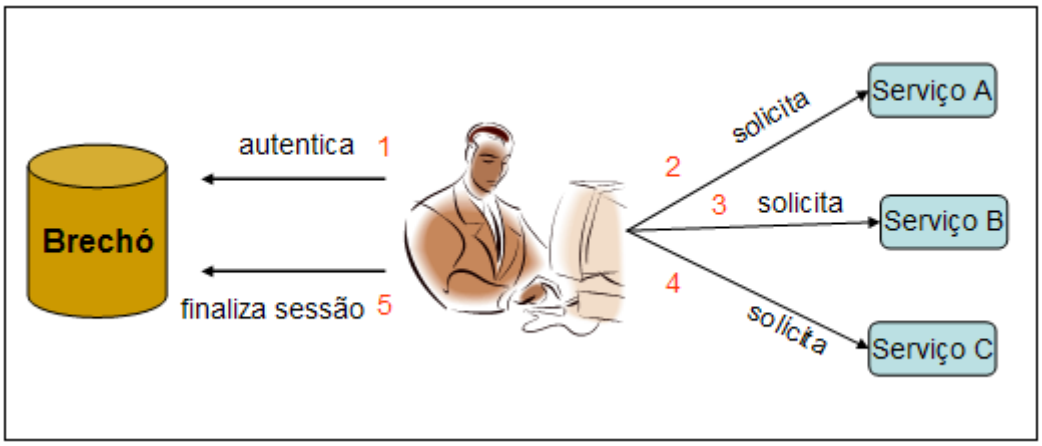


Figura 4.12 - Exemplo do padrão statefull aplicado na Brechó

Quando um serviço publicado na Brechó é acessado, de forma transparente para o usuário, existe uma comunicação entre o serviço e a Brechó. Esta comunicação é feita para realizar a autorização do usuário que está acessando o serviço, e também para realizar o registro da chamada do serviço. A Brechó dispõe de dois serviços para realizar essas tarefas: *authorizeAccess* e *registerCallService*.

O serviço *authorizeAccess* realiza a autorização do usuário para poder acessar o serviço. Essa autorização não se restringe em apenas verificar se a indentificação de sessão (*sessionId*) passada é válida. É também verificado se o usuário pode ou não utilizar o serviço devido a questões tarifárias. Por exemplo, se o usuário não possui créditos suficientes, a utilização do serviço não é autorizada. Uma discussão mais completa sobre tarifação é apresentada na Seção 4.4.

O serviço *registerCallService* tem como objetivo registrar informações de chamadas de serviços feitas pelos usuários da Brechó. Cada registro de chamada possui

informações sobre o serviço solicitado, como, por exemplo, nome, operação efetuada e o usuário que o solicitou.

4.3.5 Verificação de uso dos serviços

Uma característica importante da biblioteca Brechó é o conceito de mapeamento da utilização dos componentes. Toda solicitação de componente feita na Brechó é registrada, guardando informações sobre o componente, como, por exemplo, distribuição, release e pacote escolhido, além de informações sobre o consumidor do componente e sobre a sua forma de aquisição, como, por exemplo, o tipo de licença selecionado. A partir disto, é possível verificar o grau de utilização de qualquer componente publicado na Brechó em qualquer um dos seus níveis (e.g.: distribuição, release, etc.). Por exemplo, é possível obter o grau de utilização de uma release ou de uma distribuição de um componente.

Antes da inserção do mecanismo de serviços de componentes, os componentes eram disponibilizados apenas fisicamente através de *downloads* na Brechó. Com isso, realizar o controle do uso dos componentes não era uma tarefa difícil, já que para recuperar um componente, solicitações eram feitas diretamente para Brechó. Entretanto, com a inserção do mecanismo de serviços de componente, essa característica de mapeamento tornou-se uma preocupação. Devido ao problema do controle de acesso ao serviço, se um serviço fosse publicado na Brechó sem que este sofresse alguma alteração na sua estrutura para suportar tal característica, o seu uso não poderia ser registrado e o mapeamento da utilização do serviço seria uma tarefa quase que impossível. Entretanto, com a solução provida pelos módulos, esse problema foi solucionado.

Com isso, além dos outros níveis de abstração de componente, a partir do nível de serviços, é possível também obter informações sobre utilização do componente. A Figura 4.13 apresenta a tela de consumidores de serviço. Para cada consumidor listado, é informado o seu e-mail, a operação solicitada e a data de acesso.

Figura 4.13 - Tela de Consumidores

É possível também listar os serviços usados por um determinado usuário. Esta funcionalidade é apresentada na Seção 4.4.1.

4.4 Tarifação

A abordagem de tarifação em uma biblioteca, apresentada no Capítulo 3, tem como principais necessidades: definição do modelo de tarifação, definição de formas de precificação de um componente e rastreabilidade do consumo dos componentes. Para atender a estas necessidades, algumas características e funcionalidades foram implantadas na Brechó.

Nas próximas Seções, é apresentado um detalhamento sobre a implementação destas características e funcionalidades na Brechó. Começando pela Seção 4.4.1, que define o modelo de tarifação aplicado e as suas conseqüências na conta do usuário. A seguir, a Seção 4.4.2 detalha a forma de atribuição de valor a um componente e suas características peculiares devido à organização interna da Brechó. E, por fim, a Seção 4.4.3 explica o processo de aquisição de um componente com o mecanismo de tarifação em funcionamento.

4.4.1 Modelo de tarifação

O modelo de tarifação implementado na Brechó é uma combinação dos modelos pós-pago e pré-pago, discutidos no Capítulo 3. Na verdade, ele se assemelha ao modelo de conta bancária, onde o usuário possui um limite para gastar com aquisições de componentes e, enquanto isso, ele pode adicionar créditos para compensar o seu débito (semelhante ao modelo pós-pago), ou simplesmente para deixar acumulado (semelhante ao modelo pré-pago) na sua conta para futuras aquisições de componentes.

Por exemplo, um usuário que tem um limite de 200 pode consumir componentes até um valor total de 200, mesmo que sua quantidade de créditos seja igual à zero. No entanto, para consumir mais componentes, além do limite de 200, o usuário deverá adicionar créditos para compensar o seu débito. Já no caso onde o usuário possui créditos e limite igual à zero, o usuário deverá acrescentar créditos na Brechó antes de consumir algum componente.

Para suportar tal modelo de tarifação, as contas de usuário tiveram que ser alteradas, passando agora a guardar informações de crédito e limite do usuário. Além disso, os usuários passaram a necessitar de maior interação com as suas contas, para saber o seu limite, quanto têm de crédito, ou ver suas últimas transações realizadas.

Com isso, surgiu uma nova funcionalidade relacionada ao perfil de um usuário, chamada *Meus Créditos*, que atende as necessidades descritas acima. Segundo a Figura 4.14, esta funcionalidade fica localizada na barra de informações do usuário, no canto superior direito da tela principal da Brechó, ao lado da funcionalidade *Meu Carrinho,* que lista os componentes adicionados ao carrinho de compras do usuário.

Figura 4.14 - Tela Principal

Ao chamar a funcionalidade *Meus Créditos*, uma nova tela é exibida, como apresentado na Figura 4.15. Nesta tela, são informados o limite, o crédito e as últimas transações do usuário. Cada transação pode ser definida em dois tipos: *Download* ou Serviço. Além disso, em cada tipo é especificado o nome do componente requisitado, o valor e a data da transação. Se o usuário tiver interesse de ver mais detalhes sobre alguma transação, é possível utilizar a opção *Detalhes* da transação desejada.

Figura 4.15 - Tela Meus Créditos

Outras funcionalidades importantes para o gerenciamento da conta do usuário é a definição de limite e adição de créditos. Estas funcionalidades, porém, ficam sob a responsabilidade do administrador da biblioteca. Para definir o limite ou adicionar crédito, o administrador lista os usuários cadastrados na Brechó e utiliza a opção *Edit* para editar a conta de um usuário específico. Entretanto, para facilitar o processo de adição de créditos, já que é um processo mais requisitado, existe uma funcionalidade específica na própria tela de listagem de usuários, como apresentado na Figura 4.16.

Figura 4.16 - Adicionando créditos a um usuário

4.4.2 Atribuição de valor a um componente

A parte de atribuição de valor aos componentes fica a cargo apenas dos produtores. Nesta etapa, os produtores de componentes, ao criarem os seus componentes, estipulam um preço para a sua aquisição, através de *download*, ou pela sua utilização, através de serviços.

De acordo com a organização interna da Brechó, os níveis de pacote e os níveis de serviço são os níveis finais para a aquisição ou utilização, respectivamente, de um componente. Isto significa que são nestes níveis que o produtor, na hora de criá-los, deve atribuir um valor. Entretanto, o nível de pacotes e serviços são formas de disponibilização de uma release, onde se encontram os artefatos do componente em uma determinada versão. Estes artefatos armazenam os produtos gerados durante o momento

de desenvolvimento de software, como, por exemplo, código fonte, manuais, casos de uso, etc. Com isso, para dar uma maior precisão na precificação de um componente quando uma release é publicada na Brechó, os seus artefatos são precificados individualmente.

Essa estratégia de precificar um componente no nível de artefato é uma tentativa de automatizar a precificação no nível de pacotes e serviços, já que estes são formados a partir dos artefatos. A Figura 4.17 mostra um exemplo do esquema destes níveis. Uma release qualquer é definida por uma coleção de artefatos: código fonte (src), documentação (doc), binário (bin), etc. O pacote gerado a partir desta release é composto pelos artefatos código fonte e documentação. Por outro lado, o serviço gerado representa um conjunto das funcionalidades do artefato binário, já que este é o artefato que o serviço utiliza para rodar a aplicação. Por exemplo, um componente é composto das operações matemáticas (seno, cosseno, fatorial, etc.), no entanto, um serviço desse componente é composto apenas da operação fatorial.

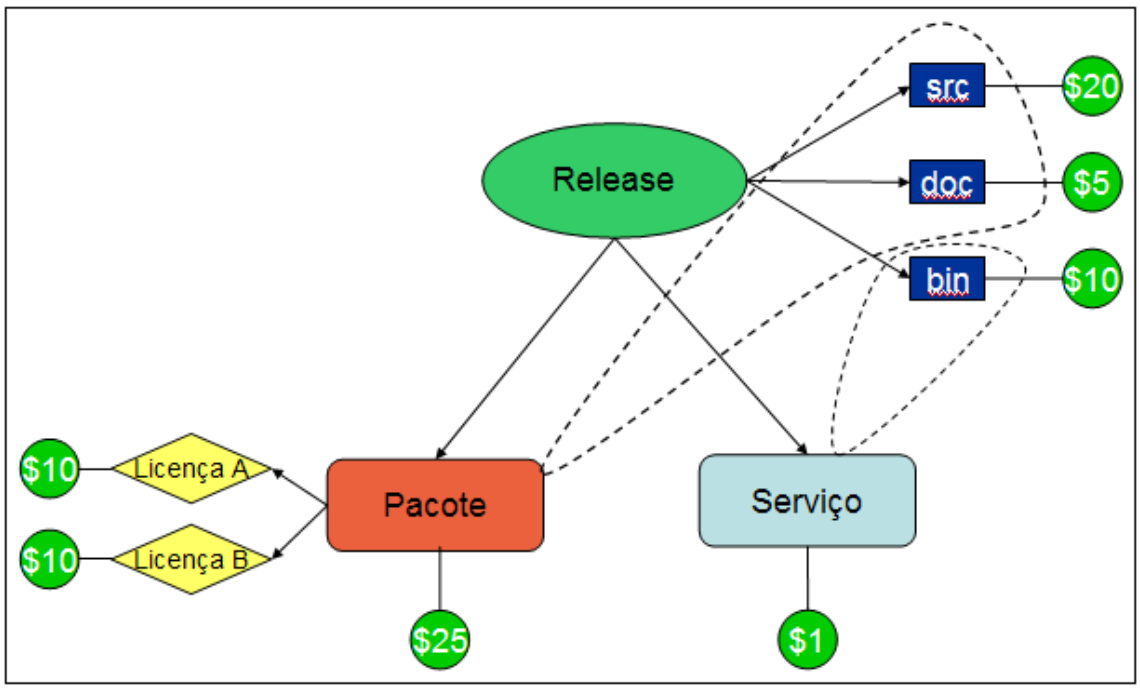


Figura 4.17 - Exemplo de precificação em um componente

Um outro elemento que agrega valor a um componente são as suas licenças associadas. Cada licença define direitos e deveres que o usuário possui com a aquisição de um componente. Por exemplo, ao escolher uma determinada licença, o usuário pode adquirir o direito de um ano de manutenção para um componente, mas pode também

estar sujeito a regras que restringem o seu uso. Na Brechó, estas licenças estão associadas no nível de pacote, definindo direitos e deveres na aquisição de um pacote.

Continuando no exemplo da Figura 4.17, a partir dos preços dos artefatos, é possível inferir um valor para um pacote, já que este representa uma configuração destes artefatos. Os artefatos src e doc têm o valor de 20 e 5, respectivamente. Então, pode-se inferir que o valor do pacote é 25. Além disso, se o pacote for adquirido com a licença A, o seu valor será acrescido de 10, e com a licença B, o seu valor será acrescido de 5. Já para o nível de serviços, como um serviço é gerado a partir do artefato binário, é possível extrair um valor adequado a partir do valor do próprio artefato binário. Uma técnica que poderia ser utilizada é a técnica do valor amortizado, comentada no capítulo de abordagem, na Seção 3.4.3. No exemplo, o serviço ficou com o valor igual a 1.

A Figura 4.18 mostra a tela de criação de releases na Brechó. Para criar uma release, é necessário fornecer: nome, descrição e seus artefatos associados. Para cada artefato associado, é necessário, também, informar o seu preço.

Figura 4.18 - Tela de Cadastro de Release

A Figura 4.19 mostra a tela de criação de pacotes. Ao estilo da tela de criação de release, é necessário fornecer nome e descrição do pacote. Outros itens necessários são os artefatos associados e as possíveis licenças que o pacote irá disponibilizar.

Sobre a questão da precificação do pacote, o seu preço pode ser informado manualmente no formulário. Entretanto, se o usuário desejar que a Brechó sugira um preço para o pacote, esta sugestão pode ser obtida através do botão *Preço Sugerido*, onde a partir dos artefatos selecionados pelo usuário, a Brechó realiza o cálculo da soma dos preços dos artefatos, definidos durante a criação da release, e retorna para o usuário. No exemplo da Figura 4.19, o usuário selecionou o artefato binário e código fonte aos quais foram estipulados os valores de 10 e 20, respectivamente, no exemplo da Figura 4.18. Com isso, ao solicitar o preço sugerido, a Brechó preencheu o campo *preço* do formulário com o valor 30.

Além da precificação do pacote, existe também a precificação das licenças associadas ao pacote. Cada licença associada deve possuir um preço que será apresentado como opção na aquisição do pacote. O valor total do pacote será o valor definido para os artefatos associados mais o valor da licença selecionada. No exemplo da Figura 4.19, o usuário associou a licença *Suporte telefônico por 1 ano* e a licença *Manutenção por 1 ano* ao pacote e atribuiu um preço para cada licença. Para a licença *Suporte telefônico por 1 ano*, foi atribuído o valor de 10, e para a licença *Manutenção por 1 ano*, foi atribuído o valor de 20.

Figura 4.19 - Tela de Cadastro de Pacote

Por último, a tela de cadastro de serviço é ilustrada na Figura 4.20, que possui também o campo *preço* para especificar o preço do serviço.

Figura 4.20 - Tela de Cadastro de serviço

4.4.3 Aquisição de um componente

A Brechó utiliza o conceito de carrinho de compras para o consumidor adquirir seus componentes. Os componentes selecionados são adicionados no carrinho e, quando o consumidor finaliza a sua compra, recebe um arquivo compactado contendo todos os componentes adquiridos.

Para adicionar um componente no carrinho, o usuário ao descobrir um componente na Brechó deve navegar nas opções que o componente oferece. Ou seja, navegar entre os níveis de distribuição, release e pacote. Chegando ao nível de pacotes, o usuário realiza a adição do componente no carrinho.

A Figura 4.21 mostra a tela *Seleção de componentes para download,* onde pacotes são selecionados e adicionados no carrinho. Cada pacote possui suas opções de licenças e, para cada configuração de pacote e licença, o preço do componente pode assumir um valor diferente. Além das opções de pacote do componente que o consumidor selecionou, a Brechó apresenta também as opções de pacote das dependências deste componente. Com isso, tornam-se desnecessárias a tarefa do usuário de procurar as dependências na Brechó e a tarefa de descobrir quais são as dependências do componente em alguma documentação do mesmo.

Figura 4.21 - Tela de Seleção de componentes para download

Para visualizar os componentes adicionados no carrinho, o usuário acessa a funcionalidade *Meu carrinho*. Dentro da tela *Meu carrinho*, para cada componente adicionado, são informados a distribuição, a release, o pacote e a licença escolhida, além de seu preço. O valor total dos componentes é informado no final da lista, como mostra a Figura 4.22.

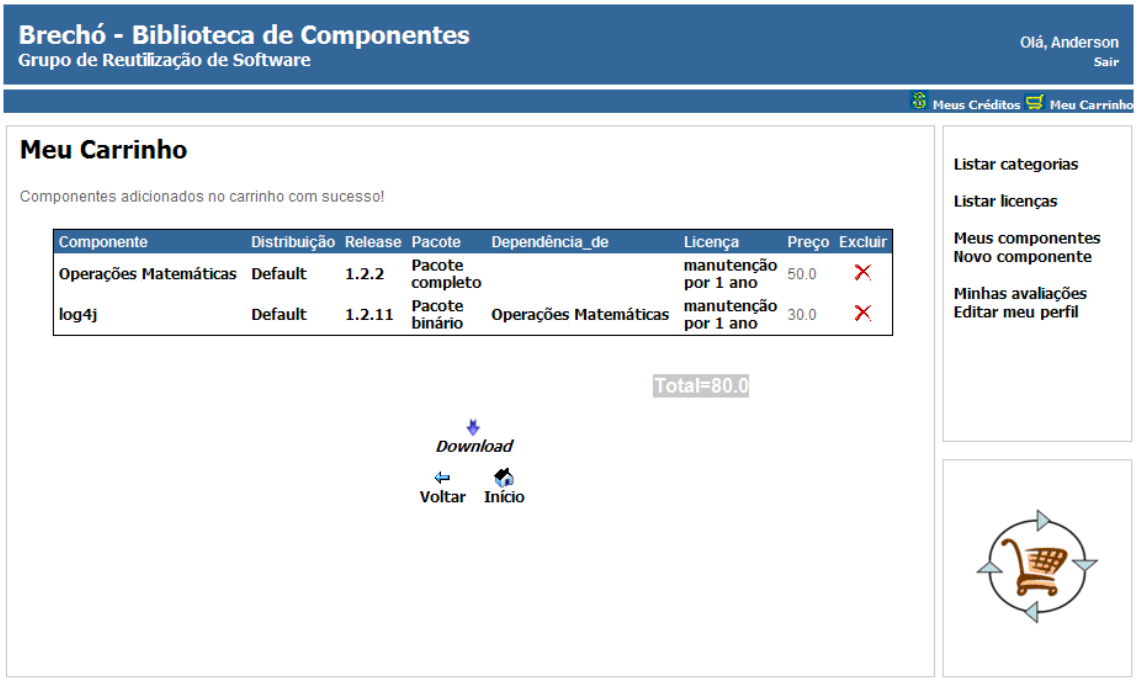


Figura 4.22 - Tela Meu carrinho

Para efetuar a compra, o usuário deve clicar no ícone *download* que se encontra na parte inferior da lista dos componentes adicionados no carrinho. Neste momento, é retornado para o usuário um arquivo compactado com todos os componentes do carrinho. Porém, antes disso, a Brechó faz uma verificação na conta do usuário para assegurar que ele possui recursos para realizar essa compra. Ou seja, se o usuário possui créditos disponíveis pra compra, ou se o limite cobre o valor da compra. Caso nenhuma destas condições seja satisfeita, a compra não é efetuada, e uma mensagem é retornada ao usuário informando o motivo.

4.5 Considerações finais

Este capítulo apresentou a forma de implementação de uma infra-estrutura de serviços de componentes e tarifação na biblioteca de componentes Brechó. O projeto Brechó foi o grande motivador deste trabalho, já que a partir do seu uso, foram detectadas as necessidades que fundamentaram este trabalho.

Em linhas gerais, a infra-estrutura implantada de serviços na Brechó, com a adição do mecanismo de serviço de um componente, manteve a compatibilidade com as funcionalidades já presentes na Brechó, como por exemplo, o controle de acesso e o mapeamento de utilização de um componente, além de aumentar as perspectivas de reutilização de um componente.

O mecanismo de tarifação implementado teve como base mecanismos aplicados em outros domínios, fora do domínio de componente de software. Por exemplo, modelos de tarifação baseados nos modelos aplicados em telefonia celular e de crédito bancário, ou a forma de aquisições de componente usando o conceito de carrinho utilizado em sites de compras. Em suma, o objetivo foi adaptar a realidade já existente de tarifação em outros domínios para o domínio de componentes de software, já que a tarifação de componentes ainda não é um assunto muito explorado na literatura.

Capítulo 5 - Conclusão

5.1 Epílogo

Com o avanço da utilização de serviços web, os componentes de software transitam da simples utilização física para a utilização remota através de serviços. Com isso, uma biblioteca, que é comumente um local de armazenamento de componentes físicos, precisa evoluir para poder também disponibilizar serviços dos componentes.

Além disso, dependendo da forma de uso de uma biblioteca, componentes e serviços precisam ser tarifados. Com isso, mecanismos de controle de acesso devem estar presentes em uma biblioteca.

Visando atender a estas necessidades, este trabalho de pesquisa apresentou uma infra-estrutura de serviços e mecanismos de tarifação para uma biblioteca de componentes de software.

5.2 Contribuições

Esta monografia apresentou os resultados de um trabalho de pesquisa que visa à aplicação de uma infra-estrutura de serviços e mecanismos de tarifação em uma biblioteca de componentes.

As principais contribuições relacionadas à infra-estrutura de serviços são:

- Especificação e implementação de mecanismos de geração automática de serviços a partir de componentes;
- Os serviços publicados em uma biblioteca podem ser tanto internos, criados pela própria biblioteca, quanto externos, fornecidos pelos produtores dos componentes;
- Especificação e implementação de mecanismos de adaptação de serviços externos que são publicados em uma biblioteca;
- Todos os serviços publicados em uma biblioteca possuem mecanismos de controle de acesso; e

- A utilização dos serviços é monitorada e registrada pela biblioteca.

Com relação à aplicação de tarifação em uma biblioteca, as principais contribuições relacionadas são:

- Tarifação aplicada em todos os meios de aquisição ou utilização de um componente em uma biblioteca, ou seja, componentes e serviços;
- Definições de formas de precificação diferentes de acordo com a forma de aquisição de um componente;
- Especificação e implementação de um modelo de tarifação;
- Especificação e implementação de mecanismos de sugestão automática de preços de componentes; e
- Especificação e implementação de mecanismos de visualização de histórico de compras e vendas de componente.

5.3 Limitações

A partir de uma análise crítica sobre a abordagem proposta e sua implementação, algumas limitações foram encontradas. Estas limitações são detalhadas nesta Seção.

5.3.1 Mapeamento de (de)serialização de um serviço

Uma das grandes limitações da geração automatizada de serviços está relacionada à falta de conhecimento suficiente sobre o componente que será transformado em serviço. Mesmo que a reflexão ajude no processo de reconhecimento do componente, indicando suas classes e métodos, essa informação ainda não é suficiente para realizar a implantação de um serviço, já que não identifica a interface de acesso das classes. Essa informação é importante para que o servidor saiba manipular os dados da classe, quando realizar algum processo de serialização ou deserialização de um objeto.

Como foi apresentado no Capítulo 4, para tentar minimizar esse problema, adotou-se que, para qualquer classe que tenha que fazer algum tipo de mapeamento, esta

seria considerada do tipo JavaBeans. Esse padrão foi escolhido, devido a ser um dos padrões de classes de dados mais usados. Com isso, a probabilidade das classes seguirem este padrão é maior. Entretanto, existe a possibilidade de erro, que culmina na geração de serviços configurados inadequadamente, que apenas serão descobertos na sua execução, já que não há nenhum tipo de teste que verifique isso à priori.

5.3.2 Pontos de acesso reduzidos na geração do serviço proxy

Quando um serviço legado possui mais de um ponto de acesso para uma ou mais operações, estes pontos de acesso são reduzidos quando um serviço *proxy* é gerado. De acordo com a implementação do serviço *proxy*, este possui apenas um ponto de acesso. Com isso, se o serviço legado possuir mais de um ponto de acesso, o serviço *proxy* selecionará um deles para fazer o redirecionamento da requisição. Essa seleção é feita apenas uma vez durante a geração do serviço *proxy*, e não a toda execução do serviço, selecionando um ponto de acesso de cada vez e realizando a distribuição de carga das requisições, como foi proposto no Capítulo 3.

Além disso, a abordagem de distribuição de carga não funciona para todos os tipos de serviço legado. Se um serviço legado possuir mais de um ponto de acesso, onde as suas definições de protocolo de comunicação forem diferentes, o serviço *proxy* não conseguirá executar a distribuição de carga para estes pontos. Por exemplo, um serviço legado possui dois pontos de acesso, onde um está configurado para receber requisições HTTP e o outro está configurado para receber requisições SMTP. Como o serviço *proxy* possuirá apenas um ponto de acesso, este ponto de acesso, dependendo da sua configuração, receberá apenas requisições HTTP ou apenas requisições SMTP, causando a perda de um dos pontos de acesso do serviço legado.

A abordagem ideal seria o serviço *proxy* possuir o mesmo número de pontos de acesso do serviço legado, onde cada ponto de acesso do serviço *proxy* redirecionasse para apenas um ponto de acesso do serviço legado e possuísse a mesma definição de protocolo de comunicação.

5.3.3 Geração automática de serviços apenas para componentes escritos em Java

Dentre os possíveis módulos específicos de linguagem de programação a serem implementados, apenas o módulo Java foi de fato implementado. No entanto, a Brechó está implementada de acordo com a arquitetura proposta por este trabalho, apresentada no Capítulo 3, onde módulos podem ser adicionados a qualquer momento, garantindo, assim, a evolução do sistema.

5.3.4 Análise de conflito de nomes durante a geração de serviço

Durante o processo de geração de um serviço a partir de um componente escrito em Java, o usuário seleciona as operações do futuro serviço a partir dos métodos das classes do componente. Nesta etapa, existe a possibilidade de alguma destas classes possuírem métodos com assinaturas iguais, porém com possíveis funcionalidades diferentes. No entanto, se estes métodos forem selecionados para formar um mesmo serviço, isso ocasionará em erro, pois um serviço não pode possuir duas operações com assinaturas iguais.

Para evitar este problema, quando o usuário selecionar métodos com nomes e assinaturas iguais, a biblioteca deve informar ao usuário sobre o conflito antes de tentar gerar o serviço e pedir um nome alternativo para os métodos.

5.4 Trabalhos futuros

A realização deste trabalho de pesquisa levou à construção de uma infra-estrutura que abre novas perspectivas de pesquisa a serem exploradas em trabalhos futuros. Alguns destes trabalhos futuros são detalhados nesta Seção.

5.4.1 Ambiente de orquestração de serviços

A partir da infra-estrutura de serviços proposta por este trabalho, implantada em uma biblioteca, um possível trabalho futuro é a geração de um ambiente de orquestração

de serviços, permitindo a construção de novos serviços a partir da combinação de serviços previamente publicados na biblioteca.

Este ambiente poderia estar incorporado à biblioteca, onde através de uma interface web, o usuário constrói um serviço a partir da combinação de outros serviços previamente publicados na biblioteca. Além disso, este serviço gerado poderia ser publicado na biblioteca para ser consumido e o seu valor mínimo calculado automaticamente a partir dos valores dos serviços incorporados. Este valor ainda poderia ser acrescido pelo produtor do serviço para obter alguma porcentagem de lucro na venda do mesmo.

5.4.2 Mecanismos de sugestões e promoções de compras

Na maioria dos sites de compras, mecanismos de sugestões e promoções de compras são bastante utilizados, informando produtos que são frequentemente comprados quando outros são adquiridos e fornecendo descontos para a compra de um conjunto de produtos.

Em uma biblioteca de componentes, estes mecanismos também poderiam ser interessantes para incentivar o consumo, informando os componentes e os serviços frequentemente usados a partir da compra de algum componente e fazendo promoções para a compra de um conjunto de componentes. Por exemplo, se um componente for comprado juntamente com as suas dependências, há um desconto de cinqüenta por cento.

5.4.3 Modelo econômico de tarifação de componentes

Este assunto foi discutido durante a apresentação da abordagem de tarifação em uma biblioteca de componentes, no Capítulo 3. Entretanto, a discussão apresentada não teve como objetivo ser completa sobre o assunto.

Um modelo econômico de tarifação de componentes é um assunto complexo e deve ser tratado mais profundadamente em um trabalho separado. Devem ser feitos estudos mais aprofundados para descobrir os verdadeiros fatores que implicam o valor de um componente tanto na sua aquisição física, quanto na utilização de um serviço. Em relação a este último, a tarifação é ainda algo mais subjetivo. Deve-se descobrir o grau

de uso de um serviço, para se estimar um valor que dê o mesmo retorno que a venda do componente. Uma solução apresentada neste trabalho foi a utilização de amortização. No entanto, não houve um estudo mais aprofundado de como aplicá-la na Brechó.

Referências Bibliográficas

APACHE, 2008, "Axis 1.4". In: http://ws.apache.org/axis, accessed in February 18.

ALTOVA, 2008, "XMLSpy". In: http://www.altova.com/products/xmlspy/xml_editor.html, accessed in May 16.

BOOTH, D., HAAS, H., MCCABE, F., *et al.*, 2004, "Web Services Architecture - W3C Working Group Note". In: http://www.w3.org/TR/ws-arch, accessed in April 18.

CAREY, M.J., 2008, "SOA What?", IEEE Computer Society Press, v. 41, n. 3 (March), pp. 92-94.

CEDERQVIST, P., 2003, *Version Management with CVS*, Free Software Foundation.

COMPONENTSOURCE, 2008, "ComponentSource". In: http://www.componentsource.com, accessed in February 18.

CURBERA, F., DUFTLER, M., KHALAF, R., *et al.*, 2002, "Unraveling the Web Services Web – An Introduction to SOAP, WSDL, and UDDI", IEEE Internet Computing, v. 6, n. 2 (March/April), pp. 86-93.

DEVSHOP, 2008, "Devshop". In: http://www.devshop.cz, accessed in April 12.

DIGITALASSETS, 2008, "DA Manager eS". In: http://www.digitalassets.com.br/da-manager, accessed in February 18.

ESTUBLIER, J., 2000, "Software Configuration Management: A Roadmap". In: *International Conference on Software Engineering (ICSE), The Future of Software Engineering*, pp. 279-289, Limerick, Ireland, June.

FLASHLINE, 2008, "Flashline Registry". In: http://www.flashline.com, accessed in April 18.

GAMMA, E., HELM, R., JOHNSON, R., *et al.*, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

GOOGLE, 2008, "Google Services". In: http://www.google.com.br/intl/pt-BR/options, accessed in February 18.

GOSLING, J., JOY, B., STEELE, G., *et al.*, 2005, *The Java Language Specification*, 3rd. ed., Addison-Wesley Professional.

HUHNS, F., SINGH, M. P., 2005, " Service-Oriented Computing: Key Concepts and Principles", IEEE Internet Computing, v. 9, n. 1 (January/February), pp. 75-

81.

IBM, 2008a, "ClearCase". In: http://www-306.ibm.com/software/awdtools/clearcase, accessed in February 18.

IBM, 2008b, "WebSphere Service Registry and Repository". In: http://www-306.ibm.com/software/integration/wsrr, accessed in February 18.

IETF, 1999, *RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1*. Internet Engineering Task Force.

IETF, 2001, *RFC 2821 - Simple Mail Transfer Protocol - SMTP*. Internet Engineering Task Force.

INFOWORLD, 2003, "ComponentSource buys part of Flashline". In: http://www.infoworld.com/article/03/04/08/HNflashline_1.html, accessed in April 18.

ISO, 1986, ISO 8879, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), International Organization for Standardization.

JACOBSON, I., GRISS, M., JONSSON, P., 1997, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley Professional.

KRUEGER, C.W., 1992, "Software Reuse", ACM Computing Surveys, v. 24, n. 2 (June), pp. 131-183.

LEE, R., JENADOSS, K., 2003, "An Architecture Model for Component Composition & Integration". In: *Proceedings of the 4th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 12-19, Lubeck, Germany, October.

LEE, R., KIM, H., YANG, H., 2004, "An Architecture Model for Dynamically Converting Components into Web Services". In: *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pp. 648-654, Busan, Korea, November.

LOGICLIBRARY, 2008, "Logidex". In: http://www.logiclibrary.com/solutions/logidex.php, accessed in February 18.

MICROSOFT, 2008, "Team Foundation Server". In: http://msdn2.microsoft.com/en-us/teamsystem/aa718934.aspx, accessed in February 18.

MURTA, L.G.P., 2006, *Odyssey-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes*, Tese de Doutorado, COPPE, UFRJ, Rio de Janeiro, Brasil.

OASIS, 2005, *Universal Description Discovery & Integration (UDDI) 3.02*, OASIS Standard.

PROLIBTOOLS, 2008, "ProLib Tools". In: http://shop.prolib.de, accessed in April 12.

RAPOSO, R.R., 2007, *Brecho-ABC: Uma Abordagem Integrada para Avaliação, Busca e Categorização de Componentes de Software*, Monografia de Graduação, IM, UFRJ, Rio de Janeiro, Brasil.

SABBOUH, M., JOLLY, S., ALLEN, D., *et al.*, 2001, "Interoperability". In: *W3C Workshop on Web Services*, San Jose, USA, April.

SCHWARTZ, J., 2008, " The Network is the Computer". In: http://blogs.sun.com/jonathan/entry/the_network_is_the_computer, accessed in April 8.

SMITH, Z., 2006, "Applied Reflection: Creating a dynamic Web service to simplify code", In: http://downloads.techrepublic.com.com/download.aspx?docid=264551, accessed in March 27.

SOFTCHOICE, 2008, "Softchoice". In: http://www.softchoice.com, accessed in April 12.

SUN, 2008a, "Sun Grid". In: http://www.network.com, accessed in March 2.

SUN, 2008b, "Solaris". In: http://www.sun.com/solaris, accessed in March 2.

SUN, 2008c, "Java Virtual Machine". In: http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html, accessed in February 27.

SUN, 2008d, "Enterprise JavaBeans Technology". In: http://java.sun.com/products/ejb, accessed in February 27.

TAVIS, G., 2008, "Understanding the Java ClassLoader". In: http://blogs.sun.com/jonathan/entry/the_network_is_the_computer, accessed in April 8.

W3C, 1998, *Document Object Model (DOM)*, World Wide Web Consortium.

W3C, 2002, *Web Services*, World Wide Web Consortium.

W3C, 2006, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, World Wide Web Consortium.

W3C, 2007a, *Simple Object Access Protocol (SOAP) 1.2 (Second Edition)*, World Wide Web Consortium.

W3C, 2007b, *Web Service Description Language (WSDL) 1.1*, World Wide Web Consortium.

W3C, 2008, "World Wide Web Consortium". In: http://www.w3.org, accessed in February 18.

WERNER, C. M. L., 2008a, "Projeto Brechó". In: http://reuse.cos.ufrj.br/site/pt/index.php?option=com_content&task=view&id=37&Itemid=46, accessed in April 18.

WERNER, C. M. L., 2008b, "Grupo de Reutilização de Software da COPPE". In: http://reuse.cos.ufrj.br, accessed in April 18.

WERNER, C.M.L., BRAGA, R.M.M., "Desenvolvimento baseado em Componentes". In: *Anais do Simpósio Brasileiro de Engenharia de Software*, *Minicursos*, João Pessoa, PB, Brasil, outubro, 2000, pp. 297-329.

WERNER, C. M. L., MURTA, L. G. P., LOPES, M., DANTAS, A., LOPES, L. G., FERNANDES, P., PRUDENCIO, J. G., MARINHO, A., RAPOSO, R., "Brechó: Catálogo de Componentes e Serviços de Software". *In: XXI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, João Pessoa, october. XXI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, 2007. p. 24-30.

WIKEPEDIA, 2008a, "Wikimedia Foundation, Inc.". In: http://en.wikipedia.org/wiki/Prepaid, accessed in April 18.

WIKEPEDIA, 2008b, "Wikimedia Foundation, Inc.". In: http://en.wikipedia.org/wiki/Postpaid, accessed in April 18.

WINDLEY, P.J., 2006, "SOA Governance: Rules of the Game". In: http://akamai.infoworld.com/pdf/special_report/2006/04SRsoagov.pdf, accessed in April 18.