DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"A utilização de serviços web providos por SOA em geradores de aplicação desenvolvidos com linguagens de padrões"

ALUNA: Kamila Rios da Hora Rodrigues
ORIENTADORA: Profa. Dra. Rosângela A. D. Penteado

São Carlos
2014

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

KAMILA RIOS DA HORA RODRIGUES

A UTILIZAÇÃO DE SERVIÇOS WEB PROVIDOS POR SOA EM GERADORES DE APLICAÇÃO DESENVOLVIDOS COM LINGUAGENS DE PADRÕES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de mestre em Ciência da Computação, área de concentração: Engenharia de Software.

Orientadora: Profª. Drª. Rosângela Ap. D. Penteado

SÃO CARLOS - SP
2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

R696us

Rodrigues, Kamila Rios da Hora.
A utilização de serviços web providos por SOA em geradores de aplicação desenvolvidos com linguagens de padrões / Kamila Rios da Hora Rodrigues. -- São Carlos : UFSCar, 2014.
134 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Engenharia de software. 2. Software - manutenção. 3. Geradores de aplicação. 4. GAwCRe. 5. Arquitetura orientada a serviços. 6. *Web Services*. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"A utilização de serviços web providos por SOA em geradores de aplicação desenvolvidos com linguagens de padrões"

KAMILA RIOS DA HORA RODRIGUES

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Rosângela Ap. Dellosso Penteado
(Orientadora - DC/UFSCar)

Prof. Dr. Antonio Carlos dos Santos
(DC/UFSCar)

Profa. Dra Maria Istela Cagnin Machado
(UFMS)

São Carlos
Junho/2009

Sem sonhos, as perdas se tornam insuportáveis,

as pedras do caminhos se tornam montanhas,

os fracassos se transformam em golpes fatais.

Mas, se você tiver grandes sonhos...

seus erros produzirão crescimento,

seus desafios produzirão oportunidades,

seus medos produzirão coragem.

Por isso...

NUNCA DESISTA DOS SEUS SONHOS.

(Augusto Cury)

Agradecimentos

Primeiramente a Deus, por ter me concedido mais essa vitória e por ter iluminado o meu caminho em todos os momentos em que humildemente pedi sabedoria, paciência e forças para continuar.

A minha mãe Zélia e meu pai Dene, pela sólida formação e pelo amor e apoio incondicionais. Obrigada por serem minha âncora nas horas de angustia, tristeza e saudade. Obrigada por celebrarem comigo cada pequena conquista obtida no decorrer da vida e em especial nesses dois anos. Obrigada por terem sido sempre dois amigos fiéis. Amo vocês!

Aos meus avós, tios e primos, que acreditaram em mim e que tantas vezes falaram as palavras que eu precisava ouvir pra continuar persistindo no caminho que tracei. Obrigada pelo carinho e por todos os abraços recebidos em cada uma das sonhadas voltas para casa.

A Profa. Dra. Rosângela Ap. D. Penteado por ter me proporcionado viver essa edificante experiência profissional e pessoal. Obrigada pela paciência, pelos conselhos dados e pelas vezes que ouviu pacientemente cada uma das minhas angustias. Obrigada, também, pelas vezes que redirecionou o meu foco. Muito obrigada ainda, pelos ensinamentos passados e pela orientação.

A cada flor da República Jardim, Cris, Li, Tati, Rê, Jú, Paula, Katia, Maria, Andréa e Deyse. Acredito que cada uma deixou um pouco de si e levou um pouco de mim. Obrigada pelos momentos que rimos e que choramos. Agradeço em especial a Cris que me acolheu, foi uma excelente companheira, um exemplo de mulher e que tornou a minha vida infinitamente mais feliz por compartilhar com as companheiras de república, a maravilhosa experiência de ser mãe e nos dar a alegria de conviver com o nosso pequeno príncipe Gui. Foi um lar perfeito!

Aos amigos do mestrado que foram mais que companheiros. Com certeza, não fosse a nossa união e amizade essa jornada teria sido muito mais árdua e triste para cada um. Somos guerreiros! Obrigada a todos pelas inúmeras risadas, festinhas e conversas. Em especial a Elis

pela paciência e amizade de sempre e a Filipe pela amizade e pelo companheirismo desde o princípio desta jornada.

Aos amigos da Bahia que acreditaram em mim, que incentivaram e acompanharam toda essa trajetória. Muito obrigada ao amigo Rogério, que mesmo distante sempre foi fiel, presente e amigo.

Aos colegas do laboratório GDMS, em especial Matheus e Ivan, pelos préstimos, pela amizade e por tantas risadas. Obrigada também ao Daniel Zardi pela valorosa ajuda e excelentes dicas.

Aos professores e funcionários do DC pela atenção, disposição e amizade. À Cris e a Dona Vera por terem sido prestativas e amigas em muitos momentos.

A coordenadora do PPG-CC da UFSCar Heloísa de Arruda Camargo pela compreensão e confiança.

A Capes pelo apoio financeiro.

Por fim, minha sincera gratidão a todas aquelas pessoas que, direta ou indiretamente, contribuíram para a realização deste sonho. Muito obrigada a todos!

Entrega o teu caminho ao Senhor,

confia n'Ele e o mais Ele fará!

(Salmo 37:5)

Esperei com paciência pelo Senhor,

e ele se inclinou para mim e ouviu o meu clamor.

(Salmo 40:1)

Resumo

O reúso de sistemas é uma técnica da Engenharia de Software que propõe a construção de sistemas de software a partir de artefatos já existentes. Uma das formas de desenvolver sistemas reutilizando código, projeto, em um domínio específico, é por meio de geradores de aplicação. Eles automatizam parte do processo de desenvolvimento, reduzem custos, possibilitam o aumento de produtividade da equipe de desenvolvimento, melhoram a qualidade dos sistemas e minimizam a inserção de erros provenientes da fase de implementação. O GAwCRe - Gerador de Aplicações baseadas na Web para o Domínio de Gestão de Clínicas de Reabilitação - foi desenvolvido com a técnica de linhas de produtos de software e com base na linguagem de padrões SiGCli (Sistema de Gestão de Clínicas de Reabilitação). Esse gerador passou por algumas manutenções para possibilitar a utilização do SGBD MySQL, bem como realizar controle de versões das aplicações geradas e ter seu domínio ampliado para domínios conexos ao seu. Atualmente, há o interesse em utilizar arquiteturas orientadas a serviços (SOA) para apoiar a construção de aplicações de software que utilizam serviços disponíveis em uma rede como a Web. Esses serviços são implementações de uma funcionalidade de negócios bem definida, que pode ser utilizada por clientes de diferentes aplicações. É cada vez mais constante a reutilização de serviços já disponíveis para a construção de novos sistemas Web, reduzindo retrabalho e facilitando a manutenção desses sistemas. Esta dissertação de mestrado apresenta uma abordagem para modificar geradores de aplicação, de modo que seu domínio seja ampliado com a utilização de serviços Web providos por uma arquitetura orientada a serviços. Essa arquitetura facilita a adaptação de sistemas, fazendo com que esses se tornem dinâmicos, uma vez que os seus serviços podem ser substituídos em tempo de execução. Também permite que futuras manutenções sejam facilitadas, pois as modificações, feitas nos serviços, são refletidas nas aplicações sem a intervenção do usuário. O GAwCRe passou por manutenções para que pudesse apoiar o uso dos serviços Web providos por uma SOA e, assim, ampliar o seu domínio. Um estudo de caso foi realizado utilizando o GAwCRe e alguns serviços Web buscando avaliar o uso conjunto de SOA e Geradores de Aplicação.

Abstract

System reuse is a Software Engineering technique that proposes the construction of software systems from existing artifacts. One way to develop systems reusing code and project in a specific domain is using application generators. These generators automate part of the development process, reduce costs, allow the increase of the development team productivity, enhance systems quality and minimize error insertion arising from the implementation phase. The GawCRe (Web based Application Generator to the Rehabilitation Clinic domain) was developed using the software product lines technique based in the SiGCli (Rehabilitation Clinics Management System) pattern language. This generator went through some maintenance in order to allow the MySQL RDBMS usage, as well as to carry out version control of the generated applications and increase its domain to others related to it. Currently, there is interest in the usage of Service Oriented Architectures (SOA) to support the construction of software applications that use services available in a network, as the Web. These services are an implementation of well-defined business functionality that can be used by clients of different applications. The reutilization of available services in the construction of new Web systems is increasing, reducing the rework and facilitating the maintenance of these systems. This master's thesis presents an approach to modify application generators in a way that its domain is increased by the use of Web Services provided by Service Oriented Architecture. This architecture eases systems' adaptation, making them dynamic, as their services can be swapped in runtime. This allows future maintenances to be eased, because modifications in the services are reflected in the applications without user intervention. The GAwCRe undergone for maintenance to support the use of Web Services provided by a SOA, thereby, broadening its domain. A case study was conducted using the GAwCRe and some Web Services attempt to evaluate the combined use of SOA and Application Generators.

Sumário

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Abreviaturas

API: *Application Programming Interface*

ARA: *Arcabouço de Reengenharia Ágil*

BPEL*: (BPEL4WS) Business Process Execution Language for Web Services*

CEP: *Código de Endereçamento Postal*

CORBA: *Common Object Request Broker Architecture*

CPF: *Cadastro de Pessoas Física*

CREFITO: *Conselho Regional de Fisioterapia e Terapia Ocupacional*

DCOM: *Distributed Component Object Model*

DOM: *Document Object Model*

EAI*: Enterprise Application Integration*

ED*: Engenharia de Domínio*

ESB*: Enterprise Service Bus*

GA: *Gerador de Aplicações*

GAWCRE: Gerador de Aplicações baseadas na Web para o domínio de gestão de Clínicas de Reabilitação

GRN*: Gestão de Recursos de Negócios*

HTTP*: Hypertext Transport Protocol*

J2EE: *Java 2 Entreprise Edition*

JPA: *Java Persistence Architecture*

JSF: *Java Server Faces*

JSP*: Java Server Pages*

LMA*: Linguagem de modelagem da aplicação*

LP*: Linha de Produto*

LPS*: Linha de Produto de Software*

MDA*: Model Driven Architecture*

MVC: *Modelo-Visão-Controle*

OMG*: Object Management Group*

ORM: *Object-Relational Mapping*

RPC: *Chamada de Procedimento Remota*

SGBD: *Sistema Gerenciador de Banco de Dados*

SIGCLI*: **Si**stemas de **G**estão de **Clí**nicas de Reabilitação*

SLA: *Service Level Agreement*

SOA*: Service Oriented Architecture*

SOAP*: Simple Object Access Protocol*

SOD*: Service-Oriented Design*

SQL*: Structured Query Language*

TI*: Tecnologia de Informação*

UDDI*: Universal Description, Discovery, and Integration*

UML*: Unified Modeling Language*

URL*: Unified Resource Location*

W3C: *World Wide Web Consortium*

WAE*: Web Application Extension*

WS-CDL*: Web Service Choreography Description Language*

WSDL*: Web Services Description Language*

WSFL*: Web Services Flow Language*

WSW: *Web ServicesWrapper*

XMI*: XML Metadata Interchange*

XML*: eXtendible Markup Language*

XSD: *XML Schema Definition*

Capítulo 1

Introdução

1.1. Considerações Iniciais

A tarefa de desenvolvimento de software tem passado por diversos desafios ao longo dos anos. Construir sistemas com qualidade, baixo custo e em um curto espaço de tempo é uma das principais metas da indústria de software. Muitas são as dificuldades em atingir essa meta, dentre elas estão a falta de experiência dos engenheiros de software em: seguir os princípios e as práticas preconizadas pela Engenharia de Software, em planejar métodos efetivos de controle e em serem resistentes às mudanças. Essas dificuldades podem acarretar falhas no cumprimento dos prazos, na estimativa de custo que não ultrapassem o orçamento, bem como na realização de manutenções nos sistemas. Podem ainda ocasionar baixa produtividade, baixa qualidade dos softwares e insatisfação do cliente (PRESSMAN, 2006).

Durante a construção de sistemas de software, técnicas podem ser aplicadas com o intuito de auxiliar na obtenção de bons resultados. O reúso de software é uma das técnicas que auxilia tanto no desenvolvimento quanto na manutenção de softwares. Essa técnica propõe a construção de sistemas de software a partir de artefatos já existentes (YING *et al*., 2006). Alguns dos recursos que podem ser utilizados para a aplicação de reúso são: *frameworks*, componentes de software, padrões e de linguagem de padrões, geradores de aplicação, dentre outros.

Os Geradores de aplicação (GA) são ferramentas de produção automatizada de artefatos que permitem reúso de código e que contribuem para a redução de tempo e dos custos empregados no desenvolvimento de software. Os GAs permitem ainda, o aumento na

produtividade durante a fase de desenvolvimento (HERRINGTON, 2003). O reúso de software por meio de GAs também ajuda a melhorar a qualidade dos sistemas produzidos minimizando a inserção de erros causada por manutenções ou adaptações no código. Essa minimização de erros ocorre devido à automatização do processo rotineiro no desenvolvimento de um software, acelerando o processo de implementação e transformando especificações de alto nível em produtos da aplicação (PAZIN, 2004). O uso de GAs permite que usuários leigos gerem aplicações de forma rápida, organizada e, que engenheiros de software tenham mais facilidade na construção de aplicações e na sua posterior manutenção.

Mesmo sendo ferramentas que facilitam a construção e manutenção de aplicações, os geradores atualmente ainda não são capazes de atender de forma rápida e eficaz às constantes mudanças de mercado e às novas demandas dos clientes. Ainda é preciso inserir algumas mudanças no gerador, executá-lo novamente e então disponibilizar ao cliente, a versão modificada da aplicação. Além disso, a qualidade da aplicação gerada está relacionada à forma utilizada na construção do gerador (RODRIGUES *et al*., 2009); caso não tenham sido realizados testes exaustivos, pode ocorrer que falhas não tenham sido detectadas quando da construção do gerador e dessa forma, os produtos gerados também estarão falhos.

Uma solução que proporciona adaptação rápida e eficaz às constantes mudanças do mercado e dos clientes é a Arquitetura Orientada a Serviços (SOA) (OASIS, 2006). A SOA é um estilo arquitetural com aplicações dinâmicas que propõe reúso de seus artefatos de forma que o processo de desenvolvimento de uma aplicação seja constituído pela identificação, pela composição e pela coordenação de serviços já implementados, aumentando o reúso, diminuindo o esforço e os custos gasto no projeto, facilitando a manutenção e aumentando o grau de satisfação dos clientes, metas importantes a serem alcançadas no desenvolvimento de software (FANTINATO *et al*., 2005; SORDI *et al*., 2006; BIANCO *et al*., 2007).

Além do reúso, a adoção de uma SOA facilita a adaptabilidade de sistemas, fazendo com que se tornem dinâmicos, ao passo que tais serviços podem ser substituídos em tempo de execução sem o conhecimento do usuário. Como atualmente as organizações estão em constante atualização, sempre buscando vantagens competitivas, a flexibilidade dos sistemas é uma questão relevante e outra importante meta a ser alcançada (PAPAZOGLOU e VAN DEN HEUVEL, 2007).

1.2. Motivação e Objetivos

Atualmente geradores de aplicação ainda são sistemas amplamente utilizados. Entretanto, boa parte desses geradores pode ser considerada sistema legado tendo passado por diversas mudanças ao longo do tempo. Tais mudanças podem ser atribuídas: à solicitação de novos requisitos por parte do cliente, à uma manutenção evolutiva; pois utilizam tecnologias e técnicas ultrapassadas ou ainda, à manutenção corretiva e essa é feita com base no código existente, sem documentação necessária o que o torna difícil de ser usado e também mantido. Além desses fatores, geradores podem apresentar: a) manutenção custosa, devido à forte dependência em relação ao domínio e baixa modularização; b) código a ser gerado entrelaçado com o código do próprio gerador; e c) dificuldade de ampliação do domínio, pois para acrescentar novas funções é necessário alterar arquivos de configuração e de código-fonte (BORGES, 2008).

A partir de um processo de engenharia reversa, Pazin (2004) desenvolveu o GAwCRe, um gerador de aplicações para a Web construído com base em Linha de Produtos de Software e linguagem de padrões de análise, para o domínio específico de clínicas de reabilitação física (Fisioterapia, Terapia Ocupacional e Educação Física). Apesar de atender às necessidades ao qual foi proposto, algumas melhorias foram realizadas para que o GAwCRe utilizasse recursos tecnológicos melhores que aqueles disponíveis em 2004, o que talvez tenha impedido que o gerador fosse implementado com uma arquitetura mais adequada, com baixo acoplamento de funções e coesão de dados. Sendo assim, algumas experiências de manutenção evolutiva, corretiva e adaptativa foram realizadas nesse gerador com diversas finalidades, dentre elas: expansão de seu domínio para atender a domínios conexos ao que foi desenvolvido (FERREIRA, 2008), modificação do banco de dados original para MySQL (RIZZO, 2005) e disponibilização do controle de versões para que haja controle de cada uma das versões geradas para os sistemas (BORGES, 2008). O GAwCRe também passou por um processo *ad hoc* de reengenharia (FREITAS, 2006) em que foi utilizado no lugar de um *framework* para a geração automatizada de artefatos no Arcabouço de Reengenharia Ágil - ARA (CAGNIN, 2005).

Mesmo com essas modificações o gerador ainda apresentava alguns problemas que impediam a expansão de seu domínio, pois ainda havia limitações arquiteturais, por exemplo, interesses da camada de negócios na camada de persistência de dados, o acesso ao banco de

dados era feito localmente, trechos de código construídos sem o uso de padrões de projeto e documentação; o que dificulta o reúso (GAMMA *et al.,* 1995), além do alto acoplamento e baixa coesão de dados.

Notou-se que o esforço em realizar uma manutenção evolutiva e corretiva no GAwCRe, que o tornasse mais flexível, seria menor do que a realização de mais uma manutenção adaptativa ou perfectiva nesse gerador. A manutenção evolutiva, fazendo uso de novos recursos e tecnologias de desenvolvimento, permitiria ao GAwCRe atender aos domínios conexos ao que foi desenvolvido, com código mais legível e estruturado; utilizar as principais normas de implementação preconizadas atualmente e com um projeto melhor documentado.

Devido à necessidade de expansão de domínio, de forma ágil e eficaz, decidiu-se por realizar a manutenção corretiva e evolutiva no GAwCRe, utilizando serviços Web providos por uma SOA. Tais manutenções visam minimizar e até eliminar os problemas acima descritos, além de permitir a ampliação do domínio de GAs que tenham sido desenvolvidos da forma como o GAwCRe foi desenvolvido.

A SOA possui características que permitem a comunicação entre aplicações de modo independente da plataforma de software e hardware ou da linguagem de programação (FANTINATO *et al*., 2005). Essa flexibilidade facilita a integração entre aplicações, permitindo o uso de serviços eletrônicos disponibilizados em uma rede de comunicação para acrescentar novas funções ao seu domínio. Para que isso aconteça é preciso criar uma interface de comunicação entre a aplicação e o serviço eletrônico para que ela possa utilizá-lo.

Com o uso de serviços eletrônicos providos por SOA, a modificação é realizada no serviço e disponibilizada de forma dinâmica e distribuída às aplicações clientes por meio da Internet. Ao acessar o serviço novamente, uma versão atualizada da aplicação é oferecida em tempo de execução. Além do dinamismo, o uso de serviços eletrônicos pode diminuir o tempo gasto com manutenções de sistemas, pois a modificação realizada em um serviço pode automaticamente ser replicada a todas as aplicações que utilizam esse serviço, diminuindo consequentemente os custos do projeto.

Geradores de aplicação modificados para suportar serviços eletrônicos terão a vantagem de ser mais flexíveis, com manutenção facilitada, possibilitando a ampliação do

domínio, independente da linguagem de programação e plataforma utilizadas na sua criação, pois, SOA aproveita o cenário tecnológico com enfoque na distribuição do processamento, oferecendo total suporte a heterogeneidade ambiental, em que diversos tipos de processadores, sistemas operacionais e linguagens de programação distintas interagem (MAHMOUD, 2005).

Esta dissertação de mestrado apresenta uma abordagem do uso de SOA junto com geradores de aplicação para a geração de sistemas, bem como para a ampliação do domínio de geradores já desenvolvidos com base em linguagens de padrões de análise. Os serviços oferecidos por uma SOA permitem ainda, que os geradores de aplicação se adéquem rapidamente às constantes mudanças de requisitos solicitados pelos clientes.

O objetivo é unir as vantagens dos geradores de aplicação com as vantagens oferecidas pelos serviços providos por uma SOA, de forma a construir aplicações com menor quantidade de erros inseridos na fase de codificação, que possuam um padrão de codificação e facilitem manutenções futuras, além de serem dinâmicas e permitirem a sua utilização em qualquer plataforma ou linguagem de programação.

Realizar uma manutenção corretiva e evolutiva em geradores de aplicação, utilizando serviços eletrônicos providos pela arquitetura orientada a serviços, permitirá a inserção de novas funções no gerador, preservando as existentes, melhorando a qualidade global do sistema e permitindo que as aplicações satisfaçam às necessidades de negócios vigentes.

1.3. Organização

Esta dissertação está dividida em sete capítulos. O Capítulo 1 contextualiza a proposta deste trabalho apresentando a motivação para realizá-lo.

O Capítulo 2 apresenta geradores de aplicação, ressaltando as vantagens do seu uso e apresenta o gerador GAwCRe a ser utilizado como estudo de caso neste trabalho.

O Capítulo 3 apresenta a Arquitetura Orientada a Serviços e as características que fazem com que essa seja uma abordagem tão discutida e utilizada atualmente. O capítulo apresenta ainda, os principais elementos utilizados por essa arquitetura e *Web Services* como opção para a construção de uma SOA.

O Capítulo 4 apresenta as manutenções realizadas no gerador de Aplicações GAwCRe para que esse pudesse prover e utilizar serviços.

O Capítulo 5 detalha a abordagem do uso de serviços providos por SOA em Geradores de Aplicação.

O Capítulo 6 apresenta um estudo de caso realizado no gerador GAwCRe utilizando a abordagem proposta.

O Capítulo 7 apresenta as conclusões e as contribuições deste trabalho, além de relatar as limitações detectadas e enumerar sugestões de possíveis trabalhos futuros.

Capítulo 2

Geradores de Aplicação

2.1. Considerações Iniciais

O esforço despendido por uma organização de desenvolvimento com a manutenção de software é estimado em 60%, sendo que cerca de 20% desses são gastos em manutenção corretiva, enquanto que os outros 80% são gastos submetendo uma aplicação a outros tipos de manutenção (PRESSMAN, 2006).

Um estudo realizado por Jones (2008) correlaciona as fases do projeto de desenvolvimento de software (requisitos, projeto, codificação, documentação ou manutenção) com a inserção de erros. O autor utilizou uma escala de níveis de criticidade para os erros, variando de 1 a 4, sendo que 1 corresponde aos erros que tornam os sistemas ou programas inoperáveis, 2 para aqueles que levam o software a ter a maioria de suas funções desativadas ou incorretas, 3 para os que possuem poucas funções do software a serem desativadas ou incorretas e 4 para aqueles erros superficiais. A maior parte dos erros detectados geralmente é inserida nas fases de projeto e codificação bem como com níveis de criticidade 2 e 3.

Nesse contexto, algumas técnicas, incluindo o uso de Geradores de Aplicação (GA), têm sido propostas com o intuito de reduzir os custos com a correção de erros e com a complexidade da manutenção de software.

Com o uso de um GA é possível evitar erros no sistema decorrentes de duplicação de código, alocação desnecessária de recursos, mal uso de APIs complexas, além de reduzir o acoplamento, garantir a coesão e facilitar a manutenção do sistema (HERRINGTON, 2003).

Os geradores construídos com padrões de software tendem a minimizar o número de erros, uma vez que esses representam soluções de sucesso para problemas recorrentes (GAMMA, 1995).

Este Capítulo apresenta Geradores de Aplicação, seus benefícios e exemplos de uso. Na Seção 2.2 é relatado o reúso por meio de padrões e linguagem de padrões, que podem ser usados na construção de um gerador de aplicações. Na Seção 2.3 é descrito o conceito de geradores e sua forma de documentação. Na Seção 2.4 é apresentado o GAwCRe gerador utilizado para estudo de caso deste projeto de mestrado e na Seção 2.5 são apresentadas as considerações finais.

2.2. Reúso por Padrões

Um padrão é uma solução de sucesso para determinado problema em um contexto específico. Esse problema pode se repetir diversas vezes e em diferentes aplicações, daí a necessidade de se aplicar um padrão (FREEMAN *et al*., 2004).

Gamma *et al*. (1995) e Freeman *et al.* (2004) citam quatro benefícios importantes quando padrões de software são aplicados: captar experiências, tornando-as acessíveis aos menos experientes; formar um vocabulário de apoio para que desenvolvedores se comuniquem melhor; ajudar engenheiros de software a entender um sistema de forma mais rápida; e facilitar a reestruturação de um sistema, ainda que o mesmo não tenha sido projetado com padrões.

O uso de padrões proporciona um vocabulário comum para a comunicação entre projetistas, criando abstrações num nível superior ao de classes e garantindo uniformidade na estrutura do software, além de atuarem como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos (GAMMA *et al*. 1995; FREEMAN *et al.,* 2004).

Uma coleção estruturada de padrões que se apoiam uns nos outros para transformar requisitos e restrições numa arquitetura é chamada de linguagem de padrões (BUSCHMANN *et al*., 2007). Os padrões que constituem uma linguagem de padrões cobrem todos os requisitos importantes em um dado domínio e pelo menos um padrão deve estar disponível para cada requisito da construção e implementação de um sistema de software. Na linguagem de padrões um problema geral e sua solução são subdivididos em um número de problemas

relacionados e suas respectivas soluções. Cada padrão da linguagem resolve um problema específico no contexto comum compartilhado pela linguagem. Os padrões podem ser usados separadamente ou com mais de um padrão da linguagem. Isso significa que um padrão sozinho é considerado útil mesmo se a linguagem não for totalmente usada (COPLIEN e HARRISON, 2004).

Braga *et al*. (1998, 1999) definem uma linguagem de padrões para Gestão de Recursos de Negócios (GRN) que é composta por quinze padrões de análise e alguns deles são aplicações ou extensões de padrões existentes na literatura. A GRN auxilia programadores menos experientes no desenvolvimento de aplicações que tratam de gestão de recursos de negócios tais como: transações de aluguel (bem ou serviço), comercialização (transferência de propriedade ou de bem) e manutenção (reparo ou conservação de produto).

Por meio de um processo de engenharia reversa de sistemas existentes, Pazin (2004), definiu um modelo de classes para seu domínio e baseado no processo para a elaboração de uma linguagem de padrões proposto por Braga (2003), elaborou a linguagem de padrões para **Si**stemas de **G**estão de **Clí**nicas de Reabilitação (*SiGCli*). Durante a construção da *SiGCli*, Pazin observou que alguns dos padrões propostos para o domínio de clínicas de reabilitação eram semelhantes aos existentes na linguagem de padrões GRN (BRAGA *et al*.,1999), mas havia padrões que a GRN não cobria ou cobria parcialmente. Ele então optou por aplicar a GRN ao domínio proposto, reutilizando os seus padrões ou fazendo adaptações quando necessário, e para suprir funções específicas da *SiGCli* não apoiadas pela GRN, Pazin criou um novo padrão. Dessa forma, a *SiGCli* representa um subdomínio da GRN (PAZIN, 2004).

A Figura 2.1 exibe o relacionamento entre os padrões da linguagem de padrões *SiGCli*. Esses padrões estão reunidos em três grupos, de acordo com sua funcionalidade. O primeiro grupo trata das informações básicas relacionadas às clínicas, a identificação dos pacientes e dos serviços prestados. O segundo grupo cuida do gerenciamento dos atendimentos, sendo possível acompanhar todas as fases do tratamento de um paciente. O terceiro grupo cuida do controle financeiro das clínicas.

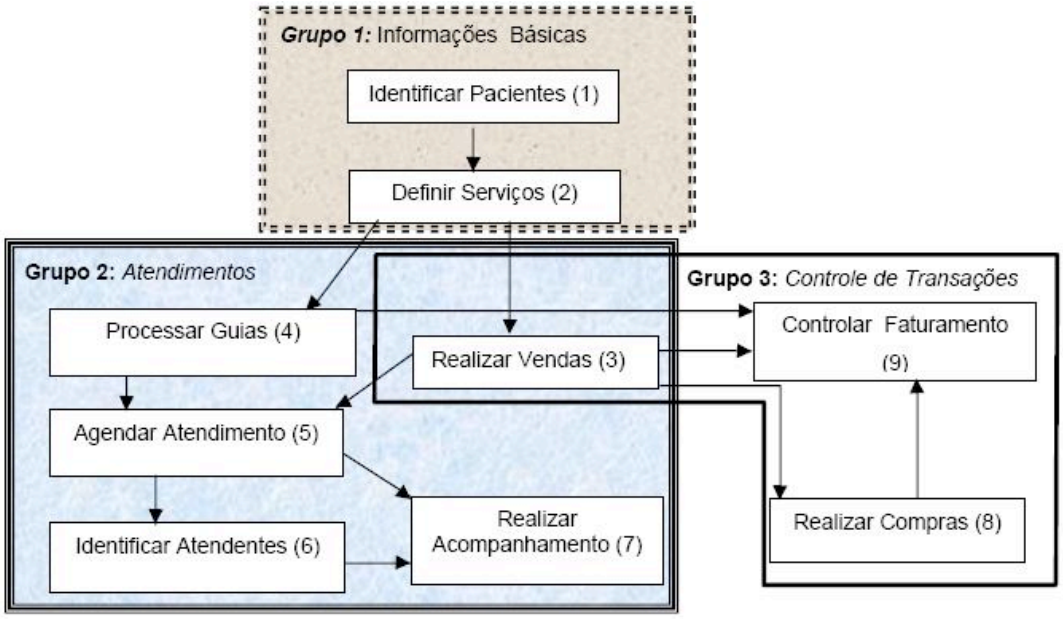


Figura 2.1 - Fluxo de aplicação dos padrões da SiGCli (PAZIN, 2004).

2.3. Gerador de Aplicações

Diminuir a quantidade de erros inseridos manualmente em um código e tornar a sua manutenção mais simples, rápida e menos custosa são alguns dos objetivos dos projetistas de software. Com o uso de Geradores de Aplicação (GA) é possível minimizar a inserção de erros durante o desenvolvimento de software em um domínio específico, aumentando sua manutenibilidade (HERRINGTON, 2003).

Geradores de Aplicação são abordagens de produção automatizada de artefatos que contribuem para a redução de tempo e dos custos empregados no desenvolvimento de software (HERRINGTON, 2003). Essas ferramentas conseguem automatizar parte de um processo rotineiro na atividade de desenvolvimento de software, acelerando o processo de implementação, transformando especificações de alto nível em produtos da aplicação (Pazin, 2004). Essa técnica de reúso, além de reduzir os custos e aumentar a produtividade, ajuda a melhorar a qualidade dos sistemas produzidos, minimizando a inserção de erros causada por manutenções ou adaptações no código (GRUBB, 2003).

A geração apropriada dos artefatos gerados por um gerador pressupõe a correta construção do mesmo. Os problemas que podem surgir estarão associados ao modo como foi

desenvolvida a geração desses artefatos. Dessa forma, a qualidade do sistema gerado não depende diretamente da qualidade do gerador ou do processo de geração, mas sim da estrutura com a qual o gerador foi construído e dos modelos utilizados (HERRINGTON, 2003).

GAs podem ser considerados compiladores para uma linguagem de um domínio específico (SMARAGDASKI e BATORY, 1998 *apud* PAZIN, 2004). Essa linguagem pode ser elaborada após a análise de domínio e pode ser representada através de uma linguagem de padrões, como por exemplo, a linguagem *SiGCli*.

Para que se justifique o uso de geradores de aplicações é preciso que haja reúso de um determinado produto de software em grande escala (PFLEEGER, 1998 *apud* FRANCA, 2000), ou seja, é preciso que haja grande quantidade de sistemas de software para um mesmo domínio (família de software).

Engenharia de Domínio (ED) é um dos enfoques mais utilizados no desenvolvimento baseado em reutilização. Os ambientes de desenvolvimento de software baseados em ED são conhecidos por permitirem a produção de aplicações por meio da reutilização do conhecimento acumulado sobre uma determinada área (MILER *et al*., 2007).

Segundo Franca (2000), o uso de geradores reduz o tempo para o mercado, pois, o desenvolvimento de um novo produto limita-se ao tempo gasto no desenvolvimento e avaliação da sua especificação. Além da redução de tempo, geradores de aplicação oferecem qualidade no produto gerado, o aumento da produtividade, pois, ocorre a geração automática das partes fixas e os novos produtos gerados são alterações das especificações fornecidas ao gerador, facilitando assim o desenvolvimento de novas versões.

Os geradores de aplicação podem ser classificados de duas formas diferentes: domínio único (aplicação única-DUAU) e múltiplos domínios (múltiplas aplicações - MDMA) (MASIERO e MEIRA, 1993 *apud* SHIMABUKURO, 2006). Os geradores do tipo DUAU sempre criam o mesmo tipo de produto no mesmo domínio e os MDMA podem ser adaptados para fornecer apoio em domínios diferentes e aplicações diferentes dentro desses domínios (SHIMABUKURO, 2006).

Shimabukuro (2006) cita como exemplos de geradores DUAU os utilitários *LEX* e *YACC*, que geram analisadores léxicos e sintáticos, respectivamente. Um exemplo de gerador MDMA é o Draco (CZARNECKI e EISENECKER, 2002). Esse gerador possui um

mecanismo para especificação de domínio e criação de regras de refinamento que servem para transformar especificações de um domínio para outros domínios conhecidos do *Draco*.

Rausch (2001) apresenta um gerador baseado em XML[1] e gabaritos (do inglês *templates*). Recebe como entrada um arquivo de XML contendo especificações de software e um conjunto de gabaritos que após o processamento produzem os arquivos de saída. Esse gerador produz um sistema que utiliza uma biblioteca de componentes. Uma especificação de XML é recebida descrevendo como esses componentes devem ser parametrizados e relacionados para em seguida ser gerado o código da aplicação com base nos gabaritos.

Aranha e Borba (2002) apresentam um processo de teste que utiliza geradores de código para melhorar a produtividade do desenvolvimento. Esses geradores são baseados na linguagem WSat (*Web System Acceptance Test language*) utilizada para realizar testes de sistemas Web como os de aceitação e de performance. Os geradores além de melhorar a produtividade, auxiliam a criação dos testes antes da implementação do próprio sistema (*Test First Design*).

Furtado *et al*. (2005) criaram o *HWSProxyGen* um gerador de *proxies* de *web services* para a linguagem funcional Haskell. Seu objetivo é abstrair detalhes de implementação de baixo-nível aos programadores dessa linguagem para a construção de aplicações consumidoras de serviços disponibilizados na Internet. A intenção é mostrar que Haskell e linguagens funcionais em geral podem ser consideradas alternativas viáveis para a implementação de aplicações e componentes distribuídos, interagindo com serviços implementados em diferentes linguagens e/ou plataformas.

O gerador *AndroMDA* (BOHLEN *et al*., 2006) é um gerador de código que recebe como entrada documentos UML[2] no formato XMI[3] e gera aplicações para diversas plataformas e tecnologias existentes como por exemplo: Java Server Faces, Enterprise Java

[1] *eXtensible Markup Language* - Uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais (W3C, 2008).
[2] *Unified Modeling Language* - Uma linguagem de modelagem não proprietária de terceira geração.
[3] *XML Metadata Interchange* - Um padrão da OMG (grupo de gerenciamento de objetos) para troca de informações baseado em XML.

Beans, J2EE, entre outros. O *AndroMDA* usa técnicas de MDA[4] e padrões de transformações abertos gerenciados pela *Object Management Group* (OMG[5]).

O gerador *Captor* (SHIMABUKURO, 2006) é um gerador de aplicação configurável para diversos domínios que fornece apoio ao desenvolvimento de diversas aplicações dentro desses domínios. A configuração do *Captor* é feita por meio da modularização da especificação em formulários ou por meio de criação de linguagens de modelagem de aplicações baseadas em formulários.

Pazin (2004) criou o gerador *GAwCRe*, um gerador de aplicações baseadas na Web para o domínio de gestão de clínicas de reabilitação que permite instanciar aplicações usando uma Linguagem de Modelagem da Aplicação (LMA[6]) definida com base na linguagem de padrões SiGCli. Para representar as informações referentes à LMA e a linguagem de padrões SiGCli, Pazin elaborou um meta-modelo utilizando a linguagem XML, em que as informações (LMA) apresentadas na interface de instanciação das aplicações do gerador são criadas de forma dinâmica. O gerador de aplicações *GAwCRe* foi utilizado como estudo de caso neste trabalho de mestrado e será detalhado na Seção 2.4. Linguagem de Modelagem de Aplicações será descrita na Seção 2.3.1.

2.3.1. Linguagem de Modelagem da Aplicação – LMA

As especificações feitas em alto nível pelos geradores de aplicação podem ser documentadas por meio de uma LMA que devem representar as abstrações (modelos) das aplicações que garantam certas propriedades importantes para o domínio (WEISS e LAI, 2004). Um gerador de aplicações criado a partir de uma LMA deve analisar as especificações escritas nessa linguagem para, a partir delas, gerar o produto final (BRAGA, 2003).

As linguagens de modelagem da aplicação podem ser definidas por uma gramática formal ou podem ser definidas por um conjunto de gráficos (que podem ser reunidos para representar uma especificação), por um conjunto de parâmetros que definem as partes

[4] *Model Driven Architecture* - é uma arquitetura para desenvolvimento de software, que dá suporte a todo o ciclo de desenvolvimento (SIEGEL, 2000).
[5] http://www.omg.org/
[6] LMA, originalmente AML – *Application Modeling Language.*

variáveis de uma aplicação ou por telas gráficas em que o usuário insere um conjunto de dados e esses dados são transformados para um formato apropriado (SHIMABUKURO, 2006).

A Figura 2.2 exibe a gramática definida para a LMA baseada na linguagem de padrões *SiGCli*. De acordo com Pazin (2004), para instanciar uma aplicação é preciso apenas optar pelas regras definidas na gramática, considerando as regras de produção determinadas.

```
LMA ::= <lista_padrões>
<lista_padrões> ::= <padrão> ; <lista_padrões>| λ
<padrão> ::= Identificar_Paciente | <Definir_Serviço> | <Realizar_Vendas>
             |Processar_Guias |Agendar_Atendimentos |
             |<Identificar_Atendente> |<Realizar_Acompanhamento>|
             |Realizar_Compra |Controlar_Faturamento
<Definir_Serviço> ::= Definir_Serviço ; <Com_Tipo_Serviço>
<Com_Tipo_Servico> ::= Com_Tipo_Serviço | λ
<Realizar_Venda> ::= Realizar_Venda ; <Com_Produto>
<Com_Produto> ::= Com_Produto | λ

                                    ...

<Tipo_Avaliação_EF> ::= Anamnese | Antropométrica | Física |

                        Controle_Diário_Atividades | Ficha_Musculação | λ
```

Figura 2.2 - Regras da gramática definida para a LMA baseada na SiGCli (PAZIN, 2004).

Com as variabilidades presentes nos padrões e representadas em uma LMA, é possível gerar um conjunto distinto de aplicações, considerando as diferentes combinações permitidas para a aplicação dos padrões e de suas variantes.

A LMA descrita na Figura 2.2 reflete as variabilidades (nome e descrição) identificadas entre os padrões da *SiGCli* apresentados na tabela abaixo.

Tabela 2.1 - Variabilidades identificadas entre os padrões da *SiGCli* (PAZIN, 2004).

Nº	Padrão *SiGCli*	Variabilidade	
		Nome	Descrição
1	Identificar Pacientes	Não possui	----------
2	Definir Serviços	Com Tipo Serviço	1. Uma clínica não precisa especificar os serviços prestados de forma detalhada.
3	Realizar Vendas	Com Produto	1. Uma clínica realiza a comercialização de serviços, mas nem sempre comercializa produtos.

4	Processar Guias	Não possui	1. Caso o padrão Controlar o Faturamento tenha sido aplicado, deve-se acrescentar alguns atributos e métodos relacionados ao valor da guia e dos atendimentos.
5	Agendar Atendimentos	Não possui	1. O agendamento de sessões pode ser efetuado por meio de uma guia ou de uma venda.
6	Identificar Atendente	Com Atributos Atendentes	1. Quando for necessário armazenar informações sobre o ganho e a especialidade do atendente.
7	Realizar Acompanhamento	Com Avaliações Educação Física Com Avaliações Fisioterapia Com Avaliações Terapia Ocupacional	1. Cada clínica possui avaliações específicas, conforme os tratamentos que realizam, com atributos específicos. Por exemplo, os atributos analisados em uma avaliação ortopédica para uma clínica de fisioterapia são diferentes dos atributos avaliados para uma clínica de educação física. 2. Cada avaliação pode possuir um conjunto de fatores que são variantes. Por exemplo, em uma avaliação o uso de medicamentos deve ser considerado; em outra alguns testes especiais devem ser realizados para a conclusão do tratamento. Esses fatores são considerados como Itens da Avaliação.
8	Realizar Compras	Não possui	----------
9	Controlar Faturamento	Não possui	1. Depende da aplicação dos padrões Realizar Vendas, Processar Guias, Identificar Atendentes e Realizar compras para definir quais classes devem ser usadas para o sistema.

Para definir uma LMA é preciso analisar as similaridades e variabilidades existentes em um domínio. É importante que as variações sejam bem definidas na LMA para que a aplicação possa ser gerada corretamente, conforme os seus requisitos.

2.4. Gerador de Aplicações GAwCRe

O GAwCRe - Gerador de Aplicações baseadas na Web para o Domínio de Gestão de Clínicas de Reabilitação - permite instanciar aplicações usando uma LMA definida com base na linguagem de padrões SiGCli (PAZIN, 2004), apresentada na Seção 2.2.

Para representar as informações referentes à LMA e a linguagem de padrões SiGCli foi elaborado um meta-modelo utilizando a linguagem XML, em que as informações (LMA) apresentadas na interface de instanciação das aplicações do gerador são criadas de forma dinâmica. Na inicialização do GAwCRe, o meta-modelo é lido e com as informações ali contidas cada um dos padrões da SiGCli com o seu conjunto de variantes (caso haja) são apresentados na interface do gerador. Para cada padrão usado na aplicação é feita a análise de

quais classes, atributos, associações e métodos devem ser criados para satisfazer os requisitos da aplicação. Segundo Pazin (2004), o total de aplicações distintas que podem ser geradas pelo GAwCRe, utilizando a SiGCli, é de seiscentas e oitenta e oito aplicações.

Esse gerador possui em sua arquitetura um conjunto de classes responsáveis por gerenciar e gerar as diferentes instanciações das aplicações, de acordo com os requisitos e as especificações da LMA (PAZIN, 2004). A Figura 2.3 mostra o diagrama de classes em alto nível desse gerador.

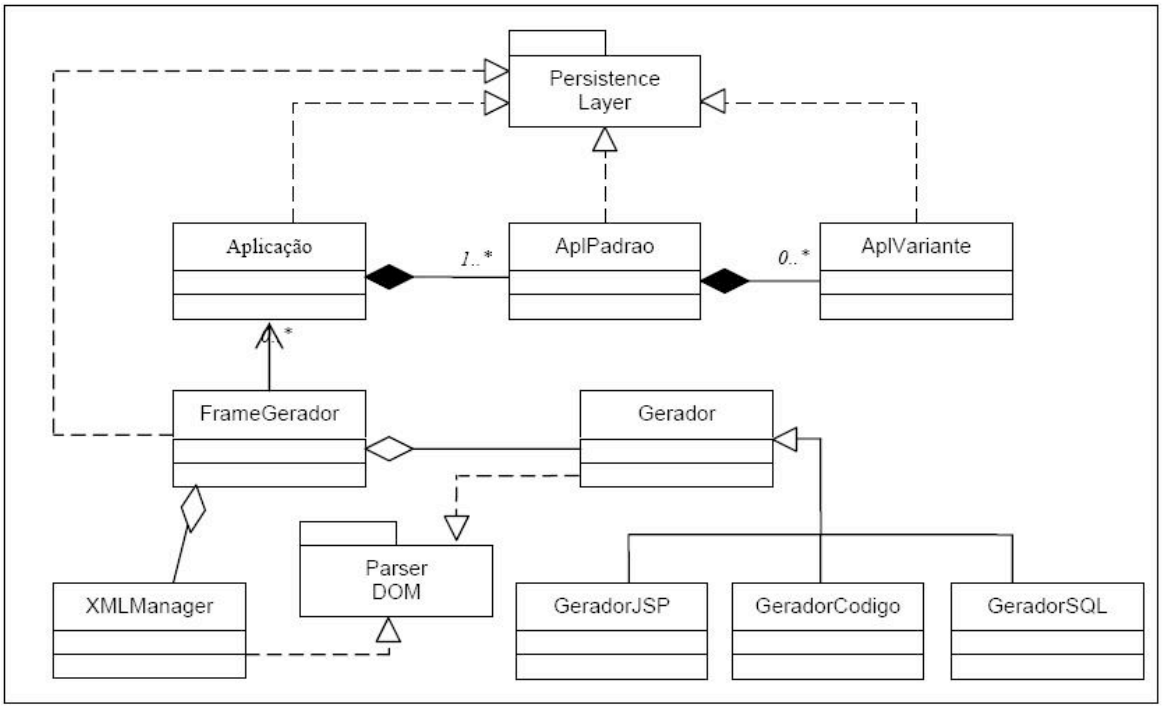


Figura 2.3 - Diagrama de Classes do gerador GAwCRe (PAZIN, 2004).

O padrão *Persistence Layer* (PAZIN, 2004) foi utilizado para armazenar as informações das aplicações no banco de dados relacional, bem como para persistir os dados das aplicações geradas pelo gerador. As classes *Aplicação*, *AplPadrao* e *AplVariante* representam, respectivamente as aplicações, os padrões e as variantes de cada aplicação instanciada pelo gerador. A classe *FrameGerador* exibe a interface de instanciação das aplicações por meio de um objeto do tipo *frame*, construído de forma dinâmica com base nas informações do XML que são referentes a LMA. A classe *XMLManager*, utilizando o *parser* DOM (lê e manipula o XML), obtém os dados a serem exibidos pela classe *FrameGerador*.

As classes *GeradorCodigo*, *GeradorSQL* e *GeradorJSP* herdam da classe Gerador os métodos que manipulam os dados do documento XML. A classe *GeradorCodigo* gera o código Java das aplicações, *GeradorSQL* gera os *scripts* de criação das tabelas e a classe *GeradorJSP* cria a interface da aplicação com o usuário (PAZIN, 2004).

Os elementos responsáveis pelo processo de geração do gerador são: 1) as Bibliotecas para as Aplicações Geradas, 2) o Instanciador da LMA, 3) o Gerador de *scripts* SQL, 4) o Gerador das Classes Java (*Beans*) da Aplicação e 5) o Gerador das Interfaces JSP.

O gerador possui bibliotecas usadas em todas as aplicações por ele geradas. Essas estão hospedadas no servidor que interpreta as interfaces JSP. A interface de instanciação das aplicações é construída de forma dinâmica a partir das definições do documento XML. Cada um dos padrões da *SiGCli* com o seu conjunto de variantes (caso haja) é apresentado na interface do gerador na forma de *checkbox*[7], e o desenvolvedor deve selecionar os padrões e respectivas variantes que deseja implementar. Sempre que um *checkbox* é selecionado, as informações são armazenadas na base de dados do gerador, possibilitando posteriormente, a recuperação e a alteração da especificação LMA para cada aplicação gerada.

O módulo Gerador de *scripts* SQL obtém as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Com as informações das aplicações armazenadas na base de dados do gerador, o módulo Gerador das Classes Java (*Beans*) da Aplicação adquire as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Dessa forma, para cada padrão usado na aplicação, é feita a análise de quais classes, atributos, associações e métodos devem ser criados para satisfazer os requisitos da aplicação. O módulo Gerador das Interfaces JSP (SUN MICROSYSTEMS, 2009) também obtém informações da aplicação gerada comparando-as com as definições do documento XML e para cada padrão usado na aplicação, faz a análise de quais interfaces de interação com o usuário devem ser geradas e quais itens do *menu* da aplicação serão disponibilizados aos usuários (PAZIN, 2004).

A seleção dos padrões ocorre com a utilização de botões de navegação. Alguns padrões são optativos, outros possuem variantes e devem ser escolhidos ou não de acordo com os

[7] *Checkbox* - Elemento de interface gráfica que permite ao usuário fazer seleções múltiplas em um conjunto de opções.

requisitos da aplicação que está sendo instanciada, entretanto outros padrões são obrigatórios e o avanço para o próximo padrão só é possível após a seleção do padrão corrente. Após a seleção do último padrão, os artefatos podem ser gerados a partir do *menu* do instanciador e são então automaticamente gravados em um diretório criado, anteriormente, pelo gerador (BORGES, 2008).

As aplicações geradas pelo GAwCRe são aplicações Web e são desenvolvidas utilizando o conceito de arquitetura em três camadas. As solicitações e interações dos usuários são feitas na camada de apresentação. As páginas são geradas pela camada de aplicação e exibidas por meio de um navegador. A camada de aplicação, quando necessário, se comunica com a camada de persistência que é responsável pelo armazenamento e pela recuperação das informações do sistema.

Após gerar a aplicação, é preciso disponibilizá-las em um servidor Web configurado para interpretar código Java (PAZIN, 2004).

2.4.1. Manutenções realizadas no GAwCRe

Em sua versão original o GAwCRe utilizava o SGBD Oracle para armazenamento de dados do gerador e das aplicações criadas a partir dele. Após uma manutenção adaptativa *ad hoc* passou a utilizar o SGBD MySQL (RIZZO, 2005).

O GAwCRe foi utilizado como ferramenta para a geração automatizada de artefatos em uma abordagem de reengenharia ágil - ARA substituindo o uso de um framework construído com base em linguagens de padrões de análise (CAGNIN, 2005). Para avaliar essa proposta o sistema exemplo utilizado foi um sistema pertencente a um domínio conexo ao seu, devido à dificuldade em encontrar um sistema legado disponível na Web no domínio de clínicas de reabilitação física. Assim, foi preciso expandir o GAwCRe para que sistemas de domínio conexo ao seu pudessem ser atendidos. Os seguintes problemas foram encontrados nesse caso: a) o código do gerador não segue padrão de codificação e não é flexível; b) há erros de lógica; c) faltam instruções para a instalação e operação do gerador; d) o registro e a validação das manutenções anteriores é superficial ou inexistente; e) a instalação dos artefatos gerados não é automatizada e faltam instruções para realizá-la; f) há presença de código duplicado e métodos extensos; e g) não há controle das versões geradas (BORGES, 2008).

Diante desses problemas, novas atividades de manutenção foram propostas e realizadas (BORGES *et al.,* 2008). O GAwCRe foi ampliado para o domínio de Clínicas de Odontologia bem como foram feitas adaptações para apoiar o desenvolvimento Web de sistemas com características semelhantes às existentes nos padrões modificados da SiGCli (FERREIRA, 2008). Também foi disponibilizado o controle de versões para que possa haver controle de cada uma das versões geradas para os sistemas (BORGES, 2008).

Mesmo com essas modificações o gerador ainda apresentava trechos de código mal construído, sem o uso de padrões, com alto acoplamento e baixa coesão de dados. Essas características dificultaram a realização de uma manutenção adaptativa no GAwCRe que o tornasse mais flexível, atendendo a outros domínios, com código mais legível e estruturado e utilizando as principais normas de implementação preconizadas atualmente. Dessa forma, o gerador passou por um processo de manutenção corretiva e evolutiva que será apresentado no Capítulo 4.

Além disso, o GAwCRe será usado neste trabalho como estudo de caso para o uso de serviços providos por uma Arquitetura Orientada a Serviços em geradores de aplicação.

2.5. Considerações Finais

Este capítulo apresentou geradores de aplicação, um dos principais conceitos utilizados nesta dissertação de mestrado. Essa técnica de reúso transforma especificações de alto nível em produtos da aplicação provendo projetos genéricos que são construídos por meio de parametrização de componentes, evitando a inserção manual de erros na fase de codificação, permitindo o aumento de produção e da qualidade dos sistemas gerados. O gerador GAwCRe (PAZIN, 2004), descrito neste capítulo, foi usado neste trabalho como estudo de caso.

No GAwCRe as especificações de seu domínio são definidas em uma LMA. Dessa forma, para acrescentar novas funções no gerador, é necessário adicioná-las em sua LMA. A geração do código das aplicações no GAwCRe é feita por uma classe que realiza a junção da LMA com as informações passadas pelo desenvolvedor na interface do gerador.

Outra maneira de gerar código é usando serviços eletrônicos providos por uma arquitetura orientada a serviços. Dessa forma, o gerador acrescentaria ao seu código as

informações sobre o serviço correspondente e esses estariam disponíveis para serem utilizados na aplicação gerada. O uso de serviços eletrônicos é uma abordagem mais flexível, pois uma modificação em sua estrutura interna não implicaria em mudanças no código dos produtos gerados.

O Capítulo 3 apresenta a arquitetura orientada a serviços, tecnologia de reúso que proverá a utilização de serviços eletrônicos em geradores de aplicação. Neste capítulo também serão descritas informações necessárias para a fundamentação do trabalho realizado.

Capítulo 3

Arquitetura Orientada a Serviços SOA

3.1. Considerações Iniciais

O segredo para o crescimento e a competitividade organizacional é a habilidade para alterar e otimizar processos de negócios com prontidão (MICROSOFT, 2009). As empresas precisam conectar processos, pessoas e informações dentro dos limites organizacionais e através desses limites. Precisam também conectar subsidiários ou parceiros comerciais. A falta ou dificuldade de integração entre os ativos (sistemas, aplicações e dados) de Tecnologia de Informação - TI - dificulta uma resposta rápida e eficaz às necessidades de negócios em constante mudança. Essa falta de flexibilidade aumenta os custos, diminui o grau de satisfação do cliente, impede a compatibilidade entre aplicações e diminui a produtividade dos funcionários (MICROSOFT, 2009). Em uma visão geral, a falta de integração é o maior desafio que as organizações enfrentam na tentativa de se manterem competitivas e crescendo.

Surge nesse contexto uma abordagem que promove melhor alinhamento entre a TI e as necessidades de negócios, e que permite aos funcionários, clientes e parceiros comerciais reagir mais rapidamente e se adaptarem às diferentes pressões de negócios. A Arquitetura Orientada a Serviço (AOS), do inglês *Service Oriented Architecture* (SOA), possibilita o fornecimento de uma nova geração de aplicações dinâmicas (também chamadas de aplicações compostas) que proporcionam aos usuários finais percepções e informações mais detalhadas e precisas de processos, além da flexibilidade necessária para acessar essas informações (IBM, 2009).

O objetivo da arquitetura orientada a serviços é o reúso intenso dos seus componentes (serviços), para que em médio prazo, a tarefa do desenvolvimento de uma aplicação seja constituído pela identificação, composição e coordenação dos serviços já implementados, aumentando o reúso, diminuindo o esforço gasto no projeto, diminuindo custos e facilitando a manutenção (FANTINATO *et al*., 2005; SORDI *et al*., 2006; BIANCO *et al*., 2007). Além do reúso, a adoção de uma arquitetura orientada a serviços facilita a adaptação de sistemas, fazendo com que eles se tornem dinâmicos na medida em que serviços possam ser substituídos em tempo de execução.

Um serviço para a arquitetura SOA é um componente de software que possui um contrato para descrever suas funções e que pode ser invocado por algum serviço ou invocar outros serviços. Deve funcionar de forma independente do estado de outros serviços e deve possuir interface bem definida. Normalmente, a comunicação entre o sistema que deseja usar o serviço (cliente) e aquele que disponibiliza o serviço (fornecedor) é realizada por meio de *Web Services* (serviços Web) (KRAFZIG *et al*., 2004).

A SOA aproveita o atual cenário tecnológico com o enfoque na distribuição do processamento, oferecendo suporte a heterogeneidade ambiental, onde diversos tipos de processadores, sistemas operacionais e linguagens de programação distintas interagem (MAHMOUD, 2005).

Neste capítulo a arquitetura orientada a serviços e as tecnologias envolvidas na sua construção são conceituadas. Na Seção 3.2 é comentada a opinião de diversos autores sobre arquiteturas de software e a escolha daquela que melhor se encaixa aos conceitos utilizados neste trabalho. Na Seção 3.3 são descritos os fundamentos de SOA. Na Seção 3.4 as características para que uma aplicação seja considerada orientada a serviços são descritas. Na Seção 3.5 os elementos envolvidos nessa arquitetura são apresentados. Na Seção 3.6 é relatada as classificações de serviços por diferentes autores. Na Seção 3.7 são retratadas as tecnologias que apoiam à implementação de uma SOA. Na Seção 3.8 os desafios existentes na construção de aplicações orientadas a serviços são destacados. Na Seção 3.9 são comentadas algumas abordagens utilizadas com SOA. Na Seção 3.10 é apresentado o modelo de referência para SOA e na Seção 3.11 são comentadas as considerações finais.

3.2. Arquitetura de Software

O desenvolvimento de sistemas de informação empresarial sofre de falta de agilidade e ineficiência, afirmaram Krafzig *et al*. (2004). Atualmente as empresas ainda sofrem desses problemas, pois dependem de sistemas de informação empresariais que estão ligados à organização, aos processos e modelos de negócios das empresas e esses mudam constantemente. São mudanças em leis, fusões de empresas, alterações nos processos para aumentar competitividade entre outras mudanças que os sistemas de informação devem acompanhar da forma mais rápida possível.

A rápida manutenção nesses sistemas de informação pode ser atingida se o sistema utilizar boa arquitetura de software, ou seja, se possuir simplicidade, flexibilidade, manutenibilidade, reusabilidade e desacoplamento tanto de funções quanto de tecnologia. A arquitetura serve como uma estrutura que permite o entendimento de componentes de um sistema e seus relacionamentos (BARAGRY e REED, 1998).

Ainda de acordo com Baragry e Reed (1998), não há uma definição unânime para arquitetura de software. Dentre as definições existentes podem-se destacar as seguintes:

- Arquitetura de software é a organização em alto nível dos elementos computacionais e a interação entre esses elementos (CHAUDET *et al*., 2000);

- Arquiteturas de software descrevem o sistema em um alto nível de abstração com o uso de três blocos de construção: componentes que representam unidades computacionais, conectores que representam protocolos de comunicação e configurações que caracterizam a topologia do sistema em termos de interconexão dos componentes via conectores (ISSARNY e BANATRE, 2001);

- Arquitetura de software de um programa ou sistema de computação é a estrutura ou estruturas do sistema, que compreende os elementos de software, as propriedades externamente visíveis desses elementos e as relações entre eles (BASS *et al*., 2003; PRESSMAN, 2006);

- Uma arquitetura de software é um conjunto de expressões que descrevem componentes de software e designa a funcionalidade do sistema para esses componentes. Descreve a estrutura técnica, restrições e características dos componentes e interfaces entre eles. Arquitetura é implicitamente o plano em alto nível para construção de sistemas (KRAFZIG *et al*., 2004);

- Arquitetura de software engloba algumas das decisões de projeto antecipadas, que são difíceis e caras para mudar em outras fases do desenvolvimento (BABAR *et al*., 2005);

- Arquitetura de software descreve a estrutura bruta do sistema de software explicitamente em termos de componente, conector, configuração, restrições e projeto (YING *et al*., 2006);

- Arquitetura de software é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles, ou seja, trata basicamente de como os componentes fundamentais de um sistema se relacionam intrinsecamente e extrinsecamente (ANSI/IEEE, 2000).

A definição de arquitetura adotada neste trabalho foi a da ANSI/IEEE (2000) e a de Krafzig *et al*. (2004), pois, são as mais abrangentes e que melhor se assemelham a este trabalho.

3.3. Fundamentos da Arquitetura Orientada a Serviços

Arquiteturas de software têm evoluído de estáticas, monolíticas e centralizadas, para arquiteturas dinâmicas, modulares e distribuídas (BARESI *et al*., 2006). Nesse contexto, SOA pode ajudar a atingir essas características em um projeto (KRAFZIG *et al*., 2004).

Mahmoud (2005) define SOA como sendo um estilo arquitetural para a construção de aplicações de software que utilizam serviços disponíveis em uma rede como a Web. Para Lewis *et al*. (2007) SOA não é uma arquitetura, é um padrão arquitetural do qual uma infinidade de arquiteturas podem ser derivadas.

Uma arquitetura orientada a serviços tem como seu componente fundamental o conceito de serviços. Esses são implementações de uma funcionalidade de negócios bem definida, que pode ser utilizada por clientes de diferentes aplicações e processos de negócios. Um serviço possui um contrato público que descreve quais funções ele oferece e permite que o serviço possa invocar outro serviço assim como ser invocado. Um serviço possui também uma ênfase em interoperabilidade e disponibilidade, podendo ligar-se dinamicamente a outros serviços em diversas plataformas computacionais (MAHMOUND, 2005).

De acordo com Papazoglou e Georgakopoulos (2003), SOA utiliza serviços eletrônicos como elementos fundamentais para o desenvolvimento de aplicações distribuídas nas quais a comunicação entre os serviços acontece via troca de mensagens. Esses serviços permitem que as empresas exponham suas competências principais em termos de programação, por meio da Internet, com o uso de linguagens e protocolos baseados em XML (W3C, 2009), implementados via interfaces auto descritivas que usam padrões abertos (padrões disponíveis para livre acesso e implementação, como o próprio XML) (PAPAZOGLOU, 2003).

Os primeiros passos para uso de SOA foram o CORBA (SIEGEL, 2000) e DCOM (THAI, 1999) que utilizavam tecnologia de middleware EAI[8] (em português - Integração de Aplicações Empresariais). Embora muito tenha sido desenvolvido, problemas como alta complexidade e alto custo associados às tecnologias EAI inviabilizaram sua utilização em maiores proporções (ALONSO *et al*., 2004).

Papazoglou *et al*. (2006) se refere a SOA como uma maneira lógica de projetar sistemas de informação para prover serviços tanto para aplicações de usuário final quanto a outros serviços distribuídos na rede, por meio de interfaces publicáveis e acessíveis. Mahmoud (2005) completa essa afirmação ressaltando que a interoperabilidade é um dos princípios mais importantes das SOAs e que essa arquitetura tem importância à medida que surgem novas tecnologias para o desenvolvimento de sistemas de software. O autor ressalta que existe a necessidade de interoperabilidade entre sistemas desenvolvidos utilizando diferentes tecnologias, incluindo a integração com sistemas legados.

[8] Do inglês *Enterprise Application Integration.*

3.4. Características da SOA

Há algumas características consideradas relevantes para que uma aplicação seja caracterizada como orientada a serviço (MACHADO, 2004; OASIS, 2006):

- **Reúso Caixa-Preta**

O reúso caixa-preta consiste na utilização de elementos cujo comportamento interno é desconhecido. O uso desses elementos ocorre por meio de um contrato bem definido que pode ser, por exemplo, uma interface Java ou um arquivo XML. Uma aplicação baseada em uma arquitetura orientada a serviços é composta de diversos elementos que se relacionam conforme a descrição de seus contratos. Essa característica facilita a manutenção e adaptação da funcionalidade da aplicação.

- **Distribuição**

Aplicações orientadas a serviços são compostas por elementos funcionais utilizados em múltiplos sistemas (provedor, consumidor, publicador) que estão localizados em pontos (máquinas) diferentes.

- **Heterogeneidade Ambiental**

Devido à distribuição, sistemas baseados em uma arquitetura orientada a serviços podem funcionar em ambientes heterogêneos, ou seja, compostos de hardwares, sistemas operacionais e linguagens de programação distintos. Portanto, a comunicação entre os elementos deve ser feita através de um protocolo padronizado e pré-definido.

- **Composição**

A composição permite a separação das funções de uma aplicação em unidades bem definidas. Assim, uma alteração no código não causa grande impacto na aplicação.

SOA pode fazer uso de alguns modelos formais e *frameworks* conceituais que deram origem a uma série de linguagens específicas para composição de componentes. Com o uso cada vez maior de linguagens reflexivas (aquelas que agem (computam) sobre si mesmas, alterando seu próprio comportamento, estrutura ou status, como Java e C#), a tarefa de composição de componentes tornou-se menos complexa, pois permitem, em tempo de

execução, carregar novos componentes e invocar métodos específicos de uma interface definida anteriormente através de reflexão.

- **Coordenação**

A coordenação é responsável pela sincronização, ordenação e temporização das aplicações. Em sistemas abertos e distribuídos, como os orientados a serviços, um modelo de coordenação é recomendável para que o dinamismo e a adaptabilidade sejam facilitados. Linguagens de coordenação foram criadas para permitir que um ou mais elementos se comuniquem de forma a coordenar operações buscando um objetivo compartilhado por ambas às partes. Exemplos de linguagens com esse intuito são BPEL/BPEL4WS (IBM, 2003; ANDREWS *et al.,* 2003).

- **Dinamismo e Adaptabilidade**

Aplicações orientadas a serviços são abertas e dinâmicas, pois conseguem se adaptar às mudanças de requisitos com facilidade. As aplicações dinâmicas permitem que empresas aprimorem e automatizem tarefas, além de orquestrar processos de negócios compatíveis com políticas internas e regulamentações externas. O resultado é que essas empresas ganham a agilidade necessária para um desempenho superior no mercado (ZHANG, 2007).

- **Estado**

Um elemento pode ter seu estado divido em persistente ou conversacional. Estado persistente é definido quando existe a possibilidade de pelo menos uma das partes persistir dados trocados na comunicação entre as partes. O estado conversacional trata do estado dos elementos somente durante a comunicação ou conversação, depois de finalizada a conversação entre os componentes, o estado é desfeito e os dados liberados. Essa classe de estado é importante para manter o andamento de operações mais complexas. As aplicações orientadas a serviços são construídas apenas com componentes que não possuem estado conversacional.

- **Sincronia**

As únicas maneiras que os elementos possuem de se comunicar e sincronizar suas ações são através da troca de mensagens, da semântica e do seu modo de transmissão e essas são partes fundamentais em sistemas distribuídos. A troca de mensagens, quanto à sincronia

entre os componentes, é classificada em: comunicação assíncrona e comunicação síncrona. A Assíncrona é definida quando o processamento do elemento que enviou a mensagem não é bloqueado a espera do fim da mensagem. Na comunicação síncrona ocorre o bloqueio do processamento até que o fim ou retorno da mensagem seja concluído.

Como não há estado conversacional na maioria dos protocolos usados para comunicação, principalmente protocolos sobre *HTTP*, as aplicações orientadas a serviços atuais costumam ser de comunicação síncrona.

- **Robustez de Protocolos**

A robustez de protocolo pode ser definida como a liberdade dada aos elementos envolvidos em uma comunicação, de evoluir sem que a comunicação seja desfeita, ou seja, se um cliente estiver acessando um servidor e esse sofrer qualquer tipo de manutenção, ainda assim o serviço deve continuar sendo fornecido ao cliente. A comunicação entre essas partes precisa de um protocolo e eles são de muita relevância em ambientes distribuídos e heterogêneos (como os orientados a serviços) onde as partes estão conectadas através de uma rede de comunicação. Nesses casos, questões como desempenho, robustez e segurança devem ser consideradas.

Por serem dinâmicas e alteradas constantemente, as aplicações orientadas a serviços precisam de protocolos robustos. A maior parte dessas aplicações utiliza o protocolo de comunicação SOAP *(Simple Object Access Protocol)*, baseado em XML. Esse protocolo será descrito na Seção 3.7.1.

De acordo com a Oasis (2006) para uma aplicação ser considerada orientada a serviços não é preciso ter todas as características descritas anteriormente. A utilização de serviços de outros componentes é feita através de interfaces que são conectáveis, de forma que o componente possa ser desconectado de um ambiente e substituído por outro que execute o mesmo serviço, sem provocar alterações nos objetivos do sistema.

Devido às características de distribuição, heterogeneidade ambiental e robustez de protocolos surgidos e utilizados na arquitetura orientada a serviços, foi possível um melhor aproveitamento de serviços criados em plataformas diferentes. Essa arquitetura propõe uma nova forma de construção de aplicações, baseada na distribuição e administração remota de

serviços. Tais serviços permitem que sejam feitas uma ou mais requisições e que sejam oferecidas uma ou mais respostas por meio de uma interface predefinida e padronizada.

3.5. Elementos de uma Arquitetura Orientada a Serviços

A SOA é um relacionamento entre três participantes: o fornecedor (provedor –> *Service Provider*) de serviços, o registro (repositório/catálogo –> *Service Registry*) de serviços e o cliente (consumidor –> *Service Requester*) de serviços. As operações que envolvem esses participantes são publicação, busca e conexão (PAPAZOGLOU, 2003). A Figura 3.1 ilustra as colaborações realizadas entre esses participantes.

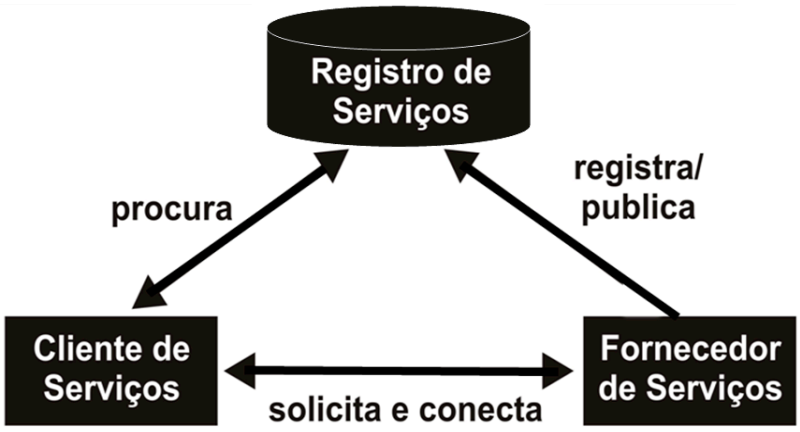


Figura 3.1 - Colaborações em SOA. Fonte Adaptada: Papazoglou e van den Heuvel (2007).

O fornecedor de serviços contém a implementação das funcionalidades do serviço e a descrição dele que são publicadas no registro de serviços. Essa publicação permite ao cliente localizá-lo e solicitá-lo ao fornecedor, que estabelece a conexão e se encarrega de executar as funções, também fazendo a distribuição de chamadas internas necessárias.

Krafzig *et al*. (2004) identificam quatro componentes em uma arquitetura orientada a serviços: 1) aplicação *frontEnd*, responsável por iniciar os processos de negócios; 2) o serviço que representa uma funcionalidade do negócio e possui interface, um contrato e sua implementação e essa por sua vez pode possuir a lógica de negócios, dados ou ambos; 3) o repositório de serviços que armazena os contratos de serviços para que possam ser buscados e;

4) o *service bus* que conecta a aplicação *frontEnd* (fornecedor) e os serviços. Na Figura 3.2 é exibida a hierarquia desses componentes.

Figura 3.2 - Elementos envolvidos em uma SOA. Fonte: Krafzig *et al.* (2004).

Krafzig *et al*. (2004) descrevem a aplicação *frontEnd* como a parte ativa da SOA. É quem inicia e controla todas as atividades e não interage necessariamente com o usuário final. Essa aplicação pode delegar responsabilidades para um ou mais serviços e sempre inicia um processo de negócio e recebe seus resultados.

Um *serviço* para Krafzig *et al*. (2004) é um componente de software que tipicamente encapsula conceitos de negócios de alto nível. O serviço, como exibido na Figura 3.2, possui: a) um contrato que provê uma especificação do propósito, das funcionalidades, restrições e uso do serviço; b) uma interface que expõe as funcionalidades dos serviços para os clientes e é apresentada fisicamente por um *service stub* (serviço conversor de dados); e c) uma implementação que provê a lógica de negócios requerida, os dados apropriados ou ambos.

O *repositório de serviços* provê facilidades para descobrir serviços e adquirir todas as informações necessárias para usá-los. Pode oferecer, além das informações contidas no contrato (WSDL[9] - ver Tabela 3.2), localização física, informações sobre o provedor, restrições técnicas, dentre outras (KRAFZIG *et al*., 2004).

[9] WSDL - *Web Services Description Language*. http://www.w3.org/TR/wsdl

O *service bus* faz a integração de diferentes tipos de serviços por meio de mensagens, manipulação de eventos e gerenciamento de performance de negócios. Conecta todos os participantes da SOA e não é composto necessariamente por uma única tecnologia. Provê técnicas como auditoria, segurança, dentre outros (KRAFZIG *et al*., 2004).

3.6. Classificação de Serviços

Os serviços podem ser classificados de acordo com a sua função, complexidade ou categoria de reúso. É importante conhecer os diferentes tipos de serviços para que possam ser utilizados durante o gerenciamento de mudanças, diante de situações em que é preciso estimar custos e prazos de desenvolvimento, ao separar segmentos de código com diferentes características de reúso e ao escolher a melhor estratégia de implementação (KRAFZIG *et al*., 2004).

Os serviços segundo Krafzig *et al*. (2004) são classificados em:

- **Serviços básicos**: podem ser centrados em dados, conter lógica de negócios ou ambos e são basicamente servidores em uma SOA. Por exemplo, empresas com serviços que validam cartão de crédito, verificam a condição do crédito do cliente. Esses serviços contêm a lógica de negócios e o poder de decisão e estão diretamente ligados aos dados requisitados e oferecidos;

- **Serviços intermediários**: podem ser clientes ou servidores em uma SOA, não possuem estados, fazem a ligação entre tecnologias inconsistentes e geralmente são específicos de cada projeto. Por exemplo, serviços que ligam duas aplicações independentes e implementadas em linguagens diferentes. O serviço intermediário permitirá a comunicação entre esses outros dois serviços que, provavelmente, fazem parte de projetos diferentes;

- **Serviços centrados em processo**: encapsula o conhecimento do processo de negócio da organização, podem ser clientes ou servidores em uma SOA, separam a lógica de processo da lógica de negócios e correspondem ao tipo mais sofisticado de serviços. Por exemplo, serviços na linguagem de descrição de negócios BPEL

(ANDREWS *et al*., 2003) que modelam a lógica de negócios e que centralizam o acesso a outros serviços;

- **Serviços de empresa pública**: provê interfaces para integração entre empresas, possui granularidade alta e exige uma maior segurança e robustez. Suas operações costumam ser controladas por um *Service Level Agreement* (SLA[10]). Por exemplo, o serviço de CEP dos correios ou de consulta de CPF da receita federal. A empresa disponibiliza o serviço e o cliente efetua um contrato com a empresa fornecedora desse serviço para que possa utilizá-lo.

Erl (2005) propõe outra classificação para serviços chamada de modelo de serviços. Por esse modelo os serviços são classificados em utilitários, de negócios, controladores, *proxy*, *wrapper*, de coordenação para transações atômicas, processos ou coordenadores para atividades de negócios.

É possível traçar um paralelo entre essas duas classificações apresentadas. Os serviços utilitários e de negócios de Erl (2005) correspondem aos serviços básicos de Krafzig *et al*. (2004). Os serviços controladores, de coordenação para transações atômicas, de processos e coordenadores para atividades de negócios correspondem aos serviços centrados em processo. *Proxy* e *wrapper* correspondem aos serviços intermediários. Na Tabela 3.1 é ilustrado esse paralelo.

[10] SLA é um contrato entre um fornecedor de serviços de TI e um cliente em que devem ser especificados, de maneira geral e em termos mensuráveis, quais serviços o fornecedor vai prestar.

Tabela 3.1 - Correspondência da classificação de serviços segundo Krafzig *et al.* (2004) e Erl (2005).

Classificação de Krafzig *et al.* (2004) para serviços.	Correspondentes segundo a classificação e Erl (2005)	Exemplo
Serviços Básicos	Serviços utilitários e de negócios	Empresas com serviço de validação de cartões de crédito. Eles contêm a lógica de negócios e o poder de decisão, pois, estão diretamente ligados aos dados requisitados e oferecidos e a validade dos mesmos. São servidores do serviço.
Serviços Intermediários	*Proxy* e *wrapper*	Serviços que fazem a ligação entre duas aplicações independentes. Por exemplo, aplicações implementadas em linguagens diferentes, por exemplo. Esse serviço intermediário permitirá a comunicação entre esses dois serviços que, provavelmente, fazem parte de projetos diferentes.
Serviços Centrados em processo	Serviços controladores, de coordenação para transações atômicas, de processos e coordenadores para atividades de negócios	Serviços em BPEL. Modelam a lógica de negócios. BPEL é um web service que centraliza o acesso a outros serviços Web de vários outros web services.
Serviços de Empresa Pública	----------	Serviço de CEP dos Correios. A empresa disponibiliza um serviço público e um cliente para utilizá-lo efetua um contrato com a empresa fornecedora do serviço.

3.7. SOA e tecnologias de suporte

Zaupa *et al.* (2007) tratam SOA como uma tecnologia capaz de oferecer infraestrutura para resolver problemas relacionados a métodos de desenvolvimento de aplicações Web como excesso de modelagem e programação, além da pouca reutilização. Esses fatores propiciam aumento de custo e de tempo de desenvolvimento de aplicações Web.

A SOA utiliza serviços eletrônicos como elementos fundamentais para o desenvolvimento de aplicações distribuídas e, atualmente, os serviços Web, que são um tipo específico de serviços eletrônicos, estão sendo apresentados como tecnologia para efetiva automação das interações interorganizacionais baseada em SOA (FANTINATO *et al*., 2005).

Embora existam diversos serviços disponibilizados na rede (Internet) que são chamados de serviços Web, parte deles são serviços eletrônicos implementados com tecnologias diferentes de *Web Services*[11] (ver Seção 3.7.1). Dessa forma, a terminologia serviços Web será adotada neste trabalho como referência aos serviços implementados com o modelo *Web Services*. A seção a seguir descreve os serviços Web e o seu uso.

[11] *Web Services* - http://www.w3.org/2002/ws

3.7.1. Serviços Web (*Web Services*)

É comum uma confusão com os termos SOA e serviços Web, muitos acham que são equivalentes. Embora frequentemente usados em conjunto, não são a mesma coisa. A SOA é uma abordagem arquitetural que visa construir sistemas a partir de um conjunto de elementos fracamente acoplados (serviços) que podem ser combinados dinamicamente. Serviço Web é uma metodologia para implementar aplicações em uma SOA e refere-se a um conjunto de tecnologias baseadas em XML que podem ser usadas para a construção de sistemas SOA (LEE *et al.,* 2006).

Os serviços Web (ALONSO *et al*., 2004; FANTINATO *et al*., 2005; LEE *et al.,* 2006) são componentes abertos e auto descritivos acessíveis a outras aplicações ou serviços Web por meio de um endereço URL[12] (em português - Localizador Uniforme de Recursos), assim ele oferece apoio à composição de soluções distribuídas, além de permitir que aplicativos comuniquem entre si de modo independente da plataforma de software e hardware ou da linguagem de programação, facilitando a integração entre aplicações. As mensagens são descritas com base em XML (*eXtendible Markup Language*) descrevendo contratos de aplicativos em uma linguagem de definição chamada WSDL, ambas obedecem aos protocolos abertos para Internet. Um serviço Web é então, um conjunto de operações acessíveis a partir de um protocolo da rede (SOAP) que são definidas por uma descrição de serviços (WSDL).

O uso desse conjunto de operações não é algo novo. O que há de novo é a utilização de protocolos abertos (XML, HTTP, SOAP, WSDL, etc.), controlados pelo *World Wide Web Consortium* (W3C[13]) e usados na tentativa de resolver um antigo desafio em sistemas distribuídos: a localização e acesso a elementos remotos.

Na Tabela 3.2 é apresenta a definição e a utilização das principais tecnologias aplicadas nos serviços Web.

[12] URL - Do inglês - *Uniform Resource Locator.*
[13] W3C - http://www.w3.org.

Tabela 3.2 - Principais tecnologias utilizadas pelos serviços Web.

Tecnologia	Descrição	Como essa tecnologia é usada no serviço Web
XML[14] (*eXtendible Markup Language*)	Linguagem de marcação de dados baseada em *tags*, que oferece um padrão para descrever dados estruturados de modo a facilitar a declaração do conteúdo.	Os contratos que descrevem os serviços são especificados em XML usando a sintaxe WSDL.
WSDL[15] (*Web Services Description Language*)	Linguagem baseada em XML utilizada para descrever os serviços Web. É um documento XML.	Responsável pela descrição dos contratos dos serviços. Todo serviço Web deve ter seu contrato descrito em WSDL. Um documento WSDL descreve a assinatura das operações (métodos) fornecidas por um serviço Web, bem como sua localização, como acessá-lo, protocolo utilizado e outras informações relevantes para o cliente.
SOAP[16] (*Simple Object Access Protocol*)	Protocolo que define o envio de objetos por meio de uma mensagem no formato de um documento XML. Portanto, é um protocolo flexível que permite a comunicação entre máquinas em ambientes heterogêneos.	Responsável pela comunicação entre cliente e fornecedor de um serviço, de acordo com a descrição de seu contrato definida em WSDL. As mensagens SOAP podem ser transportadas por meio de protocolos utilizados na Internet, principalmente o HTTP.
UDDI[17] (*Universal Description, Discovery, and Integration*)	Padrão que define a estrutura e o conteúdo dos registros de serviços. É um catálogo de serviços para publicar e pesquisar dados sobre serviços Web.	Possibilita a publicação, descoberta e invocação de serviços no registro de serviços.
HTTP[18] (*Hypertext Transport Protocol*)	Protocolo que permite transportar requisições e respostas entre clientes e fornecedores.	Os serviços são disponibilizados em plataforma Web (servidores HTTP).

Quando um fornecedor deseja disponibilizar um serviço para consumidores (clientes), ele o descreve através de WSDL e registra o serviço, opcionalmente, em um repositório (registro/catálogo) UDDI. O repositório passa então a manter referências para o descritor

[14] XML - http://www.w3.org/XML
[15] WSDL - http://www.w3.org/TR/wsdl
[16] SOAP - http://www.w3.org/TR/soap
[17] UDDI - http://www.uddi.org/pubs/uddi_v3.htm
[18] HTTP - http://www.w3.org/Protocols

WSDL e para o serviço. Quando um consumidor deseja utilizar um serviço, ele realiza uma pesquisa no UDDI para encontrar um serviço que atenda a suas necessidades, obtendo a descrição WSDL do serviço e o ponto de acesso ao mesmo. Por fim, o consumidor usa o descritor WSDL para construir uma mensagem SOAP com a qual irá se comunicar com o serviço. Caso o consumidor já conheça a URL de endereço do descritor WSDL do serviço Web, a etapa de busca através de um catálogo UDDI pode ser ignorada. Esse processo é descrito na Figura 3.3.

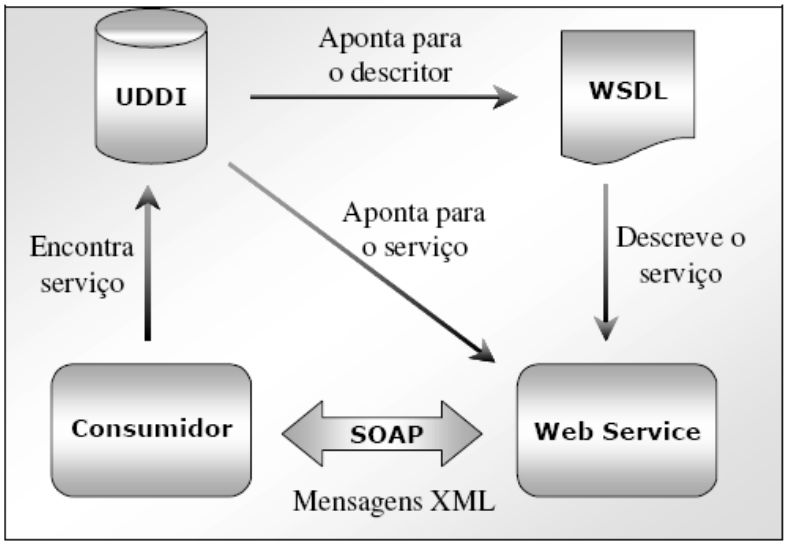


Figura 3.3 - Modelo SOA para serviços Web. Fonte: Fantinato *et al.* (2005).

3.7.2. Utilizando a API JAX-WS para criar os serviços Web

A Java XML API[19] para *Web Services* é uma biblioteca de *Web Services* para Java (JAX-WS[20]), que permite implementar serviços baseados nas normas XSD[21], WSDL[22] e SOAP[23]. A JAX-WS define o mapeamento de WSDL para Java e vice-versa. O mapeamento pode ser configurado através de anotações no WSDL ou no código Java.

[19] API - Do inglês - *Application Programming Interface*.
[20] JAX-WS - https://jax-ws.dev.java.net
[21] XSD - http://www.w3.org/XML/Schema
[22] WSDL - http://www.w3.org/TR/wsdl
[23] SOAP - http://www.w3.org/TR/soap

Com a JAX-WS é possível implementar serviços Web partindo de um contrato WSDL ou partindo de código Java. A abordagem que melhor garante a interoperabilidade é partir do contrato WSDL. Com a JAX-WS é também possível criar código cliente de invocação de serviços Web. Ela permite ao programa cliente invocar funções do programa servidor (chamadas remotas) como se fossem funções locais. Todos os aspectos de comunicação e de representação de dados são tratados pela biblioteca de RPC[24] e por objetos gerados automaticamente designados por *stubs* e *ties* que são objetos que convertem da representação de dados no programa cliente e no programa servidor (ex. objetos Java) para uma representação em formato *standard*, para transportar na rede (ex.: XML).

Os passos abaixo são executados por uma chamada remota:

1. O programa cliente invoca uma função dos *stubs*;
2. Os *stubs* convertem os objetos argumento numa mensagem;
3. Estabelecem uma ligação com o servidor através da rede;
4. A mensagem gerada pelos *stubs* é transmitida;
5. O servidor recebe a mensagem;
6. Os *ties* convertem a mensagem para objetos;
7. O servidor recebe os objetos, processa o pedido e gera objetos de resultado;
8. Os objetos de resultado são convertidos pelos *ties* numa mensagem de resposta, que é devolvida ao cliente;
9. O cliente recebe a resposta;
10. Os *stubs* convertem a resposta para objetos;
11. O programa cliente recebe os objetos de resposta, como se tivesse apenas acabado de chamar uma função local.

Enquanto a chamada remota é efetuada, o programa cliente fica bloqueado à espera, configurando assim, uma chamada síncrona.

[24] RPC - Do inglês - *Remote Procedure Call.*

3.7.3. WSDL

WSDL é uma linguagem de especificação de serviços que utiliza XML como base. Assim, um documento WSDL contém especificações em XML com regras específicas que fornecem as descrições de um serviço Web. Através dela, são descritos os serviços externos ou interfaces que são oferecidos por uma determinada aplicação, independente de sua plataforma ou linguagem de programação.

O seu principal objetivo é descrever as interfaces apresentadas e apontar a localização dos seus serviços disponíveis em um local previsível e bem conhecido na rede, permitindo que o cliente acesse de maneira confiável. Por ser um documento XML, sua leitura se torna fácil e acessível. Um documento WSDL descreve quatro partes de dados (CERAMI, 2002): 1) Interfaces: descreve as funções publicamente disponíveis; 2) Tipos de dados: descreve os tipos de dados necessários nas mensagens e nas respostas das mensagens; 3) Conexão: descreve o protocolo de transporte a ser usado no serviço; 4) Endereço: descreve onde localizar o serviço.

A especificação do documento WSDL é dividida em seis elementos principais: <*definitions*>, <*type*>, <*message*>, <*portType*>, <*binding*> e <*service*>. Na Figura 3.4 é ilustrada uma representação da especificação do documento WSDL.

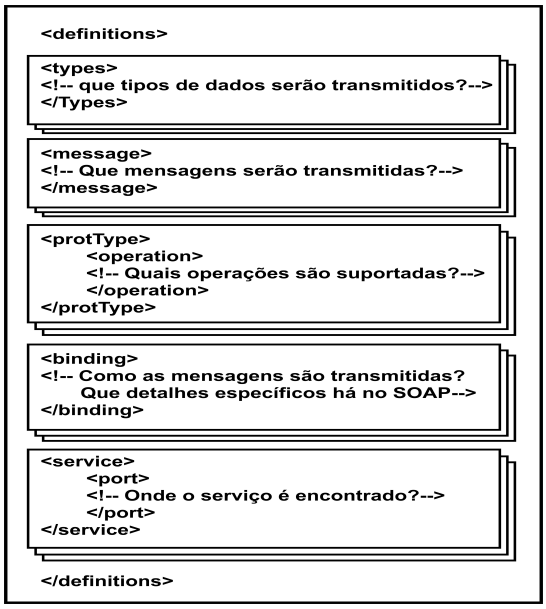


Figura 3.4 - Especificação do WSDL.

O elemento <*definitions*> (definições) é o elemento raiz do documento WSDL e contém a definição do nome do serviço e dos *namespaces* que são referências a arquivos externos ao documento.

```
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
targetNamespace="urn:server.hello" name = " helloWorld "
...
</definitions>
```

Listagem 3.1 - Exemplo do elemento <*definitions*>.

O elemento <*types*> (tipo) contém os tipos de dados que estão presentes na mensagem utilizados pelos serviços Web. Os tipos de dados são referenciados pelo elemento <*message*>.

```
<types>
 <xsd:schema targetNamespace="urn:server.hello">
  <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
 </xsd:schema>
</types>
```

Listagem 3.2 - Exemplo do elemento <*types*>.

O elemento <*message*> (mensagem) define os dados a serem transmitidos. Cada elemento <*message*> recebe um ou mais elementos <*part*>, que formam as partes reais da mensagem. O elemento <*part*> define o conteúdo da mensagem representando os parâmetros que são passados e a resposta que o serviço retorna.

```
<message name="helloRequest">
  <part name="name" type="xsd:string"/>
</message>
```

Listagem 3.3 - Exemplo do elemento <*message*>.

O elemento <*portType*> (tipo de porta) contém as operações de um serviço Web. Esse elemento combina elementos da mensagem para dar forma a uma operação de sentido único (requisição ou resposta) ou completo (requisição e resposta).

```
<portType name="server.helloPortType">
 <operation name="hello">
  <documentation>Retorna o nome</documentation>
  <input message="tns:helloRequest" />
  <output message="tns:helloResponse" />
 </operation>
</portType>
```

Listagem 3.4 - Exemplo do elemento < *portType* >.

O elemento <*binding*> (ligação) mapeia os elementos <*operation*> em um elemento <*portType*>, para um protocolo específico. Ele associa o elemento <*portType*>, ao protocolo SOAP, por exemplo, utilizando um elemento de extensão SOAP chamado <*wsdlsoap:binding*>, através de dois parâmetros: protocolo de transporte e o estilo da requisição (rpc ou *document*).

```
<binding name="server.helloBinding" type="tns:server.helloPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="hello">
    <soap:operation soapAction="urn:server.hello#hello" style="rpc"/>
    <input> <soap:body use="encoded" namespace="urn:server.hello"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/> </input>
    <output> <soap:body use="encoded" namespace="urn:server.hello"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/> </output>
  </operation>
</binding>
```

Listagem 3.5 - Exemplo do elemento < *binding* >.

O elemento <*service*> (serviço) define a localização real do serviço (endereço URL), tendo em vista que o mesmo pode conter várias portas <*ports*> e cada uma delas é específica para um tipo de ligação, descrita no elemento <*binding*>.

```
<service name="server.hello">
 <port name="server.helloPort" binding="tns:server.helloBinding">
  <soap:address location="http://localhost/imasters2/nuSOAP/server2.php"/></port>
</service>
```

Listagem 3.6 - Exemplo do elemento < *service* >.

Além dos seis elementos principais de um documento WSDL, extensões podem ser incluídas por meio de referências no elemento <*definitions*> possibilitando o uso do WSDL com outras abordagens que também dão suporte a SOA. Uma das extensões que pode ser incluída é o elemento <*partnerLinkType*> (link de parceiro) usado quando ocorre uma interação entre serviços Web parceiros e um processo de negócio BPEL (linguagem de descrição de negócio).

3.7.4. Utilizando anotações para construir serviços Web

Anotações são modificadores Java, semelhantes a *público* e *privado*, especificados no código. Um conjunto de anotações é fornecido para serviços Web pela especificação do JAX-WS. Algumas anotações são usadas para gerar artefatos. Outras são usadas para documentar seu código.

A plataforma Java EE 5[25] fornece anotações para: a) definir e usar serviços Web; b) mapear classes da tecnologia Java para XML; c) mapear classes da tecnologia Java para bancos de dados; d) mapear métodos para operações; e) especificar dependências externas; e) especificar informações de implantação, inclusive atributos de segurança, etc. (JCP, 2006).

Anotações utilizadas para fazer o mapeamento são marcadas com um caractere @. No IDE, quando um tipo que faz uso de anotações é criado, espaços reservados relacionados são fornecidos no código gerado. As anotações básicas utilizadas para a construção de serviços Web são:

- **@WebService** - Deve ser colocada na classe de implementação de *bean* de sessão. Ex.: @WebService(name,wsdlLocation,endpointInterface,portName). O *name* é o nome do serviço Web quando mapeado para WSDL e assume como padrão o nome da classe Java ou interface. O *wsdlLocation* define a url do documento wsdl. O *endpointInterface* externaliza o contrato do serviço Web na forma de uma interface Web. E *portName* é a porta WSDL utilizada;
- **@WebMethod** – Métodos anotados com @WebMethod estarão disponíveis para o serviço Web. Se a classe for anotada com @WebService e nenhum método com @WebMethod, todos os métodos estarão disponíveis no Web Service. Essa é uma boa prática de projeto para reduzir dependências entre módulos. Possui o atributo *operationName* utilizado para definir a operação WSDL que o método anotado implementa. Caso não seja especificado, o nome do método é utilizado;
- **@WebParam** – Permite controlar o WSLD gerado para um método Java sinalizado com @WebMethod. Se o estilo for RPC/LITERAL(wsdl:part), o atributo *name* configurará o nome wsdl:part;
- **@WebResult** - Fornece a mesma funcionalidade para valores de retorno que @WebParam oferece para parâmetros de métodos.

[25] Java EE 5 - http://jcp.org/en/jsr/detail?id=244

3.7.5. Processos de Negócio

Um processo de negócio é uma sequência de atividades executadas, possivelmente, por múltiplas organizações que atuam cooperativamente para atingir um objetivo comum de negócio. Tais atividades podem ser representadas por serviços (FANTINATO *et al.,* 2005).

Segundo Schmidt (2003) é necessário definir como os serviços são utilizados para implementar as atividades de um processo de negócio, bem como a forma em que o processo de negócio baseado em serviços é gerenciada. Um processo de negócio pode ser entendido como um novo serviço composto que agrega vários serviços. Por sua vez, esse novo serviço pode ser disponibilizado para uso em outros serviços, e assim sucessivamente. Um serviço composto pode ser invocado várias vezes, dentro ou fora da própria organização (LEYMANN *et al*., 2002).

3.7.5.1. Gerenciamento de Processos de Negócio

As organizações, em sua maioria, possuem sistemas heterogêneos, o que agrega maior complexidade ao gerenciamento de processos de negócio e causa impacto nas atividades de modelagem de processo, execução e monitoramento.

A integração desses sistemas heterogêneos tem sido cada vez mais utilizada entre as organizações na tentativa de redução de custos, troca de serviços, intercomunicação, reutilização, delegação e divisão de trabalho, entre outras vantagens. Isso tem produzido uma grande variedade de processos de negócios interorganizacionais, de forma que pesquisas têm sido feitas em busca de tecnologias, teorias ou técnicas para auxiliarem esse tipo de ambiente.

O gerenciamento dos processos de negócios exige troca de mensagens e a tecnologia de serviços Web tem sido atualmente uma abordagem que oferece recursos para suprir essa necessidade.

Gerenciamento de processos pode ser realizado por orquestração ou coreografia (PELTZ, 2003). Gerenciamento centralizado ou *Orquestração*: quando o gerenciamento do processo de negócio é realizado por uma única organização e o único ponto de controle fica no processo de mais alto nível. Existe um controlador central que coordena as atividades entre vários participantes passivos. Por analogia com uma orquestra musical, o controlador central é

o maestro responsável pela organização e execução de todo o processo, ao passo que os participantes são os membros da orquestra, os quais executam as suas tarefas de acordo com a delegação enviada pelo maestro. O processo é controlado sob a perspectiva individual de um participante, o controlador. Esse tem o conhecimento de como todas as tarefas individuais, que compõem o processo, funcionam. A orquestração deve ser vista como um processo de negócio executável, em que há uma implementação, um programa que controla todo o fluxo de trabalho, executando as tarefas e trocando mensagens. Os sistemas de gerência de *workflow*[26] configuram um bom exemplo desse tipo de programa (NOVAIS, 2007).

Existem diversas linguagens com sintaxe baseada em XML, para descrição de processos de negócios, dentre elas, BPEL4WS - *Business Process Execution Language for Web Service*s (ANDREWS *et al*., 2003; IBM, 2003), ou simplesmente BPEL. Essa é uma linguagem robusta que possui conceitos fortemente associados à ideia de orquestração, tendendo a ser um padrão de linguagem para esse tipo de gerenciamento de processos (ROSS-TALBOT, 2005).

Gerenciamento descentralizado ou *Coreografia*: quando o gerenciamento do processo de negócio é realizado por várias organizações, sendo que os processos estão em um único nível, precisando comunicar-se entre si para que as atividades de gerenciamento possam ser executadas. Descreve uma interação entre dois ou mais serviços para atingir um objetivo global. Diferentemente da orquestração, ela não possui a figura do controlador central. Todos os participantes estão no mesmo nível hierárquico e são tratados igualmente, cada um com suas devidas responsabilidades dentro da coreografia. Fazendo outra analogia, suponha um grupo de dança que está apresentando uma coreografia musical. Cada um dos integrantes deve saber o que fazer e quando, para que o objetivo global seja atingido por todos, porém, não é relevante para a coreografia saber como cada um dos participantes faz para executar sua parte no processo. Nesse tipo de coordenação existe também troca de mensagens entre participantes. Entretanto, a principal diferença entre coordenação e orquestração está no fato de que não há

[26] *Workflow* - Define um processo que envolve diferentes tarefas e participantes. Nesse processo, as tarefas são encadeadas e coordenadas de forma que documentos, informações ou trabalhos possam ser trocados entre participantes de acordo com algumas regras.

uma especificação segundo um único participante, e sim descrição global do processo que representa um contrato entre todos os participantes (NOVAIS, 2007).

A principal linguagem com apoio a coreografia é a WS-CDL[27] – *Web Service Choreography Description Language*. WS-CDL é uma linguagem de especificação de contratos entre os participantes. Apesar de recente, ainda como candidata recomendada pela W3C (W3C, 2009), existe uma forte tendência dela se tornar um padrão para coreografia.

Suponha um cenário em que há uma compra e venda de produto (NOVAIS, 2007). Na Figura 3.5 é apresentada uma modelagem simplificada desse problema do ponto de vista da orquestração, em que há três participantes: o comprador, o vendedor e o agente que representa o cartão de crédito. Nessa figura, pode ser observado que o problema foi separado segundo a perspectiva de cada participante (duas linhas verticais tracejadas) e, para cada um, pode ser gerado seu *workflow* correspondente, que pode ser escrito em BPEL. Cada participante atua como controlador central da sua parte e, de tempos em tempos, são feitas trocas de mensagens entre os participantes (representadas pelas setas que apontam para ou saem das linhas verticais tracejadas) para requisições de informações.

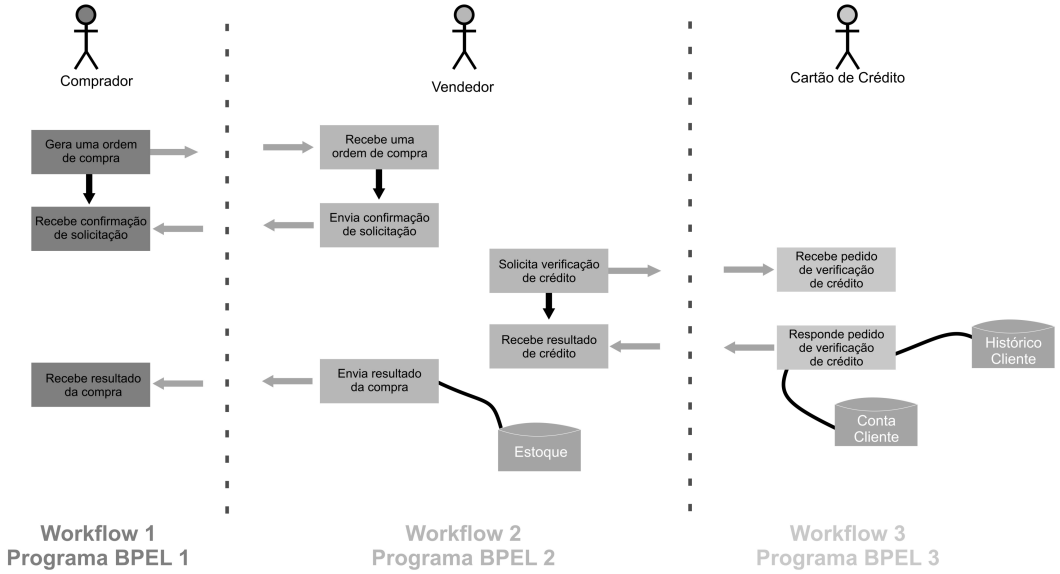


Figura 3.5 - Orquestração: Compra de Produto. Fonte: Novais (2007).

[27] WS-CDL - http://www.w3.org/TR/2004/WD-ws-cdl

É importante notar ainda que cada *workflow* pode conter etapas, omitidas na figura, que são pertinentes apenas a ele, não sendo necessário que outros participantes tenham conhecimento como, por exemplo, acesso à base de dados.

Na Figura 3.6 é apresentada a modelagem do mesmo exemplo de acordo com a coreografia. Nesse caso, o objetivo da compra é analisado do ponto de vista global, diferentemente do anterior que analisava de acordo com a perspectiva de cada participante. Porém, não há um programa executável, e sim uma especificação de como a venda do produto deve ocorrer como um todo. Essa especificação, que pode ser negociada entre analistas de negócios das empresas (os participantes) que atuam no processo, funciona como um contrato entre participantes. Os participantes podem desempenhar diferentes papéis dentro da coreografia. A partir dessa especificação, cada participante deve implementar sua parte do processo. Nesse momento há uma aproximação da orquestração, porém, não é relevante para a coreografia como cada um constrói sua parte, seja em Java, BPEL ou outra linguagem.

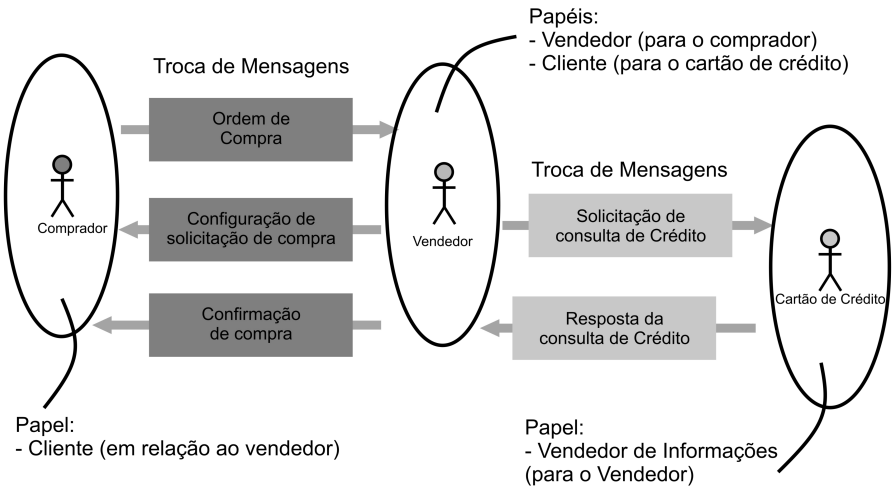


Figura 3.6 - Coreografia: Compra de Produto. Fonte: Novais (2007).

A reutilização na coreografia é mais visível que na orquestração. Suponha que um novo participante queira fazer parte dos dois processos descritos anteriormente. Na orquestração, ele terá que descobrir como é feita a troca de mensagens e desenvolver seu *workflow* correspondente. Já na coreografia, basta que ele tenha em mãos o contrato gerado. A partir dessa especificação ele já sabe o papel de cada participante e como são feitas as trocas

de mensagens entre os mesmos. Dessa maneira, ele conhece a forma de comunicação e pode, mais facilmente, integrar-se ao processo. Na Tabela 3.3 é exibida uma breve comparação entre orquestração e coreografia.

Tabela 3.3 - Orquestração X coreografia. Fonte: Novais (2007).

	Orquestração	Coreografia
Produto Alvo	Sistema executável para cada participante	Uma especificação (contrato) entre participantes
Principal Linguagem	BPEL4WS (BPEL)	WS-CDL
Reutilização	Dá-se ao nível de cada *workflow*	Ao nível do contrato
Modelo de Negócio	Segundo um controlador central	Todos os participantes possuem a mesma hierarquia
Suporte a *Workflow*	Sim	Sim
Tipo de Comunicação	Externamente: ponto-a-ponto Internamente: depende da implementação do *workflow*	Externamente: ponto-a-ponto Internamente: não especifica
Composição Recursiva	Sim	Sim

3.8. Desafios para o desenvolvimento de aplicações orientadas a serviços

Atualmente, uma quantidade cada vez maior de organizações tem transformado seus sistemas em aplicações orientadas a serviço, buscando aumentar a agilidade e a produtividade dentro da organização. Com esse aumento gradual do uso de SOA desafios têm surgido no desenvolvimento desse tipo de aplicação (LEE *et al*., 2006). Alguns desses desafios são apontados a seguir:

- Dificuldade em obter os requisitos de usuários já que eles, normalmente, não provêm de uma única fonte. Os requisitos podem vir de vários *stakeholders* que podem estar em locais diferentes;
- Dificuldade em integrar diferentes serviços uma vez que nem todos os serviços foram implementados utilizando as mesmas tecnologias. A hospedagem dos serviços em diferentes plataformas de tecnologia também contribui para as dificuldades de integração;
- Dificuldade na utilização dos serviços devido aos diferentes tipos de serviços oferecidos. Alguns serviços suportam apenas interações assíncronas, outros podem suportar interações síncronas ou assíncronas;

- Dificuldade na comunicação dos serviços devido às diferentes *interfaces*, por exemplo: troca de mensagens orientadas a documentos ou troca de mensagens orientadas a parâmetro;
- Serviços diferentes oferecem graus de acoplamento diferentes. Serviços baseados em documentos são mais fracamente acoplados do que serviços baseados em parâmetros, por exemplo;
- Dificuldade em conduzir testes em aplicações SOA já que isto requer um esforço bem coordenado de todos os provedores de serviços para assegurar que todos os serviços estão disponíveis.

A adoção de uma metodologia padrão para a implementação de aplicações orientadas a serviços diminuiria essas dificuldades e permitiria maior integração entre as aplicações orientada a serviços existentes.

3.9. SOA e outras abordagens

A arquitetura orientada a serviços tem sido utilizada em conjunto com diferentes abordagens. As subseções a seguir descrevem algumas dessas abordagens. No entanto, é importante salientar que tais abordagens não foram diretamente utilizadas neste trabalho e serão citadas apenas para fins de conhecimento.

3.9.1. SOA e componentes

Apesar de serviços serem parecidos com componentes, blocos de construção que coletivamente representam ambientes de aplicações, eles possuem um conjunto de características únicas. Um serviço é autônomo em relação a outros, isso significa que cada serviço é responsável pelo seu próprio domínio, limitando seu escopo a uma função específica de negócios. Essa abordagem, segundo Erl (2005), resulta na criação de unidades de funcionalidades de negócio isoladas e fracamente ligadas por um *framework*. Além disso, a lógica de negócios que os serviços encapsulam é independente de plataforma ou tecnologia.

Componentes e serviços diferem no tipo de comunicação, de acoplamento, de interface, de invocação, de *request brokering* [28] e ainda diferem na forma como abordam flexibilidade e reusabilidade durante a fase de desenvolvimento (PAPAZOGLOU *et al*., 2006). Enquanto serviços estão sujeitos a contínuas manutenções, aumento de escopo, desempenho e oferecem a possibilidade de seleção dinâmica de serviços, o uso de componentes instalados não permite o mesmo tipo de reúso e dinamicidade.

3.9.2. SOA e Linha de Produtos

Capilla e Topaloglu (2005) propõem um modelo para a composição e evolução de aplicações orientadas a serviços usando linha de produtos (LP) com informações de variabilidades específicas. A proposta é uma modificação na abordagem tradicional de linha de produto para suportar serviços compostos. A fase de engenharia de domínio da LP permanece igual apenas incluindo informações específicas para apoiar composição de serviço na etapa projeto de domínio que gera a arquitetura. Na fase de engenharia de aplicação, aplicações orientadas a serviços são desenvolvidas utilizando serviços já construídos na fase de engenharia de domínio.

Sistemas orientados a serviços podem ser formados por serviços únicos ou serviços compostos que oferecem maior funcionalidade. Diferentes linguagens podem especificar a composição de serviços, como por exemplo, WSFL (*Web Services Flow Language*) e BPEL. Tais linguagens descrevem a modelagem do fluxo entre as funções que são necessárias para oferecer serviços compostos como um único software. Nesse sentido, essas linguagens organizam serviços compostos de forma coordenada. Para Havey (2006), esse estilo de composição reduz a complexidade da evolução do software e o sucesso de sua evolução depende da habilidade de separar componentes de softwares estáveis, ou seja, separar aqueles que sofrem pouca mudança dos que estão sujeitos a uma maior quantidade de mudança. Ainda no contexto de sistemas orientados a serviços, Havey (2006) propõe alguns pontos de variação para modelar serviços Web e diz que essa abordagem foi usada no desenvolvimento de diversos sistemas Web.

---

[28] *Request brokering* - É um componente de software ORB (*Object Request Broker*) cuja função é facilitar a comunicação entre objetos.

van Gurp e Savolainen (2006) proveem um método para incorporar variabilidade em linhas de produto de software. Os autores apresentam uma técnica e um processo para serem usados na implementação de características variantes em arquitetura de conjuntos de serviços Web. Nas técnicas descritas por van Gurp e Savolainen (2006) são aplicadas as variabilidades e apesar de terem sido projetadas para ambientes de grades de serviço inicialmente, a maioria delas pode ser estendida para a implementação de LPs com SOA.

Chen *et al.* (2005) apresentam uma abordagem que procura encontrar uma forma de relacionar características e serviços. Também pode ser usada no desenvolvimento de uma linha de produtos com SOA, uma vez que a definição das características implementadas por certo serviço é fundamental para um projeto da LP.

Ye *et al*. (2007) fizeram uma análise das variabilidades de processos de negócios enquanto Gomaa e Saleh (2005) pensaram em variabilidades como um serviço. Quando a análise de características ocorre, as características variantes em termos de processo são separadas daquelas variantes em termos de serviço. Essas duas abordagens podem ser combinadas para desenvolver LPs com SOA, porém, nenhum dos autores se preocupou com a forma de implementação das variabilidades como serviços. Não há na literatura ainda uma técnica consolidada, que envolva a análise de muitos fatores e de comprovada eficiência para a implementação de uma linha de produtos com SOA.

Com base nesse contexto e considerando que uma LP é um conjunto de artefatos pré-definidos que podem ser compostos de forma a gerar um produto e que esses artefatos podem fazer parte do núcleo da linha ou ser uma variabilidade dela, é possível propor uma forma de implementação onde cada uma dessas variabilidades seja considerada como um serviço e utilizando uma linguagem de composição como BPEL (que também especifica processos de negócio), tais serviços sejam unidos de forma a gerar um serviço composto que seria um produto da linha de produtos.

3.9.3. Análise de características para Reengenharia Orientada a Serviços

Sistemas legados contêm grande quantidade de dados críticos e de lógica de negócio de uma empresa, não podendo ser facilmente substituídos. Chen *et al*. (2005) consideram um desafio transformar completamente ou parcialmente a funcionalidade de um sistema legado em

serviços. A reengenharia de sistemas desempenha, portanto, um papel importante na migração para ambientes orientados a serviços e é aplicável a sistemas legados que têm algumas das seguintes características (ZHANG e YANG, 2004):

- O sistema legado precisa ser migrado para um ambiente distribuído e ser publicado como um serviço Web;
- Possui funcionalidade reusável e confiável com uma valiosa lógica de negócio embutida, disponível no próprio sistema legado;
- A funcionalidade deve ser útil para ser publicada como um serviço independente e que atenda a determinada exigência;
- Os componentes do sistema alvo podem ser usados em diferentes plataformas;
- Alguns componentes legados devem ser substituídos aos poucos de forma que não afete o cliente do serviço;
- O sistema alvo deve operar sob a Internet onde não pode ser garantida confiança e velocidade.

Diante da dificuldade de transformar sistemas legados em sistemas orientados a serviços, Chen *et al*. (2005) propõem uma técnica de reengenharia de aplicações que utiliza análise de características para fazer essa transformação. Analisar características é um processo que inclui a identificação de características do sistema, a construção do modelo de características para organizar as características identificadas de forma consistente, o rastreio dos relacionamentos entre as características e a implementação de sistema legado por meio de técnicas de localização de características (CHEN *et al*., 2005). Os autores escolheram a análise de características para identificação de serviços, pois serviços e características possuem aspectos em comum.

De acordo com Lee e Muthig (2006) uma característica é uma qualidade importante e especial, uma particularidade do sistema. A definição de característica usada por Chen *et al*. (2005) em sua técnica diz que essa é um conjunto de funcionalidades do sistema, coerentes e identificáveis que são visíveis ao usuário através de uma interface. Os autores definem ainda

característica como uma funcionalidade essencial do sistema, mas salientam que nem toda funcionalidade é uma característica.

Para determinar se uma funcionalidade é uma característica, Chen *et al*. (2005) sugerem os seguintes critérios:

- Se a funcionalidade pode ser usada para especificar uma das capacidades de um sistema;
- Se uma funcionalidade pode ser visível ou identificável na perspectiva do usuário final;
- Se uma funcionalidade é uma instância de uma característica do domínio;
- A funcionalidade deverá estar em um alto nível de abstração sem preocupação com detalhes computacionais;
- Se a funcionalidade, baseada na perspectiva do usuário, é identificada como uma característica e é indivisível, então ela é chamada de característica atômica;
- Duas ou mais características atômicas podem consistir em uma característica composta baseado nas regras de negócio.

Se uma funcionalidade atende a esses critérios, então pode ser considerada como uma característica que pode coexistir com outras características em um modelo por meio de seus relacionamentos. Um modelo de características é resultado de um processo combinado de identificação e classificação das características, organização dessas como um conjunto de modelos coerentes e validação desses modelos (CHEN *et al*., 2005).

Para construir uma SOA, Chen *et al*. (2005) usam o paradigma de Projeto Orientado a Serviço (*Service-Oriented Design* – SOD) que tenta explorar um processo híbrido. Esse processo analisa o sistema de forma *top-down* envolvendo análise do domínio, processos de negócio e outros métodos avançados e de forma *bottom-u*p a fim de investigar os componentes e estratégias de refatoração.

O primeiro passo na abordagem de Chen *et al.* (2005) é escolher características intermediárias durante o processo de reengenharia orientada a serviços. Para essa escolha são observados os seguintes itens:

- Granularidade alta: a funcionalidade da característica identificada pelo usuário possui granularidade alta e tem uma afiliação natural com o serviço. Ambos, característica e serviço, são orientados a negócio. Detalhes computacionais podem ser negligenciados;

- Rastreabilidade: como uma regra no ciclo de vida do software, características podem ser mapeadas para outros artefatos através de diferentes níveis de abstração. A identificação de serviço pode ser benéfica nesse ponto;

- Semântica: as dificuldades nas atividades de modelagem semântica necessitam ser superadas, a fim de combinar requisições do serviço cliente e respostas do provedor do serviço;

- Interação de características: tal interação é proposta para solucionar a coordenação de características. Serviços Web devem ser integrados em alguns modelos e interações entre si para que a sua composição alcance o resultado esperado no nível de aplicação.

Por meio da análise de característica, operações de serviços (funcionalidades) de sistemas legados podem ser identificadas e refatoradas como implementações orientadas a serviço.

Depois de identificados os serviços, esses são empacotados e publicados. Para Chen *et al*. (2005), empacotar significa enquadrar os serviços extraídos e encapsulados de um código-fonte legado. É um passo necessário para transformar serviços legados candidatos em serviços funcionais de software. Supõe-se aqui que um serviço alvo será provido como um serviço Web.

As operações de serviço identificadas são implementadas pela definição de um método público em um diagrama de classe como uma interface de serviço exposto para o cliente do serviço. Em seguida, os desenvolvedores de serviços Web devem definir as propriedades e métodos que especificam as operações dos serviços. As propriedades são nome do serviço, descrição e *namespace*. As propriedades dos métodos são *MessageName, CacheDuration, EnableSession, TransactionOption, BufferResponse* e *Description*. Uma definição rigorosa da

funcionalidade do serviço pode ser provida automaticamente pelo WSDL. Depois o serviço pode ser registrado no UDDI e então ser dinamicamente encontrado e usado através da Internet e acessado com o protocolo SOAP.

Uma grande parte da abordagem de Chen *et al*. (2005) é apoiada por ferramentas. Os autores utilizaram a ferramenta *Captain Feature*[29] para construir o modelo de características e a ferramenta FEAT[30] para localizar características. O resultado da análise de características é armazenado como um perfil onde outra ferramenta usará para gerar os serviços. O protótipo da ferramenta WSW[31] (*Web Services Wrapper*) que apoia a implementação de serviços Web é usada para gerar os serviços.

3.10. Modelo de Referência para SOA 1.0

Desenvolvido pelo Comitê de Especificação da organização Oasis, o modelo de referência para Arquitetura Orientada a Serviço é considerado um *framework* abstrato, que permite o entendimento das entidades significativas e os relacionamentos entre essas entidades em um ambiente orientado a serviço, além de permitir o desenvolvimento de padrões consistentes ou especificações que suportem esse ambiente. É baseado nos conceitos unificados de SOA e pode ser usado por arquitetos no desenvolvimento específico de arquiteturas orientadas a serviço ou em treinamento e exposição de SOA (OASIS, 2006).

Esse modelo não está ligado de forma direta a nenhum padrão, tecnologia ou outro detalhe de implementação. Ele procura oferecer uma semântica comum que pode ser usada de forma não ambígua entre implementações diferentes. A Figura 3.7 ilustra o relacionamento entre o Modelo de Referência criado pela Oasis e as arquiteturas e tecnologias particulares (OASIS, 2006).

O modelo idealizado pela empresa Oasis tem enfoque no campo de arquitetura de software, enquanto a orientação a serviço pode ser um conceito popular encontrado em ampla variedade de aplicações. Os conceitos e relacionamentos descritos podem ser aplicados a

[29] *Captain Feature* - https://sourceforge.net/projects/captainfeature
[30] FEAT - http://www.cs.mcgill.ca/~swevo/feat
[31] WSW - http://www.actional.com/resources/whitepapers/SOA-Worst-Practices-Vol-I/Web-Services-Wrapper

outros ambientes que utilizam serviços, mas a sua especificação não considera o uso fora do domínio de software.

Um modelo de referência define a essência da arquitetura orientada a serviço e propõe um vocabulário e um entendimento comum de SOA. Esse modelo oferece uma referência normativa que permanece relevante para SOA como um modelo abstrato e poderoso, independente das várias inevitáveis evoluções tecnológicas que venham a influenciar a realização dessa arquitetura (OASIS, 2006).

A Figura 3.7 mostra a relação de um modelo de referência para SOA com outras arquiteturas de sistemas distribuídos. Os conceitos e relacionamentos definidos pelo modelo de referência são base para descrição de arquiteturas de referências e padrões que definem as categorias mais específicas de projetos SOA. As arquiteturas concretas vêm de uma combinação de arquiteturas de referência, padrões de arquitetura e requisitos adicionais, incluindo aqueles impostos pelos ambientes tecnológicos (OASIS, 2006).

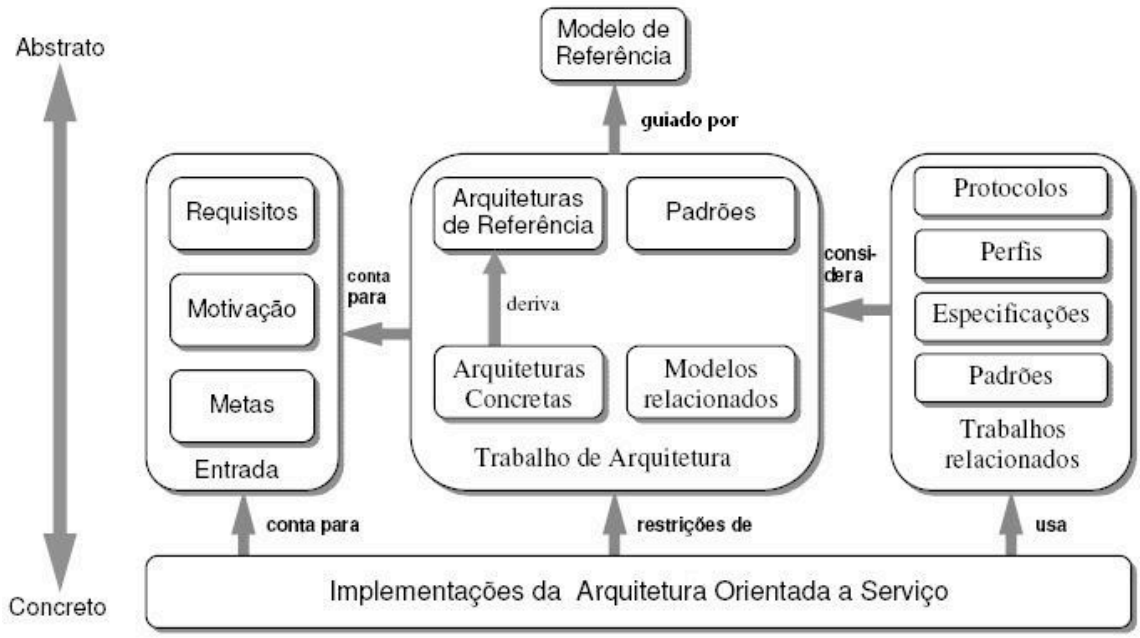


Figura 3.7 - Como o Modelo de Referência relaciona-se com os outros trabalhos. Fonte: Oasis (2006).

A arquitetura deve considerar as metas, motivações e requisitos que definem os problemas reais que estão sendo estudados, enquanto que as arquiteturas de referências podem

formar as bases de classes de soluções e as arquiteturas concretas irão definir abordagens de soluções específicas (OASIS, 2006).

Arquiteturas são geralmente desenvolvidas no contexto de um ambiente pré-definido, usando protocolos, perfis, especificações e padrões que sejam pertinentes. A arquitetura orientada a serviços combinam todos esses elementos desde os mais genéricos aos mais específicos, de forma que atendam aos requisitos necessários para construção e uso de uma aplicação SOA. Essa combinação é possível porque SOA provê serviços independentemente de plataformas ou tecnologias.

3.11. Considerações Finais

Neste capítulo foram apresentados os conceitos necessários para o entendimento da arquitetura orientada a serviços, bem como as tecnologias utilizadas para prover serviços eletrônicos. Destacou-se que serviços Web (*Web Services*) tem se tornado o padrão pela W3C para construção de uma SOA.

SOA é uma técnica em ascensão, pois provê o uso de seus serviços independente de tecnologia. Por utilizar padrões abertos, essa arquitetura permite interoperabilidade entre aplicações. Essa característica permite que aplicações desenvolvidas em diferentes plataformas e por linguagens de programação possam estabelecer comunicação e troca de mensagens. Essa flexibilidade permite que muitas organizações adaptem seus sistemas legados para atender melhor às necessidades atuais de seus clientes.

O GAwCRe, em sua versão original, não atende às características de SOA. Dentre as características apresentadas neste capítulo, na Seção 3.4, algumas poderão ser identificadas no gerador após manutenção realizada no mesmo para apoiar serviços Web, tais como: reuso caixa-preta, dinamismo, adaptabilidade e sincronia.

No Capítulo 4 é apresentada a manutenção realizada no gerador de aplicações GAwCRe, com o objetivo de adaptá-lo para apoiar o uso de serviços Web providos por SOA.

Capítulo 4

Manutenções realizadas no GAwCRe

4.1. Considerações Iniciais

As atualizações são sempre necessárias para que um software seja adaptado e atenda a novas funcionalidades. Para que o GAwCRe (PAZIN, 2004) possa apoiar o uso de serviços Web providos por SOA, algumas modificações foram identificadas a partir da realização de estudos de casos com o uso do gerador. Vários problemas foram encontrados no seu código e na sua interface gráfica, como por exemplo, trechos de código mal construídos, camadas entrelaçadas, a não apresentação de quais padrões poderiam ser selecionados na tela da interface, entre outras. A solução adotada foi realizar uma manutenção evolutiva e corretiva para proporcionar ao gerador GAwCRe mais funcionalidade e flexibilidade.

Este capítulo relata os problemas encontrados bem como todas as modificações efetuadas, denominadas de manutenções corretiva e evolutiva. Na Seção 4.2 são comentados os problemas encontrados no GAwCRe que conduziram a manutenção evolutiva nesse gerador. Na Seção 4.3 são detalhadas as manutenções evolutiva e corretiva realizadas na arquitetura do gerador. Na Seção 4.4 são descritos alguns problemas de usabilidade que foram observados a partir do uso do gerador e da aplicação gerada, indicando a necessidade de modificações na sua interface. Na Seção 4.5 a estrutura do projeto de implementação é ilustrada e, na Seção 4.6 são comentadas as considerações finais.

4.2. Problemas encontrados no GAwCRe

O GAwCRe foi utilizado para a geração de sistemas pertencentes ao domínio de clínicas médicas, que é conexo ao domínio para o qual foi desenvolvido, para possibilitar o entendimento e familiarização com esse tipo de software. Alguns testes funcionais foram realizados inicialmente no GAwCRe com o objetivo de inserir um serviço Web que gerasse sistemas pertencentes ao domínio de clínicas médicas.

Manutenções já haviam sido realizadas anteriormente por Rizzo (2005), Freitas (2006), Ferreira (2008) e Borges (2008), mas o gerador ainda apresentava problemas no código-fonte, o que inviabilizava a realização de manutenção evolutiva no GAwCRe para acoplar ao gerador e as aplicações geradas, um código capaz de utilizar e prover serviços Web disponibilizados na Internet.

Ao inserir um serviço Web no GAwCRe de validação de dados, por exemplo, alguns dos problemas, que já haviam sido identificados por Borges (2008), foram ratificados e outros foram identificados. Dentre os que foram identificados pela primeira vez é possível citar (BORGES, 2008): a) a falta de instruções para a instalação e operação do gerador; b) não havia um padrão de codificação para o código-fonte do gerador; c) falta de flexibilidade; d) erros de lógica; e) trechos de código duplicado e métodos extensos o que indicara a necessidade de refatoração; f) alto acoplamento e baixa coesão de dados; g) o não uso de padrões de projeto, que possibilita reuso e melhor documentação (GAMMA, 1995).

4.3. Manutenção Corretiva e Evolutiva no GAwCRe

O primeiro passo para a adaptação do GAwCRe para que ele pudesse utilizar serviços Web providos por SOA foi a análise do código fonte existente. Assim, a correção dos problemas citados na Seção 4.2 foi iniciada.

A estratégia de manutenção utilizada pode ser aplicada para outros geradores de aplicação também desenvolvidos com linguagem de padrões e, além dos serviços web providos por SOA, é possível também modificar o banco de dados e a linguagem de programação.

Após analisar o gerador, a solução adotada para dar mais flexibilidade ao GAwCRe foi a criação de um modelo para metadados (YODER e JOHNSON, 2002), independente de linguagem de programação; que permite a utilização de novas linguagens de padrões e facilita

manutenções futuras. Para a construção do modelo de metadados foi utilizado um sistema de *templates* chamado FreeMarker[32]. Detalhes sobre o FreeMarker e sobre a utilização do mesmo neste trabalho serão fornecidos na seção a seguir.

4.3.1. Utilizando o FreeMarker na definição dos metadados

O FreeMarker é um motor (*engine*) que gera um arquivo de saída baseado em *templates* pré-definidos, como o ilustrado na Figura 4.1. Esse arquivo de saída pode ser HTML, XML, texto simples, classes Java, dentre outros.

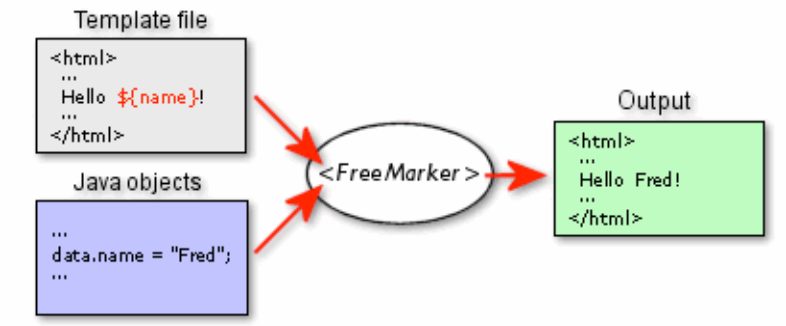


Figura 4.1 - Informações de entrada e saída do FreeMarker. Fonte: FreeMarker (2009).

O FreeMarker pode ser utilizado em qualquer aplicação que precise gerar uma saída de texto variável. As páginas textuais que apresentam os dados definidos são geradas a partir de *templates* (FREEMARKER, 2009).

Os *templates* são criados para receber informações das entidades (representações dos objetos que existem no mundo real e que são persistidos no banco de dados) existentes na linguagem de padrões SiGCli (médico, paciente, etc.). O usuário desenvolvedor do gerador informa os nomes e as propriedades (atributos) dessas entidades. A partir dessas informações, ocorre a execução do *template* e as classes Java, referentes a cada entidade, são então geradas.

O uso do FreeMarker permite reúso, pois a definição das informações comuns a todas as entidades são feitas na construção do *template* e replicadas ao rodá-lo para as outras entidades. Há também diminuição do tempo gasto na implementação do sistema, pode-se evitar possíveis inserções de erros na fase de codificação e há padronização em todo o código-fonte.

[32] http://freemarker.org/

Na Figura 4.2 é apresentada a listagem que ilustra o *template* de criação da entidade Paciente, uma das que existe na linguagem de padrões SiGCLi para o domínio que foi desenvolvida e que também atende ao domínio de clínicas médicas. A classe *Paciente* será criada com os parâmetros passados, as respectivas propriedades (nome, sexo, data de nascimento, endereço, etc.), os *gets* e *sets*.

```
//Template para criação da entidade Paciente
package br.ufscar.dc.gdms.entities;

@ModelMetadataProvider(context = "SiGCli")
public class Paciente {

      @EntityMetadataProvider a
    public static EntityMetadata getEntity() {
        String packageName = "br.ufscar.gdms";
        String entityName = "Paciente";

        EntityMetadata myClass = new EntityMetadata(entityName, packageName);

    b   PropertyMetadata nome = new RequiredPropertyMetadata(new
         StringPropertyMetadata("nome"));
        myClass.addProperty(nome);
        PropertyMetadata sexo = new RequiredPropertyMetadata(new
         EnumerationReferencePropertyMetadata("sexo", "br.ufscar.gdms.Sexo")); …
        myClass.addProperty(sexo);
        return myClass; } }
```

Exemplos de propriedades da classe Paciente

Figura 4.2 - *Template* para criação da entidade Paciente.

Qualquer entidade que for criada tem essa estrutura sendo necessário modificar apenas o nome da entidade em questão e suas propriedades. É possível observar que a propriedade *sexo* pertence ao tipo de dado *enum* (*enumeration*), uma constante representada pelos valores MASCULINO e FEMININO. As classes *EntityMetadata* e *PropertyMetadata*, modificadas neste projeto e representadas no trecho de código abaixo por a e b, respectivamente, são as classes *templates* onde estão as definições comuns a todas as entidades e propriedades das entidades (veja a implementação dessas classes no Apêndice A).

A seção seguinte apresenta a arquitetura construída para o GAwCRe durante o processo de manutenção realizado.

4.3.2. Arquitetura do GAwCRe

Em sua versão original, para instanciar uma nova aplicação, o GAwCRe inicialmente fazia a leitura das definições existentes em um arquivo XML, em que estava definida a linguagem de padrões SiGCli. Em seguida, juntamente com as informações passadas pelo usuário na escolha

dos padrões, o gerador iniciava a geração dos artefatos da aplicação utilizando módulos específicos para a criação de cada artefato. Esses módulos eram carregados durante a execução da aplicação gerada. Os artefatos gerados pelo GAwCRe são: os *scripts* SQL, as classes Java e as páginas JSPs. A Figura 4.3 exibe a arquitetura original do gerador GAwCRe.

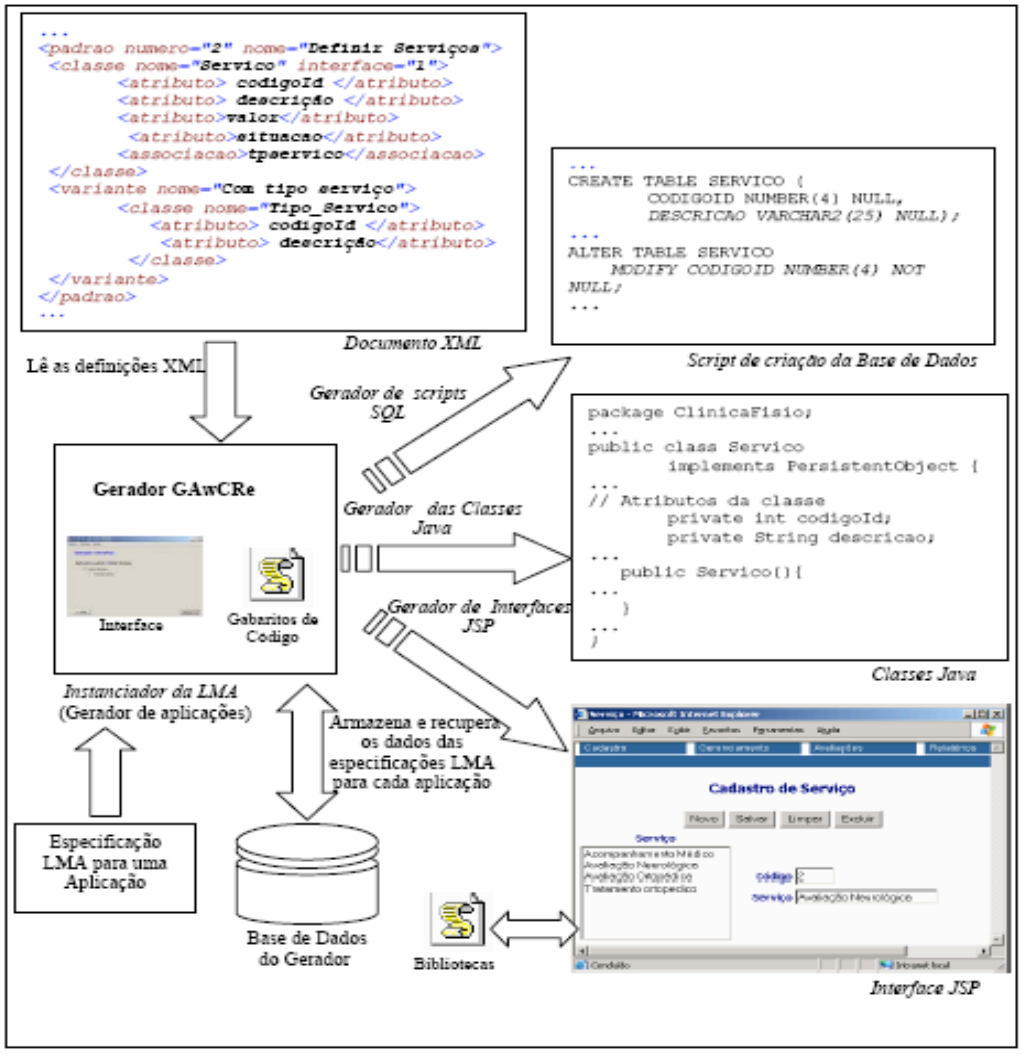


Figura 4.3 – Arquitetura original do GAwCRe. Fonte: Pazin (2004)

Diante dos problemas encontrados no GAwCRE, conforme descritos na Seção 4.2, foi realizada uma manutenção evolutiva e corretiva nesse gerador para torná-lo mais flexível e com suporte ao serviços web. Uma nova arquitetura foi construída para esse gerador. A Figura 4.4 exibe a arquitetura do GAwCRe após as manutenções realizadas.

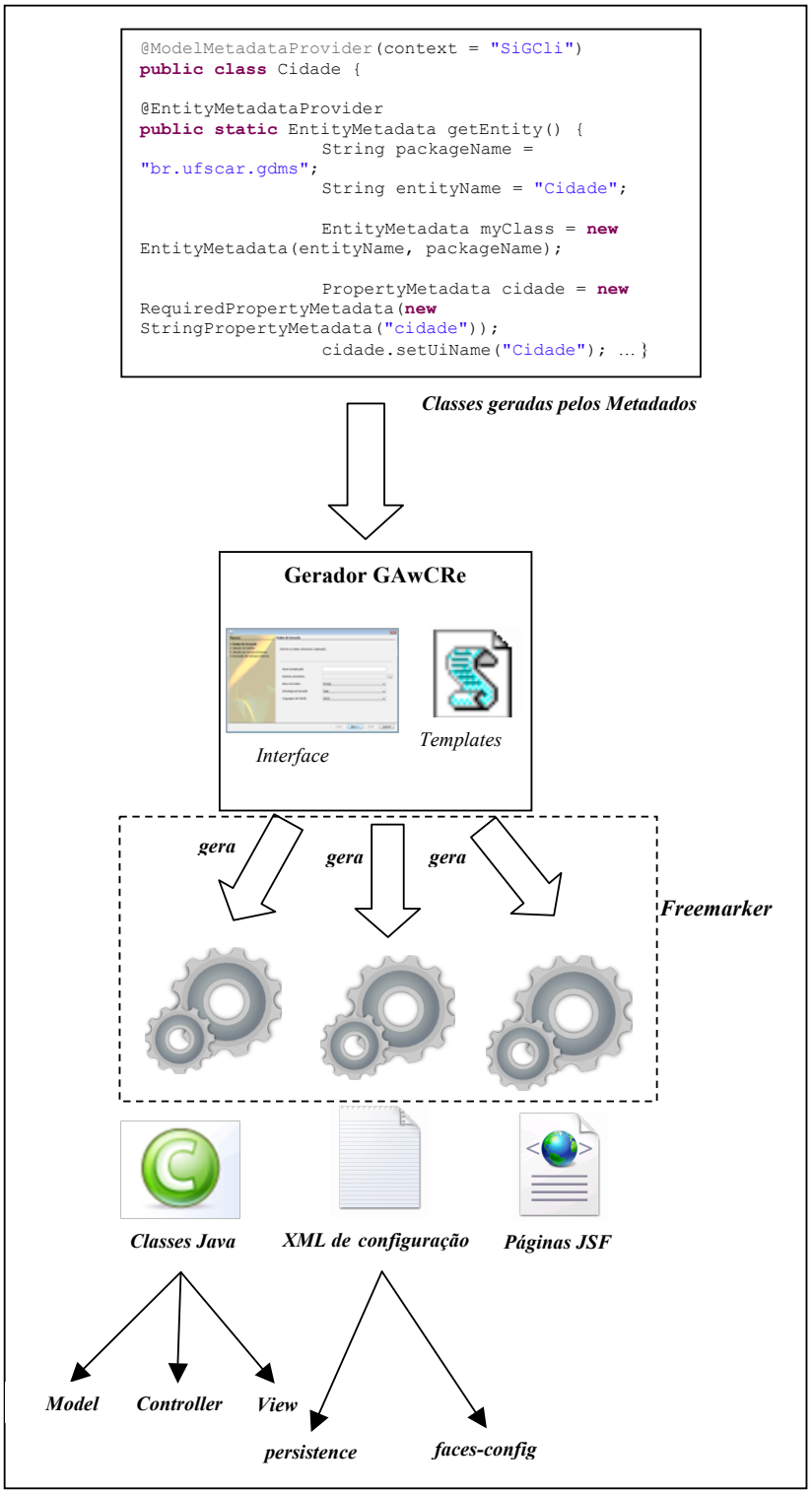


Figura 4.4 - Nova Arquitetura do GAwCRe.

A estrutura de metadados construída durante a manutenção do GAwCre utiliza *templates* do Freemarker com estruturas pré-existentes de código, usadas para definir os produtos que se deseja gerar por intermédio de um gerador. Eles possuem partes fixas,

presentes em todos os artefatos gerados, e partes variáveis que possibilitam a geração de diferentes artefatos usando o mesmo gabarito.

Os metadados contendo as informações da SiGCli que servem como parâmetros para a execução dos *templates* e os padrões selecionados pelo usuário na interface do GAwCRe são passados ao gerador que utiliza o Freemaker[33] para gerar os artefatos da aplicação especificada pelo usuário. Esses artefatos são as classes Java, os XMLs de configuração e as páginas JSF da aplicação gerada. As classes Java geradas são classes de modelo (*model*) que definem as estruturas de dados das entidades, classes de negócio (*controller*) responsável por realizar persistência e consulta dos dados e a classe de visão (*view)* que realiza a interface entre as páginas JSF e a camada de negócio. O arquivo XML de configuração *persistence* possui as especificações do banco de dados utilizado na aplicação e o arquivo *faces-config* define o fluxo de navegação da aplicação gerada. Na camada de visualização há duas páginas, uma responsável pela listagem dos dados cadastrados e outra responsável pelo cadastro de novos dados.

Na arquitetura original do GAwCRe não havia separação entre a listagem dos dados cadastrados e o cadastro de novos dados na interface da aplicação gerada, além disso, a camada de modelo e negócios estavam entrelaçadas, bem como a camada de negócio estava entrelaçada com a camada de visualização. Para separar essas camadas e prover a independência dos tipos de dados foram criados metadados que mapeiam um tipo abstrato para um tipo específico em determinada plataforma. Seguindo a estrutura de metadados, entidades (classes que armazenam as propriedades de um objeto persistente) e enumerações (tipos compostos por constantes que representam valores possíveis de um determinado conjunto) também são representadas por metadados.

A modificação realizada na estrutura de classes do GAwCRe constou da criação de classes para armazenar metadados de entidades (*EntityMetadata*). As entidades representam os objetos persistidos na aplicação e são compostas por propriedades que possuem um identificador e um tipo de dado (ex.: String, Decimal). Os objetos persistidos, na linguagem de padrões SiGCli, são por exemplo: médico, paciente. Os tipos de dados que compõem as entidades possuem uma combinação de valores e de operações que uma variável pode

[33] http://freemarker.org/

executar e pode variar conforme a linguagem de implementação utilizada. Esses tipos são definidos pela implementação da classe *PropertyMetadata* utilizada, e podem ser um tipo primitivo como String ou Decimal ou uma referência a outra entidade ou enumeração (tipos compostos por constantes). A classe *PropertyMetadata* é uma interface que estende a classe *Metadata* e é implementada pela classe *PropertyMetadataDecorator* uma classe abstrata que serve de base para a implementação dos tipos de dados. A Figura 4.5 exibe o trecho de código correspondente à classe *PropertyMetadata* e os metadados das propriedades que ela define e que serão referências para as entidades e enumerações.

```
// Classe que define metadados das propriedades.
public interface PropertyMetadata extends Metadata {

	public boolean isRequired();
	public String getName();
	public Class<? extends TypeMetadata> getType();
	public TypeFamily getTypeFamily();
	public boolean isCollection();
	public boolean isPrimitive();
	public boolean isBoolean();
	public boolean isEntityReference();
	public boolean isEnumerationReference();
	public boolean isReference();
	public void setUiName(String uiName);
	public String getUiName();
	public String getUiDescription();…  }
```

Figura 4.5 - Trecho de código correspondente à classe *PropertyMetadata*.

A classe *EntityMetadata* estende a classe de metadados de objeto (*ObjectMetadata*) que armazena informações como nome da aplicação a ser gerada e que serão usados pela classe *EntityMetadata* para implementar as configurações das entidades. A classe *EntityMetadata* possui uma lista das propriedades que compõe a entidade, validadores de dados e restrições de unicidade (define um conjunto de propriedades que não podem ter valores iguais) para uma ou mais propriedades da entidade. Essa lista será utilizada pela classe *PropertyMetadata*, para implementar as propriedades, validadores e restrições de cada entidade. O trecho de código que contém partes dessa lista é o exibido na Figura 4.6.

```
// Classe que armazena metadados de entidades.
public class EntityMetadata extends ObjectMetadata {

	private List<PropertyMetadata> properties;
	private List<EntityValidator> validators;
	private List<UniqueConstraintMetadata> uniqueConstraints; ...}
```

Figura 4.6 - Trecho de código correspondente à classe *EntityMetadata*.

Algumas entidades possuem propriedades de preenchimento obrigatório obtidas através da classe *RequiredPropertyMetadata,* que é implementada utilizando o padrão de projeto *decorator*. Para a linguagem de padrões SiGCli, entidades como paciente possui, por exemplo, o atributo *nome* do tipo primitivo *String* e *profissão* do tipo constante *EntityReference* (referência para outra entidade, nesse caso a entidade profissão) como propriedades obrigatórias. A Figura 4.7 exibe o trecho de código com a classe *RequiredPropertyMetadata.*

```
// Classe que encapsula uma propriedade da entidade para torná-la requerida
(decorator)
public class RequiredPropertyMetadata implements PropertyMetadata {

        private PropertyMetadata property;
        ...
public boolean isEntityReference() {
                return property.isEntityReference();
        } ...
public boolean isPrimitive() {
                return property.isPrimitive();
        }...
}
```

Define a existência de propriedades referências para outras entidades que são de preenchimento obrigatório para essas.

Define a existência de tipos primitivos que são de preenchimento obrigatório para determinada entidade.

Figura 4.7 – Trecho de código correspondente à classe *RequiredPropertyMetadata*.

Além da definição de entidades é possível definir metadados para enumerações (*EnumerationMetadata*) e seus atributos (*EnumerationConstantMetadata*). Enumerações são tipos compostos por constantes com o objetivo de representar os valores possíveis de um determinado conjunto. As enumerações são utilizadas na arquitetura atual do GAwCRe para definir os valores de determinadas propriedades. *Situacao*, por exemplo, é uma propriedade utilizada por mais de uma entidade do gerador e que recebe os valores ATIVO e INATIVO. Implementá-la como uma enumeração permite ao desenvolvedor flexibilidade e agilidade ao utilizar essas constantes que serão exigidas mais de uma vez no sistema.

A classe *EnumerationMetadata* armazena as informações sobre essas constantes. O código-fonte exibido na Figura 4.8 exibe trechos correspondente a classe *EnumerationMetadata* e a Figura 4.9 exibe trechos correspondente a classe *EnumerationConstantMetadata*.

```
// Classe que define tipos enumerados
public class EnumerationMetadata extends ObjectMetadata {

       private Set<EnumerationConstantMetadata> constantsMetadata;
         ...
}
```

Figura 4.8 – Trecho de código correspondente à classe *EnumerationMetadata*.

```
// Classe que define metadados das propriedades.
public class EnumerationConstantMetadata implements Metadata {

       private String name;
       private String description;
       private EnumerationMetadata enclosingEnum;
       ...
}
```

Figura 4.9 - Trecho de código correspondente à classe *EnumerationConstantMetadata*.

Existem no sistema, também, entidades que contém propriedades de valores únicos. Como exemplo, para uma mesma clínica de reabilitação física não deveria existir dois pacientes com o mesmo nome e a mesma mãe. Dessa forma, as propriedades *mae* e *nomeMae* da entidade *Paciente* são rotulados como uma restrição *UniqueConstraintMetadata* (restrições de unicidade). Essas restrições podem ser definidas para uma ou para um conjunto de propriedades (*PropertyMetadata*) de uma entidade. A Figura 4.10 exibe o trecho de código correspondente à classe *UniqueConstraintMetadata.*

```
// Classe que define um conjunto de propriedades que não podem ter valores
iguais.
public class UniqueConstraintMetadata {

       private List<PropertyMetadata> uniqueProperties;
       ...
```

Figura 4.10 - Trecho de código correspondente à classe *UniqueConstraintMetadata*.

Para a utilização de outra linguagem de padrões devem ser definidas as entidades e as enumerações que a compõe. Para cada entidade são criados metadados para suas propriedades e restrições de unicidade. As propriedades podem, por sua vez, incluir referências a outras entidades ou enumerações. As enumerações são compostas apenas por constantes, que possuem um identificador e descrição. Na Figura 4.11 é mostrado o relacionamento entre as classes criadas neste projeto para o desenvolvimento do novo gerador. Com essa arquitetura é

possível adicionar metadados para a validação dessas entidades com a implementação da interface *EntityValidator*.

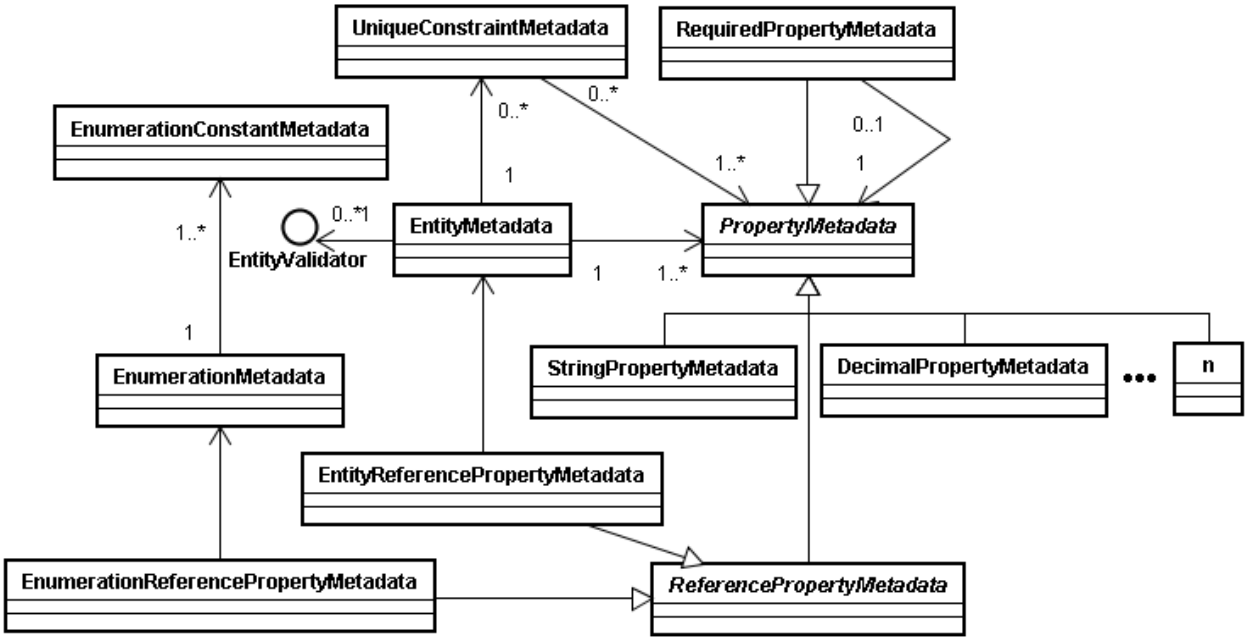


Figura 4.11 - Modelo de Classes do gerador após manutenções corretiva e evolutiva.

As definições do modelo de metadados construído, descritas acima, serão exemplificadas considerando uma aplicação gerada pelo GAwCRe para clínicas médicas. Os requisitos dessa clínica referem-se à venda de serviços (consultas) e de produtos (muletas, cadeiras de roda, etc.). A clínica não realiza atendimento por convênios médicos e faz controle de faturamento.

A classe *ModelMetadataProvider* é do tipo provedora de metadados e é a partir dela que são fornecidos, ao gerador, as informações sobre a linguagem de padrões que está sendo instanciada.

Para exemplificar a definição de uma entidade será considerada a de um médico. Essa entidade pertence à linguagem de padrões SiGCli e tem propriedades, restrições de unicidade e propriedades requeridas indicadas pelas letras a, b e c, respectivamente, na Figura 4.12.

```
//Classe que define os metadados para entidade médico
@ModelMetadataProvider(context = "SiGCli")
public class Medico {
        @EntityMetadataProvider
   public static EntityMetadata getEntity() {
      String packageName = "br.ufscar.gdms";
      String entityName = "Medico";

      EntityMetadata myClass = new EntityMetadata(entityName, packageName);

a     PropertyMetadata nome = new RequiredPropertyMetadata(new StringPropertyMetadata("nome"));
      myClass.addProperty(nome);                                                    b
      PropertyMetadata situacao = new RequiredPropertyMetadata(new
EnumerationReferencePropertyMetadata("situacao", "br.ufscar.gdms.Situacao"));
      myClass.addProperty(situacao);
      PropertyMetadata especialidade = new RequiredPropertyMetadata(new
StringPropertyMetadata("especialidade"));
      myClass.addProperty(especialidade);

      myClass.addUniqueConstraint(new UniqueConstraintMetadata(nome, especialidade));
                                  c
      return myClass;
   }
```

Figura 4.12 - Representação da Entidade Médico.

A entidade **Medico** pertence ao contexto SiGCli e possui as propriedades **nome** do tipo String, **situação** do tipo constante Enumeration (que receberá os valores ATIVO ou INATIVO) e **especialidade** do tipo String. Todas as propriedades para a entidade **Medico** são de preenchimento obrigatório no momento do cadastro, portanto, essas propriedades são rotuladas como propriedades requeridas (*RequiredPropertyMetadata*). As propriedades nome e especialidade não devem ter valores iguais para a entidade Medico, por isso são rotuladas como *UniqueConstraint*, ou seja, o sistema não permitirá o cadastro de mais de um médico com o mesmo nome e a mesma especialidade nessa entidade.

Além da modificação do modelo de dados, a geração dos artefatos no GAwCRe também foi alterada possibilitando a utilização de *templates*, o que aumenta a manutenibilidade e a flexibilidade do gerador. A informação não está mais inserida no código da aplicação e sim em arquivos dedicados que possuem a mesma estrutura dos artefatos a serem gerados. A seção seguinte trata da escolha dos padrões no gerador de aplicações após a manutenção.

4.3.3. Definição dos padrões da Linguagem de Padrões

O GAwCRe original possui em sua arquitetura um conjunto de classes responsáveis por gerenciar e gerar as diferentes instanciações das aplicações, de acordo com os requisitos especificados no XML contendo a linguagem de padrões SiGCli.

Após a manutenção o gerador passou a ter classes que definem os metadados de um padrão e que são processados pelo FreeMarker. Essa modificação além de permitir o desacoplamento das funções do gerador, permite também que a camada de negócios não esteja entrelaçada na camada de modelo, pois, a escolha dos padrões no GAwCRe passa a não ser mais feita por três classes (*AplPadrao*, *AlpVariante* e *Aplicação*) que contém, além das estruturas de dados, toda a regra de negócio do gerador. Com a manutenção, há arquivos de configuração separados para a estrutura de dados e para as regras de negócio desse gerador.

A Figura 4.13 ilustra trechos de código da classe que contém a definição do padrão seis da SiGCli (Identificar Atendente).

```
// Classe que define metadados de um padrão
@ModelMetadataProvider(context = "SiGCli")
public class Pattern6 {                                                    a

        @PatternMetadataProvider
        public static PatternMetadata getPattern6(ModelMetadataRegistry context) {
                PatternMetadata patter6= new PatternMetadata();                                    b
                EntityMetadata atendente = context.getEntity("br.ufscar.gdms.Atendente");
                patterA.addRequiredEntity(atendente);
                ...
                return pattern6;    }
        c
        @VariantMetadataProvider(pattern = "Pattern6")
        public static VariantMetadata getVariant1(ModelMetadataRegistry context) {
                VariantMetadata variant1 = new VariantMetadata();
                variant1.requireEntity("br.com.gdms.ComAtributosAtendente");                       d
                variant1.setProcess(new MetadataModifier() {
                        public static void modify(ModelMetadataRegistry context) {                      e
                        EntityMetadata profissao = context.getEntity("br.ufscar.gdms.Profissao");
                        profissao.addProperty(new ReferencePropertyMetadata("Fisioterapeuta",
"br.com.gdms.Profissao")); }                                                                       f
                });
                return variant1;    }
        }
```

Figura 4.13 - Definição dos Padrões no GAwCRe modificado.

O padrão **Identificar Atendente** é composto pela entidade *Atendente* e tem uma variante denominada *ComAtributosAtendente* em que a clínica pode optar por armazenar informações sobre o ganho e a especialidade do atendente. O trecho de código apresentado, ilustra a entidade *Atendente* que utiliza informações da entidade *Profissao* (definido no padrão 1 da SiGCli) e essa entidade pode ter, dentre outros valores, o valor *Fisioterapeuta*. A Figura 4.13 indica essas características pelas letras: a; b; c; d; e; e f, respectivamente.

Após definidos os dados dos padrões nos metadados de configuração, esses metadados são executados pelo FreeMarker gerando as classes de modelo, controle e visão para cada padrão. Essas classes serão carregadas pelo gerador dependendo dos padrões e variantes escolhidos pelo usuário no momento de instanciar uma aplicação. Isso é possível pois o FreeMarker permite a modificação em seus *templates* originais.

A seção seguinte trata das modificações realizadas na interface do gerador de aplicações GAwCRe.

4.4. Manutenção na Interface do GAwCRe

A interface do GAwCRe e da aplicação gerada apresentava alguns problemas de usabilidade, como por exemplo, a forma de escolha dos padrões, que não permitia a visualização dos padrões selecionados. A tela exibida na Figura 4.14 era pela qual o usuário do gerador fazia a seleção dos padrões que desejava.

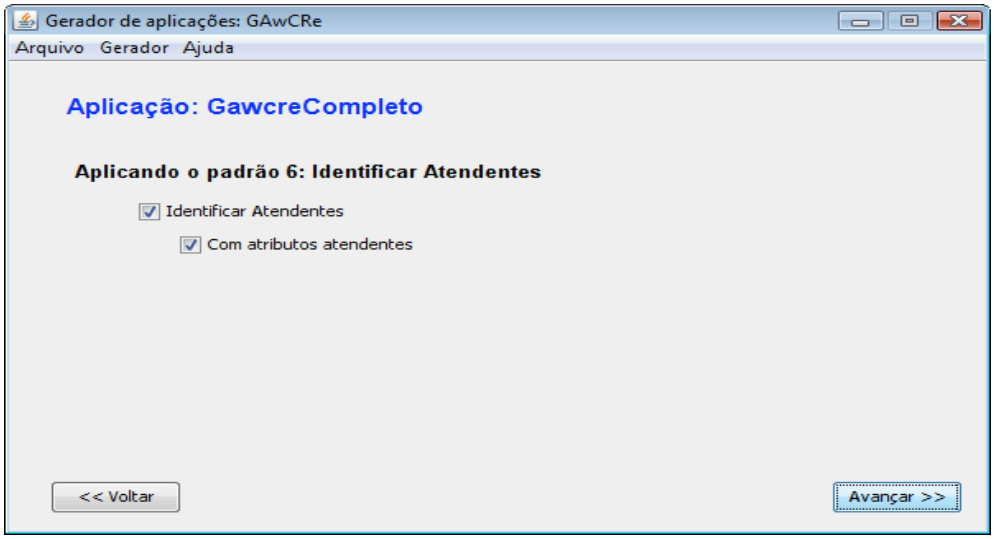


Figura 4.14 - Seleção de padrões feita pelo GAwCRe.

Além do usuário poder apenas visualizar os padrões que foram escolhidos, ao concluir a escolha também não era intuitiva, ou seja, se as variantes existentes em alguns dos padrões eram obrigatórias ou não. Não ficava explícito para o usuário (desenvolvedor da aplicação) quais os padrões existentes na linguagem de padrões SiGCli e quais podiam ser escolhidos.

A solução adotada, para esse caso, foi utilizar um modelo de árvores, possibilitando ao usuário a escolha do padrão desejado, com destaque para os que são obrigatórios, bem como a exibição da lista de padrões. Ao selecionar um padrão que possui variantes e, caso seu uso seja obrigatório, essas são carregadas imediatamente. Caso a variante não seja obrigatória, o usuário não seleciona o nó correspondente a ela na árvore.

Anteriormente a essa manutenção, um padrão que fosse obrigatório deveria ser escolhido e só após a escolha o usuário tinha permissão para selecionar o próximo padrão. Atualmente, esse evento ocorre, porém de forma mais intuitiva, pois, todos os padrões são exibidos por meio de uma árvore e aqueles que são obrigatórios possuem uma descrição. Caso, no momento da seleção, o padrão a ser escolhido seja obrigatório, ele fica em destaque. Além disso, a lista de padrões selecionados também pode ser visualizada como mostra a tela exibida na Figura 4.15.

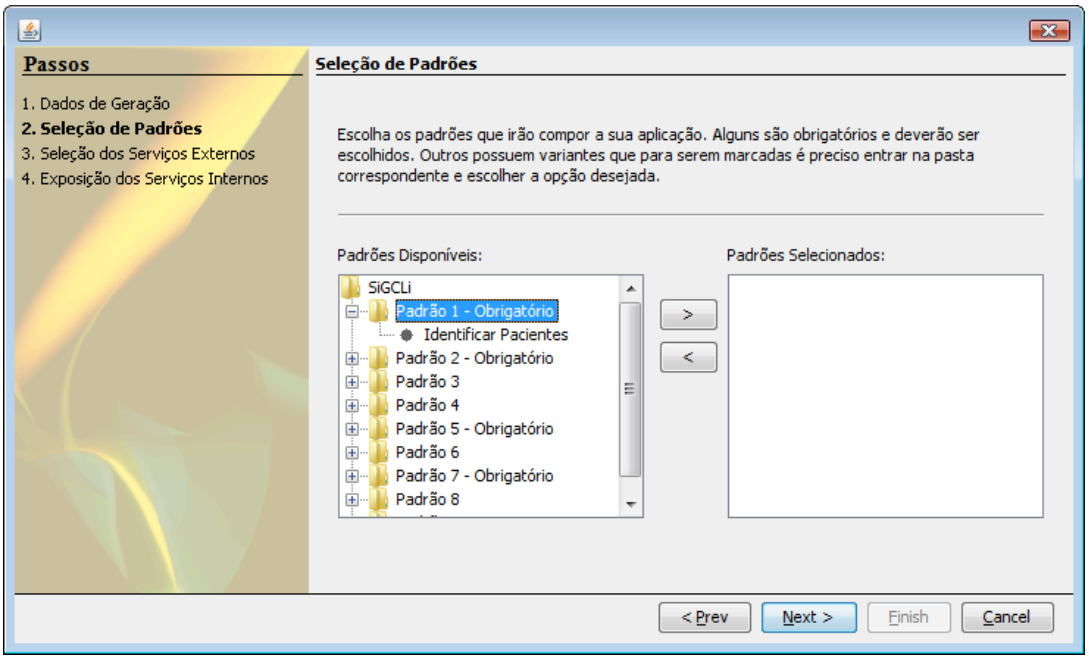


Figura 4.15 - Seleção de padrões no GAwCRe depois da manutenção evolutiva e corretiva.

O ambiente de desenvolvimento utilizado para implementar a nova interface do gerador foi o NetBeans[34]. Desse modo, os botões que possibilitam a mudança de telas por parte do usuário (Próximo, Anterior e Cancelar), têm seus nomes em inglês, como pode ser visto na Figura 4.15.

A aplicação gerada também sofreu modificações decorrentes das mudanças feitas no gerador. A interface gráfica tornou-se mais flexível, permitindo ao usuário fazer modificações de acordo com as suas necessidades e preferências. Anteriormente, a interface gráfica da aplicação gerada no GAwCRe possuía algumas limitações, como a ausência de um mecanismo que permitisse ao desenvolvedor da aplicação personalizar o *layout* da página. A página gerada pelo GAwCRe antes da manutenção é ilustrada na Figura 4.16.

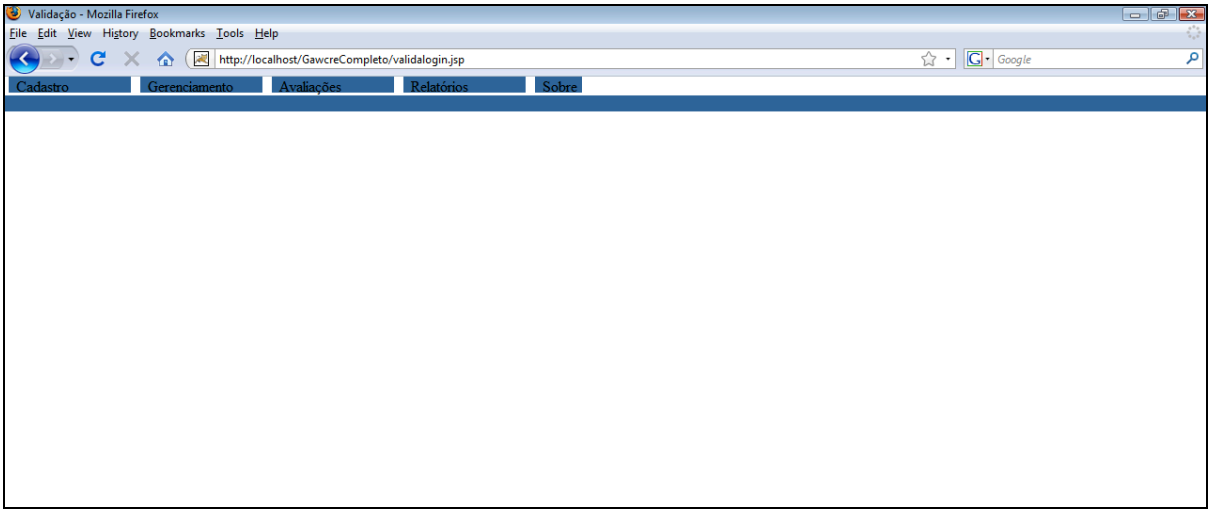


Figura 4.16 - Interface gráfica de uma aplicação gerada pela versão original do GAwCRe.

Com as modificações efetuadas é permitido ao usuário customizar sua aplicação, inserindo o logotipo de sua empresa e de parceiros, formatar o cabeçalho e o plano de fundo da página, configurar um texto de descrição da empresa, além de organizar as telas de cadastro do sistema de acordo com as suas preferências. A Figura 4.17 exibe o exemplo de uma nova interface de uma aplicação gerada pelo GAwCRe para a empresa fictícia chama FysioSys Web, que possui o logo exibido.

[34] http://netbeans.org/

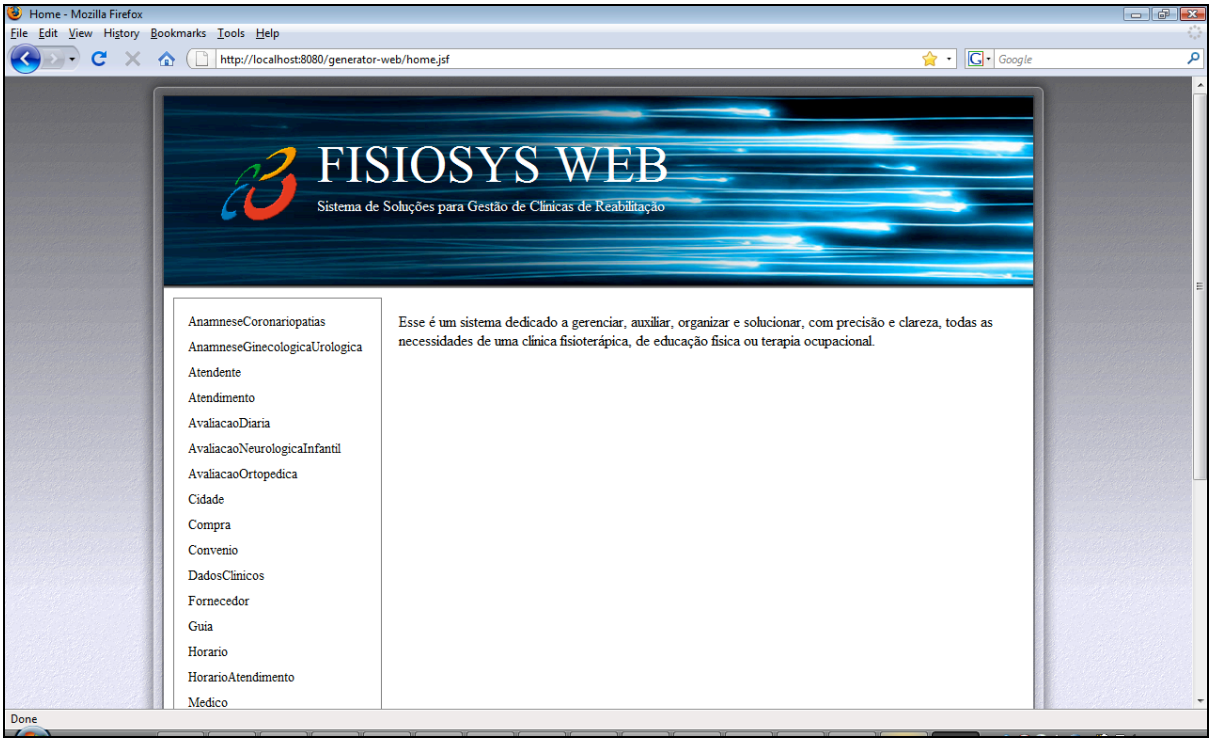


Figura 4.17 - Interface gráfica de uma aplicação gerada pela versão atual do GAwCRe.

Outro problema que existia era que o usuário da aplicação gerada deveria conhecer a ordem na qual os cadastros deveriam ser realizados e quando essa não era seguida, todos os dados já digitados eram perdidos. Por exemplo, ao cadastrar um serviço realizado por uma clínica, de um tipo não existente, na base de dados, antecipadamente deveria ter sido cadastrado o tipo desse serviço. Se isso não ocorresse, as informações inseridas na aplicação eram perdidas. A modificação feita foi inserir um *link* para essas dependências, assim, o usuário ao perceber que ao cadastrar um serviço ainda não há determinado tipo de serviço cadastrado, ele pode selecionar o *link,* cadastrar o tipo, e retornar à tela para completar as outras informações juntamente com as inseridas anteriormente. A Figura 4.18 ilustra o uso dos *links* na interface do cadastro de serviços.

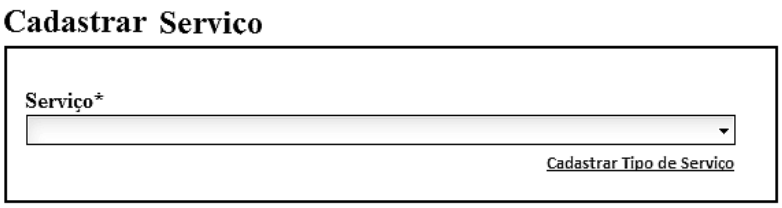


Figura 4.18 - Link para cadastrar tipo de serviço no cadastro de serviço.

A ausência de uma listagem com informações relevantes sobre os atributos já cadastrados para as entidades da aplicação gerada foi outro problema encontrado. No GAwCRe original era exibido ao usuário a lista de um dos atributos da entidade em questão. Na Figura 4.19 está ilustrada uma listagem das informações apresentadas aos usuários pela interface do GAwCRe original para a entidade Médico. Nela estão listados os nomes dos médicos cadastrados no sistema. Caso o usuário desejasse realizar uma busca e não soubesse o atributo nome, mas soubesse o atributo especialidade, por exemplo, ele não poderia realizar a busca, pois, não havia essa flexibilidade na aplicação gerada.

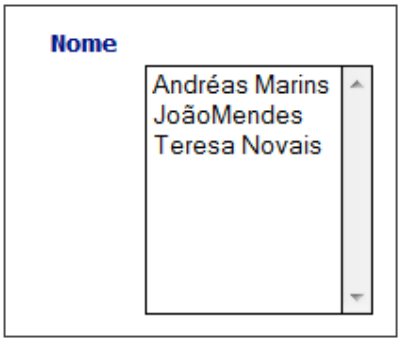


Figura 4.19 - Seleção Listagem das informações no GAwCRe original.

Com a modificação realizada, o usuário visualiza outras informações além do nome do médico, por exemplo, o seu identificador, a sua situação e especialidade. Com isso, há a possibilidade de busca por qualquer uma dessas informações o que torna a aplicação gerada mais flexível e com mais usabilidade. A Figura 4.20 ilustra a listagem dos médicos cadastrados e de suas respectivas especialidades, para uma clínica médica que está sendo gerada.

Identificador	Nome	Situação	Especialidade
1	João Mendes	ATIVO	Ortopedista
2	Teresa Novais	INATIVO	Neurologista
3	Andréa Marins	ATIVO	Cardiologista

Figura 4.20 - Listagem dos atributos relevantes ao usuário na entidade Médico para um sistema de clínica médica.

Como consequência das modificações realizadas houve reestruturação do código gerado, tornando-o mais intuitivo para desenvolvedores. O padrão de projeto MVC (*Model View Controller*) foi usado e também foi criada uma nomenclatura para classes geradas para garantir homogeneidade no código. Além disso, como na versão original do gerador não existia apoio para internacionalização, na nova versão esse recurso foi inserido. A linguagem inglesa é o padrão adotado, o que permite que o código fonte possa ser utilizado por usuários de diferentes países.

Devido à estrutura de metadados construída durante a manutenção evolutiva do GAwCre, descrita na Seção 4.3, é possível que ele use outras linguagens de padrões além da SiGCli. Para isso é necessário que essas linguagens estejam disponíveis na interface do gerador, ou seja, que as entidades e regras de negócios sejam definidas, antecipadamente, nos arquivos de configuração dos metadados.

Quando um usuário desejava iniciar uma nova aplicação no GAwCRe original, o arquivo XML da linguagem de padrões deveria ser escolhido e quando esse fosse carregado, o usuário selecionava os padrões que cobrissem as necessidades exigidas pela aplicação. As telas exibidas na Figura 4.21 são exemplo de telas apresentada nesse caso.

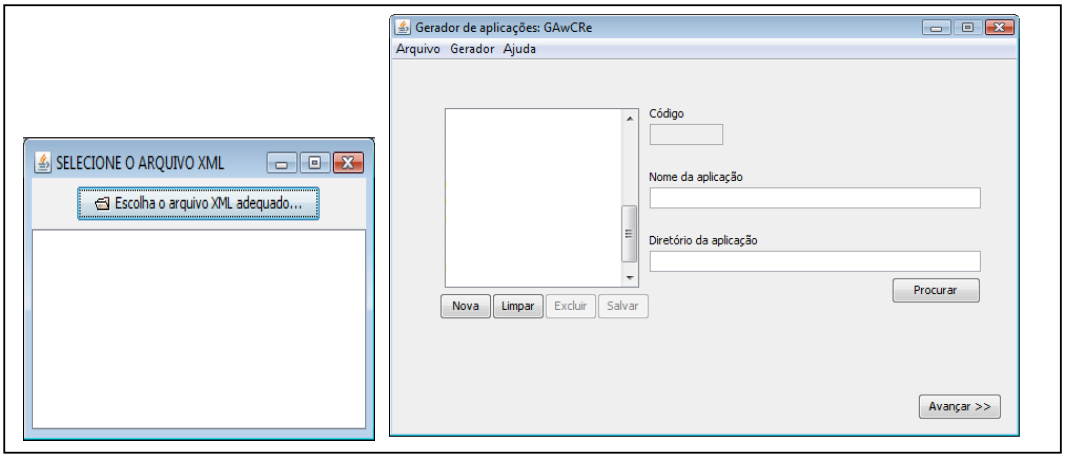


Figura 4.21 - Telas iniciais para gerar uma aplicação no GAwCRe original.

Atualmente, após as modificações realizadas, a tela exibida na Figura 4.22 é apresentada ao usuário para que ele inicie a geração de sua aplicação. Pode-se observar que há a possibilidade de escolher além da linguagem de padrões, o banco de dados e a estratégia de geração (Web ou Desktop) para a nova aplicação. A escolha do banco de dados e da estratégia de geração, assim como descrito para a linguagem de padrões, há necessidade que tenham sido anteriormente configurados nos arquivos de configuração dos metadados. Depois de configurados, as novas opções de banco de dados e a estratégia de geração são disponibilizadas na interface do gerador.

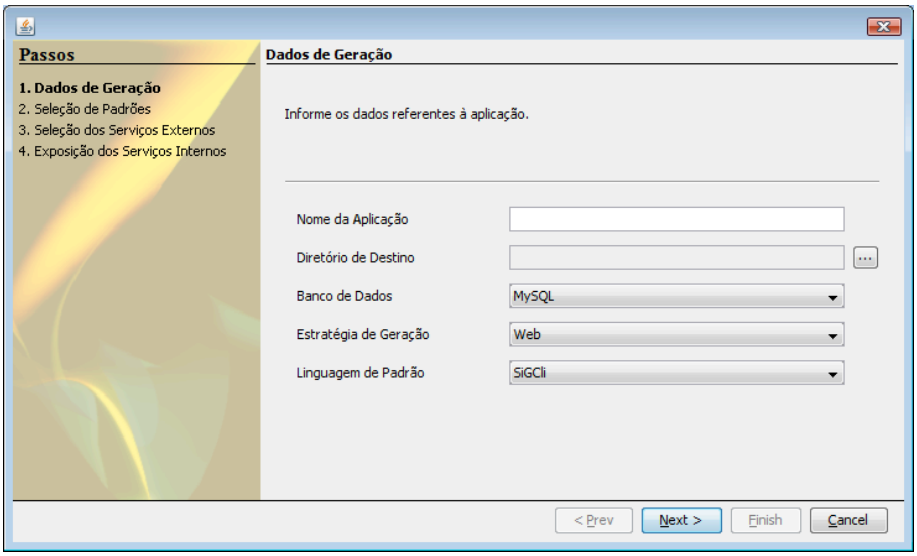


Figura 4.22 - Instanciação de uma aplicação no GAwCRe após manutenção evolutiva.

Na próxima seção será comentado como foi o projeto de manutenção para realização das modificações ocorridas no GAwCRe.

4.5. Estrutura do Projeto de Manutenção

As modificações no GAwCRe foram realizadas com o apoio do ambiente integrado de desenvolvimento Eclipse[35] e foi dividido em cinco projetos: *generator*, *generator-sigcli*, *generator-sources*, *generator-web* e *validation* que são exibidos na Figura 4.23 e serão comentados a seguir.

[35] http://www.eclipse.org/

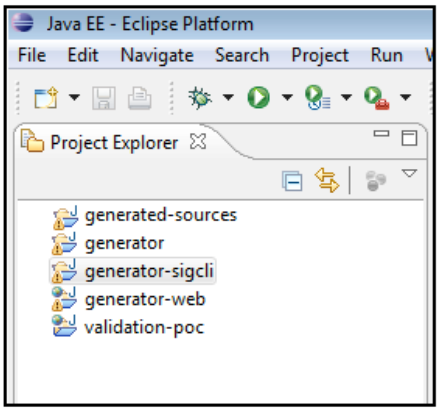


Figura 4.23 - Estrutura do Projeto Eclipse.

A Tabela 4.1 exibe os projetos construídos durante a manutenção do GAwCre com os seus respectivos pacotes e as características referentes a cada um desses pacote.

Tabela 4.1 - Projeto no Eclipse contendo as modificações feita no GAwcRe.

Projeto	Pacote	Características
generator-sigcli	***default***	Contém a classe *main* que executa a leitura dos metadados e instancia o gerador.
	entities	Contém as pré-definições para todas as entidades, enumerações e padrões da linguagem de padrões SiGCli;
	enumerations	
	pattern	
generator	***generation.java***	Contém o mapeamento dos tipos do gerador para a linguagem Java e uma estratégia de geração Java para Web.
	generator	É responsável pela definição de geração da aplicação e dos artefatos.
	loader	É responsável pelo carregamento das

generator		configurações.
	loader.entity	Contém implementações de carregadores de configuração para as entidades.
	loader.enumeration	Contém implementações de carregadores de configuração para as enumerações.
	loader.type	Contém implementações de carregadores de configuração para tipos de dados.
	metadata	Contêm classes abstratas para a definição de metadados, anotações para a definição de dados de geração, escopo e classes para o registro (*registry*) e consulta dos metadados.
	metadata.application	Contém classes para a definição de metadados da aplicação a ser gerada.
	metadata.entity	Contém classes para a definição de metadados de entidade, restrições de unicidade para propriedades das entidades e uma anotação para indicar o fornecimento de metadados da entidade.
	metadata.enumeration	Contém classes para a definição de metadados de enumerações, constantes de enumerações e uma anotação para indicar o fornecimento de metadados de enumeração.
	metadata.impl.property	Contém as implementações dos tipos de dados que serão utilizados pelas propriedades

generator		das entidades.
	metadata.impl.type	Contém as definições dos tipos de dados suportados pelo gerador. Esses tipos servem para fazer a ligação entre os tipos de dados abstratos do gerador e os que serão utilizados no programa gerado (linguagem de programação de destino), dependendo da configuração de tipo fornecida.
	metadata.object	Contém classes que abstraem metadados de entidades e enumerações.
	metadata.property	Contém classes para a definição de metadados de propriedades de entidades, incluindo restrições de preenchimento, preenchimento por serviço e o tipo de dado (referência, primitivo, etc...).
	metadata.service	Contém classes para a definição de metadados de localização (endereço) de serviços.
	metadata.type	Contém classes para a definição de metadados de tipos de dados.
	util	Contém classes utilitárias.
	validator	Contém classes para a definição de validadores de dados (entidades).
generator-sources	***default***	Contém a classe *main*
	model	Contém as classes Java com os dados da aplicação, ou seja, dados de cada entidade e

generator-sources		enumeração da SiGCli geradas pelo *template* e toda a sua lógica de negócios.
	controller	Contém as classes Java com o comportamento da aplicação. Essas classes interpretam as ações do usuário e as mapeia para chamadas do modelo. Também são as responsáveis por criar, deletar, atualizar e listar as entidades e seus dados.
	view	Contém as classes Java que acessam os dados do modelo via controlador (*controller*) e define como esses dados devem ser apresentados.
generation-web (projeto Web)	***model***	Semelhante ao projeto **generator-sources**, porém, usando notações que transformam as classes Java em serviços Web.
	controller	
	view	
	template	*Template* de configuração da página Web a ser exibida ao usuário e que pode ser alterada de forma a adequar-se às necessidades e *layout* do usuário.
validation (projeto Web)	***default***	Contém a classe *main.*
	validation	Contém as classes dos serviços externos inseridos no gerador.

4.5.1. Configurando o banco de dados

Para armazenar e recuperar informações no banco de dado foi utilizada a técnica de desenvolvimento ORM[36] (mapeamento objeto-relacional) usada para reduzir impedimentos da programação orientada a objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

Com essa técnica, o desenvolvedor não manipula comandos em linguagem SQL, ele usará uma interface de programação simples que faz todo o trabalho de persistência.

Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa. A relação entre as tabelas onde estão os dados e o objeto que os disponibiliza é configurada pelo desenvolvedor.

A forma como este mapeamento é configurado depende da ferramenta utilizada. Para a manutenção do GAwCRe foi utilizado o Hibernate[37] e o sistema de anotações (rótulos) que a linguagem Java disponibiliza.

Para configurar o banco de dados que será utilizado pelo gerador de aplicações GAwCRe, foi criado o arquivo de configuração *persistence.xml* (usado pela especificação JPA *Java Persistence Architecture*). Esse arquivo contém uma propriedade *persistence-unit* que encapsula um banco de dados e define o provedor JPA (implementação da especificação) e o *datasource* (link para um *pool* de conexões com o banco de dados). A implementação da especificação precisa da definição do banco de dados utilizado (*driver*). O *driver* é responsável por mapear as classes e os relacionamentos para as tabelas e as chaves.

A Figura 4.24 exibe o trecho de código responsável pela configuração do banco de dados. Nesse arquivo está configurada a aplicação de nome: *fisioSys Web*, com o tipo de transação: JTA[38], com o link para um *pool* de conexões com o banco de dados: *jdbc/fisioSysWeb* e o *driver*: *com.mysql.jdbc.Driver,* (para o banco de dados MySQL). Essas configurações estão representadas na Figura 4.24 pelas letras a, b, c e d, respectivamente.

---

[36] Do inglês - *Object-Relational Mapping.*
[37] *Framework* escrito na linguagem Java que facilita o mapeamento dos atributos entre uma base tradicional de dados relacional e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) ou anotações para estabelecer esta relação. https://www.hibernate.org
[38] http://jta.org - Uma API pertencente a plataforma Java EE que disponibiliza uma interface para a demarcação de transações em aplicações escritas na linguagem Java.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
        <persistence-unit name="fisiosysWeb" transaction-type="JTA">
                                a                                  b
        <provider>mysql.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider</provider
>                       c
                <jta-data-source>jdbc/fisiosysWeb </jta-data-source>
                <exclude-unlisted-classes>false</exclude-unlisted-classes>
                                        <properties>
                        d       <property name="toplink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
                                <property name="toplink.ddl-generation" value="drop-and-create-
tables"/>
                </properties>
        </persistence-unit>
</persistence>
```

Figura 4.24 - Configuração do banco de dados no GAwCRe atual.

Para alterar o banco de dados utilizado, o *driver* deve ser modificado especificando o *driver* do banco desejado.

A Tabela 4.2 exibe uma comparação entre as características do GAwCRe original e do GAwCRe atual em que podem ser observados os benefícios da manutenção realizada.

Tabela 4.2 - Comparação entre características do GAwCRe original e do atual.

	GAwCRe Original	GAwCRe Atual
Configuração e Leitura da SiGCli	A SiGCli é criada a partir de uma LMA que é carregada pelos *templates* de acordo com a seleção do usuário.	A SiGCLi é mapeada nos metadados e as partes correspondentes a seleção do usuário na interface do gerador são geradas pelos *templates* (freemarker).
Ampliação para Domínios Conexos	Era preciso adaptar a linguagem de padrões.	É preciso modificar os metadados referentes à linguagem de padrões (LP) utilizada com as informações específicas da LP a ser usada.
Geração de Artefatos	Com as informações das aplicações armazenadas na base de dados do gerador, os módulos referentes a cada artefato obtêm as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML. Cada módulo gera o artefato no local definido pelo usuário. Para acessar a aplicação gerada o desenvolvedor	A partir das configurações sobre a linguagem de padrões definidas nos metadados e os *templates* carregados a partir das informações obtidas por meio do usuário, os artefatos são gerados automaticamente cada um no seu devido contexto sem precisar da intervenção do usuário.

Geração de Artefatos	da aplicação deve disponibilizar cada artefato gerado no seu devido contexto. Esse processo é feito manualmente.	
Interface do Gerador	As telas do gerador permitiam ao usuário ver apenas os padrões disponíveis no momento da escolha. Não era possível acompanhar os padrões selecionados para a aplicação. Somente após escolher todos os padrões o usuário tinha acesso à lista de padrões escolhidos.	Foi implementada uma estrutura de árvores para a escolha dos padrões que tornou possível acompanhar quais padrões foram escolhidos, quais possuem variantes, bem como os que são obrigatórios.
Listagem dos dados	Não havia separação entre a listagem dos dados cadastrados e o cadastro de novos dados na interface da aplicação gerada. Além disso, listava-se apenas um atributo de cada entidade.	A interface dos dados a serem cadastrados está separada da interface com a listagem daqueles que já foram cadastrados, e o usuário tem acesso aos dados relevantes de cada aplicação não apenas a um atributo como o nome.
Interface da Aplicação Gerada	Não customizável.	Adaptado para as preferências e necessidades do usuário.
Camadas MVC	Interesses da camada de modelo na camada de negócios e da camada de negócio na camada de visualização.	Bem definidas.
Tipo de Dados aceitos	Apenas primitivos para a linguagem de implementação Java.	Tipos primitivos e constantes criados a partir de metadados que mapeiam um tipo abstrato para um tipo específico em determinada plataforma, podendo aceitar tipos para Java, para C, dentre outros.
Banco de dados	Com as informações das aplicações armazenadas na base de dados do gerador, o módulo *Gerador de Scripts SQL* obtém as informações sobre a aplicação a ser gerada comparando-as com as definições do documento XML e gera os scripts do banco de dados os quais o usuário deve manualmente inserir no banco de dados, assim como o *script* para criação do gerador que também é criado manualmente pelo desenvolvedor antes de instanciar qualquer aplicação.	Utiliza tecnologias como, ORM, Hibernate e JPA em que o desenvolvedor não manipula comandos em linguagem SQL, ele usa uma interface de programação simples que faz todo o trabalho de persistência.

4.6. Considerações Finais

Este capítulo apresentou o processo de manutenção evolutiva e corretiva realizado no gerador de aplicações GAwCRe para prepará-lo para apoiar o uso de serviços Web providos por uma arquitetura orientada a serviços (SOA).

As manutenções realizadas possibilitaram que, com o código fonte atual do GAwCRe, fossem obtidos ganhos tanto para o gerador quanto para a aplicação gerada: maior usabilidade, manutenibilidade, entendimento do que o desenvolvedor/usuário está realizando, entre outras.

Outro ganho que também pode ser obtido tanto para os usuários das aplicações geradas quanto para os desenvolvedores, que possam vir a utilizar o gerador, é a facilidade para configurar os *templates* ou para instanciar novas linguagens de padrões, usando a *interface* que foi construída que é mais intuitiva e amigável do que a que existia anteriormente.

Devido à nova estrutura, manutenções posteriores nesse gerador tendem a ser menos custosas, pois, além de um código padronizado, os dados estão mais coesos e as funções com menor acoplamento.

O destaque da manutenção evolutiva realizada é que ela foi projetada para implementar uma arquitetura orientada a serviços. Dessa forma, as funções do sistema, *create, delete, find, listAll* e *update,* foram desenvolvidas como serviços Web que podem ser publicados em um repositório de serviços e utilizados na Internet por outras aplicações.

Outro ponto relevante é que como as funções do sistema foram construídas no formato de serviços Web, caso ocorra alguma mudança de requisitos a manutenção é mais fácil de ser realizada, pois somente o serviço é disponibilizado no repositório na Internet. Em seguida, quando uma aplicação for gerada ou acessada novamente, utilizando o serviço Web do repositório, o usuário já está usando a versão atualizada do sistema. Ou seja, o usuário fica totalmente isolado das modificações que ocorrem, não percebendo a manutenção realizada ao utilizar o sistema.

O Capítulo 5 descreve uma abordagem para o uso de serviços Web providos por uma arquitetura orientada a serviços em geradores de aplicação.

Capítulo 5

Utilização de uma SOA em Geradores de Aplicação

5.1. Considerações Iniciais

Este trabalho propõe a utilização de serviços providos por SOA em GAs construídos a partir de linguagem de padrões de análise. Esses serviços possuem características como distribuição, reúso caixa-preta, dinamismo, adaptabilidade dentre outros, que aplicados aos GAs os tornam mais flexíveis e com o domínio ampliado. A união dessas duas abordagens também permitirá às aplicações geradas, rápida adaptação às mudanças de negócio, além de diminuir a inserção manual de erros provenientes da fase de codificação, oferecendo qualidade, consistência e maior produtividade aos projetos de desenvolvimento de software.

Este capítulo relata o uso de serviços Web providos por SOA em geradores de aplicação. Na Seção 5.2 a abordagem proposta é apresentada. Na Seção 5.3 são detalhados os tipos de serviços específicos que podem ser inseridos em geradores de aplicação. Na Seção 5.4 é relatada a manutenção realizada no código-fonte do gerador de aplicações para que ele apoie o uso de serviços Web providos por SOA e na Seção 5.5 são comentadas as considerações finais.

5.2. Geradores de Aplicação e serviços Web providos por SOA

A ampliação do domínio de um gerador de aplicação pode ser feita utilizando serviços Web providos por uma arquitetura orientada a serviços. A utilização desses serviços em geradores de aplicação pode deixá-los mais flexíveis de modo a facilitar a expansão do seu domínio ao

inserir novas funções sem que haja a necessidade de alterar todos os arquivos de configuração e de código-fonte do gerador.

Além de tornar os geradores mais flexíveis, o uso de serviços facilita a manutenção desses geradores, pois, caso ocorra uma solicitação de mudança de requisitos feita pelo cliente, essa é realizada no serviço Web pelo fornecedor desse serviço e disponibilizada na Internet. Ao usar novamente esse serviço Web, o usuário já estará utilizando a versão atualizada do serviço.

Embora essa abordagem apresente vantagens, não foram encontrados na literatura trabalhos que relatam o uso conjunto delas. Dessa forma, esta dissertação de mestrado descreve uma proposta para o uso de geradores de aplicação com serviços Web providos por uma SOA.

5.2.1. Abordagem Proposta

Para que seja possível a utilização de serviços Web externos em geradores de aplicação, é preciso, inicialmente, estabelecer uma comunicação entre as duas aplicações (gerador de aplicação e serviço Web externo). Dessa forma, o código-fonte do GA deve ser capaz de invocar os serviços Web. Ao invocar esses serviços o gerador precisa conhecer as informações do serviço Web como o nome, onde está disponibilizado e quais operações o serviço realiza. Essas informações constam no WSDL dos serviços Web externos e devem ser especificados no código-fonte do gerador para que o mesmo consiga utilizar o serviço Web externo.

Depois de especificadas as informações necessárias, o gerador é capaz de invocar o serviço Web externo estabelecendo uma comunicação entre as duas aplicações. Em seguida o gerador fornece os dados de entrada informados pelo usuário ao serviço para que esse serviço realize uma operação e retorne as informações necessárias para o gerador.

A Figura 5.1 exibe o esquema de comunicação entre o gerador de aplicação e o serviço Web externo. O serviço Web pode retornar ao gerador de aplicação informações como: dados persistidos, uma resposta booleana ou não fornecer retorno de informações (vide seta tracejada na Figura 5.1). Esses tipos de serviços serão detalhados na Seção 5.3.

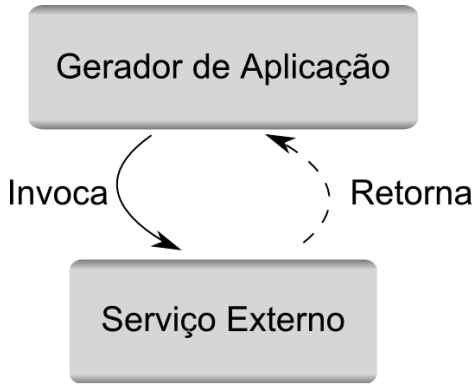


Figura 5.1 - Esquema genérico de comunicação entre o GA e os serviços Web externos.

5.3. Serviços Web externos abrangidos pela proposta

Os tipos de serviços Web construídos e disponibilizados na Internet são, em sua maioria, os serviços que validam dados, os que obtêm dados ou os que fornecem dados para aplicações. Esses serão os serviços abrangidos por esta proposta e estão detalhados a seguir:

- **Serviços de validação** – Serviço bidirecional em que dados de entrada são fornecidos ao serviço e esse devolve como resposta um valor booleano de validação dos dados. Ex.: Um serviço que consulta saldo de cartão de crédito para autorização de compra. Tem-se como resposta a aprovação ou não da compra, caso seja aprovado significa que havia saldo disponível no cartão;

A Figura 5.2 exibe o esquema de comunicação entre serviços Web externos de validação de dados e o gerador de aplicações.

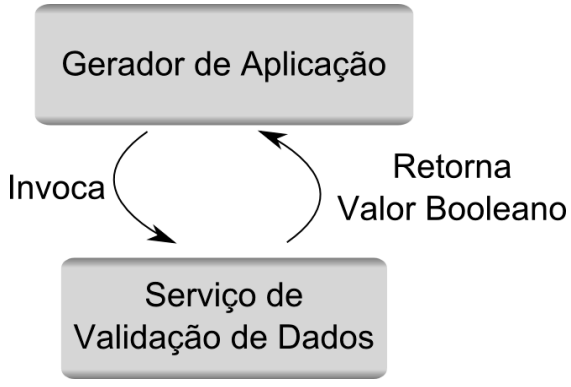


Figura 5.2 - Esquema de comunicação do serviço Web externo de validação de dados.

- **Serviço de Obtenção de Dados** – Serviço também bidirecional em que são fornecidos dados de entrada e a partir desses, são esperados outros dados de saída. Ex.: Um serviço em que o dado de entrada é o CEP e como saída espera-se o endereço completo referente.

A Figura 5.3 exibe o esquema de comunicação entre serviços Web externos de obtenção de dados e o gerador de aplicações.

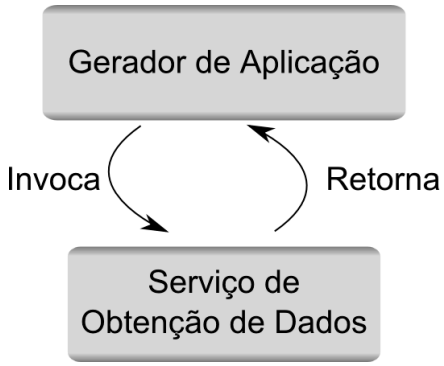


Figura 5.3 - Esquema de comunicação do serviço Web externo de obtenção de dados.

- **Serviço de Fornecimento de Dados** - Serviço unidirecional, ao fornecer novos dados de entrada esse serviço é invocado e realiza sua operação sem devolver nenhum valor (dado) de retorno. Ex.: Sempre que um novo dado é cadastrado no sistema, um serviço dispara uma notificação ao administrador sobre esse novo dado.

A Figura 5.4 exibe o esquema de comunicação entre serviços Web externos de fornecimento de dados e o gerador de aplicações.

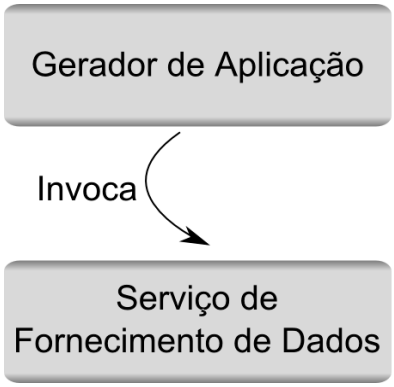


Figura 5.4 - Esquema de comunicação do serviço Web externo de fornecimento de dados.

Esses serviços correspondem aos serviços utilitários e de negócios pela classificação de Erl (2005), descrita no Capítulo 3.

A Seção 5.4 descreve as modificações que devem ser realizadas no gerador de aplicações para que esse possa prover o uso de serviços Web externos em suas aplicações.

5.4. Manutenção adaptativa no gerador de aplicações

Estabelecer a comunicação entre o gerador de aplicações e o serviço Web externo, pode não ser uma tarefa simples de ser implementada. Geradores construídos com alto acoplamento e baixa coesão de dados dificultarão uma manutenção em seu código-fonte para que seja permitida a inserção de um código que realize a comunicação entre o gerador e o serviço Web externo.

O trecho de código a ser implementado no gerador de aplicações deve conter parâmetros que recebam o WSDL do serviço externo, bem como o seu nome, a porta e as operações que o serviço realiza.

A Figura 5.5 exibe um exemplo de classe implementada no código-fonte de um gerador de aplicações (neste caso, o GAwCRe), que contém os parâmetros para o WSDL. Essa classe recebe as informações necessárias do serviço Web externo, estabelecendo uma comunicação com o serviço e permitindo ao gerador gerar aplicações provendo o uso desse serviço.

```
public class ServiceLocationMetadata extends ObjectMetadata {

	private String port;

	private String operation;

	public ServiceLocationMetadata(String wsdlLocation, String name,
String port, String operation) {
		super(name);
		this.port = port;
		this.operation = operation;
	}

	public String getPort() {
		return port; }

	public String getOperation() {
		return operation; }
}
```

a b c d

Figura 5.5 - Trecho de código que permite a integração com serviços Web externos.

Os trechos inscritos nas elipses destacam os parâmetros que devem ser passados ao gerador para que ele possa prover o uso do serviço Web externo. Esses parâmetros são a localização do WSDL contendo as informações do serviço oferecido, o nome do serviço, a porta (porta de conexão) e as operações que ele oferece, representados na Figura 5.5 pelas letras a; b; c e d, respectivamente.

Depois de obtidas essas informações e estabelecida a comunicação com o serviço externo, sempre que uma aplicação gerada solicitar esse serviço o gerador o invocará no repositório de serviços onde ele estiver hospedado e a troca de mensagens é realizada conforme o tipo de serviço invocado pela aplicação.

Além de manutenções no código-fonte do gerador de aplicações, modificações também deverão ser feitas na sua *interface* de forma a permitir a configuração de novos serviços Web externos para as aplicações geradas.

A Figura 5.6 exibe o fluxo a ser seguido para prover a inserção e o uso de serviços Web externos em geradores de aplicação.

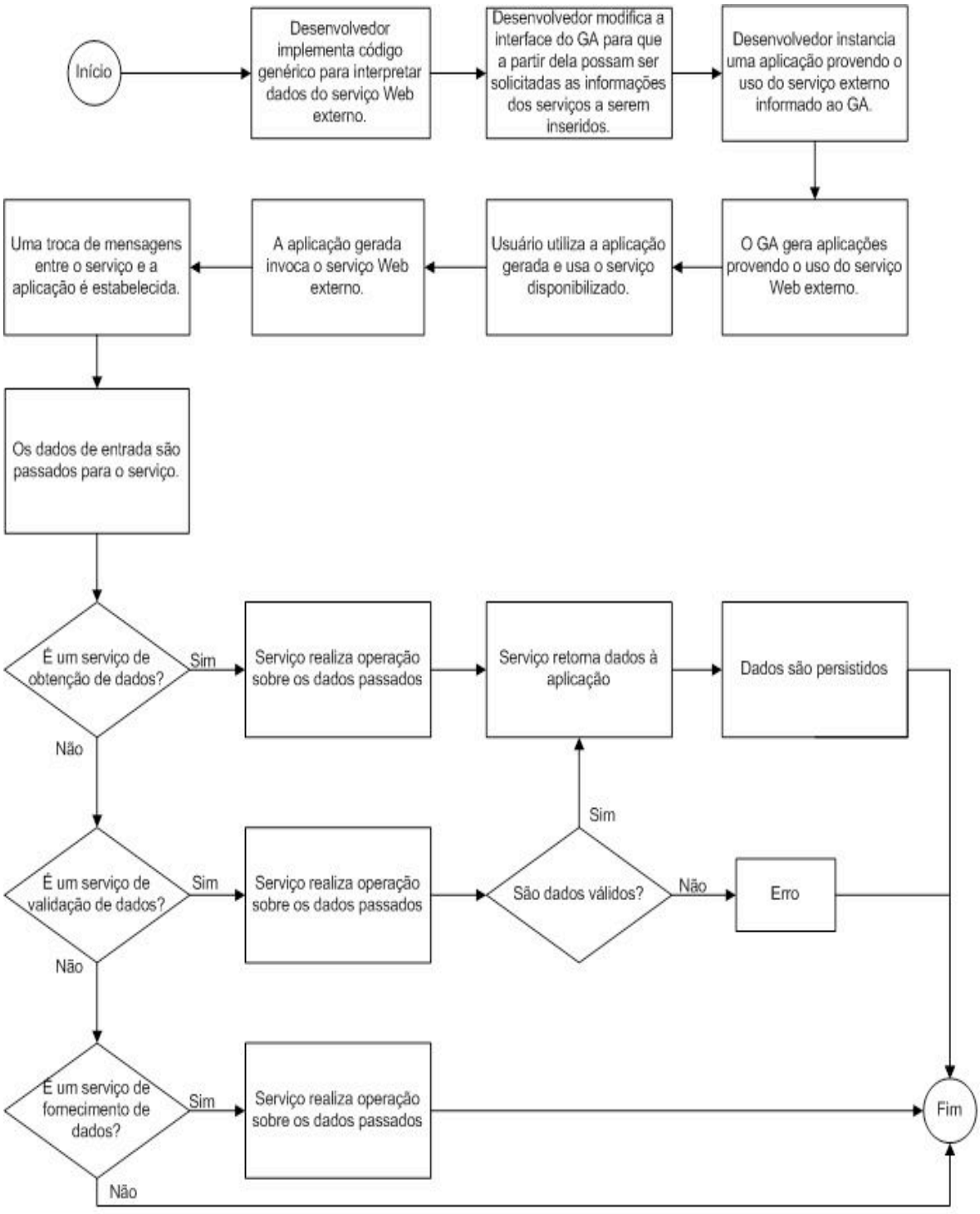


Figura 5.6 – Fluxo para a inserção de serviços Web externo em GAs.

5.5. Considerações Finais

Este capítulo apresentou uma abordagem genérica para a inserção de serviços Web externos em geradores de aplicação, com o intuito de torná-los mais flexíveis e de fácil manutenção.

A utilização de serviços Web providos por uma SOA trará às aplicações geradas pelo gerador mais dinamismo e facilidade de adaptação a mudanças, além de prover a essas aplicações novas funções que são processadas em tempo de execução, permitindo assim, a ampliação do domínio dos geradores.

Para que a abordagem proposta seja utilizada, um estudo de viabilidade dever ser, previamente, realizado no gerador de aplicações. Esse estudo deve analisar a arquitetura utilizada pelo gerador e as tecnologias que foram aplicadas para construí-lo. Além de analisar o esforço necessário para realizar as manutenções que terão por objetivo, adaptar o gerador para prover o uso de serviço Web.

Após esse estudo é possível verificar a possibilidade de realizar uma manutenção adaptativa no sistema para que ele permita o uso de serviços Web.

O Capítulo 6 apresenta um estudo de caso realizado no gerador de aplicações GAwCRe. Esse gerador atende ao domínio de clínicas de reabilitação física e foi utilizado para prover o uso de serviços Web em geradores de aplicação.

Capítulo 6

Integração de serviços Web providos por SOA ao GAwCRe

6.1. Considerações Iniciais

Após as manutenções realizadas no gerador de aplicações GAwCRe é possível inserir serviços Web providos por uma arquitetura orientada a serviços, buscando expandir o domínio desse gerador e permitir que manutenções posteriores sejam facilitadas em decorrência da nova arquitetura construída.

Além de permitir o uso de serviços Web externos, as funções para criar, deletar, alterar, listar e buscar, desenvolvidas para as entidades do GAwCre, foram construídas como serviços Web que podem ser disponibilizadas em repositórios de serviços. Dessa forma, outros sistemas ou serviços, desde que pertençam ao mesmo domínio ou de domínios conexos, podem utilizá-los. Esses serviços oferecidos pelo GAwCRe são chamados de serviços Web internos e para que sejam disponibilizados na Web, o desenvolvedor responsável por instanciar a nova aplicação deve deixar essa opção ativa no sistema.

Este capítulo relata a implementação dos serviços Web externos e internos realizados no GAwCRe. Na Seção 6.2 são comentados os serviços eletrônicos disponibilizados na Internet e suas restrições para que possam ser utilizados no GAwCRe. Na Seção 6.3 é mostrada a integração dos serviços externos inseridos no GAwCRe e quais são os tipos de serviços suportados por esse gerador em consonância com os tipos listados na Seção 5.3. Na Seção 6.4 são apresentados os serviços Web internos oferecidos pelo GAwCRe. Na Seção 6.5

são ilustradas as modificações realizadas na interface do GAwCRe para prover o uso de serviços Web. Na Seção 6.6 são apresentados os usuários existentes no GAwCRe após as manutenções realizadas. Alguns testes realizados para verificar o comportamento do gerador são apresentados na Seção 6.7 e na Seção 6.8 são comentadas as considerações finais.

6.2. Serviços eletrônicos disponíveis na Internet

A quantidade de serviços eletrônicos disponíveis na rede (Internet) atualmente é grande, porém não são todos os serviços disponíveis que são providos por uma arquitetura orientada a serviços. Os serviços que não são providos por uma SOA não possuem um padrão definido, com um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída, especificado de forma a ser independente de qualquer modelo de programação ou outra implementação específica. Esses serviços também não possuem um contrato (WSDL) ou documento descrevendo o que é oferecido, especificando como acessá-lo e quais as operações ou métodos estão disponíveis para o uso. Também não há garantias de que esse serviço esteja sempre disponível para que a aplicação possa utilizá-lo, pois não existe um acordo entre o fornecedor do serviço e o cliente sobre o uso do mesmo.

Dentre os modelos para construção de serviços providos por uma SOA é possível destacar os seguintes: Web Services (WS[39]), CORBA[40], DCOM[41], JSON[42], dentre outros. Esses serviços são construídos utilizando características da arquitetura orientada a serviços como reúso caixa-preta, dinamismo e adaptabilidade. Porém, alguns deles não utilizam padrões abertos como XML ou HTTP e, portanto, não proveem a característica de heterogeneidade ambiental difundida por SOA, que permite a integração entre aplicações construídas em diferentes plataformas e linguagens de programação.

O serviço eletrônico candidato pela W3C[43] a ser o padrão para implementação de uma SOA é o *Web Service*, utilizado nesta dissertação como serviços Web. Os serviços Web utilizam padrões abertos permitindo o seu uso em plataformas e linguagens de programação

[39] http://www.webservices.org
[40] http://www.omg.org
[41] http://www.microsoft.com/com
[42] http://json.org
[43] *World Wide Web Consortium* - http://www.w3.org

diferentes, eles também disponibilizam um documento (WSDL) descrevendo as operações, métodos e localização do serviço.

Diante dessas características, serviços Web foi o modelo para construção de serviços eletrônicos escolhido para realizar a integração de serviços providos por SOA no gerador de aplicações GAwCRe.

Existem ainda disponíveis na Internet serviços eletrônicos construídos a partir do padrão serviços Web (WS), que são de empresas ou instituições que atendem exclusivamente seus clientes e que cobram por esses serviços, não sendo permitido a terceiros utilizá-los.

Por fim, existem os serviços Web que proveem funcionalidade específica, podendo não atender determinados domínios. Exemplos desse tipo de serviço são aqueles que fornecem previsão do tempo, conversão de valores monetários, localização a partir de latitude e longitude, dentre outros. Esses serviços foram encontrados durante a busca por serviços Web que pudessem ser acoplados ao gerador escolhido para ser usado como estudo de caso. No entanto, tais serviços não oferecem ganhos para esse gerador, pois não permitem ampliar o seu domínio.

Para que seja possível a interoperabilidade entre aplicações utilizando serviços Web, é necessário o estabelecimento de uma interface de comunicação entre o serviço e o sistema. Dessa forma, para que os sistemas possam utilizar esses serviços são necessárias manutenções em seu código-fonte para que haja a troca de informações.

Uma das tentativas em inserir serviços externos ao GAwCRe enfocou a utilização de um serviço de e-mail. Nesse serviço, ao cadastrar uma entidade na aplicação gerada, o administrador do sistema ou outro usuário específico receberia uma notificação no seu e-mail informando a adição dos novos dados. Não foi possível utilizar o serviço de e-mail, pois não foi possível estabelecer comunicação com esse serviço de e-mail proposto devido ao fato do serviço não ter sido projetado como um serviço Web, padrão suportado atualmente pelo gerador de aplicações GAwCRe.

6.3. Inserindo serviços no GAwCRe

Após ter passado pelas manutenções evolutivas e corretivas descritas no Capítulo 4, o GAwCRe é capaz de aceitar a inserção de serviços Web em suas aplicações desde que esses

estejam no formato de serviços Web. Esse formato, como descrito na Seção 6.2, foi escolhido por ter se destacado como plataforma interoperável que oferece suporte a SOA tendendo a ser um padrão para implementação dessa arquitetura.

O GAwCRe foi configurado para aceitar em suas aplicações os três tipos de serviços Web descritos no Capítulo 5.

Diante das dificuldades descritas na Seção 6.2 de utilizar os serviços Web disponíveis na Internet, optou-se por criar serviços Web dos três tipos básicos descritos no Capítulo 5, publicá-los em um servidor Web e utilizá-lo em uma aplicação gerada pelo GAwCRe de forma a confirmar a viabilidade da proposta deste trabalho. Esse processo será descrito na próxima seção.

6.3.1. Criando um serviço de Validação de Dados no GAwCRe

Os serviços externos inseridos no GAwCRe são publicados no servidor de aplicações Glassfish[44] da Sun[45] Microsystems que possui suporte às novas especificações Web Java e é responsável por hospedar as aplicações e prover os serviços Web às aplicações clientes.

Como descrito na Seção 6.3 os serviços de validação retornam dados de saída respondendo às condições da operação provida pelo serviço. Esses dados podem ser positivos ou negativos caso essas condições tenham sido ou não aceitas. Para exemplificar o uso desses serviços no gerador, foi desenvolvido um serviço externo que recebe os dados de entrada: cidade, estado e país; e retorna as coordenadas de latitude e longitude referentes a esses dados, porém, a condição para o retorno dessas coordenadas é que a cidade, o estado e o país sejam São Paulo, São Paulo e Brasil. Se os dados digitados forem diferentes desses, o serviço retorna o valor zero para a latitude e longitude daquela localização. As condições de aceitação para esse serviço podem ser customizadas a depender do que se queira validar.

A Figura 6.1 ilustra esse serviço em uma aplicação gerada pelo GAwCRe.

[44] https://glassfish.dev.java.net
[45] http://br.sun.com

Figura 6.1 - Serviço do tipo Validação de Dados inserido no GAwCRe.

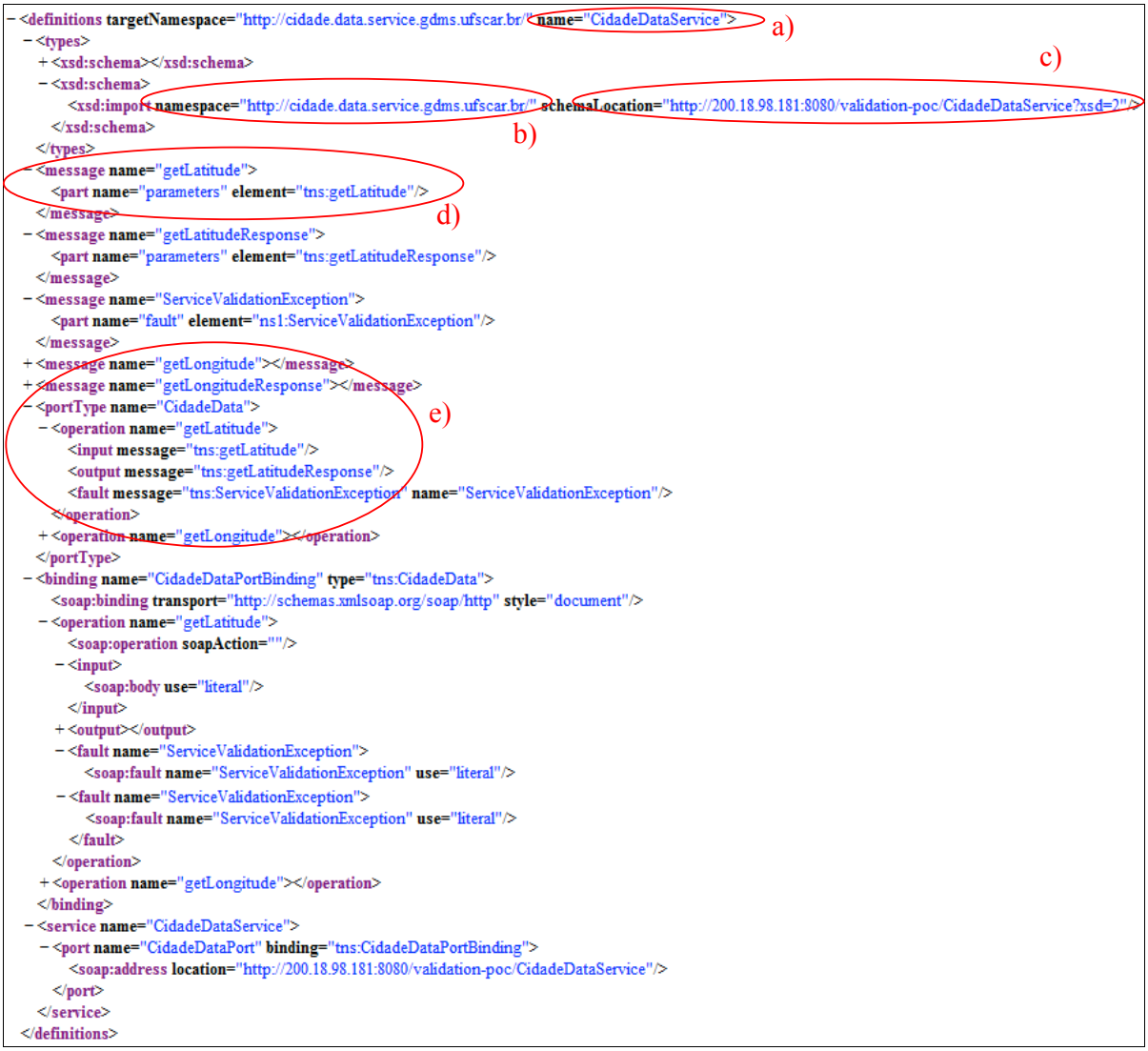
Observe que o sistema só forneceu a latitude e longitude da ocorrência na qual a cidade cadastrada corresponde a São Paulo, o estado São Paulo e o país Brasil conforme a operação definida no serviço, em que somente a latitude e longitude desses locais são definidas.

A Figura 6.2 abaixo ilustra o código-fonte do serviço de validação implementado. As partes com elipses exibem a implementação das condições para operação desse serviço.

```
// Classe que expõe o serviço para a obtenção de dados
@WebService(serviceName = "CidadeDataService", portName = "CidadeDataPort", targetNamespace =
"http://cidade.data.service.gdms.ufscar.br/")
public class CidadeData {
        @WebMethod
        @RequestWrapper(className = "br.ufscar.gdms.data.service.validation.cidade.CidadeLatitudeDataRequest")
        @ResponseWrapper(className =
"br.ufscar.gdms.data.service.validation.cidade.CidadadeLatitudeDataResponse")
        public Double getLatitude(@WebParam(name = "cidade")String cidade, @WebParam(name =
"estado")String estado, @WebParam(name = "pais")String pais) throws ServiceValidationException {
                Double latitude = 0D;
                if (cidade.equals("São Paulo") && estado.equals("São Paulo") && pais.equals("Brasil")) {
                        latitude = -23.488439; }
                return latitude; }
        @WebMethod
        @RequestWrapper(className =
"br.ufscar.gdms.data.service.validation.cidade.CidadeLongitudeDataRequest")
        @ResponseWrapper(className =
"br.ufscar.gdms.data.service.validation.cidade.CidadadeLongitudeDataResponse")
        public Double getLongitude(@WebParam(name = "cidade")String cidade, @WebParam(name =
"estado")String estado, @WebParam(name = "pais")String pais) throws ServiceValidationException {
                Double longitude = 0D;
                if (cidade.equals("São Paulo") && estado.equals("São Paulo") && pais.equals("Brasil")) {
                        longitude = -46.639709; }
return longitude;       } }
```

Figura 6.2 - Código-fonte do serviço Web tipo Validação de Dados implementado para usar no GAwCRe.

O WSDL é o documento em que é possível ver todos os dados fornecidos por um serviço. A Figura 6.3 ilustra trechos do WSDL gerado para o serviço de Validação de Dados. Com ele é possível saber quais operações serão realizadas, os parâmetros a serem passados, a URL onde se encontra esse serviço, o *namespace*, dentre outras informações.




```
- <definitions targetNamespace="http://cidade.data.service.gdms.ufscar.br/" name="CidadeDataService">
  - <types>
    + <xsd:schema></xsd:schema>
    - <xsd:schema>
        <xsd:import namespace="http://cidade.data.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeDataService?xsd=2"/>
      </xsd:schema>
    </types>
  - <message name="getLatitude">
      <part name="parameters" element="tns:getLatitude"/>
    </message>
  - <message name="getLatitudeResponse">
      <part name="parameters" element="tns:getLatitudeResponse"/>
    </message>
  - <message name="ServiceValidationException">
      <part name="fault" element="ns1:ServiceValidationException"/>
    </message>
  + <message name="getLongitude"></message>
  + <message name="getLongitudeResponse"></message>
  - <portType name="CidadeData">
    - <operation name="getLatitude">
        <input message="tns:getLatitude"/>
        <output message="tns:getLatitudeResponse"/>
        <fault message="tns:ServiceValidationException" name="ServiceValidationException"/>
      </operation>
    + <operation name="getLongitude"></operation>
    </portType>
  - <binding name="CidadeDataPortBinding" type="tns:CidadeData">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="getLatitude">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      + <output></output>
      - <fault name="ServiceValidationException">
          <soap:fault name="ServiceValidationException" use="literal"/>
        </fault>
      - <fault name="ServiceValidationException">
          <soap:fault name="ServiceValidationException" use="literal"/>
        </fault>
      </operation>
    + <operation name="getLongitude"></operation>
    </binding>
  - <service name="CidadeDataService">
    - <port name="CidadeDataPort" binding="tns:CidadeDataPortBinding">
        <soap:address location="http://200.18.98.181:8080/validation-poc/CidadeDataService"/>
      </port>
    </service>
</definitions>
```

Figura 6.3 - WSDL do serviço Web tipo Validação de Dados implementado para usar no GAwCRe.

As partes inscritas nas elipses da Figura 6.3 representam: a) o nome do serviço; b) seu *namespace*; c) o seu endereço de localização (URL); d) mensagens como os parâmetros passados; e e) operações realizadas pelo serviço.

A utilização ou construção de serviços Web externos depende da disponibilidade de serviços que atendam ao domínio do sistema. No caso do gerador GAwCRe que possui um domínio específico (clínicas de reabilitação física), não foi possível encontrar serviços Web externos que, além de serem no formato aceito pelo gerador, não fossem proprietários e atendessem a esse domínio. Dessa forma, optou-se por projetar e publicar serviços que permitissem nova funcionalidade ao sistema e que comprovasse a abordagem proposta. A evolução dessa abordagem está condicionada ao surgimento de novo serviços eletrônicos disponibilizados, gratuitamente, na Internet e que estejam no formato de serviços Web.

Para o domínio do GAwCRe poderiam ser acoplados serviços de *Validação de Dados* que, por exemplo, checassem o número do CREFITO (Conselho Regional de Fisioterapia e Terapia Ocupacional) dos profissionais e retornasse se são válidos e se esses profissionais estão aptos a exercer a profissão. Ou ainda um serviço que, fornecido o nome do paciente, checasse se esse está em débito com a clínica e assim permitir ou não um novo atendimento ou compra. O controle de acesso dos usuários do sistema poderia ser uma terceira opção de serviço para *Validação de Dados,* em que o usuário digitaria o nome do usuário e sua respectiva senha e o serviço checaria a legitimidade dos dados permitindo a entrada no sistema, com permissões de acesso restritas a cada tipo de usuário.

6.3.2. Criando um serviço de Obtenção de Dados no GAwCRe

Os serviços de *Obtenção de Dados* retornam dados de saída a partir de dados de entrada fornecidos ao sistema. O serviço usado no GAwCRe de *obtenção de dados* possui restrições no seu uso, pois, o serviço de *validação de dados* citado na Seção 6.3.1, usa os mesmos dados de entrada. Nos serviços de *Obtenção de Dados* são passados: cidade, estado e país como dados de entrada e a saída é a latitude e longitude do lugar referente aos dados de entrada. Devido à condição de operação do tipo de serviço *Validação de Dados* somente a latitude e longitude de São Paulo (cidade), São Paulo (Estado) e Brasil (país) estão sendo impressos na interface da aplicação gerada. Mas, esse serviço pode ser generalizado para devolver a latitude e longitude de qualquer outra localidade.

A diferença entre o sistema de *Validação de Dados* e o de *Obtenção de Dados* descrito é que o primeiro valida uma operação, ou seja, só exibe a latitude e longitude se as entradas

forem São Paulo, São Paulo e Brasil, outras regras poderiam ter sido adotadas. No serviço de *Obtenção de Dados*, a partir dos valores de entrada fornecidos ao serviço, ele retornará a latitude e a longitude da localização, porém, como há a condição do primeiro serviço, só está sendo retornada uma localização. Como esses serviços foram criados apenas para ilustrar a aplicabilidade dos serviços Web externos no GAwCRe, eles não possuem regras de negócio complexas.

A Figura 6.4 exibe a latitude e longitude apresentada na tela da aplicação gerada para os dados de entrada fornecidos pelo usuário.

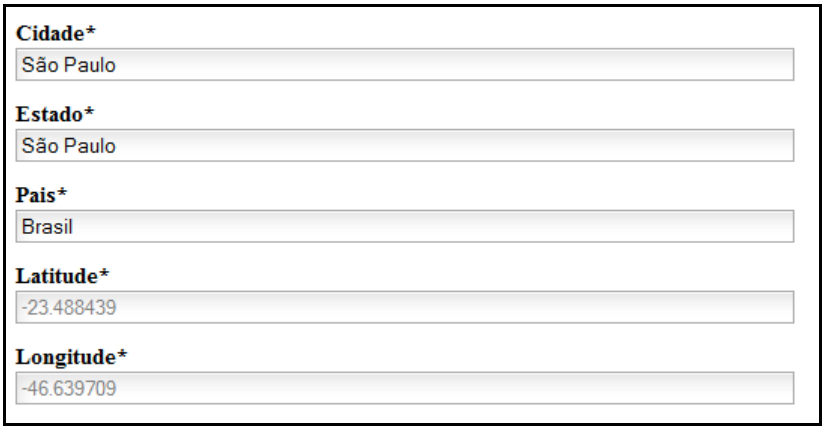


Figura 6.4 - Parte da interface da aplicação gerada onde a latitude e longitude são mostradas.

Outros serviços de *Obtenção de Dados* podem ser inseridos no GAwCRe. Por exemplo, dado um atendente (fisioterapeuta, terapeuta) o sistema retorna ao usuário todos os atendimentos agendados para esse atendente em determinada data.

O código-fonte e o WSDL referentes ao serviço de *Obtenção de Dados* encontram-se nos Apêndices A e B, respectivamente.

6.3.3. Criando um serviço de Fornecimento de Dados no GAwCRe

O serviço de Fornecimento de Dados como descrito na Seção 6.3 não precisa de um valor de retorno para continuar alguma operação no sistema. Nesse serviço, dados de entrada são fornecidos e uma operação é realizada. O serviço utilizado no GAwCRe recebe os dados de

entrada: cidade, estado e país e realiza a operação que é mandar uma mensagem ao desenvolvedor da nova ocorrência para a entidade *cidade* e que foi cadastrada no sistema.

A Figura 6.5 ilustra o uso desse serviço no GAwCRe. A elipse destaca todas as ocorrências cadastradas no sistema para a entidade *cidade*.

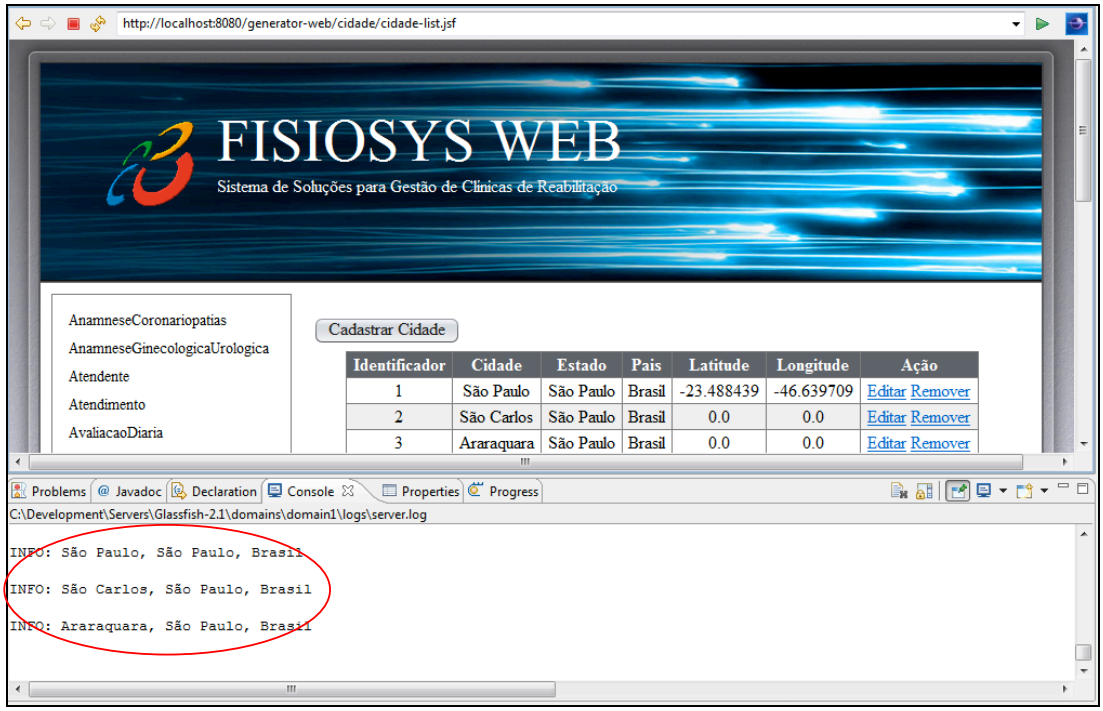


Figura 6.5 - Serviço do tipo Fornecimento de Dados inserido no GAwCRe.

Gerar relatórios a partir de determinados dados de entrada pode ser um serviço de *Fornecimento de Dados* relevante a ser inserido no GAwCRe.

O código-fonte e o WSDL construídos para esse serviço podem ser encontrados nos Apêndice I e II, respectivamente.

6.4. Implementando uma função interna como um serviço Web.

As funções de criar, deletar, atualizar, buscar e listar de todas as entidades do GAwCRe, após manutenção realizada, foram implementadas como serviços Web e podem ser disponibilizadas na rede (Internet) caso o desenvolvedor da aplicação selecione essa opção para ser utilizada em sua aplicação.

A opção de permitir que o serviço Web seja oferecido na rede está disponível na interface do gerador de aplicações e caso seja selecionada, um código referente à criação de serviços é gerado. Caso essa opção não seja a selecionada pelo usuário, as funções são geradas da forma tradicional (classes Java).

Ao selecionar essa opção são gerados também os WSDLs dos serviços internos (*create*, *delete*, *findById*, *listAll* e *update*) e para que outro usuário da rede (Internet) possa utilizá-lo ele buscará esse serviço pelo endereço descrito no WSDL do serviço. Esse é o endereço da aplicação gerada pelo GAwCRe onde o serviço estará disponibilizado. Para que seja encontrado pelo usuário, antes esse serviço deve ser exposto em um repositório de serviços como o UDDI[46].

O UDDI é um Web Service que gerencia informação sobre provedores, implementações e metadados de serviços. Para que um usuário ou outro serviço encontre o serviço desejado, deve ser realizada a busca pelo nome e pelo tipo de dado de saída esperado para esse serviço. Caso o usuário já conheça o WSDL do serviço, esse é passado para o repositório UDDI, por exemplo, e ele devolve o endereço, nome, operações e métodos daquele serviço. A Figura 6.6 ilustra o trecho de código gerado para uma entidade implementada como serviço web.

```
// Classe implementada a camada de negócio expondo suas operações como serviços Web
@WebService(serviceName = "CidadeService", portName = "CidadePort", targetNamespace =
"http://cidade.service.gdms.ufscar.br/") a)
public class CidadeHome {        b)  …
        @WebMethod                                                              c)
        @RequestWrapper(className = "br.ufscar.gdms.service.cidade.CreateCidadeRequest")
        @ResponseWrapper(className = "br.ufscar.gdms.service.cidade.CreateCidadeResponse")
        public Cidade create(@WebParam(name = "cidade") Cidade cidade) throws
ServiceException {
                EntityManager manager = managerFactory.createEntityManager();
                try {
                        transaction.begin();
                        manager.joinTransaction();
                        ...
                        manager.persist(convenio);
                        transaction.commit();
                } catch (PersistenceException e) {
                        try { if (transaction.getStatus() == Status.STATUS_MARKED_ROLLBACK)
{
                                        transaction.rollback();} … }
```

Figura 6.6 - Trechos do código-fonte gerado para a entidade cidade como serviço Web.

[46] Do inglês - *Universal Description, Discovery and Integration.*

As partes inscritas nas elipses mostram pelas letras a; b; e c, respectivamente: o nome do serviço Web gerado, o endereço onde ele será disponibilizado e os métodos que ele implementa. O uso das anotações *@WebService*, *@WebMethod* e outras são as responsáveis por transformar classes Java da aplicação em classes Java para serviços Web.

A Figura 6.7 mostra trechos do WSDL referente ao código-fonte apresentado na Figura 6.6.

```
- <definitions targetNamespace="http://cidade.service.gdms.ufscar.br/" name="CidadeService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/generator-web/CidadeService?xsd=1"/>
      </xsd:schema>
    + <xsd:schema></xsd:schema>
    </types>
  - <message name="delete">
      <part name="parameters" element="tns:delete"/>
    </message>
  + <message name="deleteResponse"></message>
  + <message name="ServiceException"></message>
  + <message name="create"></message>
  + <message name="createResponse"></message>
  + <message name="update"></message>
  + <message name="updateResponse"></message>
  + <message name="findById"></message>
  + <message name="findByIdResponse"></message>
  + <message name="listAll"></message>
  + <message name="listAllResponse"></message>
  + <portType name="CidadeHome"></portType>
  + <binding name="CidadePortBinding" type="tns:CidadeHome"></binding>
  - <service name="CidadeService">
    - <port name="CidadePort" binding="tns:CidadePortBinding">
        <soap:address location="http://200.18.98.181:8080/generator-web/CidadeService"/>
      </port>
    </service>
</definitions>
```

Serviços Web disponibilizados pela entidade *cidade*.

Figura 6.7 - Trechos WSDL criado para a entidade cidade.

A seção seguinte relata as adaptações feitas na interface do gerador GAwCRe para que esse provesse e utilizasse serviços Web.

6.5. Adaptações feitas na interface do GAwCRe para prover e usar serviços Web.

Para permitir o uso de serviços Web externos e a exposição dos seus serviços Web internos na rede, o gerador de aplicações GAwCRe sofreu modificações em sua interface gráfica. Inserir serviço Web externos ou expor os serviços Web internos são os últimos passos executados na interface do gerador para instanciar uma nova aplicação. A Figura 6.8 ilustra a interface do

GAwCRE atual em que é apresentada ao usuário a oportunidade de utilizar um serviço externo para prover uma nova funcionalidade para a sua aplicação.

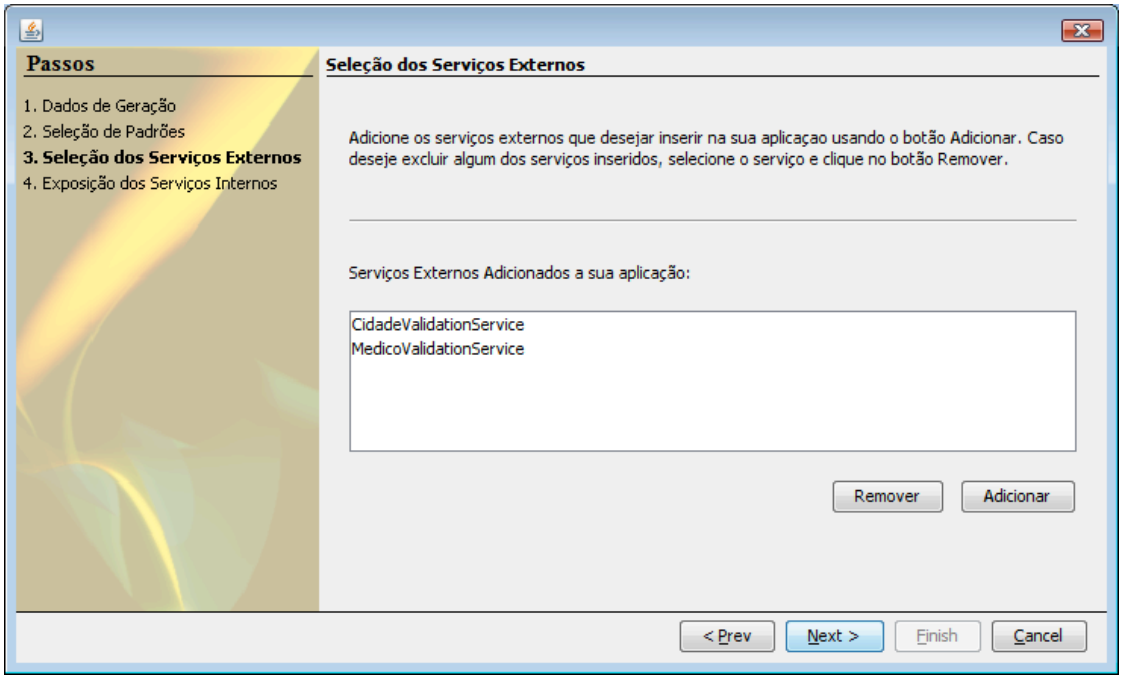


Figura 6.8 - Seleção dos serviços externos no GAwCRe atual.

Para que um serviço Web externo seja adicionado ao gerador, é necessário que ele tenha sido construído utilizando padrões de serviço Web semelhantes aos que foram utilizados na manutenção evolutiva do GAwCRe, pois tal gerador ainda não provê o uso de serviços eletrônicos que não seja no padrão de serviço acima referenciado.

Se o serviço a ser utilizado for um serviço Web, o próximo passo exige do desenvolvedor, que está instanciando a nova aplicação, conhecimento sobre o domínio do gerador de aplicações, além de conhecimento sobre os dados referentes ao serviço que ele está inserindo na aplicação. Os dados exigidos são: a) o nome do serviço, b) a porta de resposta, c) seu *namespace*, d) operações que desempenha, e) parâmetros a serem passados, f) valores de retorno, g) endereço do WSDL e h) a qual tipo de serviço (validação, obtenção de dados ou fornecimento de dados) ele pertence. Essas informações podem ser encontradas no WSDL do serviço provido.

A Figura 6.9 ilustra a interface do GAwCRe atual com as informações solicitadas.

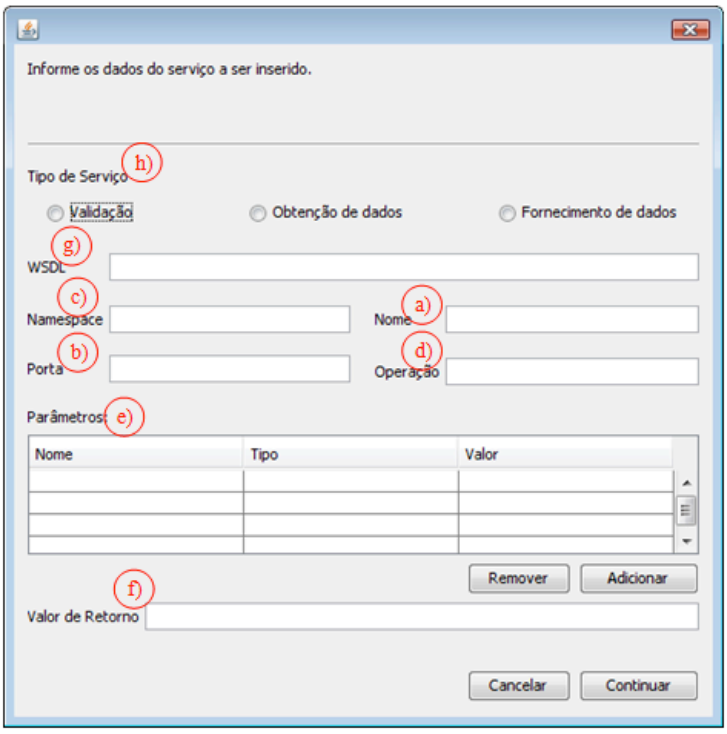


Figura 6.9 - Dados necessários para a inserção de um serviço externo no GAwCRe atual.

Quando o desenvolvedor adiciona um parâmetro ele precisa conhecer ainda o tipo (que pode ser fixo ou uma referência para a propriedade de uma entidade) e o valor desse parâmetro. A Figura 6.10 exibe a tela da interface gráfica do GAwCRe na qual essas informações são passadas para o gerador.

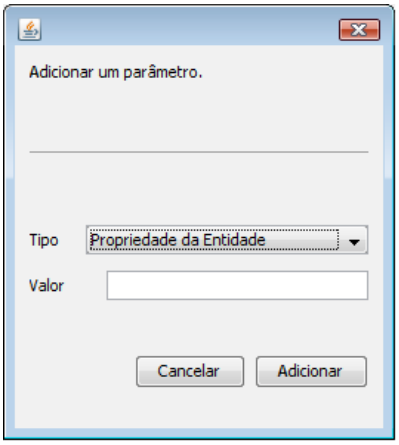


Figura 6.10 - Dados para adicionar um parâmetro à lista.

O último passo para a instanciação de uma nova aplicação é a exposição dos serviços Web internos. Ao escolher essa opção o desenvolvedor permitirá que os serviços Web daquela aplicação sejam disponibilizados na Internet e que outras aplicações ou serviços possam utilizá-los. A Figura 6.11 exibe a opção de expor os serviços internos na interface atual do GAwCRe.

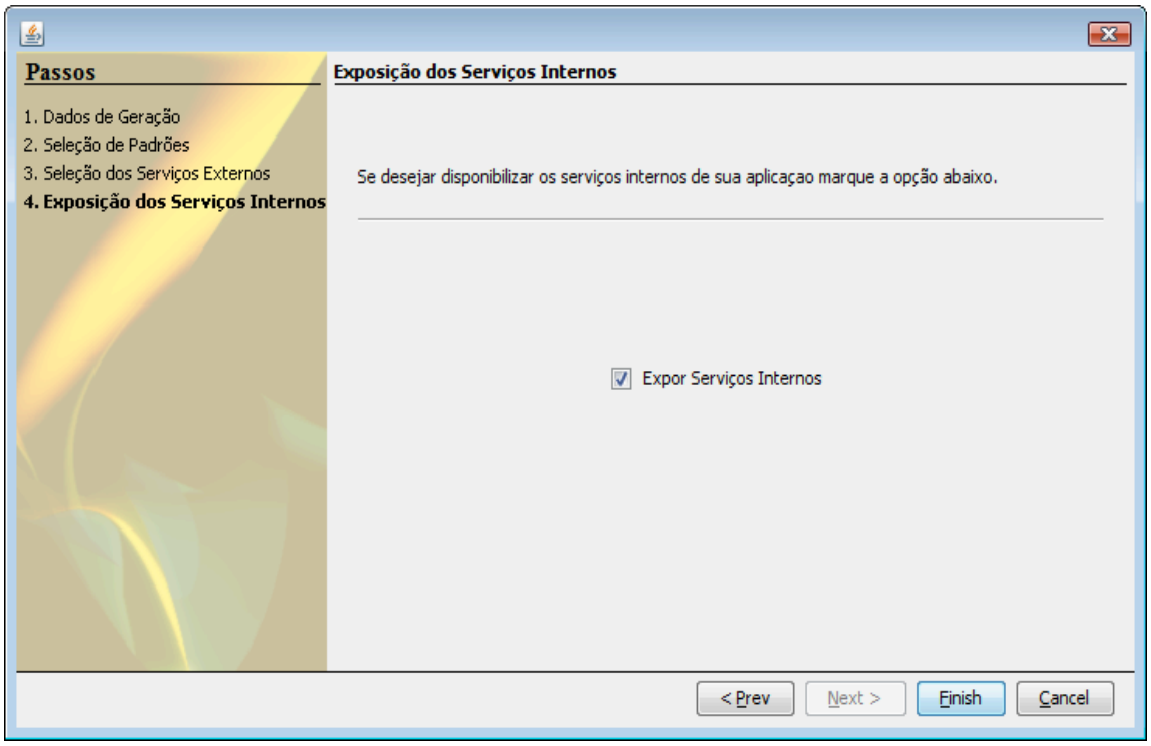


Figura 6.11 - Exposição dos serviços internos da aplicação.

Quando o desenvolvedor marca essa opção, o gerador de aplicações executa a parte do código-fonte referente à geração das funções da aplicação como serviços Web. Se o desenvolvedor não selecionar essa opção, as funções serão geradas como classes Java tradicional.

6.6. Usuários do sistema

Após as modificações descritas no Capítulo 4, o GAwCre passou a ter três tipos de usuários.

- O usuário *Desenvolvedor do Sistema* - Aquele que melhor conhece o domínio do gerador de aplicações e é o responsável por configurar os metadados para aceitar

novas linguagens de padrões, bancos de dados, novos tipos de serviços e estratégias de geração. É responsável ainda por inserir novas entidades e enumerações, configurar novos idiomas e configurar a interface gráfica da aplicação a ser instanciada, de acordo com as necessidades e opções do cliente;

- O usuário *Desenvolvedor da Aplicação* - Responsável por instanciar novas aplicações. Esse usuário pode criar uma nova aplicação, escolher a linguagem de padrões a ser utilizada, escolher o banco de dados, escolher a estratégia de geração, escolher quais padrões aplicar, quais serviços Web externos quer inserir na sua aplicação, além de expor os serviços Web internos na Internet;

- O *Usuário Final* - Aquele que utiliza a aplicação gerada pelo GAwCRe. É o responsável por cadastrar os dados das entidades e por utilizar o sistema.

O usuário *Desenvolvedor da Aplicação* pode realizar as funções do *Usuário Final* e o usuário *Desenvolvedor do Sistema* pode realizar as funções do *Desenvolvedor da Aplicação.* O Diagrama de Caso de Uso especificado na Figura 6.12 exibe os usuários do sistema e suas funções.

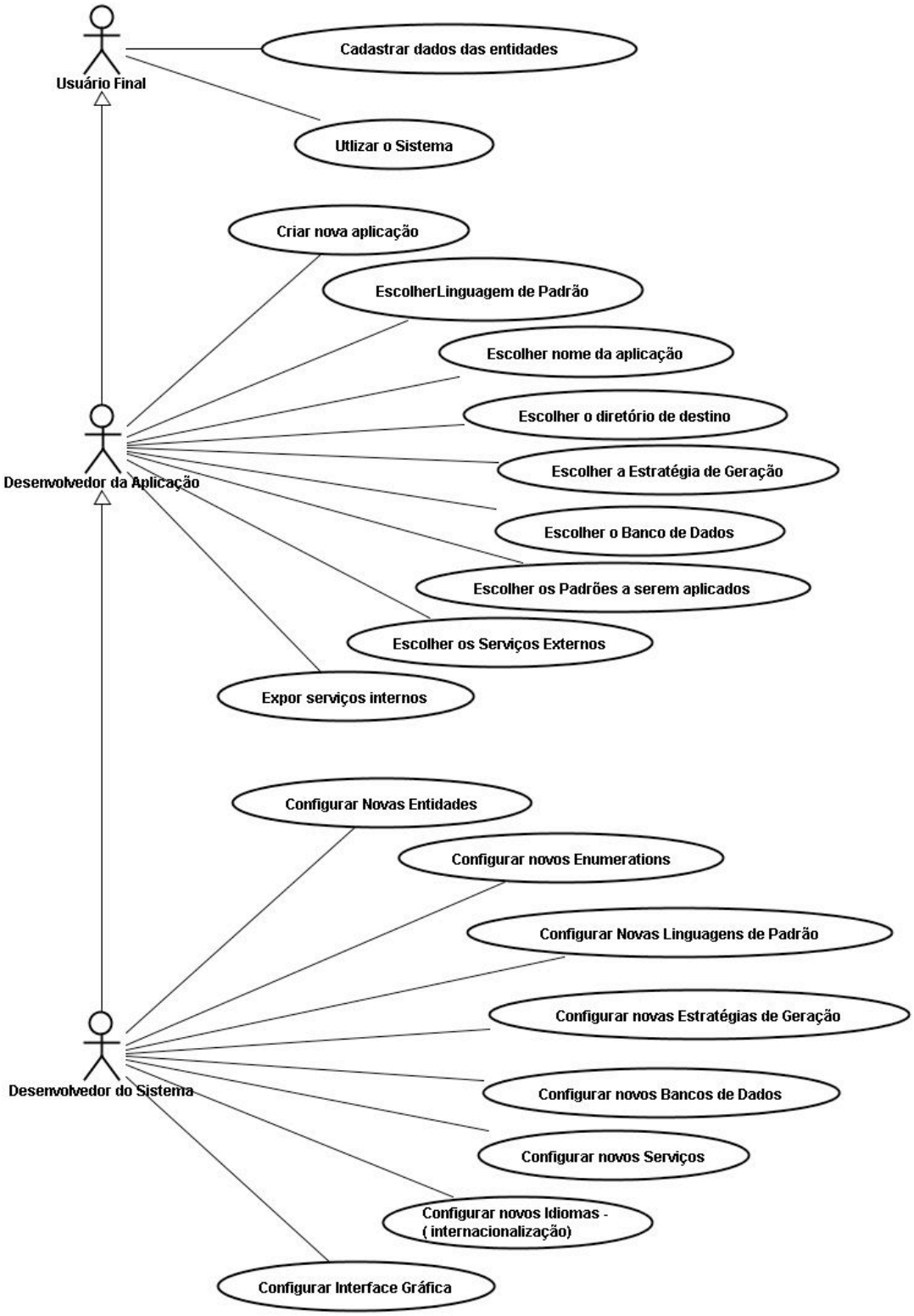


Figura 6.12 - Diagrama de Casos de Uso dos usuários do GAwCRe atual.

6.7. Testes realizados no GAwCRe

Com a finalidade de verificar as funções do gerador GAwCRe após as modificações realizadas, conforme aqui descritas, foram realizados testes utilizando a ferramenta SoapUI[47].

A SoapUI é uma ferramenta *open source* escrita em Java, que utiliza e testa serviços Web e que facilita todo o processo de criação e depuração dos testes, por meio de uma interface gráfica visual e intuitiva. O desenvolvedor fornece um WSDL existente e a partir dele a ferramenta carrega todas as operações expostas pelo serviço. Para cada operação realizada pelo serviço a ferramenta cria requisições contendo o pacote SOAP em que os dados e parâmetros dos serviços são preenchidos pelo desenvolvedor.

A partir das requisições são criados os casos de teste para os quais o desenvolvedor fornece a requisição e a resposta para essa requisição. A ferramenta depura o código e confirma se o serviço está executando corretamente a operação.

Os serviços Web internos implementados no GAwCRe são: *create*, *delete*, *update*, *listAll* e *findById*. Para esses serviços os testes foram realizados em duas etapas. Na primeira, casos de teste de unidade foram feitos e na segunda, realizou-se um teste com todos os serviços Web da primeira etapa em uma sequência definida.

O objetivo desses testes é verificar a validade dos dados cadastrados pelos serviços Web internos. Os itens em teste são os serviços Web internos de validação e as classes envolvidas com a entidade testada em cada caso de teste (cidade, médico, paciente, etc). O procedimento a ser realizado utiliza a ferramenta SoapUI para carregar o WSDL, criar as requisições, definir as entradas e saídas. A ferramenta faz a depuração e retorna falhas ou acertos. Caso ocorram falhas o SoapUI as detalha e diz onde elas foram detectadas.

A Tabela 6.1 ilustra os casos de teste criados para verificar os serviços Web internos: *create*, *findById*, *listAll*, *delete* e *update*.

[47] http://www.soapui.org

O termo entidade diz respeito a uma classe que armazena as propriedades de um objeto persistente e o termo ocorrência refere-se a um cadastro feito para uma entidade.

Tabela 6.1 - Caso de teste para os serviços Web internos do GAwCRe.

Caso de Teste para o serviço *create*	
Nome	Verificar dados cadastrados.
Descrição	Verifica a validade dos dados cadastrados.
Objetivo	Validar o serviço Web *create.*
Itens de Teste	O serviço *create* e as classes envolvidas com a entidade testada. Ex.: Classes envolvidas com criação da entidade cidade.
Procedimento	Fornecer à ferramenta especificações de entrada para que uma ocorrência seja criada. Verificar se todos os campos estão corretos. Verificar se a ocorrência foi criada com sucesso.
Caso de Teste para o serviço *findById*	
Nome	Buscar dados cadastrados.
Descrição	Verifica a busca dos dados cadastrados.
Objetivo	Buscar um objeto a partir do seu Id para validar o serviço Web *findById.*
Itens de Teste	O serviço *findById* e as classes envolvidas com a entidade testada. Ex.: Classes envolvidas com busca de uma ocorrência existente na entidade cidade.
Procedimento	Fornecer à ferramenta especificações de entrada (Id da ocorrência) para que determinada ocorrência seja procurada. Em caso de sucesso, o serviço deve retornar a ocorrência existente na classe (cidade, por exemplo) com o Id fornecido.
Caso de Teste para o serviço *listAll*	
Nome	Listar ocorrências cadastradas.
Descrição	Lista todas as ocorrências cadastradas para uma entidade

Objetivo	Validar o serviço Web *listAll.*
Itens de Teste	O serviço *listAll* e as classes envolvidas com a entidade testada. Ex.: Classes envolvidas com a listagem de todas as ocorrências cadastradas para a entidade cidade.
Procedimento	Fornecer à ferramenta especificações de entrada para que todas as ocorrências de determinada entidade sejam listadas. Verificar se todas as ocorrências cadastradas para aquela entidade foram listadas.
Caso de Teste para o serviço *delete*	
Nome	Deletar dados cadastrados.
Descrição	Verifica a exclusão de dados cadastrados.
Objetivo	Deletar ocorrências cadastradas e validar o serviço Web *delete*.
Itens de Teste	O serviço *delete* e as classes envolvidas com a entidade testada. Ex.: Classes envolvidas com a exclusão de uma ocorrência cadastrada para a entidade cidade.
Procedimento	Fornecer à ferramenta especificações de entrada (Id da ocorrência) para que determinada ocorrência seja deletada. Em seguida realizar um *findById* para verificar se a ocorrência foi de fato deletada.
Caso de Teste para o serviço *update*	
Nome	Alterar dados cadastrados.
Descrição	Verifica a alteração de dados cadastrados.
Objetivo	Validar o serviço Web *update* alterando dados cadastrados.
Itens de Teste	O serviço *update* e as classes envolvidas com a entidade testada. Ex.: Classes envolvidas com alteração de ocorrências da entidade cidade.
Procedimento	Fornecer à ferramenta especificações de entrada como, por exemplo, o Id da ocorrência, e também as alterações a serem efetuadas em determinada ocorrência. O serviço deve fazer o *update*. Em seguida, realizar um *findById* para verificar se a alteração na ocorrência de fato ocorreu.

Esses testes unitários foram realizados para cada entidade do gerador GAwCRe (Etapa 1 - casos de teste de unidade). Ao final dessa etapa outro teste foi realizado na seguinte sequência: o *create*, *findById*, *listAll*, *delete*, *finById*, *update* e *findById*. Ou seja, inicialmente ocorrências são cadastradas, em seguida determinada ocorrência é procurada pelo seu Id; são listadas as ocorrências encontradas para a entidade em teste; uma dessas ocorrências é deletada verifica-se se a exclusão foi realizada com sucesso; e por fim, uma ocorrência é alterada. Outra verificação é realizada para comprovar a alteração. Para esses casos de teste não foram identificados erros nas entidades do gerador.

6.8. Considerações Finais

Este capítulo apresentou o processo de inserção de serviços Web no gerador de aplicações GAwCRe, bem como a manutenção evolutiva realizada nas funções internas desse gerador para prover um serviço Web.

Foram encontradas diversas dificuldades para a inserção dos serviços Web, dentre elas destaca-se a falta de serviços eletrônicos, disponíveis na Internet, que sejam gratuitos e que tenham sido construídos utilizando a tecnologia de serviços Web (*Web Services*). Essa tecnologia tem se tornado o padrão pelo W3C para construção de uma SOA.

Diante da dificuldade em encontrar serviços Web disponíveis na Internet, um protótipo com serviços Web externos foi construído e utilizado para exemplificar a viabilidade da abordagem proposta.

Não foi possível implementar no GAwCRe, em tempo hábil, uma solução que acoplasse todos os tipos de serviços eletrônicos disponíveis atualmente na Internet aumentando assim, as possibilidades de ampliação do domínio desse gerador.

Após a construção e uso dos serviços Web externos no GAwCRe, foi possível perceber que a utilização desses serviços providos por uma SOA em geradores de aplicação pode trazer ganhos para o desenvolvedor e para o usuário da aplicação, mas esse uso está restrito as condições tecnológicas vigentes. Como a tendência para construção de serviços eletrônicos tem sido o padrão de serviço Web, é possível que, no futuro, haja mais serviços disponíveis

para serem acoplados ao GawCRe e à outros sistemas que ofereçam apoio a essa tecnologia. Consequentemente, as possibilidades de ampliação do domínio desses sistemas serão maiores.

Outra solução para o uso de serviços externos em geradores de aplicações pode ser obtida construindo aplicações intermediárias entre o gerador e o serviço a ser acoplado. Essa solução também demanda esforço, conhecimento dos protocolos utilizados em ambos os lados e, sendo assim, exige um estudo prévio de viabilidade.

Capítulo 7

Conclusão

7.1.Visão Geral

Geradores de aplicação quando modificados para suportar serviços Web podem ser tornar mais flexíveis, além de ter a sua manutenção facilitada, possibilitar a ampliação do domínio e serem independentes da linguagem de programação e plataforma utilizadas na sua criação.

O uso de serviços Web providos por SOA em GAs permite a inserção de novas funções nesses geradores, mas preserva as funções existentes, o que possibilita a melhoria de sua qualidade global e faz com que suas aplicações satisfaçam às necessidades de negócios vigentes.

A união de geradores de aplicação e arquitetura orientada a serviços irá resultar em aplicações com o código-fonte padronizado, com a redução de erros inseridos na fase de codificação, com baixo acoplamento e alta coesão dos dados e que podem ser utilizados tanto por usuários que possuem afinidade com as tecnologias envolvidas, como por usuários menos experientes. Os usuários com menos experiência, entretanto, precisam conhecer o domínio do gerador de aplicação para melhor utilizá-lo. Para os mais experientes, além do conhecimento do domínio, é preciso também conhecer os serviços candidatos a serem inseridos no sistema e suas particularidades, uma vez que terão que gerá-lo.

A redução de erros inseridos na fase de codificação é consequência da tecnologia utilizada. O GA fornece ao usuário os passos a serem seguidos e quais são as informações necessárias tornando o processo mais automatizado e guiado.

O trabalho realizado permitiu a ampliação do domínio de geradores utilizando SOA. Ao utilizá-la, vantagens foram proporcionadas aos sistemas, tais como o dinamismo das aplicações ou a adaptabilidade das mesmas. Porém, não é garantido que haja sempre serviços Web disponíveis que se adequem às necessidades dos clientes. Quando isso acontecer, o desenvolvedor deverá desenvolver um serviço Web que atenda aos requisitos exigidos, analisando o custo/benefício do desenvolvimento. É preciso analisar a viabilidade da utilização de serviços Web como forma de ampliação do domínio, principalmente se o gerador exigir trabalhos adicionais, como foi o caso do GAwCRe que precisou de um processo de manutenção evolutiva, para em seguida, possibilitar o uso de serviços como recurso para a expansão.

7.2. Contribuições do Trabalho

- O uso de serviços providos por uma arquitetura orientada a serviços na construção de geradores de aplicação ou na ampliação do domínio desses tornaram essas ferramentas mais flexíveis, com fácil manutenção e se adequando rapidamente às mudanças de requisitos solicitadas pelos clientes, pois, para inserir uma nova funcionalidade na aplicação e expandir o seu domínio, o engenheiro de software deve encontrar (em um repositório de serviços) um serviço Web que atenda aos requisitos desejados e acoplá-lo ao sistema. Para realizar uma manutenção, a alteração é feita no serviço Web pelo fornecedor desse serviço e disponibilizada ao cliente em tempo de execução. Na próxima vez que o usuário utilizar esse serviço na aplicação, a manutenção já estará disponível;
- A união de geradores de aplicação com SOA (por meio dos serviços Web) permite a construção de aplicações dinâmicas e que podem ser utilizadas em qualquer plataforma ou linguagem de programação. O código-fonte dessas aplicações possui um padrão de codificação facilitando assim manutenções futuras, além de possuir a quantidade de erros, inseridos na fase de codificação, reduzida em virtude do uso de um gerador;
- Com a manutenção corretiva e evolutiva, que resultou na construção de uma arquitetura mais genérica, o GAwCRe passou a ser mais genérico. Atualmente o gerador aceita outras linguagem de padrões que não apenas a SiGCli e outros banco de dados, além do MySQL. Até mesmo estratégias de geração podem ser disponibilizadas, mas para isso é preciso que o desenvolvedor responsável pelo gerador configure o modelo de metadados

construído com a manutenção evolutiva e corretiva, inserido as informações necessárias para a instanciação de novas aplicações. Assim, o gerador poderá instanciar aplicações de domínios conexos ao seu, com o banco de dados relacional que o cliente tiver em sua empresa, com a estratégia não apenas Web como também Desktop e com a linguagem de programação Java ou outra a escolha do cliente;

- A manutenção realizada também eliminou os trechos de código construídos de forma inadequada, aumentou a coesão dos dados e diminuiu o acoplamento das funções, além de possibilitar reúso e melhorar a documentação, devido ao uso de padrões de projeto como o MVC;
- A interface gráfica, tanto do gerador quanto da aplicação gerada, também passou por manutenções que as tornaram mais intuitiva e com maior usabilidade. Outro resultado obtido no processo de manutenção está relacionado à requisitos não-funcionais que vão além da usabilidade tais como: manutenibilidade, configurabilidade e reusabilidade. Na nova versão do GA tais requisitos também puderam ser tratados. Neste projeto, a engenheira de software pôde comprovar melhoras com relação à esses requisitos, pois qualquer modificação no código ou interface foi facilitada. Além disso, foi possível o reuso de código em mudanças realizadas. No entanto, ainda é preciso uma investigação mais detalhada com relação à esses requisitos e o novo gerador.

Espera-se que as manutenções futuras que ocorrerem no GAwCRe possam ser mais fáceis de ser realizadas e com menor esforço que as aqui descritas, devido à arquitetura atual obtida e o uso de técnicas e tecnologias aqui citadas.

7.3. Limitações do Trabalho

A seguir são listadas as limitações existentes na arquitetura construída:

- A arquitetura implementada no GAwCRe não apoia o uso de serviços eletrônicos providos por uma SOA que não estejam no formato de serviço Web;
- A estrutura de metadados construída não permite o uso de determinados tipos de dados básicos como aqueles que utilizam imagens ou arquivos;

- A arquitetura desenvolvida não provê, atualmente, funcionalidade que permita o uso de relatórios ou de regras complexas de negócios. Apenas as regras já previstas no GAwCRe.

7.4. Trabalhos Futuros

A seguir são listadas algumas possibilidades de trabalhos que podem dar continuidade ao realizado nesta dissertação:

- Configurar outras linguagens de padrões nos metadados dos *templates*, para verificar a consistência e a abrangência da arquitetura elaborada;
- Configurar outras estratégias de geração (Desktop, por exemplo) nos metadados dos *templates*, para verificar a consistência e a abrangência da arquitetura elaborada;
- Avaliar a viabilidade de inserir outros serviços eletrônicos externos que estejam disponíveis e que atendam aos requisitos solicitados;
- Modificar e evoluir o código-fonte para que receba serviços eletrônicos construídos com outra estrutura de implementação para SOA (Jini[48], JavaSpaces[49], CORBA[50], DCOM[51], etc.), que não apenas serviços Web (Web Services). Essa evolução pode ser feita utilizando *proxys*, em que um servidor atende as requisições repassando os dados a outros servidores;
- Alterar o gerador para permitir a inclusão de funções que hoje não são contempladas como cálculos, relatórios ou funções com regras de negócio mais complexas;
- Com a criação de uma plataforma independente de linguagem e preparada para adição de novas funções, também podem ser adicionados novos tipos de dados como arquivos e imagens;
- Validar o uso de outras Linguagens de Programação e realizar novos testes com os serviços Web, tais como: de estresse, escalabilidade, entre outros;

[48] http://www.jini.org
[49] http://java.sun.com
[50] http://www.omg.org
[51] http://www.microsoft.com/com

Referências Bibliográficas

ALONSO, G.; CASATI, F.; KONU, H.; MACHIRAJU, V. (2004). Web *Services – Concepts, Architecture e Applications*. Bookmaker Springer.

ANDREWS, *et al*., (2003). *BPEL4WS*: *Business Process Execution Language for Web Services Version 1.1*, IBM – Specifications, May-2003. Disponível em: http://msdn.microsoft.com/en-us/library/ee251594(v=bts.10).aspx. Acesso em: Fevereiro de 2009.

ANSI/IEEE 1471-2000 (2000). *Recommended Practice for Architecture Description of Software-Intensive Systems*. Disponível em: http://ieeexplore.ieee.org/servlet/opac?punumber=7040. Acesso em: Fevereiro de 2009.

ARANHA, E.; BORBA, P. (2002). *Testes e Geração de Código de Sistemas Web.* In: XVI Simpósio Brasileiro de Engenharia de Software - SBES 2002, p. 114-129, Gramado/RS.

BABAR, M.A.; GORTON, I.I.; JEFFERY, R. (2005). *Capturing and using software architecture knowledge for architecture-based software development*. In: QSIC'05: Proceedings of the Fifth International Conference on Quality Software, Washington, DC, USA: IEEE Computer Society, p.169 – 176.

BARAGRY, J.; REED, K. (1998). *Why is it so hard to define software architecture?* In: APSEC'98: Proceedings of the Fifth Asia Pacific Software Engineering Conference, Washington, DC, USA: IEEE Computer Society, p.28-36.

BARESI, L.; NITTO, E.D.; GHEZZI, C. (2006). *Towards open-world software: Issue and challenges*. In: SEW'06: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, Washington, DC, USA: IEEE Computer Society, p.249-252.

BASS, L.; CLEMENTS, P.; KAZMAN, R. (2003). *Software Architectures in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

BIANCO, P.; KOTERMANSKI, R.; MERSON, P. (2007). *Evaluating a Service-Oriented Architecture*. Technical Report - CMU/SEI-2007-TR-015, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

BOHLEN, M,; BRANDNON, C.; ZOONS, W. (2006). *AndroMDA*. 2006. Disponível em: http://mysql.com. Acesso em: Fevereiro de 2009.

BORGES, S. S. (2008). *Apoio de Gerência de Configuração de Software ao ARA-GAwCRe*. Dissertação de Mestrado apresentada ao PPG-CC, DC-UFSCar, São Carlos – SP.

BORGES, S. S., FERREIRA, T. T., PENTEADO, R. A. D. (2008). *Manutenção e Evolução de um Gerador de Aplicações Desenvolvido com Linha de Produtos de Software*. In: V Workshop de Manutenção de Software Moderna - WMSWM, Florianópolis – SC.

BRAGA, R. T. V. (2003). *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico*. Tese de Doutorado, ICMC/USP, São Carlos – SP.

BRAGA, R. T. V.; GERMANO, F. S. R.; e MASIERO, P. C. (1999). *A Pattern Language for Business Resource Management*. In: Proceedings of the 6th Annual Conference on Pattern Languages of Programs (PLoP'99), p. 1-33.

BRAGA, R. T. V.; GERMANO, F.S.R.; MASIERO, P.C. (1998). *A Family of Patterns for Business Resource Management.* In: Proceedings of the 5th Annual Conference on Pattern Languages of Programs (PLoP'98), Washington University in ST. Louis – Missouri,USA, p. 4.

BRAGA, R.T.V.; GERMANO.F.S.R.; MASIERO, P.C.; MALDONADO, J.C. (2001). *Introdução aos Padrões de Software*. Notas Didáticas. Universidade de São Paulo, USP-São Carlos.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. (2007). *Past, Present, and Future Trends in Software Patterns Software.* IEEE, 24, p. 31-37.

CAGNIN, M. I. (2005). *PARFAIT: Uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks*. Tese de Doutorado, ICMC/USP.

CAPILLA, R.; TOPALOGLU, N. (2005). *Product lines for supporting the composition and evolution of service oriented applications.* In: Proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE'05), Washington, DC, USA: IEEE Computer Society, p. 53-56.

CERAMI, E. (2002). *Web Services Essential, Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. O'Reilly Media, 304 pages, Chapter 6.

CHAUDET, C.; GREENWOOD, R.M.; OQUENDO, F.; WARBOYS, B. (2000). *Architecture-driven software engineering: Specifying, generating, and evolving component – based software systems*. In: IEEE Proceedings - Software, v. 147, n.6, p. 203-214.

CHEN, F.; LI, S.; CHU, W.C.C. (2005). *Feature analysis for service-oriented reengineering*. In: APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference, Washington, DC, USA: IEEE Computer Society, p. 201-208.

COPLIEN, J. O.; HARRISON, N. B. (2004). *Organizational Patterns of Agile Software Development.* Prentice-Hall, Inc, 401 pages.

CZARNECKI, K.; EISENECKER, U.W. (2002). *Generative programming - Methods, Tools, and Applications.* Addison-Wesley Publishing Co, 864 pages.

ERL, T. (2005). *Service-oriented architecture: Concepts, technology and design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 792 pages.

FANTINATO, M.; TOLEDO, M.; GIMENES, I; (2005). *Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços*. Relatório técnico, UNICAMP.

FERREIRA, T. T. (2008). *Avaliação de um Processo de Manutenção Usando Gerador de Aplicações*. Relatório Final. PIBIC/CNPq-UFSCar, 5 páginas.

FRANCA, L.P.A. (2000). *Um Processo para a Construção de Geradores de Artefatos.* Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro – RJ.

FREEMAN, E.; FREEMAN, E.; SIERRA, K. e BATES, B. (2004). *Head First Design Patterns.* O'Reilly Media, 678 pages.

FREEMARKER. (2009). *Freemarker*. Disponível em: http://freemarker.org. Acesso em: Abril de 2009.

FREITAS, R.G. (2006). *Utilização de Geradores de Aplicação em Processos Ágeis de Reengenharia.* Dissertação de Mestrado apresentada ao PPG-CC, DC-UFSCar, São Carlos – SP.

FURTADO, A.; SILVA, A.; NASCIMENTO, C.; SANTOS, G.; SANTOS, A.L.; FERRAZ, C.A.G. (2005). *HWSProxyGen: um Gerador de Proxies de Web Services para a Linguagem Haskell*. In: IX Simpósio Brasileiro de Linguagens de Programação, Recife – PE, p. 1-14.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented software.* Addison-Wesley Publishing Co, 431 pages.

GOMAA, H.; SALEH, M. (2005). *Software product line engineering for Web services and UML Computer Systems and Applications*. In: 3rd ACS/IEEE International Conference on, p. 110-vii.

HAVEY, M. (2006). *BPEL SOA and Web Services For Java*. Disponível em: http://wldj.sys-con.com/read/169346.htm. Acesso em: Janeiro de 2009.

HERRINGTON, J. (2003). *Code Generation in Action*. Manning Publications, 368 pages.

IBM Corporate. *BPEL Business Process Execution Language* (2003). Disponível em: http://www.ibm.com/developerworks/websphere/downloads/bpel/bpel.html. Acesso em: Março de 2009.

IBM Corporate. *SOA Development Survival* Guide (2009). Disponível em: http://www.ibm.com/br/services/mdl/soa.phtml. Acesso em: Março de 2009.

ISSARNY, V.; BANATRE, J. (2001). *Architectural-based exception handling*. In: HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34) Volume 9, Washington, DS, USA: IEEE Computer Society, 9058 pages.

JCP. (2009). *Java Community Process*. Disponível em: http://jcp.org/en/home/index. Acesso em: Janeiro de 2009.

JONES, C. (2008). *Applied Software Measurement*. 3ª Edição, McGraw-Hill, p. 471-475.

KRAFZIG, D.; BANKE, K.; SLAMA, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 1 edition, 408 pages.

LEE, J.; MUTHIG, D. (2006). *Feature-oriented variability management in product line engineering.* In: *Commun*, ACM, New York, USA, v. 49, p. 55-59.

LEE, S. P.; CHAN, L. P. & LEE, E. W. (2006). *Web Services Implementation Methodology for SOA Application.* In: IEEE International Conference on Industrial Informatics, p. 335-340.

LEWIS, G. A.; MORRIS, E.; SIMANTA, S.; WRAGE, L. (2007). *Common Misconceptions about Service-Oriented Architecture*. Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07. In: 6th International IEEE Conference on Commercial-off-the-Shelf, Washington, DS, USA: IEEE Computer Society, p.123-130.

LEYMANN, F., ROLLER, D., SCHMIDT, M.-T.(2002). *Web services and business process management*. In: IBM Systems Journal, v. 41, no.2, p. 198-211.

MACHADO, J.C. (2004). *Um estudo sobre o Desenvolvimento Orientado a Serviços*. Dissertação de Mestrado. Departamento de Informática - PUC-Rio.

MAHMOUD, Q. H. (2005). *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*. Disponível em: http://java.sun.com/developer/technicalArticles/WebServices/soa. Acesso em: Janeiro de 2009.

MICROSOFT Corporation. *Learn About Service Oriented Architecture (SOA)*. (2009). Disponível em: http://www.microsoft.com/biztalk/solutions/soa/overview.mspx#E5. Acesso em: Fevereiro de 2009.

MILER, N. J.; WERNER, C. M.; BRAGA, R. M. (2007). *O uso de Modelos de Features na Engenharia de Aplicações*. SBCARS 2007: Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software.

NOVAIS R.L.; (2007). *Coordenação de workflows em ambientes com suporte a dispositivos móveis*. Tese de Doutorado. Pontifícia Universidade Católica do Rio de Janeiro.

OASIS. (2006). *Modelo de Referência para Arquitetura Orientada a Serviço 1.0.* Comitê de Especificação 1, 2006. Disponível em: http://www.pcs.usp.br/~pcs5002/oasis/soarm-csbr.pdf. Acessado em: Dezembro de 2008.

ORACLE. (2009). *Oracle*. Disponível em http://www.oracle.com/global/br/index.html. Acesso em: Março de 2009.

PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F.; KRÄMER, B. J. (2006). *Service-Oriented Computing: A Research Roadmap Service Oriented Computing (SOC)*. In: Int\'l J. Cooperative Information Systems, vol. 17, nº. 2, pp.223 -255.

PAPAZOGLOU, M. P.; van den HEUVEL, W; (2007). *Service oriented architectures: approaches, technologies and research issue*s. In: VLDB Journal The International Journal on Very Large Data Bases, vol. 16, nº 3, p. 389-415.

PAPAZOGLOU, M.P. (2003). *Service-oriented computing: concepts, characteristics and directions.* Conference on Web Information Systems Engineering, 2003. In: WISE 2003 - Proceedings of the Fourth International, p. 3-12.

PAPAZOGLOU, M.P. e GEORGAKOPOULOS, D. (2003). *Service-oriented Computing.* Guest Editions. In: Communications of the ACM, vol. 46, nº 10, 392 pages.

PAZIN, A. (2004). *GAwCRe*: *Um Gerador de Aplicações baseadas na Web para o Domínio de Gestão de Clínicas de Reabilitação*. Dissertação de Mestrado. Departamento de Computação - Universidade Federal de São Carlos – UFSCar.

PELTZ, C. (2003). *Web Services Orchestration and Choreography*. Hewlett-Packard Company, vol. 36, Issue: 10, p. 46-52.

PRESSMAN, R. S. (2006). *Engenharia de Software.* 6ª Edição, McGraw-Hill.

RAUSCH, A. (2001). *A proposal for a code generator based on XML and code templates*. In: Proceedings of the Workshop of Generative Techniques for Product Lines, 23rd International Conference on Software Engineering.

RIZZO, J. A. M. (2005). *Validação de Processos Ágeis de Reengenharia e de Geradores de Aplicações*. Relatório de Iniciação Científica. PIBIC/CNPq-UFSCar.

RODRIGUES, K. R. H., FERNANDES, D. Z, PENTEADO, R.A.D (2009). *Aplicação de manutenção corretiva e evolutiva em um Gerador de Aplicações.* Artigo submetido ao VI Workshop de Manutenção de Software Moderna, Ouro Preto – MG.

ROSS-TALBOT. S; (2005). *Orchestration and Choreography: Standards, Tools and Technologies for Distributed Workflow*. In: Workshop Network Tools and Applications in Biology (NETTLAB ',05).

SCHMIDT, R. (2003) *Web Services Based Architectures to Support Dynamic Inter-organizational Business Processes*. In: The International Conference on Web Services – Europe, ICWS-Europe'03, vol. 2853, p.123-136.

SHIMABUKURO, M. H. J. (2006). *Captor: Um gerador de aplicações configurável*. Dissertação de Mestrado. ICMC/USP – São Carlos.

SIEGEL, J. (2000). *Corba 3 fundamentals and programming*. Wiley Computer Publishing, 2nd edition, 928 pages.

SORDI, J.S.O.; MARINHO, B.L.; NAGY, M. (2006). *Benefícios da Arquitetura de Software Orientada a Serviços para as empresas: Análise da experiência do ABN AMRO Brasil*. Revista de Gestão da Tecnologia e Sistemas de Informação. In: Journal of Information Systems and Technology Management, vol. 3, nº. 1, p. 19-34. ISSN online: p. 1775-1807.

SUN Microsystems, Inc. (2009). Sun Developer Network (SDN) - *Javadoc Tool Home Page*, 2008. Disponível em: http://java.sun.com/j2se/javadoc/. Acesso em: Março de 2009.

THAI, T.L. (1999). *Learning DCOM.* O'Reilly Media, 504 pages, CA, USA.

van GURP, J.; SAVOLAINEN, J. (2006). *Service grid variability realization.* In: SPLC '06: Proceedings of the 10th International on the Software Product Line Conference, Washington, DS, USA: IEEE Computer Society, p.85-94.

W3C – *World Wide Web Consortium*. (2008). Disponível em: http://www.w3c.org. Acesso em: Abril de 2009.

WEISS, D. M.; LAI, C. T. R. (2004). *Software product-line engineering: a family-based software development process.* Addison-Wesley Longman Publishing Co., Inc., 448 pages, Boston, MA, USA.

YE, E.; MOON, M.; KIM, Y.; YEOM, K. (2007). *An Approach to Designing Service-Oriented Product-Line Architecture for Business*. In: 9th International Conference on Advanced Communication Technology, vol. 2, p. 999-1002, Gangwon-Do.

YING, S.; LIANG, Z.; WANG, J.; WANG, F. (2006). *A Reflection Mechanism for Reusing Software Architecture*. In: QSIC '06: Proceedings of the Sixth International Conference on Quality Software, IEEE Computer Society, p. 235-243, Beijing.

YODER. J.W.; JOHNSON. R. (2002). *The adaptive object model architectural style*. In: Proceeding of the Proceeding of The Working IEEE/IFIP Conference on Software Architecture 2002 (WICSA3 '02), p. 3-27, Montreal, Canada.

ZAUPA, F.; GIMENES, I. M. S.; COWAN, D.; ALENCA, P.; LUCENA, C. (2007). *Um Processo de Desenvolvimento de Aplicações Web baseado em Serviços*. In: SBCARS 2007: Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, Campinas - SP, Brasil.

ZHANG, Z.; YANG, H. (2004). *Incubating services in legacy systems for architectural migration.* In: 11th Asia-Pacific Software Engineering Conference, 2004, p. 196-203, Leicester, UK.

ZHANG, L. J. (2007). *SOA Solution Reference Architecture*. In: IEEE International Conference on Web Services - ICWS07, p.1420-1425.

Apêndice A

Classe abstrata *PropertyMetadataDecorator*: Serve de base para a implementação dos tipos de dados.

```
package br.ufscar.dc.gdms.gamp.metadata.property;
import br.ufscar.dc.gdms.gamp.metadata.entity.EntityMetadata;
import br.ufscar.dc.gdms.gamp.metadata.type.TypeMetadata;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamAsAttribute;
import com.thoughtworks.xstream.annotations.XStreamOmitField;

public abstract class PropertyMetadataDecorator implements PropertyMetadata {

	@XStreamAsAttribute
	@XStreamAlias("name")
	private String name;
	@XStreamOmitField
	private Class<? extends TypeMetadata> typeClass;
	@XStreamOmitField
	private TypeFamily typeFamily;
	@XStreamOmitField
	private EntityMetadata enclosingObject;
	private String uiName;
	private String uiDescription;
	public PropertyMetadataDecorator(String name, Class<? extends TypeMetadata> typeClass,
TypeFamily family) {
		this(name, typeClass, family, null);
	}

	public PropertyMetadataDecorator(String name, Class<? extends TypeMetadata> typeClass,
TypeFamily family, EntityMetadata enclosingObject) {
		setName(name);
		setType(typeClass);
		setTypeFamily(family);
		setEnclosingObject(enclosingObject);
		setUiName(name);
		setUiDescription(name);
	}

	public boolean isRequired() {
		return false;
	}

	public final String getName() {
		return name;
	}
```

```
public final void setName(String name) {
	this.name = name;
}

public final Class<? extends TypeMetadata> getType() {
	return typeClass;
}

public String getUiName() {
	return uiName;
}

public void setUiName(String uiName) {
	this.uiName = uiName;
}

public String getUiDescription() {
	return uiDescription;
}

public void setUiDescription(String uiDescription) {
	this.uiDescription = uiDescription;
}

public final void setType(Class<? extends TypeMetadata> typeClass) {
	this.typeClass = typeClass;
}

public final TypeFamily getTypeFamily() {
	return typeFamily;
}

public final void setTypeFamily(TypeFamily family) {
	this.typeFamily = family;
}

public final EntityMetadata getEnclosingObject() {
	return enclosingObject;
}

public final void setEnclosingObject(EntityMetadata enclosingObject) {
	this.enclosingObject = enclosingObject;
}

public final boolean isTemporal() {
	return typeFamily.equals(TypeFamily.TEMPORAL);
}

public final boolean isCollection() {
	return typeFamily.equals(TypeFamily.COLLECTION);
}

public final boolean isPrimitive() {
	return typeFamily.equals(TypeFamily.PRIMITIVE);
```

```
}

public final boolean isBoolean() {
	return typeFamily.equals(TypeFamily.BOOLEAN);
}

public final boolean isEntityReference() {
	return typeFamily.equals(TypeFamily.ENTITY_REFERENCE);
}

public final boolean isEnumerationReference() {
	return typeFamily.equals(TypeFamily.ENUMERATION_REFERENCE);
}

public final boolean isReference() {
	return isEntityReference() || isEnumerationReference();
}

public String getQualifiedName() {
	return enclosingObject.getQualifiedName() + '.' + name;
}

@Override
public int hashCode() {
	final int prime = 31;
	int result = 1;
	result = prime * result + ((name == null) ? 0 : name.hashCode());
	result = prime * result
			+ ((typeClass == null || typeClass.getName() == null) ? 0 :
			typeClass.getName().hashCode());
	result = prime * result
			+ ((typeFamily == null) ? 0 : typeFamily.hashCode());
	return result;
}

@Override
public boolean equals(Object obj) {
	if (this == obj)
		return true;
	if (obj == null)
		return false;
	if (getClass() != obj.getClass())
		return false;
	PropertyMetadataDecorator other = (PropertyMetadataDecorator) obj;
	if (name == null) {
		if (other.name != null)
			return false;
	} else if (!name.equals(other.name))
		return false;
	if (typeClass == null || typeClass.getName() == null) {
		if (other.typeClass != null || typeClass.getName() != null)
			return false;
	} else if (!typeClass.getName().equals(other.typeClass.getName()))
		return false;
	if (typeFamily == null) {
```

```
                        if (other.typeFamily != null)
                                return false;
                } else if (!typeFamily.equals(other.typeFamily))
                        return false;
                return true;
        }

        @Override
        public String toString() {
                return name;
        }

}
```

Classe *EntityMetadata:* Estende a classe de metadados de objeto (*ObjectMetadata*) que armazena informações como nome da aplicação a ser gerada.

```
package br.ufscar.dc.gdms.gamp.metadata.entity;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import br.ufscar.dc.gdms.gamp.metadata.object.ObjectMetadata;
import br.ufscar.dc.gdms.gamp.metadata.property.PropertyMetadata;
import br.ufscar.dc.gdms.gamp.validator.EntityValidator;

import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamImplicit;

@XStreamAlias("entity")
public class EntityMetadata extends ObjectMetadata {

        @XStreamImplicit
        private List<PropertyMetadata> properties;

        private List<EntityValidator> validators;

        @XStreamImplicit
        private List<UniqueConstraintMetadata> uniqueConstraints;

        public EntityMetadata(String entityName) {
                this("", entityName);
        }

        public EntityMetadata(String entityName, String containerName) {
                super(entityName, containerName);
                setProperties(Collections.<PropertyMetadata>emptyList());
                setUniqueConstraints(Collections.<UniqueConstraintMetadata>emptyList());
                setValidators(Collections.<EntityValidator>emptyList());
        }

        public void addProperty(PropertyMetadata metadata) {
                metadata.setEnclosingObject(this);
```

```
		this.properties.add(metadata);
	}

	public void setProperties(List<PropertyMetadata> properties) {
		this.properties = new ArrayList<PropertyMetadata>();
		this.properties.addAll(properties);
	}

	public Collection<PropertyMetadata> getProperties() {
		return Collections.unmodifiableList(properties);
	}

	public void addUniqueConstraint(UniqueConstraintMetadata metadata) {
		this.uniqueConstraints.add(metadata);
	}

	public void setUniqueConstraints(List<UniqueConstraintMetadata> properties) {
		this.uniqueConstraints = new ArrayList<UniqueConstraintMetadata>();
		this.uniqueConstraints.addAll(properties);
	}

	public Collection<UniqueConstraintMetadata> getUniqueConstraints() {
		return Collections.unmodifiableList(uniqueConstraints);
	}

	private void setValidators(List<EntityValidator> emptyList) {
		this.validators = new ArrayList<EntityValidator>();
		this.validators.addAll(validators);
	}

	public void addValidator(EntityValidator validator) {
		this.validators.add(validator);
	}

	public List<EntityValidator> getValidators() {
		return Collections.unmodifiableList(validators);
	}

	public PropertyMetadata getProperty(String propertyName) {
		PropertyMetadata target = null;
		for (PropertyMetadata currentProperty : properties) {
			if (currentProperty.getName().equals(propertyName)) {
				target = currentProperty;
				break;
			}
		}
		return target;
}

}
```

Classe *PropertyMetadata*: Uma interface que estende a classe *Metadata* e é implementada pela classe *PropertyMetadataDecorator*.

```
package br.ufscar.dc.gdms.gamp.metadata.property;
import br.ufscar.dc.gdms.gamp.metadata.Metadata;
import br.ufscar.dc.gdms.gamp.metadata.entity.EntityMetadata;
import br.ufscar.dc.gdms.gamp.metadata.type.TypeMetadata;

public interface PropertyMetadata extends Metadata {

	public boolean isRequired();

	public String getName();

	public Class<? extends TypeMetadata> getType();

	public TypeFamily getTypeFamily();

	public EntityMetadata getEnclosingObject();

	public boolean isTemporal();

	public boolean isCollection();

	public boolean isPrimitive();

	public boolean isBoolean();

	public boolean isEntityReference();

	public boolean isEnumerationReference();

	public boolean isReference();

	public void setEnclosingObject(EntityMetadata enclosingObject);

	public void setUiName(String uiName);

	public String getUiName();

	public void setUiDescription(String uiDescription);

	public String getUiDescription();
}
```

Classe do Serviço de Obtenção de Dados

```
package br.ufscar.gdms.service.validation;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

@WebService(serviceName = "CidadeEchoService", portName = "CidadeEchoPort", targetNamespace =
"http://cidade.echo.service.gdms.ufscar.br/")
public class CidadeEcho {

	@WebMethod
	@RequestWrapper(className = "br.ufscar.gdms.service.validation.cidade.EchoRequest")
	@ResponseWrapper(className = "br.ufscar.gdms.service.validation.cidade.EchoResponse")
	public void echo(@WebParam(name = "cidade")String cidade, @WebParam(name = "estado")String
estado, @WebParam(name = "pais")String pais) throws ServiceValidationException {
		System.out.println(cidade + ", " + estado + ", " + pais);
	}

}
```

Classe do Serviço de Fornecimento de Dados

```
package br.ufscar.gdms.service.validation;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

@WebService(serviceName = "CidadeValidationService", portName = "CidadeValidationPort",
targetNamespace = "http://cidade.validation.service.gdms.ufscar.br/")
public class CidadeValidation {

	@WebMethod
	@RequestWrapper(className = "br.ufscar.gdms.service.validation.cidade.ValidateCidadeRequest")
	@ResponseWrapper(className = "br.ufscar.gdms.service.validation.cidade.ValidateResponse")
	public Boolean validate(@WebParam(name = "cidade")String cidade, @WebParam(name =
"estado")String estado, @WebParam(name = "pais")String pais) throws ServiceValidationException {
		Boolean valid = false;
		if (pais.equals("Brasil")) {
			valid = true;
		}
		return valid;
	}

}
```

Apêndice B

WSDL Serviço de Fornecimento de Dados

```
- <definitions targetNamespace="http://cidade.echo.service.gdms.ufscar.br/" name="CidadeEchoService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://validation.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeEchoService?xsd=1"/>
      </xsd:schema>
    - <xsd:schema>
        <xsd:import namespace="http://cidade.echo.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeEchoService?xsd=2"
      </xsd:schema>
    </types>
  - <message name="echo">
      <part name="parameters" element="tns:echo"/>
    </message>
  - <message name="echoResponse">
      <part name="parameters" element="tns:echoResponse"/>
    </message>
  - <message name="ServiceValidationException">
      <part name="fault" element="ns1:ServiceValidationException"/>
    </message>
  - <portType name="CidadeEcho">
    - <operation name="echo">
        <input message="tns:echo"/>
        <output message="tns:echoResponse"/>
        <fault message="tns:ServiceValidationException" name="ServiceValidationException"/>
      </operation>
    </portType>
  - <binding name="CidadeEchoPortBinding" type="tns:CidadeEcho">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="echo">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      - <fault name="ServiceValidationException">
          <soap:fault name="ServiceValidationException" use="literal"/>
        </fault>
      </operation>
    </binding>
  - <service name="CidadeEchoService">
    - <port name="CidadeEchoPort" binding="tns:CidadeEchoPortBinding">
        <soap:address location="http://200.18.98.181:8080/validation-poc/CidadeEchoService"/>
      </port>
    </service>
</definitions>
```

WSDL Serviço de Validação de Dados

```
- <definitions targetNamespace="http://cidade.data.service.gdms.ufscar.br/" name="CidadeDataService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://validation.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeDataService?xsd=1"/>
      </xsd:schema>
    - <xsd:schema>
        <xsd:import namespace="http://cidade.data.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeDataService?xsd=2"/>
      </xsd:schema>
    </types>
  - <message name="getLatitude">
      <part name="parameters" element="tns:getLatitude"/>
    </message>
  - <message name="getLatitudeResponse">
      <part name="parameters" element="tns:getLatitudeResponse"/>
    </message>
  - <message name="ServiceValidationException">
      <part name="fault" element="ns1:ServiceValidationException"/>
    </message>
  - <message name="getLongitude">
      <part name="parameters" element="tns:getLongitude"/>
    </message>
  - <message name="getLongitudeResponse">
      <part name="parameters" element="tns:getLongitudeResponse"/>
    </message>
  - <portType name="CidadeData">
    - <operation name="getLatitude">
        <input message="tns:getLatitude"/>
        <output message="tns:getLatitudeResponse"/>
        <fault message="tns:ServiceValidationException" name="ServiceValidationException"/>
      </operation>
    - <operation name="getLongitude">
        <input message="tns:getLongitude"/>
        <output message="tns:getLongitudeResponse"/>
        <fault message="tns:ServiceValidationException" name="ServiceValidationException"/>
      </operation>
    </portType>
  - <binding name="CidadeDataPortBinding" type="tns:CidadeData">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="getLatitude">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      - <fault name="ServiceValidationException">
          <soap:fault name="ServiceValidationException" use="literal"/>
        </fault>
      </operation>
    - <operation name="getLongitude">
      </binding>
    - <service name="CidadeDataService">
      - <port name="CidadeDataPort" binding="tns:CidadeDataPortBinding">
          <soap:address location="http://200.18.98.181:8080/validation-poc/CidadeDataService"/>
        </port>
      </service>
    </definitions>
```

WSDL Serviço de Obtenção de Dados

```
- <definitions targetNamespace="http://cidade.echo.service.gdms.ufscar.br/" name="CidadeEchoService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://validation.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeEchoService?xsd=1"/>
      </xsd:schema>
    - <xsd:schema>
        <xsd:import namespace="http://cidade.echo.service.gdms.ufscar.br/" schemaLocation="http://200.18.98.181:8080/validation-poc/CidadeEchoService?xsd=2"/>
      </xsd:schema>
    </types>
  - <message name="echo">
      <part name="parameters" element="tns:echo"/>
    </message>
  - <message name="echoResponse">
      <part name="parameters" element="tns:echoResponse"/>
    </message>
  - <message name="ServiceValidationException">
      <part name="fault" element="ns1:ServiceValidationException"/>
    </message>
  - <portType name="CidadeEcho">
    - <operation name="echo">
        <input message="tns:echo"/>
        <output message="tns:echoResponse"/>
        <fault message="tns:ServiceValidationException" name="ServiceValidationException"/>
      </operation>
    </portType>
  - <binding name="CidadeEchoPortBinding" type="tns:CidadeEcho">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="echo">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      - <fault name="ServiceValidationException">
          <soap:fault name="ServiceValidationException" use="literal"/>
        </fault>
      </operation>
    </binding>
  - <service name="CidadeEchoService">
    - <port name="CidadeEchoPort" binding="tns:CidadeEchoPortBinding">
        <soap:address location="http://200.18.98.181:8080/validation-poc/CidadeEchoService"/>
      </port>
    </service>
</definitions>
```