UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
ESCOLA DE INFORMÁTICA APLICADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Uso de ferramenta de gestão de regras de negócio em uma arquitetura orientada a serviços

Nome do autor:
Diego Alexandre Aranha Duarte

Nome do Orientador:
Leonardo Guerreiro Azevedo

Agosto/2010

Uso de ferramenta de gestão de regras de negócio em uma arquitetura orientada a serviços

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Nome do autor:
Diego Alexandre Aranha Duarte

Nome do Orientador:
Leonardo Guerreiro Azevedo

Uso de ferramenta de gestão de regras de negócio em uma arquitetura orientada a serviços

Aprovado em _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Leonardo Azevedo, D.Sc. (UNIRIO)

_____

Prof. Márcio Barros, D.Sc. (UNIRIO)

_____

Prof. Fernanda Baião, D.Sc. (UNIRIO)

Os autores deste Projeto autorizam a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, _____ de _____ de _____

_____

Diego Alexandre Aranha Duarte

AGRADECIMENTOS

À todos que possibilitaram a realização do trabalho e que sempre me apoiaram durante todos os momentos de minha vida desde os momentos mais tristes aos mais felizes.

Agradeço primeiramente a Deus pelo dom da minha vida e por sempre me guiar e renovar minhas esperanças mostrando sempre sua enorme graça e amor. A Ele, toda minha gratidão.

Agradeço aos meus pais e minha irmã pela minha formação como pessoa, por estarem sempre acreditando em mim. Durante toda minha vida sempre pude contar com vocês, sempre me dando apoio para que eu pudesse me desenvolver. Muito obrigado pelo amor, carinho e dedicação por tanto tempo!

Agradeço a uma pessoa que é muito especial para mim, a Jessica. Que já está há muito tempo comigo e que durante todo esse tempo me deu todo apoio, carinho e amor necessário para que eu pudesse seguir em frente. E me mostrou que caminhando juntos podemos ir muito mais longe. Muito obrigado por ser a pessoa que você é e por estar comigo.

Agradeço ao meu orientador, Leonardo Azevedo, por contribuir de forma significativa para a minha formação profissional e acadêmica, por toda dedicação e apoio na elaboração deste trabalho, sem os quais com certeza não conseguiria obter o mesmo resultado.

Agradeço também a todos os docentes da Unirio, que durante esses 4 anos ajudaram na minha formação, e que sempre estiveram dispostos a ajudar e buscar o melhor de mim.

Agradeço a todos que trabalharam comigo no NP2Tec, principalmente aqueles que estavam sempre comigo no laboratório da NP2Tec.

Agradeço aos meus colegas da Unirio, que estiveram comigo esse tempo todo, nos trabalho, nos estudos, nas conversas, nas brincadeiras e “zoações”, em especial ao Alexandre Souza, Brunno Silveira, Sergio Puntar e Thiago Yusuke.

Agradeço a todos os meus amigos, que me proporcionaram momentos muito felizes nesse tempo todo, aos amigos da igreja, do prédio e aqueles que fiz em Portugal e que até hoje sinto saudades.

Por fim, agradeço a todos que possa ter esquecido de citar aqui.

SUMÁRIO

Resumo 7
Capítulo 1: Introdução 8
1.1 Motivação 8
1.2 Objetivo 12
1.3 Estrutura do trabalho 13
Capítulo 2: Conceitos 14
2.1 Regras de Negócio 14
2.1.1 Definição de regras de negócio 14
2.1.2 Classificação das regras de negócio 15
2.1.3 Gestão de regras de negócio 18
2.2 BRMS 20
2.2.1 Definição 20
2.2.2 Arquitetura de um BRMS 21
2.2.3 Por que um BRMS? 23
2.2.4 Ferramentas BRMS 24
2.3 Arquitetura Orientada a Serviços (SOA) 26
2.3.1 Definição 26
2.3.2 Serviços 28
2.3.3 *Web Services* 30
2.3.4 ESB 31
2.4 Resumo 33
Capítulo 3: Análise das ferramentas 35
3.1 Processo de avaliação de ferramentas 36
3.2 Análise dos resultados 40
3.3 Resumo 41
Capítulo 4: WebSphere Ilog JRules BRMS 43
4.1 Conceitos da Ferramenta 43
4.1.1 Business Object Model (BOM) 43
4.1.2 Execution Object Model (XOM) 44
4.1.3 Ruleset e RuleApp 45
4.1.4 Rule Model 46
4.1.5 Template 47
4.1.6 RuleFlow 48
4.1.7 Modos de Execução 49
4.2 Arquitetura da ferramenta 51
4.2.1 Desenvolvimento de Aplicações de Regras de Negócio 53
4.2.2 Gestão e Autoria de Regras de Negócio pelo Usuário do Negócio 55
4.2.3 Integração, Monitoramento e Auditoria nas Aplicações da Organização 57
4.2.4 Validação de Regras de Negócio 59
4.3 Disponibização de regras como serviços 60
4.3.1 *Transparent Decision Services* 62
4.4 Resumo 67
Capítulo 5: Estudo de Caso 69
5.1 Levantamento de regras 70
5.2 Criação do modelo que representa os objetos 73

5.3 Desenvolvimento do projeto de regras 75
5.3.1 checkCadastro 79
5.3.2 calcularLimiteCredito 81
5.3.3 verificarLimiteCredito 82
5.3.4 comprometerLimiteCredito 82
5.3.5 calcularJuros 83
5.3.6 calcularValorTotal 84
5.4 Disponibilização dos projetos de regras como *Monitored Transparent Decision Service* 86
5.5 Publicação dos serviços no OSB 90
5.6 Criação da aplicação cliente 92
5.7 Execução da aplicação 95
5.8 Análise dos resultados do estudo de caso 98
5.9 Resumo 100
Capítulo 6: Conclusão 102
Capítulo 7: Referências Bibliográficas 105
Apêndice 1 Critérios Genéricos 109
Apêndice 2 Critérios específicos para gestão 112

LISTA DE FIGURAS

Figura 1 – Ciclo de vida de gestão de regras de negócio [Nelson *et al*., 2008] 20
Figura 2 – Componentes de um BRMS [Deitert e McCoy, 2007] 22
Figura 3 – Arquitetura simplificada de um ESB [Papazouglou *et al.*, 2007] 32
Figura 4 – Processo para avaliação de ferramentas [Azevedo *et al*., 2010b] 36
Figura 5 – Legenda para a Tabela 1 39
Figura 6 – Utilização dos objetos do BOM pela regra 44
Figura 7 – Execução de regras utilizando o XOM 45
Figura 8 – Exemplo de *template* 48
Figura 9 – Exemplo de um ruleflow 48
Figura 10 – Operação do modo de execução RetePlus 50
Figura 11 – Operação do Sequential 51
Figura 12 – Arquitetura do WebSphere Ilog JRules BRMS 52
Figura 13 - Decision Validation Service integrado ao BRMS 60
Figura 14 – Possibilidades de geração de código para integração das regras 62
Figura 15 – Arquitetura para um serviço de decisão transparente hospedado 64
Figura 16 – Arquitetura da Ilog JRules para os serviços de decisão transparentes monitorados 66
Figura 17 – Fluxo inicial do processo de Analisar pedido de crédito 72
Figura 18 – Classes que compõem o modelo de objetos de execução 73
Figura 19 – Projetos de regras criados 74
Figura 20 – BOM do Cliente 75
Figura 21 – Parâmetros do projeto checkCadastro 76
Figura 22 – Parâmetros do projeto calcularLimiteCredito 77
Figura 23 – Parâmetros do projeto verificarLimiteCredito 77
Figura 24 – Parâmetros do projeto comprometerLimiteCredito 78
Figura 25 – Parâmetros do projeto calcularJuros 78
Figura 26 – Parâmetro do projeto calcularValorTotal 79
Figura 27 – Regra checkClienteNovo 80
Figura 28 – Regra checkAtualização 80
Figura 29 – Fluxo de execução do projeto checkCadastro 81
Figura 30 – Regra calculaLimiteCredito 82
Figura 31 – Regra verificaLimiteCredito 82
Figura 32 – Regra comprometerLimite 83
Figura 33 – Tabela de decisão calcularJuros 84
Figura 34 – Regra calculaTaxas 85
Figura 35 – Regra calculaValorTaxas 85
Figura 36 – Regra calculaValorJuros 85
Figura 37 – Fluxo de execução calcularValorTotal 86
Figura 38 - RuleApps 87
Figura 39 – Entrada e saída dos métodos dos serviços 87
Figura 40 - Componentes criados a partir do *wizard* 88
Figura 41 - RuleApps publicados no Rule Execution Server 89
Figura 42 – WSDL do ServicoClienteWS 89
Figura 43 - Classe cliente do ServicoClienteWS 90
Figura 44 – Configuração do Business Service 91
Figura 45 - Arquitetura de execução dos *web services* utilizando o OSB 92

Figura 46 – Tabela de clientes no banco de dados 92
Figura 47 – Método executarServicos da classe Principal 93
Figura 48 - Tela principal da aplicação 94
Figura 49 - Tela de resposta da aplicação 94
Figura 50 – Mensagem de erro de cliente novo 95
Figura 51 - Mensagem de erro de cliente desatualizado 95
Figura 52 – Método de ação do botão “Enviar” 95
Figura 54 - Mensagem de erro de limite de crédito insuficiente 96
Figura 53 – Método sendProposta 96
Figura 55 - Classe ServicoLimiteCreditoClient 97
Figura 56 – Classe ServicoLimiteCreditoRunnerImplService 98
Figura 57 - Restrições e funcionalidades das implementações de *web services* 99

LISTA DE TABELAS

Tabela 1 – Resumo dos resultados obtidos com a avaliação das ferramentas de Azevedo et al. [2010a], considerando componente de gestão 39
Tabela 2 – Tipos suportados pelos serviços de decisão transparentes 64
Tabela 3 – Distribuição 109
Tabela 4 – Escalabilidade 109
Tabela 5 – Flexibilidade 109
Tabela 6 - Integração com outros sistemas 110
Tabela 7 - Plataforma tecnológica 110
Tabela 8 - Qualidade da documentação 110
Tabela 9 – Suporte 111
Tabela 10 – Edição de regras 112
Tabela 11 - Consulta às regras 114
Tabela 12 - Visualização de regras 115
Tabela 13 - Exportação / Importação 115
Tabela 14 - Repositório e versionamento de regras 116
Tabela 15 - Integração de regras 116
Tabela 16 - Validação de regras 117
Tabela 17 – Simulação de regras 117
Tabela 18 – Segurança 117
Tabela 19 – Qualidade de regras 118
Tabela 20 – Administração 118

LISTA DE ABREVIATURAS

BOM – *Business Object Model*

BRMS – *Business Rules Management System*

ESB – *Enterprise Service Bus*

OSB – *Oracle Service Bus*

SOA - *Service Oriented Architecture*

SOAP - *Simple Object Access Protocol*

TI - *Tecnologia da Informação*

UDDI - *Universal Description, Discovery and Integration*

WSDL - *Web Services Description Language*

XML - *(eXtensible Markup Language)*

XOM – *Execution Object Model*

XSD – *XML Schema Definition*

RESUMO

As regras de negócio representam um importante ativo intelectual para as organizações, e por isso devem ser geridas de forma eficiente. Para a gestão das regras de negócio é fundamental a utilização de ferramentas de apoio. Os BRMS são ferramentas que apóiam a gestão de regras de negócio, permitindo a externalização das regras que antes estavam embutidas nas aplicações. Por outro lado, é necessário que essas regras sejam integradas com as aplicações de forma simples. Uma das possibilidades de integração das regras de negócio, gerenciadas por um BRMS, é a disponibilização das regras como serviços em uma arquitetura orientada a serviços (SOA). Este trabalho apresenta e analisa o uso de ferramentas BRMS em uma abordagem SOA. Ferramentas BRMS existentes no mercado foram avaliadas e uma delas foi escolhida para analisar como seria a disponibilização de regras como serviços em uma arquitetura SOA. Como o Enterprise Service Bus (ESB) é apontado como principal infra-estrutura em SOA, também foi avaliado o uso desse componente como mediador das trocas de mensagens entre aplicação e os serviços que implementam as regras.

Capítulo 1: INTRODUÇÃO

Este capítulo apresenta a motivação para o trabalho, contextualiza o assunto abordado, bem como apresenta o objetivo do trabalho e a estrutura dos capítulos.

1.1 Motivação

Segundo [BRG, 2000], regra de negócio é uma declaração que define ou restringe algum aspecto de uma organização, sendo atômicas, de forma que não podem ser divididas. Tem como objetivo afirmar a estrutura de um negócio ou controlar ou influenciar o comportamento deste. Segundo Crerie [2009], as regras de negócio são importantes porque guiam o funcionamento de uma organização, tornando explícitos as regras e os conceitos envolvidos com as atividades realizadas, uniformizando a visão sobre o negócio realizado. Um exemplo de regra de negócio é: "Se o cliente possui saldo superior a 1000 e não tem empréstimos ativos então sua categoria é Premium".

Em geral, regras de negócio aparecem implementadas em aplicações ou mesmo presentes no conhecimento tácito das pessoas, não estando explícitas..O fato das regras não estarem explicitas não significa que a organização não possui regras; na verdade, ao contrário das regras explícitas que estão expostas, claramente definidas e documentadas, as regras que estão implícitas são difíceis de capturar, documentar, gerenciar e evoluir. As regras de negócio implícitas estão fortemente acopladas às aplicações e sistemas [Deitert e McCoy, 2007]. Considerando que as regras são voláteis e a cada dia novas regras de negócio são criadas, torna-se muito custosa a manutenção dessas regras. Por isso torna-se fundamental a externalização das regras de negócio, possibilitando sua gestão, documentação e evolução.

Segundo Halle [2002] e Graham [2007], a externalização das regras de negócio apresenta algumas vantagens como:

- Possibilidade da gestão de regras ser feita de forma eficiente, proporcionando uma rápida adaptação às mudanças nas regras de negócio;

- Aumento do grau de entendimento sobre as regras de negócio, facilitando o diagnóstico de problemas;
- Reutilização das regras por diferentes aplicações; e
- Diminuição do custo e esforço para a manutenção e criação de regras de negócio, sem necessidade de se gerar uma nova versão de um sistema.

Bajec e Krisper [2004] afirmam que pesquisadores e profissionais estão convencidos que as regras de negócio são muito sensíveis às mudanças do negócio e exigem um tratamento explícito durante o desenvolvimento do SI (sistema de informação) para assegurar sua agilidade. Caso contrário, muitos problemas podem ocorrer. Por exemplo:

- Uma vez que não foram adquiridas de forma sistemática e completa, regras de negócio não refletem as condições reais do ambiente de negócio. Conseqüentemente, os aplicativos desenvolvidos não atendem às necessidades do negócio;
- Há uma falta de documentação relativa às regras de negócio;
- As regras de negócio estão misturadas no código do programa. Não está claro que tipos de regras regem a aplicação, quando as regras são acionadas e como elas são implementadas;
- É difícil manter a lógica de negócio já que as regras são distribuídas em toda a lógica do aplicativo;
- As regras de negócio são de difícil controle, uma vez que não há nenhum objetivo comum e único para armazená-las.

Segundo Deitert e McCoy [2007], a gestão de regras de negócio é uma disciplina estruturada que guia a definição, categorização, governança e implantação das regras de negócio e é praticada durante o ciclo de vida de uma regra de negócio. Ela é a ciência de expor e gerir as políticas de uma organização de maneira explícita e geralmente apoiada por uma tecnologia. Com a percepção de que as regras de negócio não deveriam mais estar implícitas nas aplicações e sim externas a elas, de

maneira que se tornassem reutilizáveis, ferramentas denominadas BRMS (*Business Rule Management System*) foram criadas para gestão de regras.

A gestão de regras de negócio é uma tarefa de difícil execução, porém com planejamento, estudo e definição de ferramentas de apoio essa tarefa torna-se mais ágil. Além disso, surgem novas possibilidades para a organização, como a utilização e manutenção dessas regras de negócio no contexto de uma Arquitetura Orientada a Serviços (SOA), o que discutiremos a seguir.

Segundo Graham [2007], a gestão de regras de negócio pode ser feita de diversas maneiras, porém a mais econômica é com a utilização de um BRMS (*Business Rules Management Systems*), que pode ser definido como um conjunto de ferramentas para apoio à gestão de regras de negócio que permite a criação, registro, classificação, verificação, desenvolvimento e execução de regras.

Para Graham [2007] um BRMS deve ter as seguintes funcionalidades e responsabilidades:

- Armazenar as regras de negócio;
- Separar a lógica do negócio, ou seja, as regras das aplicações;
- Integrar com as aplicações da organização;
- Organizar as regras em *rulesets* (conjunto de regras) e realizar inferências sobre esses *rulesets;*
- Permitir que os analistas de negócio e até mesmo os usuários de negócio criem, mantenham e entendam as regras, com o mínimo de aprendizado necessário;
- Automatizar e facilitar os processos de negócio; e
- Possibilitar a criação de aplicações de regras ou projetos de regras, que interajam com os usuários, através de diálogos naturais, lógicos e compreensíveis.

Disponibilizando as regras de negócio separadas do código, as regras devem ser disponibilizadas em uma maneira que produza baixo acoplamento, permita a invocação por diferentes tecnologias, utilize um protocolo padrão e permita interoperabilidade. Estas características podem ser alcançadas empregando uma abordagem orientada a serviços (SOA) e *web services*. *Web services* é a principal tecnologia para implementação de serviços em uma SOA [Erl, 2005].

Segundo Graham [2007], a utilização de um BRMS em SOA permite o reuso das regras através dos serviços. Isso torna o desenvolvimento mais rápido e a manutenção mais fácil do que seria utilizando SOA sem um BRMS, pois quando um aspecto do negócio muda, as regras podem ser mudadas sem a necessidade de intervenção nos códigos dos serviços. A arquitetura orientada a serviços e os sistemas gerenciadores de regras de negócio são componentes fundamentais dentro da agilidade necessária no mundo dos negócios hoje em dia. SOA e BRMS são tecnologias paralelas e complementares.

Em SOA os recursos de software são chamados de Serviços, que são módulos bem definidos e auto contidos que provêm funcionalidades do negócio e são independentes de estado e contexto de outros serviços [Papazoglou *et al.*, 2007].

Os *web services* podem ser definidos como programas modulares, geralmente independentes e auto-descritivos que podem ser localizados e invocados através da internet ou de uma intranet corporativa. Eles tipicamente são construídos com especificações de tecnologias XML, SOAP e WSDL [W3C, 2004].

Newcomer e Lomow [2005] define SOA como um estilo de projeto que guia todos os aspectos de criação e uso de serviços de negócio através de todo o ciclo de vida de desenvolvimento (desde a fase de concepção até a aposentadoria de serviços).

A principal infra-estrutura de SOA é o Enterprise Service Bus [Hewitt, 2009]. Papazoglou *et al.* [2007] apresenta que o ESB é um barramento de mensagens, projetado para possibilitar a implementação, desenvolvimento e gerenciamento de soluções baseadas em SOA com o foco no empacotamento, desenvolvimento e gestão de serviços distribuídos. É responsabilidade do ESB permitir que consumidores (clientes) invoquem os serviços oferecidos por provedores de serviços, o que envolve várias tarefas, tais como: prover conectividade; transformação de dados; roteamento de mensagens baseado em conteúdo; tratar segurança; tratar confiabilidade; gerência de serviços; monitoramente e *log* de serviços; balanceamento de carga etc. Estas tarefas devem ser consideradas para diferentes plataformas de software e hardware, para diferentes *middleware* e protocolos.

Portanto, uma boa ferramenta BRMS deve possibilitar que suas regras, *rulesets* ou projetos de regras sejam disponibilizados em uma arquitetura orientada a

serviços (SOA). Isto é, o motor de execução deve estar disponível como um serviço e as regras, *rulesets* ou projetos de regras também devem ser disponibilizados como serviços, por exemplo, como *web services* [Graham, 2007]. Além disso, argumentamos que para conseguir alcançar sucesso na disponibilização das regras, estas devem ser disponibilizadas em um Enterprise Service Bus, permitindo obter os benefícios descritos anteriormente.

1.2 Objetivo

O objetivo deste trabalho é avaliar, na prática, a integração de BRMS x ESB. Ele apresenta a implementação e implantação de regras de negócio utilizando a tecnologia BRMS em um contexto orientado a serviços. Além disso, um estudo de caso será realizado apresentando a definição de regras de negócio em um BRMS com disponibilização das mesmas como *web services*, os quais foram publicados em um ESB.

Para execução do estudo de caso, dentre as ferramentas utilizadas, destacamos o uso do BRMS Websphere Ilog JRules [1]e o OSB[2] (Oracle Service Bus – ESB da Oracle).

Com o intuito de validar a proposta foi realizado um estudo de caso em domínio fictício ("Analisar pedido de crédito"), apresentado por Diirr *et al.* [2010], o qual contempla os tipos de regras de negócio considerados na proposta.

Para atingir esses objetivos este trabalho executará as seguintes atividades:

- Entender conceitos de Regras de negócio, ferramentas de gestão de regras de negócio, SOA e ESB;
- Escolher um processo de avaliação de ferramentas;
- Executar uma avaliação de ferramentas BRMS que permitam a disponibilização de regras de negócio como serviços;
- Estudar detalhadamente a ferramenta BRMS escolhida para o cadastro de regras e disponibilização como serviço;
- Escolher uma ferramenta ESB;

[1] http://www-01.ibm.com/software/integration/business-rule-management/jrules/
[2] http://www.oracle.com/us/technologies/soa/service-bus-066459.html

- Escolher um processo de negócio com regras bem explicitadas para utilização no estudo de caso;
- Escrever as regras na ferramenta BRMS;
- Disponibilizar as regras como serviços no servidor de aplicação e publicar no ESB;
- Implementar a aplicação cliente para aplicação das regras;
- Executar experimentos.

1.3 Estrutura do trabalho

Este trabalho está dividido da seguinte forma. O capítulo 1 é a presente introdução. O capítulo 2 apresenta os conceitos sobre regras de negócio, ferramentas de gestão de regras de negócio (BRMS) e sobre a arquitetura orientada a serviços (SOA), estes são os principais conceitos utilizados e apresentados neste trabalho. O capítulo 3 apresenta uma análise de ferramentas BRMS, considerando os critérios para a utilização das mesmas em uma arquitetura orientada a serviços. No capitulo 4 são apresentadas as características da ferramenta escolhida durante a avaliação, além de apresentar como as regras são disponibilizadas como serviços. O capitulo 5 apresenta um estudo de caso, onde os conceitos apresentados neste trabalho são colocados em prática. Por fim, o capítulo 6 apresenta a conclusão do trabalho e o capitulo 7 as referências bibliográficas.

Capítulo 2: CONCEITOS

Este capítulo tem como objetivo apresentar os conceitos de regras de negócio, BRMS e SOA que serão utilizados nesse trabalho.

2.1 Regras de Negócio

Esta seção tem como objetivo apresentar as definições de regras de negócio, sua importância e o porquê deve ser feito o gerenciamento das regras de negócio.

2.1.1 Definição de regras de negócio

Existem diversas definições de regras de negócio na literatura, como a de Halle [2002], onde as regras de negócio são definidas como um conjunto de condições que regem um evento do negócio para que este ocorra de forma aceitável para a organização. Também podem ser definidas como políticas que definem e descrevem ações do negócio [Azevedo *et al.*, 2009].

Em uma organização, as regras de negócio representam o conhecimento gerado por especialistas do negócio e, devido a esta característica, elas representam um importante ativo intelectual para as organizações, gerando a necessidade de organizá-las em uma documentação formal com o objetivo de efetivamente tornar essas regras uma fonte de divulgação de conhecimento organizacional. Isto viabilizaria um fortalecimento do tratamento uniforme e eficiente do negócio por parte dos profissionais envolvidos, devido à facilitação do entendimento do negócio [Morgado *et al*., 2007 *apud* Crerie 2009].

Utilizaremos neste trabalho a definição do BRG [2000] onde regras de negócio são definidas como declarações que definem ou restringem alguns aspectos do negócio, sendo atômicas, de forma que não podem ser divididas. Tem como objetivo afirmar a estrutura de um negócio ou controlar ou influenciar o comportamento deste.

2.1.2 Classificação das regras de negócio

Apesar da existência de diversas classificações para as regras de negócio, neste trabalho é utilizada a classificação da Business Rule Group, uma organização não comercial formada por pesquisadores importantes, que atua desde 1989 com foco em regras de negócio em organizações públicas e privadas. Segundo BRG [2000] as regras de negócio estão divididas em quatro categorias:

- Definições de termos de negócio: corresponde ao elemento mais básico, definindo um termo utilizado na linguagem dentro do domínio do negócio. A própria definição do termo é uma regra de negócio. Termos são tradicionalmente documentados em glossários ou como entidades em um modelo entidade-relacionamento. Eles se dividem em termos de negócio (ligados a contextos específicos e precisam ser definidos através de fatos) e termos comuns (significado aceito independente de conceito).
    - Exemplos:
        - (termo comum): Aluno;
        - (termo comum): Professor;
        - (termo de negócio): Graduação;
        - (termo de negócio): Disciplina.
- Fatos relacionando termos entre si: a natureza ou a estrutura operacional de uma organização pode ser definida com base nos fatos que relacionam os termos entre si. Os fatos são chamados de fatos base quando representam um caso do negócio, ou de fatos derivados quando construídos baseados em uma ou mais regras de negócio através de derivação.
    - Exemplos:
        - (Base): Um modelo de automóvel está em uma categoria de automóveis;

        - (Derivado): A taxa de locação do automóvel é baseada no preço do automóvel, na taxa de seguro e na taxa de combustível.

    Uma segunda classificação, divide os fatos em 3 tipos: atributo, generalização e participação. Um atributo expressa um fato onde um termo descreve algum aspecto de outro termo, uma generalização é o fato que descreve que um termo é subconjunto de outro termo e uma participação é o fato que associa um conjunto de termos de forma significativa para o negócio.

    - Exemplos:
        - (Atributo): Cor é atributo de carro;
        - (Generalização): Gerente de vendas é um empregado;
        - (Participação): Um grupo de locação é composto por modelo de carros.

- Restrição (Sentença de ação): toda organização restringe ou condiciona comportamentos de alguma forma, ou seja, enquanto as sentenças estruturais (fatos e termos) descrevem possibilidades, as sentenças de ações impõem restrições. As sentenças de ações podem se classificadas de duas formas distintas: em classes ou por tipos (que descrevem a natureza da regra).

    As sentenças de ação são classificadas em três classes: Restrição de Integridade, Condição e Autorização.

    - Restrição de Integridade: Classifica as sentenças que precisam sempre ser verdadeiras. Normalmente, elas representam elementos (atributos ou relacionamentos, por exemplo) que são obrigatórios. Por exemplo:
        - Um estudante precisa ter uma matricula;
        - Um cliente precisa estar cadastrado.

- Condição: Refere-se às sentenças onde caso uma condição seja verdadeira outra regra de negócio torna-se aplicável. Por exemplo:
    - Se o aluno está com a matricula trancada, então ele não pode se inscrever nas disciplinas.
- Autorização: Classificação que se refere a uma sentença que restringe quem pode executar uma determinada ação. Por exemplo:
    - Somente o gerente tem acesso ao volume de vendas.

As sentenças de ação também podem ser classificadas em três tipos: Habilitadora, Temporizadora e Executora.

- Habilitadora: Tipo que se refere a uma sentença que, se é verdadeira, permite a existência de determinado objeto ou execução de determinada ação. Por exemplo:
    - O Aluno só pode cursar Cálculo II após ser aprovado em Cálculo I.
- Temporizadora: Tipo que representa a sentença que ocorre de acordo com um controle de tempo. Funciona como um contador que assinala o momento da parada ou execução de uma ação. Por exemplo:
    - Os carros devem ser devolvidos antes das 18 horas do dia da entrega.
- Executora: Tipo que se refere à sentença que causa a execução de uma ou mais ações. Por exemplo:
    - Se a conta for paga após o vencimento é cobrada uma multa de 1% ao dia.

- Derivações: definem como uma regra representada em uma forma pode ser transformada em outra. As derivações geram como resultado os fatos derivados.

- Exemplos:
    - (Derivação): Preço de venda = Preço de custo + Margem de lucro;
    - (Fato derivado): O preço de venda é baseado no preço de custo e na margem de lucro.

  As derivações também podem utilizar cálculos matemáticos e inferências (deduções e induções lógicas) para gerarem fatos derivados.

Essas regras de negócio podem estar embutidas em códigos de aplicações, documentadas em modelos de processos de negócio ou podem ser gerenciadas por ferramentas de gestão de regras de negócio (*Business Rules Management Systems* - BRMS). Com o apoio de um BRMS as regras tornam-se mais flexíveis, podendo ser alteradas/ajustadas de acordo com a demanda do negócio com uma maior velocidade e praticidade do que se estivesse embutida em uma aplicação do negócio. Além disso, é possível que o usuário do negócio crie e mantenha suas próprias regras, sem que um desenvolvedor ou programador tenha que mexer em algum código de programa.

2.1.3 Gestão de regras de negócio

Assim como já há a muito tempo o consenso de que os dados e informações de uma organização precisam ser gerenciados como ativos, de forma independente das aplicações que os acessam, formando o conceito de Arquitetura de Informação [Botto, 2004; Evernden e Evernden, 2003], as Regras de negócio também vêm sendo vistas cada vez mais como ativos da organização por possuírem informação sobre como o negócio da organização está estruturado e como ele deve se comportar, o que faz com que surja a necessidade de gestão destas regras. [Azevedo *et al.*, 2009]

Muitas regras de negócio estão inacessíveis, ou pior, são desconhecidas. Este é o caso quando as regras estão embutidas nos códigos das aplicações, para as quais não há quase ou nenhuma documentação [Halle, 2002]. Quando essas regras não são conhecidas, as pessoas muitas vezes assumem diferentes regras que não são as regras de negócio da organização, e isso leva a erros, tanto humanos quanto tecnológicos.

Para que a divulgação do conhecimento organizacional através das regras de negócio seja efetiva é necessário que estas sejam organizadas e gerenciadas utilizando uma linguagem natural que possa ser entendida por toda a organização, sem a necessidade de intervenção dos profissionais de Tecnologia da Informação (TI), como propõe Ross [2003].

Além disso, as organizações necessitam reagir rapidamente às mudanças, e para isso é necessário que as regras de negócio sejam mudadas de forma eficiente e rápida. Para isso é necessário que exista uma gestão eficiente das regras de negócio.

Segundo Graham [2007], gestão de regras de negócio é a prática de implementar sistemas baseados em uma abordagem de regras de negócio. Para Azevedo *et al*. [2009], gestão de regras de negócio refere-se às atividades necessárias para tratar as atividades relacionadas a um ciclo de vida de regras de negócio.

Deitert e McCoy [2007] apontam que a tecnologia de regras de negócio vem sendo utilizada há décadas. Por exemplo, regras de negócio têm sido aplicadas para nortear o trabalho entre participantes de uma organização ou de um processo para determinar como uma organização deve responder a um evento do negócio, para guiar o uso e validação de dados através de diferentes sistemas, entre outros. Com isso a utilização de uma ferramenta de suporte é essencial. Mas além de uma ferramenta de suporte é necessário que haja uma série de questionamentos como, por exemplo, uma proposta de governança dessas regras, qual a melhor forma de representação, qual será o ciclo de mudança dessas regras, além de outras questões apresentadas por Sinur e McCoy [2007].

Nelson *et al*. [2008] apresentam uma proposta de ciclo de vida de gestão de regras de negócio, os quais são: (1) alinhamento/planejamento dos domínios de regras de negócio dentro do modelo de dados e estratégia da organização; (2) capturar regras das diferentes fontes; (3) organizar as regras; (4) autoria de regras; (5) distribuir (ou compartilhar) as regras, depois que estas foram armazenadas e geridas em um repositório central; (6) testar as regras para interoperabilidade; (7) aplicar as regras para automação; (8) manter as regras, apresentado na Figura 1.

Este trabalho tem como premissa que as regras de negócio já tenham sido identificadas e definidas conceitualmente. O foco se encontra nas fases de autoria de regras (criação das regras definidas na ferramenta), armazenamento em um

repositório, disponibilização da regra como serviço e testes do uso das mesmas. Logo, são contemplados os passos 4, 5, 6 e 7 do ciclo de vida proposto por Nelson *et al*. [2008].

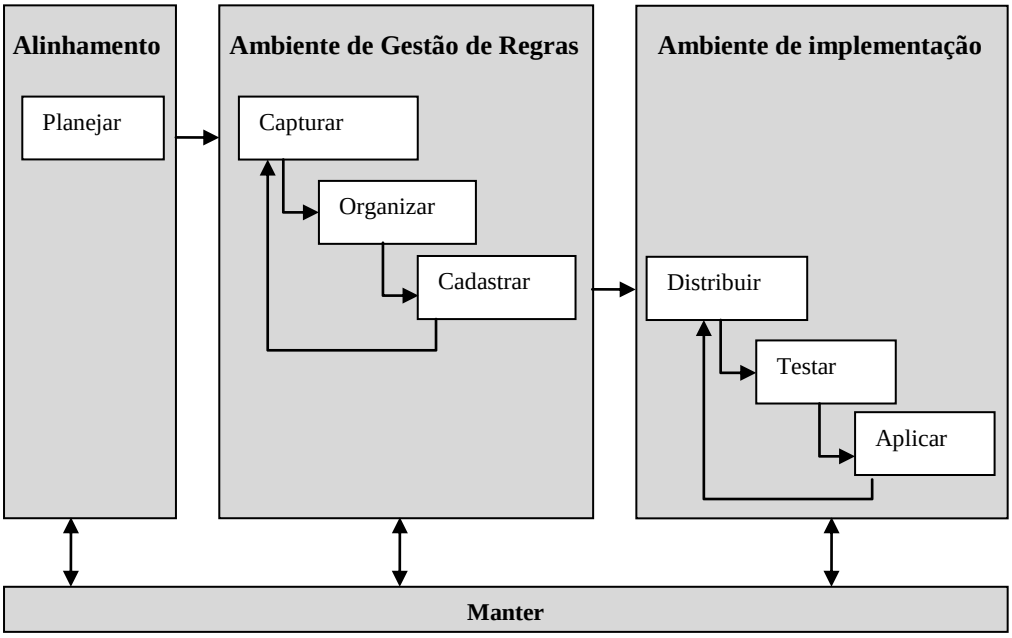


Figura 1 – Ciclo de vida de gestão de regras de negócio [Nelson *et al*., 2008]

2.2 BRMS

Esta seção apresenta os conceitos de BRMS: definição, arquitetura, quando adquirir a ferramenta e exemplos de ferramentas existentes.

2.2.1 Definição

A gestão das regras de negócio de uma organização demanda um suporte computacional efetivo, devido não apenas à abrangência de tais regras (que devem contemplar todo o ambiente corporativo da organização, permitindo uma visão global do gestor sobre todas as regras de todas as unidades funcionais da organização), mas também em função da complexidade envolvida [Azevedo *et al.*, 2009]. Por exemplo, as regras de negócio por definição contemplam definições de vários tipos (estruturais, cálculos e inferências, condicionais de ação, de autorização), que podem ser combinadas entre si, produzindo efeitos e restrições de comportamento que são inviáveis de serem processadas manualmente. Surgem então

ferramentas para gestão de regras de negócio. Deitert e McCoy [2007] apresentam que, no início do advento de tais ferramentas, elas eram chamadas de BRE (*Business Rule Engine*) e, por anos, consistiam em um ambiente para desenvolvimento das regras e um motor de execução das regras. Apesar de estes dois componentes proverem um *baseline* para ferramentas de modo a facilitar a criação de regras e a execução de regras complexas, mais funcionalidades foram requeridas a fim de construir uma estratégia de gestão de regras de negócio completa, surgindo a necessidade da criação de uma arquitetura mais ampla, implementada por sistemas denominados de BRMS.

O objetivo de um BRMS é prover um conjunto de funcionalidades capaz de apoiar uma estratégia de gestão de regras de negócio completa, ou seja, contemplando as atividades de criação, registro, classificação, verificação, desenvolvimento e execução de regras de negócio. A ferramenta pode ser utilizada em tempo de projeto (quando as regras são planejadas, definidas e especificadas) e de execução (quando as regras devem ser monitoradas para garantir que não estão sendo violadas), permitindo à organização explicitar, definir, analisar, executar, auditar e manter uma grande variedade de regras de negócio. A equipe de TI e do negócio, utilizando a ferramenta, tem facilidades para definir regras usando diferentes formas de representação, tais como, árvores de decisão, tabelas de decisão, linguagem natural, código semelhante à linguagem de programação ou outras técnicas de representação. Isto permite isolar a representação da regra da execução da lógica do negócio – provendo um mecanismo explícito para gerência das regras. Além disso, é possível analisar conflitos entre regras, consistência entre regras e outras questões de qualidade [Azevedo *et al.*, 2009].

2.2.2 Arquitetura de um BRMS

Segundo Deitert e McCoy [2007], a arquitetura de um BRMS é composta pelos seguintes componentes: Motor de execução, Repositório, Ambiente de desenvolvimento integrado, Modelo de simulação de regras, Monitoramento e análise, Gestão e administração e *Template* de regras, como pode ser observado na Figura 2.

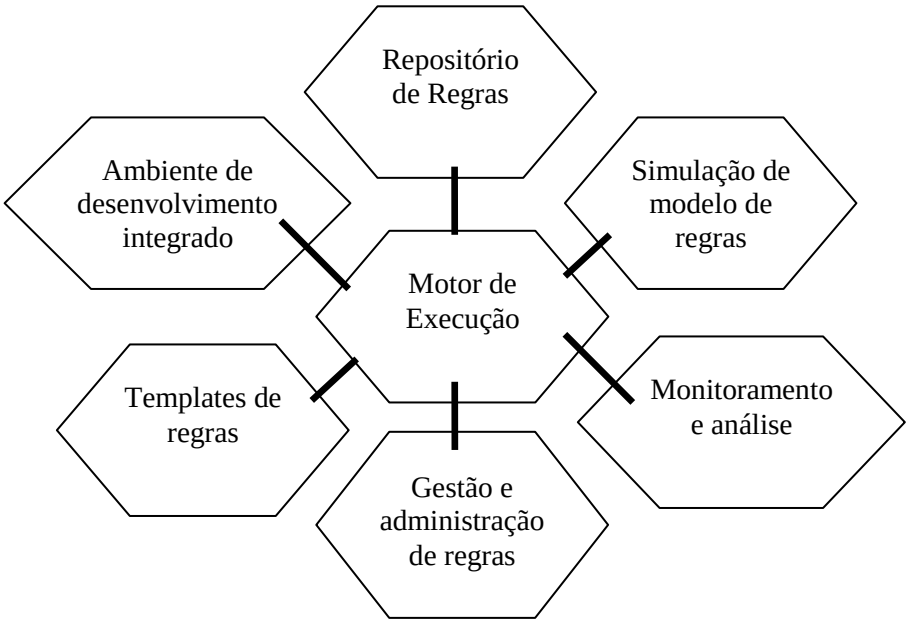


Figura 2 – Componentes de um BRMS [Deitert e McCoy, 2007]

Deitert e McCoy [2007] também apresentam as características de cada um dos componentes de um BRMS.

- **Motor de execução**: É o coração do BRMS, responsável pela execução das regras de negócio. O motor de execução geralmente é baseado no algoritmo RETE [Forgy, 1982]. A execução é baseada em decisões do tipo SE x => ENTÃO y, onde na primeira parte estão representadas as condições necessárias para a execução de uma ação, descrita na segunda parte. É possível também executar regras que tenham somente a parte da ação, ou seja, regras que não tenham condições.

- **Repositório**: É a base de dados onde ficam armazenadas as regras de negócio. Os repositórios podem oferecer mecanismos de versionamento e agrupamento de regras, além de segurança, funcionalidades avançadas para pesquisa/consulta de regras, controle de edição/concorrência e proteção de edição através de senha.

- **Ambiente de desenvolvimento integrado (IDE)**: Uma IDE é uma ferramenta gráfica que permite ao usuário de negócio desenvolver, testar e depurar as regras. As IDEs são projetadas especialmente para

que os usuários de negócio e gerentes de TI possam participar da gestão de regras de negócio.

- **Simulação do modelo de regras:** O componente de simulação de regras oferece visualização gráfica da seqüência de execução das regras e suas dependências, podendo ser feita uma análise do comportamento das regras usando dados atuais ou antigos. Permite realizar verificações com as regras antes destas serem disponibilizadas em produção.
- **Monitoramento e análise:** Este componente provê funcionalidades para a análise do histórico de execução das regras, bem como o monitoramento em tempo real da execução das regras. Com isso, é possível saber se as regras estão executando como era esperado ou então saber a razão de alguma falha na execução das mesmas.
- **Gestão de administração da regra:** No componente de gestão e administração da regra é possível publicar a regra, controlar o acesso à regra, promover novos conjuntos de regras e monitorar o desempenho dos sistemas e servidores de execução de regras.
- ***Template* de regras:** O componente de *template* de regras fornece regras pré-definidas para agilizar o processo de desenvolvimento de regras. Além disso, neste componente é possível desenvolver novos *templates*.

2.2.3 Por que um BRMS?

A não ser que as aplicações e processos sejam perfeitamente estáveis, a introdução de uma tecnologia para gerir explicitamente as regras faz sentido. Halle [2002] e Graham [2007] apresentam as vantagens para externalização de regras de negócio, enquanto Bajec e Krisper [2004] listam os problemas caso esta externalização não seja realizada. Estas vantagens e problemas foram descritos na Seção 1.1. A necessidade pela adaptação rápida às mudanças exige que a gestão de regras seja eficiente, e com a utilização de um BRMS isso se torna muito mais fácil [Azevedo *et al.*, 2009].

Sinur e McCoy [2007] apresentam 10 questões que devem ser feitas e respondidas antes de se optar pela escolha de uma ferramenta de gestão de regras de negócio. As mais relevantes para este trabalho são:

- Quais são as possibilidades de utilização de um BRMS? – Com essa pergunta a organização que está pretendendo adquirir um BRMS deve avaliar a importância de se externalizar as regras em uma ferramenta, o grau de entendimento que a externalização das regras pode trazer, além de analisar a possibilidade de acompanhar o fluxo de execução dos processos através das regras.
- Todas as regras devem ser colocadas em um BRMS? – Nesse caso a organização deve pensar em quais regras deverão estar no BRMS, geralmente opta-se por colocar as regras que mudam com maior freqüência e aquelas que são mais importantes.
- Quem deve ser o "dono" das regras de negócio? – Nesta pergunta a organização deve avaliar quem vai ser o responsável por gerenciar as regras.
- Quem deve modificar as regras? – Deve-se pensar em quem pode modificar as regras que já estão sendo utilizadas, visto que a mudança nas regras afeta diretamente as aplicações. Nesse caso a organização deve pensar como será feita a governança dessas regras.

2.2.4 Ferramentas BRMS

O mercado de ferramentas BRMS vem crescendo cada vez mais, anteriormente essas ferramentas chamadas de BRE (Business Rules Engine) passaram a ser chamadas de BRMS devido a evolução das mesmas, implementando mais funcionalidades, como simulação e teste das regras, versionamento, análise das regras etc [Deitert e McCoy, 2007].

Existem várias ferramentas no mercado para gestão de regras de negócio como, por exemplo, as ferramentas enumeradas por Sinur [2005] e por Rymer e Gualtieri [2008], cujos trabalhos apresentam avaliações de ferramentas existentes no mercado. Também são apresentadas algumas ferramentas BRMS em [BRG, 2000] e [BRCommunity, 2009]. Além destes, Bajec e Krisper [2005] listam algumas

ferramentas existentes e classificam as ferramentas em 3 tipos: ferramentas para desenvolvimento de sistemas de informação baseados em regras, ferramentas para o desenvolvimento de aplicações baseadas em conhecimento e ferramentas para o gerenciamento de regras de negócio em toda a empresa.

A partir da análise dessas ferramentas que foram listadas por esses trabalhos, verificou-se a volatilidade do mercado de BRMS. Muitas ferramentas que foram listadas e/ou avaliadas por Sinur [2005] e Bajec e Krisper [2005] foram compradas por outras empresas ou então foram descontinuadas. Após essa constatação, analisamos quais ferramentas ainda estão presentes no mercado e foram encontradas as seguintes ferramentas:

- Corticon Business Rules Management System[3];
- ESI Logist[4];
- FICO Blaze Advisor[5];
- WebSphere Ilog BRMS[6];
- JBoss Enterprise BRMS[7];
- RuleXpress[8];
- SAP NetWeaver BRM[9];
- Sapiens eMerge[10];
- Ness Usoft[11];
- Visual Rules[12];
- G2 Plataform[13];
- Versata BRMS[14];

---

[3] http://www.corticon.com/Products/Business-Rules-Management-System.php
[4] http://www.esi-knowledge.com/products_logist.aspx
[5] http://www.fico.com/en/Products/DMTools/Pages/FICO-Blaze-Advisor-System.aspx
[6] http://www-01.ibm.com/software/websphere/products/business-rule-management/
[7] http://www.jboss.com/products/platforms/brms/
[8] http://www.rulearts.com/RuleXpress
[9] http://www.sap.com/platform/netweaver/components/brm/index.epx
[10] http://www.sapiens-tech.com/Dev2Go.web?Anchor=206975
[11] http://www.usoft.com
[12]http://www.visual-rules.com/business-rules-management-enterprise-decision-management.html
[13] http://www.gensym.com/index.php?option=com_content&view=article&id=47&Itemid=54
[14] http://www.versata.com/index2.php?option=com_content&task=view&id=124

- PegaRules[15];
- InRule[16];
- Oracle Business Rules[17],
- ARIS Business Rule Designer[18].

Sendo que as seguintes ferramentas permitem a disponibilização das regras de negócio como um serviço (implementado na tecnologia de *web services*), a qual é a proposta que este trabalho irá avaliar.

- Corticon BRMS;
- Fico Blaze Advisor;
- WebSphere Ilog BRMS;
- JBoss Enterprise BRMS;
- Sapiens eMerge;
- Visual Rules;
- Versata BRMS;
- PegaRules;
- InRule;
- Oracle Business Rules.

2.3 Arquitetura Orientada a Serviços (SOA)

Esta seção apresenta a definição de SOA e seus conceitos, como *Web Services* e ESB.

2.3.1 Definição

Existem muitas definições diferentes sobre SOA na literatura, uma vez que os projetos em Sistemas de Informação possuem particularidades únicas, e isso permite que a aplicação prática de SOA resulte em diferentes visões de acordo com as experiências de cada profissional e/ou pesquisador [Erickson e Siau, 2008].

---

[15] http://www.pega.com/Products/RulesTechnology.asp
[16] http://www.inrule.com/products/InRule.aspx
[17] http://www.oracle.com/technology/products/ias/business_rules/index.html
[18]http://www.ids-scheer.com/en/ARIS/ARIS_Platform/ARIS_Business_Rules_Designer/3747.html

Segundo Newcomer e Lomow [2005], SOA é um estilo de projeto que guia todos os aspectos de criação e uso de serviços de negócio através de todo o ciclo de vida de desenvolvimento (desde a fase de concepção até a aposentadoria de serviços), bem como trata da definição e do provisionamento da infra-estrutura de TI que permite que diferentes aplicações troquem dados e participem de processo de negócio independente dos sistemas operacionais onde estas aplicações estão executando ou linguagens de programação utilizadas para suas implementações.

Segundo Josuttis [2007], a arquitetura orientada a serviços (SOA) é um paradigma para a realização e a manutenção dos processos corporativos que se encontram em grandes sistemas distribuídos. SOA não é uma arquitetura concreta: é algo que conduz a uma arquitetura concreta. Você pode chamá-la de estilo, paradigma, conceito, perspectiva, filosofia ou representação. Ou seja, SOA não é uma ferramenta ou *framework* que se possa comprar. É uma abordagem, uma maneira de pensar, um sistema de valores que leva a certas decisões concretas quando se projeta uma arquitetura concreta de software.

É importante deixar claro algumas definições de SOA que NÃO procedem. SOA não é um produto, não é uma solução, não é uma tecnologia, não pode ser reduzido a produtos de software e, finalmente, não atende todos os desafios tecnológicos aos quais estão submetidos os negócios de hoje [Josuttis, 2007].

SOA tem o propósito de tratar os requisitos de baixo acoplamento, desenvolvimento baseado em padrões, computação distribuída independente de protocolo, mapeamento dos sistemas de informação da organização para todos os seus fluxos de processos de negócios, integração de aplicações, gerência de transações, políticas de segurança e coexistência de sistemas em múltiplas plataformas e também sistemas legados [Papazoglou *et al.*, 2007].

Para Marks e Bell [2006], o objetivo de SOA é permitir às organizações realizarem seus negócios e ter vantagens tecnológicas por meio da combinação de inovação de processos, governança eficaz e estratégia de tecnologia, as quais giram em torno da definição e reutilização de serviços. Um dos benefícios mais importantes que SOA fornece é a melhora da produtividade e agilidade tanto para o negócio quanto para TI, e conseqüentemente, a redução de custos no desenvolvimento e manutenção dos sistemas envolvidos.

2.3.2 Serviços

Serviços são módulos de negócio ou funcionalidades das aplicações que possuem interfaces expostas e que são invocados via mensagens. Por exemplo, em um sistema bancário teríamos os seguintes serviços: serviço de nomes e endereços, serviço de abertura de conta, serviço de balanço de contas, serviço de depósitos etc. Serviços correspondem a recursos de software bem definidos através de uma linguagem padrão, são auto-contidos, provêem funcionalidades padrões do negócio, independentes do estado ou contexto de outros serviços [Erl, 2005].

SOA é focado em processos de negócio. Esses processos são executados em diferentes passos (atividades ou tarefas) e em diferentes sistemas. O primeiro objetivo de um serviço é representar um passo de uma funcionalidade do negócio. Isto é, de acordo com o domínio para qual ele é desenvolvido, um serviço deve representar uma funcionalidade auto-contida que corresponde a uma atividade do mundo real [Josuttis, 2007].

Na utilização dos serviços existem duas entidades fundamentais: *provedor* e *consumidor*. O provedor é aquele que oferece e executa o serviço em resposta a uma requisição do consumidor, que por sua vez é aquele que se utiliza do serviço para obter a funcionalidade processada pelo provedor [Josuttis, 2007].

Serviços são disponibilizados aos clientes através de interfaces. Assim como na programação orientada a objetos busca-se "esconder" a complexidade do código e não permitir acesso à estrutura interna, o objetivo da interface em SOA é o mesmo: esconder e não permitir acesso à implementação do serviço. A interface é uma assinatura, que descreve parâmetros de entrada, parâmetros de saída e possíveis exceções [Josuttis, 2007].

Marks e Bell [2006] apresentam as seguintes características para serviços:

- Granularidade adequada - O serviço deve possuir a granularidade adequada a fim de resolver o problema do negócio, possibilitar o reuso e permitir que os serviços sejam implementados do ponto de vista tecnológico;

- Contratos de serviços bem definidos – O contrato de serviço especifica o que o serviço faz e como proceder pra utilizá-lo. Nos *web services* os contratos são definidos pelo documento WSDL;
- Acoplamento fraco – O acoplamento fraco é o conceito utilizado para tratar requisitos de escalabilidade, flexibilidade e tolerância a falha. Acoplamento fraco significa projetar serviços de forma que implementações específicas dos serviços possam ser substituídas, modificadas e evoluídas sem corromper os serviços;
- Capacidade de serem localizados – Serviços devem ser bem projetados e seus contratos devem ser publicados e estar visíveis para os possíveis clientes poderem acessá-los;
- Durabilidade – Os serviços devem existir ao longo do tempo, sendo mapeados a temas de processos duradouros;
- Composição – Serviços devem ser projetados a fim de permitir que possam ser utilizados por outros serviços;
- Alinhamento ao negócio - Serviços devem representar conceitos do negócio e estar de acordo com as necessidades do negócio, como definido na estratégia e no plano do negócio, além de estarem associados aos processos de negócio identificados;
- Reuso - Serviços devem ser implementados com reuso claro em processos de negócio;
- Interoperabilidade - A interoperabilidade deve ser garantida pela aplicação de políticas, padrões e outros critérios de projeto durante o ciclo de vida do serviço: identificação de serviço, análise de serviços, projeto, implementação, teste e integração e implantação.

Neste trabalho são utilizados os conceitos de serviço e regras de negócio, dado que se pretende avaliar a disponibilização das regras de negócio como serviços. As regras serão criadas e gerenciadas por uma BRMS e serão disponibilizadas como serviços em um barramento de serviços (ESB).

2.3.3 *Web Services*

Os *web services* podem ser definidos como programas modulares, geralmente independentes e auto-descritivos que podem ser localizados e invocados através da internet ou de uma intranet corporativa. Eles tipicamente são construídos com especificações de XML, SOAP, WSDL e UDDI [W3C, 2004].

A XML (*Extensible Markup Language*) descreve uma classe de objetos de dados, chamados documentos XML, e parcialmente descreve o comportamento dos programas que processam eles. XML é um perfil de aplicação ou uma forma restrita do SGML (*Standard Generalized Markup Language*). Seu propósito principal é a facilidade de compartilhamento de informações através da Internet [XML, 2008].

O WSDL (*Web Services Description Language* – Linguagem de Descrição de Serviços Web) é uma linguagem baseada em XML utilizada para descrever *web services*. Trata-se de um documento escrito em XML que, além de descrever o serviço, especifica como acessá-lo e quais são as operações ou métodos disponíveis [WSDL, 2001 e 2007].

O UDDI (*Universal Description, Discovery and Integration* – Descrição Universal, Descoberta e Integração) é um protocolo baseado em XML que provê um diretório distribuído com listas de negócios na Internet e descoberta de serviços [UDDI, 2004]. Através da interface do UDDI é possível pesquisar os serviços publicados e obter as informações necessárias para acessá-los.

O SOAP (*Simple Object Access Protocol* – Protocolo Simples de Acesso a Objeto) é independente de plataforma e independente de implementação. Permite baixo acoplamento entre requisitante e provedor e permite comunicação entre serviços de diferentes organizações [SOAP, 2007].

A arquitetura de *web services* é a mais utilizada para implementar serviços em uma arquitetura SOA, já que é baseada em um conjunto padrões que permitem a interoperabilidade. Os padrões de desenvolvimento utilizados em *web services* permitem que funções de negócios possam ser invocadas remotamente. Muitos analistas, fabricantes e autores hoje em dia recomendam apenas uma maneira apropriada para realizar um ambiente SOA: com *web services* [JOSUTTIS, 2007].

2.3.4 ESB

Segundo Papazoglou *et al.* [2007], *web service* é uma tecnologia importante para implementação de SOAs, mas outras linguagens de programação convencionais ou *middlewares* podem ser adotados. Na verdade, todas as tecnologias que implementam as interfaces de um serviço diretamente com WSDL e se comunicam com mensagens XML podem ser envolvidas em SOA. Para resolver essas questões de interoperabilidade de diferentes tecnologias e informações existem duas soluções:

- Construir o módulo cliente para que este esteja de acordo com as características de todos os módulos servidores que ele for invocar; ou
- Inserir uma camada de comunicação e integração lógica entre os módulos cliente e servidor.

A primeira solução requer o desenvolvimento de uma interface para cada conexão, resultando em uma topologia ponto a ponto. Esta topologia é conhecida por ser difícil de gerenciar e manter, visto que a mesma introduz um forte acoplamento. Isto torna a mudança mais difícil, além de não ser escalável e ser muito complexo [Papazoglou *et al*, 2007].

Com isso, a segunda opção torna-se a alternativa mais viável. Ela introduz uma camada de integração que deve suportar a interoperabilidade entre a infra-estrutura disponibilizada e as aplicações. Os requisitos de prover uma infra-estrutura de integração apropriada e gerenciável para *web services* e SOA se encontram no conceito de ESB (*Enterprise Service Bus*) [Hewitt, 2009; Chappell, 2004 e Schulte, 2002 *apud* Papazoglou *et al*, 2007].

Segundo Papazoglou *et al* [2007], o ESB é um barramento de mensagens baseado em padrões abertos que permite a implementação, disponibilização e gerenciamento de soluções SOA. O ESB provê processamento distribuído, integração baseada em padrões e é a infra-estrutura central necessária para as empresas que vão implementar uma SOA. Ele é projetado para prover interoperabilidade entre aplicações através de adaptadores e interfaces. Uma arquitetura simplificada de um ESB pode ser observada na Figura 3. Na figura está exemplificada a integração entre aplicações de diferentes plataformas (Java, .Net e aplicações legadas) e também aplicações de empresas, utilizando adaptadores. Além

disso, o ESB permite integrar aplicações externas e diversas fontes de dados através do uso de *web services*.

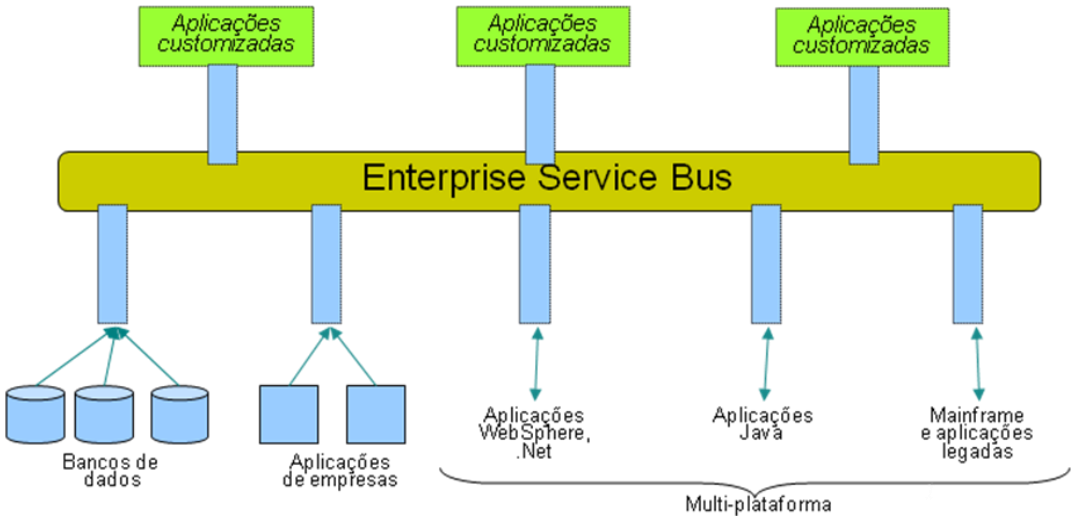


Figura 3 – Arquitetura simplificada de um ESB [Papazouglou *et al.*, 2007]

Segundo Josuttis [2007], é responsabilidade do ESB possibilitar que os consumidores chamem os serviços provedores. Dependendo das abordagens técnicas e organizacionais escolhidas para implementar o ESB, essa responsabilidade envolve as seguintes tarefas:

- Prover conectividade: permitir a conexão entre aplicações desenvolvidas em diferentes plataformas, como ilustrado na Figura 3;
- Transformar dados: Por integrar diferentes plataformas e linguagens de programação, a transformação de dados é parte essencial do ESB. Transformações são especificadas no ESB de forma a permitir, por exemplo, que mensagens enviadas em um determinado formato a partir de um requisitante sejam transformadas para o formato esperado pelo serviço provido e vice-versa;
- Roteamento (Inteligente): Outra tarefa fundamental do ESB é o roteamento, isto é, o ESB é responsável por encaminhar as mensagens do provedor ao consumidor e as respostas de volta. Além disso, o roteamento inteligente realiza o balanceamento de carga, roteamento baseado em conteúdo e tratamento de falhas;
- Tratar de segurança: permitir que aplicações se comuniquem de forma segura;

- Tratar de confiabilidade: garantir a confiabilidade na comunicação entre aplicações;
- Gerenciamento de serviços: permitir a publicação de serviços e suas respectivas versões;
- Monitoramento e *log*: permitir acompanhar a execução dos serviços como, por exemplo, fluxo de mensagens, carga de dados trafegando no ESB, apresentar alertas caso algum evento aconteça.

2.4 Resumo

Este capítulo apresentou os principais conceitos sobre regras de negócio, sistemas gerenciadores de regras de negócio (*Business Rules Management System* – BRMS) e sobre arquitetura orientada a serviços (SOA).

Regras de negócio são declarações que definem ou restringem alguns aspectos do negócio e são classificadas em quatro categorias: definições de termos de negócio, fatos relacionando termos entre si, restrições (sentenças de ação) e derivações. Para que a divulgação do conhecimento organizacional através das regras de negócio seja efetiva é necessário que estas sejam organizadas e gerenciadas, esse gerenciamento de regras de negócio necessita de um apoio computacional, que é fornecido pelos BRMS.

Os sistemas gerenciadores de regras de negócio (BRMS) têm como objetivo prover um conjunto de funcionalidades capaz de apoiar uma estratégia de gestão de regras de negócio completa. A arquitetura de um BRMS é composta pelos seguintes componentes: Motor de execução, Repositório, Ambiente de desenvolvimento integrado, Modelo de simulação de regras, Monitoramento e análise, Gestão e administração e *Template* de regras. Um BRMS torna-se cada vez mais necessários de acordo com o aumento da necessidade de reação as mudanças de forma efetiva e rápida. Neste trabalho, foram listadas 16 ferramentas BRMS, entre as quais 10 possibilitam a disponibilização das regras de negócio como serviço.

Existem diversas definições para arquitetura orientada a serviços. Segundo Josuttis [2007], a arquitetura orientada a serviços (SOA) é um paradigma para a realização e a manutenção dos processos corporativos que se encontram em grandes sistemas distribuídos. SOA é focado em processos de negócio. Esses processos são

executados em diferentes passos (atividades ou tarefas) em diferentes sistemas. Um serviço tem como objetivo primário representar um passo de uma funcionalidade do negócio. Serviços são módulos de negócio ou funcionalidades das aplicações que possuem interfaces expostas, e que são invocados via mensagens.

A arquitetura de *web services* é a mais utilizada para implementar serviços em uma arquitetura SOA, já que é baseada em um conjunto padrões que permitem a interoperabilidade. Eles tipicamente são construídos com especificações de XML, SOAP, WSDL e UDDI.

A principal infra-estrutura de SOA é o *Enterprise Service Bus* (ESB). O ESB é um barramento de mensagens baseado em padrões e aberto que permite a implementação, disponibilização e gerenciamento de soluções SOA. Um ESB deve prover conectividade entre os serviços, interoperabilidade entre diferentes plataformas, roteamento inteligente, ele trata segurança e confiabilidade, gerencia os serviços, monitora e faz o *log*.

No próximo capitulo será apresentada uma análise das ferramentas BRMS, incluindo o processo de avaliação de ferramentas e a análise dos resultados do mesmo.

Capítulo 3: ANÁLISE DAS FERRAMENTAS

Este capítulo tem como objetivo analisar as ferramentas BRMS que estão presentes no mercado e, utilizando um processo de avaliação de ferramentas, avaliar qual ferramenta é mais adequada para utilização dentro de uma SOA.

O estudo e prospecção de ferramentas de apoio computacional às atividades em uma organização são de responsabilidade da área de Arquitetura de Tecnologia da Informação (TI) [Botto 2004]. Muitas vezes a organização possui ferramentas que atendem de forma parcial ou integral às suas necessidades, mas deseja encontrar novas ferramentas mais adequadas que tragam maior agilidade para atender às necessidades do mercado. Outras vezes, organizações não possuem apoio computacional às suas atividades e querem introduzí-lo no processo.

No entanto, avaliar e selecionar uma ferramenta não são tarefas simples. De modo geral, devem ser avaliadas todas as ferramentas existentes com potencial para atendimento dos requisitos definidos. Vários fatores podem dificultar esta avaliação. Primeiro, em muitos casos os requisitos não estão definidos de forma clara pela organização, o que torna a avaliação subjetiva. Segundo, não é trivial identificar quais ferramentas disponíveis devem ser avaliadas, pois a quantidade de informação disponível por seus fabricantes pode não ser suficiente ou, mesmo extensa demais para o período da avaliação. Terceiro, o número de ferramentas disponíveis no mercado que podem atender aos requisitos definidos pode ser muito grande e tornar inviável o tempo para executar a avaliação detalhada de todas as ferramentas. Quarto, é freqüente existir um grande número de ferramentas que dizem atender a subconjuntos distintos dos requisitos definidos, tornando injusta uma comparação direta entre elas. Finalmente, é desejado que esta avaliação não se restrinja a critérios técnicos de cada ferramenta individualmente, mas também considere a arquitetura de TI atual da organização, planejando interoperabilidade com as demais ferramentas existentes na organização e migração dos dados manipulados por ferramentas legadas que forem substituídas. Para endereçar a todas estas questões, é essencial que exista uma metodologia para avaliação e seleção de ferramentas computacionais, a

fim de tornar estas importantes atividades parte de um processo sistemático, eficiente e menos subjetivo.

A prospecção de ferramentas realizada neste trabalho segue o processo de avaliação de ferramentas proposto por Azevedo *et al.* [2010b]. Os detalhes de execução deste processo para avaliação de ferramentas para gestão de regras de negócio em um ambiente SOA são apresentados na seção 3.1. A seção 3.2 apresenta uma análise dos resultados obtidos no processo de avaliação.

3.1 Processo de avaliação de ferramentas

O processo utilizado para avaliação das ferramentas corresponde a uma adaptação do processo proposto pela SEI (*Software Engineering Institute*) apresentado em [Comella-Dora, 2004] e do processo apresentado em [Azevedo *et al.*, 2010b], descrito em [Azevedo *et al.*, 2010a]. O processo é ilustrado na Figura 4.

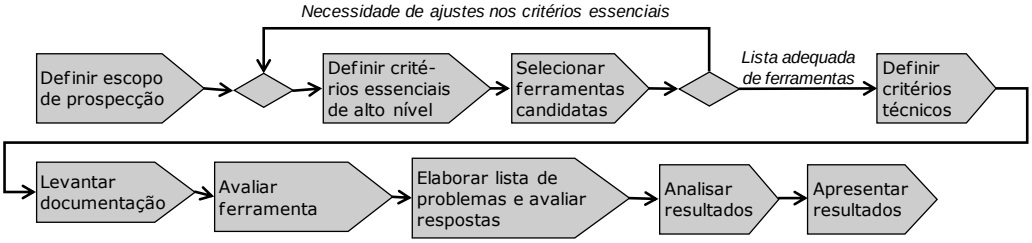


Figura 4 – Processo para avaliação de ferramentas [Azevedo *et al.*, 2010b]

1. **Definir escopo de prospecção:** O objetivo deste trabalho de prospecção de ferramentas foi encontrar uma ferramenta BRMS que permitisse a gestão de regras de negócio e execução das mesmas dentro de uma arquitetura orientada a serviços.
2. **Definir critérios essenciais de alto nível:** Os critérios essenciais definidos foram: (i) possuir um ambiente de desenvolvimento de regras (IDE) que permita que um usuário comum, sem conhecimentos de linguagens de programação, possa criar e manter suas regras de negócio; (ii) permitir a disponibilização das regras de negócio como serviço.
3. **Selecionar ferramentas candidatas:** A escolha das ferramentas para avaliação foi baseada nas referências às empresas que possuem ferramentas BRMS apresentadas por [BRG, 2000] e [BRCommunity, 2009], além das

ferramentas avaliadas por [Sinur, 2005] e [Rymer e Gualtieri, 2008] e pesquisas realizadas.

As seguintes ferramentas foram escolhidas para avaliação: Corticon Business Rules Management System[19]; ESI Logist[20]; FICO Blaze Advisor[21]; WebSphere Ilog BRMS[22]; JBoss Enterprise BRMS[23]; RuleXpress[24]; SAP NetWeaver BRM[25]; Sapiens eMerge[26]; Ness Usoft[27]; Visual Rules[28]; G2 Plataform[29]; Versata BRMS[30]; PegaRules[31]; InRule[32]; Oracle Business Rules[33], ARIS Business Rule Designer[34].

Com esse conjunto de ferramentas levantado, foi feita uma pesquisa sobre cada ferramenta para adquirir as informações necessárias para responder aos critérios essenciais definidos no passo 2.

A maioria das ferramentas atendeu aos critérios essenciais: no caso das 16 ferramentas avaliadas 10 permitem a disponibilização das regras de negócio como serviços e possuem uma IDE para edição e manutenção das mesmas. No caso, as 10 ferramentas selecionadas são:

- Corticon BRMS;
- Fico Blaze Advisor;
- WebSphere Ilog JRules BRMS;
- JBoss Enterprise BRMS;
- Sapiens eMerge;

---

[19] http://www.corticon.com/Products/Business-Rules-Management-System.php
[20] http://www.esi-knowledge.com/products_logist.aspx
[21] http://www.fico.com/en/Products/DMTools/Pages/FICO-Blaze-Advisor-System.aspx
[22] http://www-01.ibm.com/software/websphere/products/business-rule-management/
[23] http://www.jboss.com/products/platforms/brms/
[24] http://www.rulearts.com/RuleXpress
[25] http://www.sap.com/platform/netweaver/components/brm/index.epx
[26] http://www.sapiens.com/Dev2Go.Web?id=207028
[27] http://www.usoft.com
[28]http://www.visual-rules.com/business-rules-management-enterprise-decision-management.html
[29] http://www.gensym.com/index.php?option=com_content&view=article&id=47&Itemid=54
[30] http://www.versata.com/index2.php?option=com_content&task=view&id=124
[31] http://www.pega.com/Products/RulesTechnology.asp
[32] http://www.inrule.com/products/InRule.aspx
[33] http://www.oracle.com/technology/products/ias/business_rules/index.html
[34] http://www.ids-scheer.com/en/ARIS/ARIS_Platform/ARIS_Business_Rules_Designer/3747.html

- Visual Rules;
- Versata BRMS;
- PegaRules;
- InRule;
- Oracle Business Rules.

Porém, devido a restrições de tempo e ao fato das avaliações detalhadas serem custosas, foi necessário um ajuste nos critérios essenciais, visando reduzir o número de avaliações detalhadas. Para isso foi incluído um critério essencial adicional: o bom posicionamento da ferramenta nas avaliações realizadas por [Sinur, 2005] e [Rymer e Gualtieri, 2008]. A partir deste ajuste, as seguintes ferramentas foram selecionadas para serem avaliadas detalhadamente:

- WebSphere Ilog JRules BRMS;
- InRule;
- Corticon;
- FICO Blaze Advisor;

4. **Avaliação detalhada:** Neste trabalho utilizamos os resultados da avaliação realizada por Azevedo *et al.* [2010a], na qual são avaliadas ferramentas BRMS de acordo com a arquitetura de BRMS para tratar regras de autorização proposto por Azevedo *et al.* [2010c]. A arquitetura é dividida em dois componentes: componente para gestão de regras de autorização, que são responsáveis por cadastro, edição, alteração, versionamento, implantação, simulação de regras, etc.; e componente para execução de regras de autorização, que são responsáveis por aplicar a regra de acordo com os privilégios do usuário que a está executando. Neste trabalho, busca-se uma ferramenta que atenda às características propostas para o componente de gestão e com disponibilização das regras como serviço para execução em um ambiente SOA. Logo, apenas os resultados referentes à gestão de regras de autorização foram utilizados neste trabalho.

   Nela, as 4 ferramentas selecionadas após a aplicação dos critérios essenciais foram avaliadas e os critérios técnicos utilizados atendem ao escopo deste trabalho. Os critérios foram elaborados a partir de critérios propostos em

outros trabalhos de avaliação de ferramentas [Azevedo *et al.* 2008], características das ferramentas enumeradas em comunidades de pesquisa na área [BRG 2000; BRCommunity 2009] e avaliações de ferramentas BRMS [Sinur 2005; Rymer e Gualtieri 2008]. Os critérios técnicos utilizados são apresentados no Apêndice 1 e Apêndice 2 . Como resultado desta a avaliação, considerando apenas as ferramentas selecionadas neste trabalho, foram encontrados os resultados apresentados na Tabela 1.

A Tabela 1 apresenta um resumo dos resultados obtidos com a avaliação das ferramentas. As cores das células estão de acordo com a legenda apresentada na Figura 5. Os percentuais entre parênteses (e marcados com asterisco) representam valores que a ferramenta conseguiria atingir caso fosse atribuída nota máxima para os critérios para os quais não foram encontradas informações para respondê-los. Ou seja, alguns dos critérios receberam nota 0 (zero) pois não havia informações suficientes para respondê-los. Se estes critérios recebessem nota 1, as ferramentas teriam uma melhor pontuação. No entanto, vale ressaltar que os resultados e dúvidas foram enviados para os fornecedores das ferramentas e só foi mantida nota zero para os itens que eles não apresentaram resposta suficiente para avaliá-los.

Tabela 1 – Resumo dos resultados obtidos com a avaliação das ferramentas de Azevedo *et al.* [2010a], considerando componente de gestão

		Documentação	**Laboratório**	
Ferramenta	**Critérios genéricos**	**Gestão**	**Gestão**	**Resultado**
ILOG	80%	61%	60%	65%
Corticon	77%	68% *(90%*)*	X	X
FICO Blaze Advisor	66% *(94%*)*	58% *(88%*)*	47%	54%
InRule	32% *(57%*)*	47% *(51%*)*	47%	43%

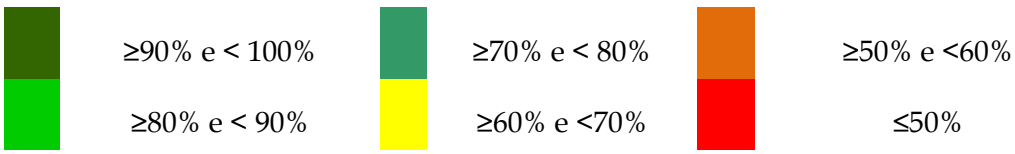


Figura 5 – Legenda para a Tabela 1

3.2 Análise dos resultados

Com relação aos critérios genéricos, a ferramenta Ilog apresenta vários pontos fortes, tais como facilidade de integração de novas versões, bom desempenho atestado, facilidade de extensão, integração com banco de dados, boa documentação e diferentes formas de suporte. Considerando os critérios de gestão, a ferramenta obteve boa pontuação. Ela apresentou pontos positivos importantes em relação às funcionalidades para edição, versionamento, validação e simulação, segurança, gestão de regras em diferentes ambientes e qualidade de regras. Os pontos negativos apareceram em relação aos mecanismos para consultas às regras, importação de usuários da organização, linguagem e forma de armazenamento serem proprietárias, visualização de regras ser limitada e a ferramenta não permitir a combinação de regras.

A ferramenta Corticon apresenta pontos fortes como solidez do fornecedor, escalabilidade e integração com banco de dados e ferramenta de modelagem. Apesar de obter uma boa pontuação nos critérios genéricos e na avaliação da documentação quanto aos critérios de gestão, a avaliação em laboratório não foi executada, pois não foi disponibilizado acesso aos componentes de execução de regras. Dessa forma, a ferramenta só poderia ser utilizada para a gestão sem considerar execução. Foi analisada então a possibilidade de integrar a ferramenta com outra ferramenta de execução. Isto foi descartado porque a Corticon não permite integração com outra ferramenta, pois a forma de armazenamento de regras é proprietária.

A ferramenta Fico Blaze Advisor possui pontos fortes importantes como solidez do fornecedor, flexibilidade para ser estendida, boa documentação e parceiro no Brasil. No entanto, a integração com o banco de dados não é simples e muitas informações importantes não puderam ser obtidas da documentação ou mesmo através de contato com o fornecedor. A ferramenta obteve pontuação razoável como ferramenta de gestão. Muitos critérios não puderam ser respondidos a partir da documentação e o fornecedor não retornou respostas para os mesmos. A ferramenta tem pontos positivos em relação a IDE para edição, capacidade de estender a ferramenta via API para consulta e visualização, validação e simulação de regras e possui mecanismos para publicar regras em diferentes ambientes. Pontos negativos aparecem no fato da ferramenta não permitir importar usuários da empresa e associá-

los às regras, a linguagem e a forma de armazenamento de regras serem proprietárias, e não ter atendidos bem critérios de segurança e qualidade.

Quanto a ferramenta InRule, muitas informações não puderam ser obtidas através da documentação para realizar a avaliação genérica. Logo, a dificuldade em se obter estas informações já dá indícios de não ser interessante o uso desta ferramenta. Ela obteve pontuação ruim quando avaliada como ferramenta para gestão de regras. Como pontos positivos levantados durante a avaliação estão o fato de permitir ser acessada por API para consulta e visualização de regras, permitir a importação de usuários e associação dos mesmos às regras, versionamento, validação e simulação, ter atendido importantes critérios de segurança e possuir mecanismos para publicar regras em diferentes ambientes. Como pontos negativos, a ferramenta não possui controle de concorrência dos usuários que a acessam para edição de regras, mecanismos limitados para consultas às regras, a linguagem e a forma de armazenamento são proprietárias, visualização de forma textual, não permite combinar regras e atendeu de forma ruim os critérios de qualidade de regras.

A partir do resultado e da análise do mesmo, a ferramenta que teve o melhor desempenho dentro da avaliação foi a ILOG BRMS[35], e por isso a mesma será detalhada, apresentando seus principais conceitos e sua arquitetura, no Capítulo 4: deste trabalho. Além disso, a mesma será utilizada no estudo de caso apresentado no Capítulo 5: deste trabalho.

3.3 Resumo

Este capítulo apresentou uma análise das ferramentas BRMS com foco da sua utilização dentro de uma arquitetura orientada a serviços, demonstrando a execução do processo de avaliação e uma análise dos resultados obtidos por essa avaliação.

A avaliação de ferramentas não é trivial, devido a diversos problemas como o grande número de ferramentas presentes e a falta da definição de requisitos. Para atender a essa demanda torna-se necessária a aplicação de uma metodologia para avaliação e seleção de ferramentas.

[35] http://www-01.ibm.com/software/websphere/products/business-rule-management/

A prospecção de ferramentas realizada neste trabalho segue o processo de avaliação de ferramentas proposto por Azevedo *et al.* [2010b]. Este processo compreende as seguintes etapas: Definir escopo de prospecção, Definir critérios essenciais de alto nível, Selecionar ferramentas candidatas, Definir critérios técnicos, Levantar documentação, Avaliar ferramentas, Elaborar lista de problemas e avaliar respostas, Analisar resultados e Apresentar resultados. Neste trabalho utilizamos como base a avaliação realizada por Azevedo *et al.* [2010a].

Como etapa final da avaliação de ferramentas, foi realizada uma análise consolidando os resultados. Esta análise detectou os pontos fortes e fracos de cada ferramenta. Com isso, constatou-se que a ferramenta que seria mais adequada para ser utilizada no estudo de caso apresentado no Capítulo 5: deste trabalho seria a WebSphere Ilog BRMS.

Capítulo 4: WEBSPHERE ILOG JRULES BRMS

Nesta seção será caracterizada a ferramenta escolhida através do processo de avaliação de ferramentas. Primeiramente serão apresentados alguns conceitos básicos que a ferramenta introduz e posteriormente a arquitetura da ferramenta.

4.1 Conceitos da Ferramenta

Serão detalhados nas subseções desta seção alguns conceitos, apresentados por [Ilog, 2010], utilizados pela ferramenta e que são fundamentais para o entendimento e aplicação de uma gestão de regras de negócio utilizando a mesma.

4.1.1 Business Object Model (BOM)

O *Business Object Model* (BOM), ou Modelo de objetos do negócio, permite a edição das regras de uma forma amigável, pois utiliza um vocabulário em linguagem natural. Com esse vocabulário, os usuários de negócio podem descrever a lógica do negócio em uma linguagem de regras de negócio (Figura 6).

O BOM contém as classes e métodos sobre os quais os artefatos das regras atuam. Como um modelo de objetos, o BOM é muito similar a um modelo de objetos JAVA, consistindo em classes agrupadas em pacotes. Cada classe possui um conjunto de atributos, métodos e classes aninhadas.

A Figura 6 exemplifica como é a associação dos elementos da regra com o modelo de objetos.

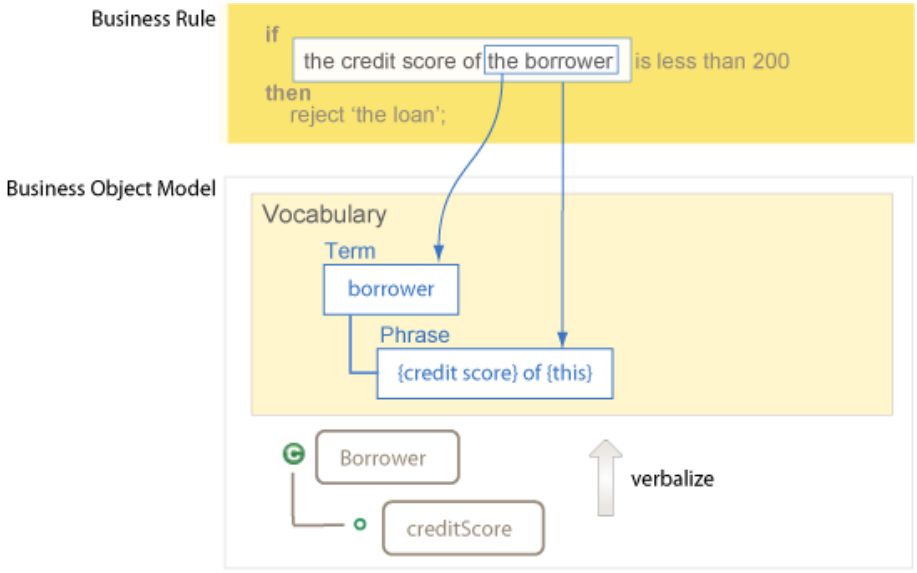


Figura 6 – Utilização dos objetos do BOM pela regra

4.1.2 Execution Object Model (XOM)

O *Execution Object Model* (XOM), ou modelo de execução do objeto, define como cada regra será executada. Ele referencia objetos da aplicação e dados e é a base da implementação do *Business Object Model* (BOM). O projeto de regras referencia um XOM.

O XOM é importado para o projeto de regras a partir de classes JAVA ou XML *Schema* (XSD). Essas duas formas definem dois tipos de XOM: *Java Execution Object Model* e *Dynamic Execution Object Model*. Ou seja, o mapeamento dos objetos da aplicação com os objetos do BOM pode ser feito via classes JAVA ou XML *Schema*.

Através do XOM, o motor de execução pode acessar objetos e métodos da aplicação, que podem ser objetos JAVA, dados XML ou dados de outras fontes. Em tempo de execução as regras que foram escritas baseadas no BOM são executadas de acordo com o XOM, como demonstrado na Figura 7.

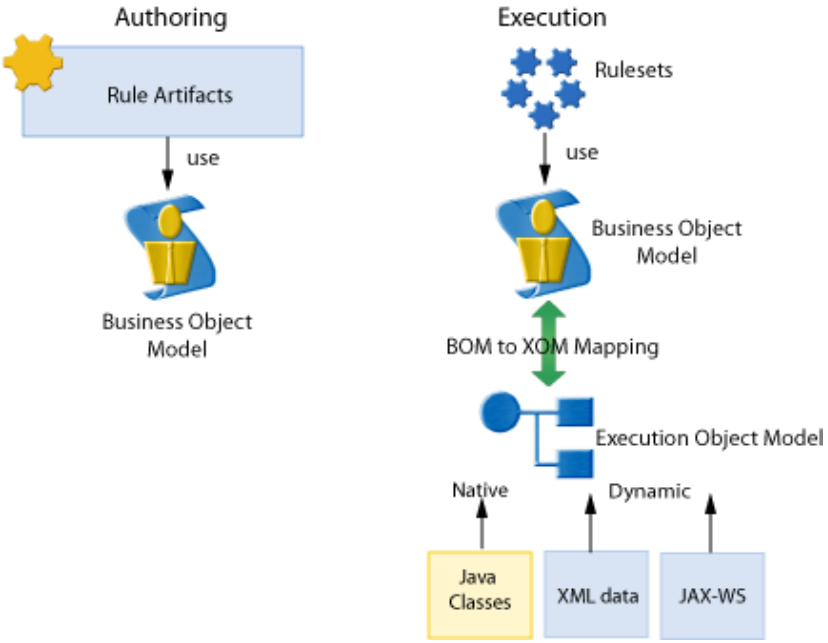


Figura 7 – Execução de regras utilizando o XOM

4.1.3 Ruleset e RuleApp

Um *ruleset* é um conjunto de regras que pode ser executado pelo motor de regras. É um pacote executável que inclui as regras e os artefatos necessários para execução das regras. Um arquivo *ruleset* é gerado a partir da extração de um projeto de regras.

No estágio de desenvolvimento das regras, as regras são escritas e testadas no Rule Studio e depois disponibilizadas como um RuleApp. Um RuleApp é uma unidade de publicação e gerenciamento para o Rule Execution Server, que pode conter um ou mais *rulesets*. O RuleApp geralmente contém *rulesets* que compartilham o mesmo modelo de dados.

Cada *ruleset* dentro de um RuleApp pode ser invocado utilizando o caminho do mesmo. O caminho do *ruleset* age como ponto de entrada para os clientes que querem acessar a lógica do negócio encapsulada no RuleApp. Os RuleApps, após publicados em um servidor de aplicação suportado, podem ser acessado por múltiplos clientes de forma distribuída e concorrente.

4.1.4 Rule Model

O *Rule Model* define os artefatos que serão gerenciados num projeto de regras. Esses artefatos estão associados a propriedades com vários tipos. Os artefatos são:

- *Action Rule*: uma regra de negócio, escrita em BAL (*Business Action Language*), que permite o usuário do negócio escrever a regra em uma linguagem quase natural;
- *BRLRule*: uma regra de negócio, sem nenhuma linguagem associada, que é utilizada como base de todas as regras;
- *Decision Table*: uma tabela de decisão que permite a criação de regras com múltiplas condições de maneira mais fácil de visualizar;
- *Decision Tree*: uma árvore de decisão que permite a visualização da execução da regra de outra forma, no formato de árvore;
- *Function*: uma função que pode ser usada dentro das regras para qualquer tipo de cálculo ou tarefa;
- *Rule Artifact*: qualquer artefato de regra, por exemplo, uma tabela de decisão;
- *Rule Flow*: um fluxo de regra que define a ordem de execução das regras;
- *Rule Package*: um pacote, onde ficam armazenadas as regras;
- *Technical Rule*: uma regra técnica, escrita em IRL (Ilog *Rule Language*) que permite a criação de regras mais complexas.

O *Rule Model* pode ser estendido adicionando novas classes de artefatos, que representam um ou mais artefatos, e novas propriedades. Um exemplo seria definir uma nova classe de artefato que representasse *Decision Table* e *Decision Tree* ao mesmo tempo, definir propriedades para esses artefatos. Além disso, com o *Rule Model* é possível adicionar novas propriedades a artefatos já existentes. Não é possível criar novos tipos de artefatos, apenas definir novas classes de artefatos, ou seja, a partir dos artefatos existentes pode-se agrupar alguns dentro de uma classe.

Por exemplo, criar uma classe de artefato que representa *Decision Table* e *Decision Tree* ao mesmo tempo e definir propriedades para esse artefato.

O *Rule Model* também permite a criação de códigos personalizados que permitem, por exemplo, esconder uma propriedade específica ou chamar algum código após o valor de uma propriedade ser alterado. Nesse caso, o código, que é chamado de callback no JRules, serve apenas para esconder propriedades dos artefatos ou classes de artefatos (Action Rule, BRLRule, Funcion, Rule Flow e etc) do JRules. Por exemplo, após definir a valor da propriedade "Revisão" para TRUE, pode aparecer uma nova propriedade "Estado da Revisão".

O *Rule Model* é descrito por arquivos XML (XSD). O extensiomodel.xsd é o arquivo principal que define a estrutura do *Rule Model*. É possível associar um arquivo (extensiondata.xsd) que define os valores dos tipos *enumerations* (tipo de dados que define um conjunto de valores fixos, ex.: País – Brasil, EUA, Itália, Argentina) e *hierarchies* (semelhante ao *enumerations*, porém com níveis, ex.: Estado -> Cidade : RJ -> Búzios, Petrópolis; SP -> Santos, Guarujá). Esses dois arquivos devem ser registrados no Rule Studio e no Rule Team Server.

As propriedades padrão da ferramenta para cada regra são: effectiveDate (*Date*), expirationDate (*Date*), status (*Status*: *new, defined, validated, rejected, deployable*), priority (*String*).

4.1.5 Template

Um *template* é uma regra de negócio ou tabela de decisão parcialmente escrita, utilizada como ponto de partida para a criação de múltiplas regras ou tabelas de decisão com estruturas e conteúdos semelhantes. Além disso, se for definido no *template,* parte da regra pode ser bloqueado para edição, permitindo apenas inserir informações novas na regra.

Os *templates* são aplicados no momento da criação da regra ou da tabela de decisão, caso o *template* seja modificado posteriormente, as regras terão de ser atualizadas manualmente. Um exemplo de *template* pode ser visto na Figura 8.

Figura 8 – Exemplo de *template*

4.1.6 RuleFlow

O *RuleFlow* define a ordem de execução dos artefatos de regras dentro de uma decisão maior. Para as aplicações, o *ruleflow* é chamado como se fosse apenas uma decisão. No *ruleflow* existirá uma lógica que norteia o fluxo das decisões menores, as quais são identificadas como tarefas e o caminho das decisões como transações. Um exemplo de *ruleflow* é ilustrado na Figura 9.

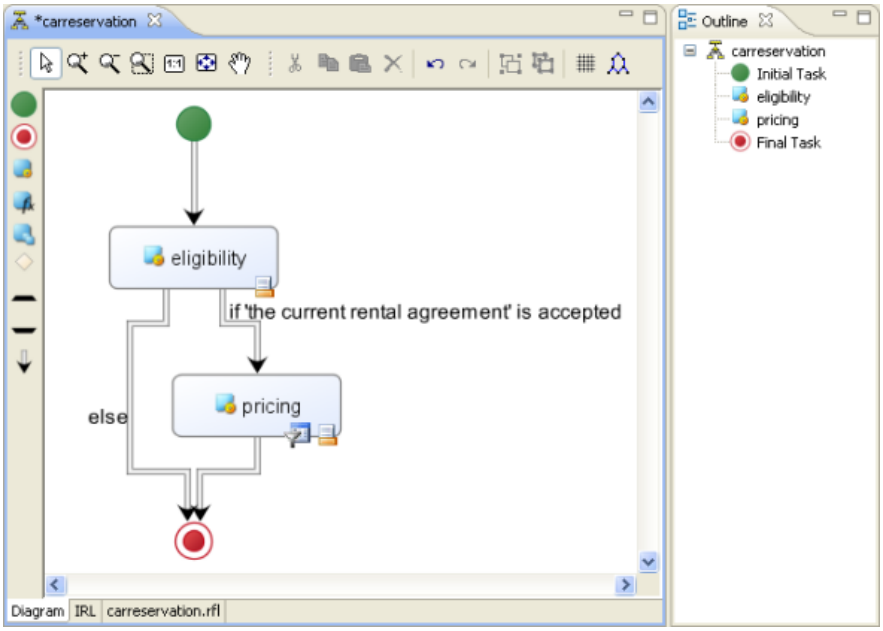


Figura 9 – Exemplo de um ruleflow

4.1.7 Modos de Execução

Os modos de execução determinam como as regras serão disparadas e em qual seqüência. A seguir serão apresentados alguns modos de execução que a ferramenta oferece.

4.1.7.1 RetePlus

O RetePlus [Reteplus, 2010] é a extensão do algoritmo RETE [Forgy, 1982] para a ferramenta ILOG JRules. Ele é o modo de execução padrão da ferramenta.

O modo de execução RetePlus provê meios para que o motor de execução minimize o número de regras e condições que precisam ser avaliadas, definindo quais precisam ser executadas e identificando a ordem em que elas precisam ser executadas.

No RetePlus o motor de execução utiliza uma *working memory* e uma *agenda* para armazenar e manipular objetos da aplicação. A *working memory* armazena as referências para os objetos da aplicação e a *agenda* lista e ordena as instâncias de regras que estão aptas a serem executadas.

A Figura 10 apresenta a operação do RetePlus. O RetePlus executa 3 passos básicos:

- Passo 1 – O RetePlus associa os objetos da aplicação na *working memory* às condições das regras no *ruleset*. Durante esse passo o RetePlus cria uma rede baseada nas relações semânticas entre os testes de condições de regras e os objetos que estão na *working memory*;
- Passo 2 – Para cada associação do Passo 1, uma instância da regra é criada e colocada na *agenda*, que seleciona as regras que serão executadas baseado em alguns princípios de ordenação. Um critério de ordenação é atribuir prioridade às regras. Neste caso, cada regra terá um atributo que define sua prioridade e quanto maior for esse número, maior será a prioridade de execução da regra. Regras de maior prioridade são executadas primeiro. Outro critério para seleção de execução de uma regra é o *refraction*, ou seja, uma regra que já foi executada só pode ser executada novamente se algum novo fato

acontecer (modificação do objeto utilizado pela regra, nova associação de objeto);

- Passo 3 – Por fim, a instância da regra é executada, e após isso a *working memory* é modificada adicionando, removendo ou modificando os objetos.

Este processo continua até que não exista mais nenhuma instância dentro da *agenda* para ser executada. Em cada iteração desse processo as associações são refeitas, pois, para cada instância de regra executada, os dados podem ser modificados. Sendo assim o RetePlus é incremental e orientado a dados.

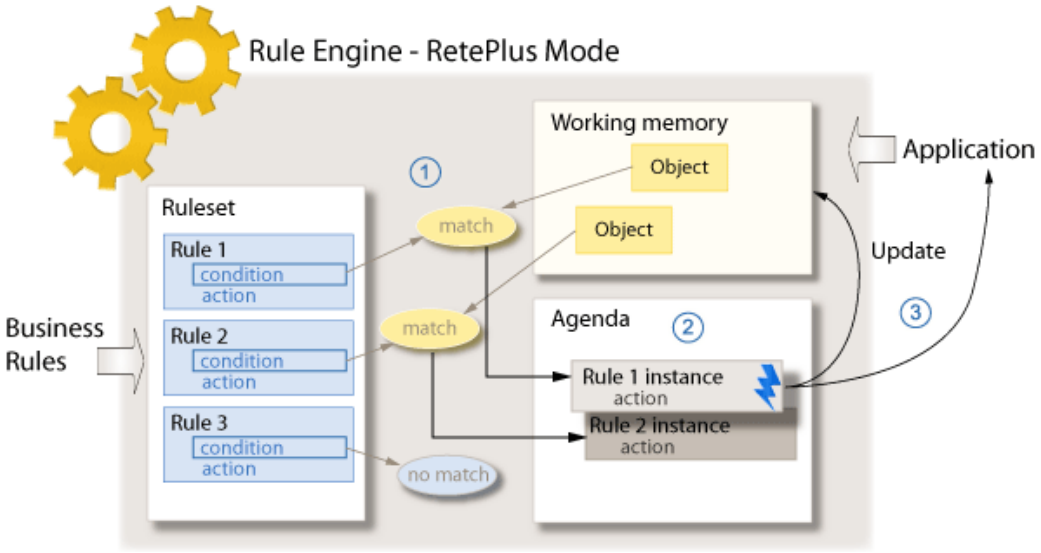


Figura 10 – Operação do modo de execução RetePlus

4.1.7.2 Sequential

O modo de execução *Sequential* [Sequential, 2010] executa na sequência todas as regras aptas a serem executadas dentro de uma tarefa.

Diferentemente do RetePlus, o *Sequential* executa apenas 2 passos:

- Passo 1 – Associa as regras que estão no *ruleset* aos objetos da aplicação na *working memory*.
- Passo 2 – Para cada associação, uma instância da regra é criada e imediatamente executada, podendo alterar o valor de algum atributo ou saída do parâmetro do *ruleset*.

Esses passos são demonstrados na Figura 11.

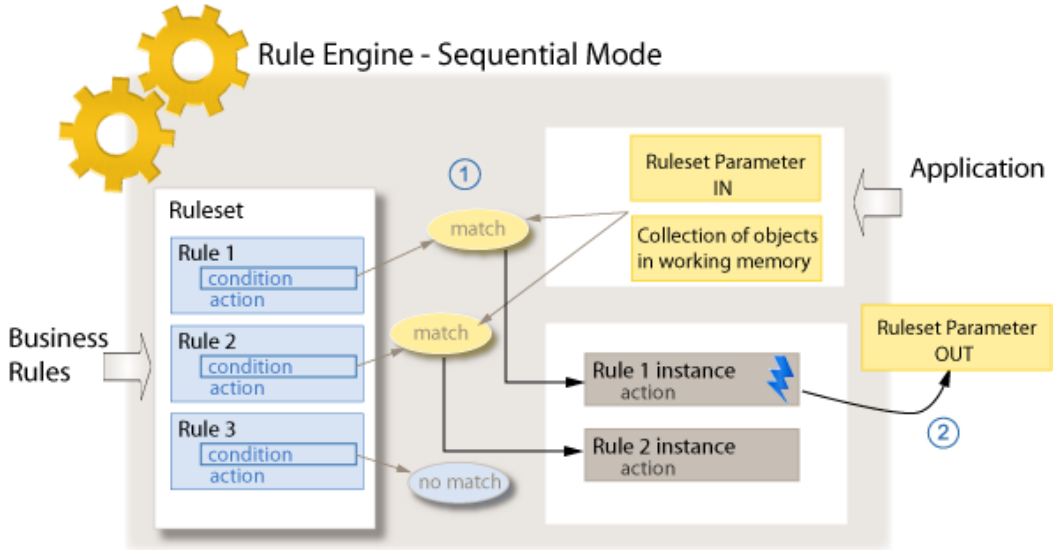


Figura 11 – Operação do Sequential

Dessa forma o *Sequential* é similar a uma pilha de execução, onde a associação das condições das regras com os objetos só acontece uma vez. Devido a sua natureza sistemática, o *Sequential* possui um desempenho melhor nas aplicações de validação, onde a aplicação utiliza regras de negócio para assegurar a validade de um determinado conceito, se ele está sendo aplicado da maneira correta, e aplicações de conformidade *(compliance*), onde a aplicação está preocupada em monitorar, auditar e implantar controles sobre requisitos de conformidade baseados em normas, legislações, procedimentos, boas práticas e etc.

4.1.7.3 FastPath

O FastPath é uma combinação dos modos de execução RetePlus e *Sequential*. Ele é um modo de execução seqüencial, mas ele realiza a associação de forma semelhante ao RetePlus.

Ele executa 2 passos assim como o *Sequential*, porém seu Passo 1 é semelhante ao RetePlus, criando uma árvore baseada na relação semântica entre os testes de condições de regras. Seu Passo 2 é exatamente igual ao Passo 2 do *Sequential*.

4.2 Arquitetura da ferramenta

O Ilog JRules é composto por módulos que operam em diferentes ambientes e trabalham em conjunto para prover as funcionalidades de um BRMS. A Figura 12,

apresentada por [Ilog, 2010], ilustra a arquitetura do Ilog JRules, os papéis propostos para utilizarem a ferramenta e suas atividades, além de como os componentes da arquitetura operam em conjunto.

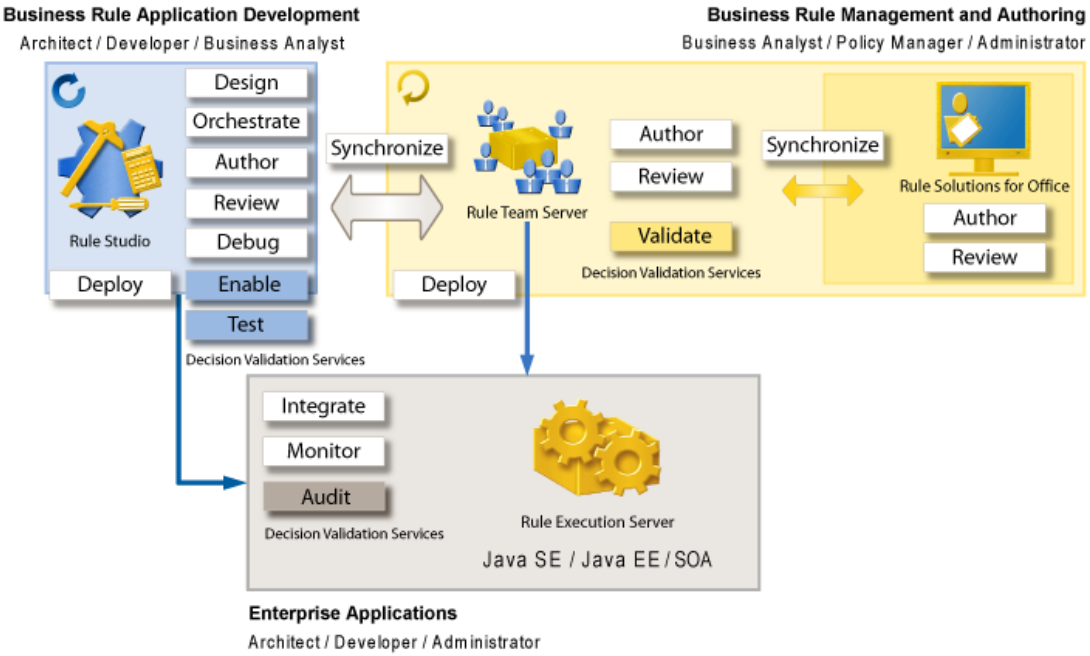


Figura 12 – Arquitetura do WebSphere Ilog JRules BRMS

A arquitetura separa a ferramenta em 4 grandes módulos, cada um deles específico para um conjunto de atividades executadas por diferentes papéis, propostos por [Ilog, 2010], os quais são:

- Arquiteto - é o responsável pelo gerenciamento das regras da organização no ambiente de um BRMS e por assegurar que sua execução seja feita da melhor forma possível.

- Desenvolvedor – é o responsável pelo desenvolvimento, teste, depuração e implantação de aplicações de regras de negócio, ou seja, aplicações (ou sistemas) que utilizam as regras de negócio gerenciadas pelo Ilog JRules para aplicar políticas do negócio.

- Analista de negócio - atua como uma ponte entre o pessoal de negócio e de TI. Ele formaliza as políticas da organização em regras de negócio de forma que possam ser entendidas pelos desenvolvedores e valida a implementação dessas regras.

- Gestor de Políticas - é um especialista do negócio, responsável pelas políticas de negócio dentro da organização. No processo de criação da regra, ele extrai as informações necessárias (por exemplo, a partir de documentos) para que o Analista de negócio possa criar os elementos de vocabulário que serão utilizados no desenvolvimento das regras de negócio.
- Administrador - é o responsável pela configuração e manutenção do ambiente de gestão de regras, de auditoria e, se necessário, ele é o responsável pela restauração do ambiente de regras.

4.2.1 Desenvolvimento de Aplicações de Regras de Negócio

O módulo de Desenvolvimento de Aplicações de Regras de Negócio é utilizado pelos Desenvolvedores, Arquitetos e Analistas de negócio para a execução das seguintes atividades:

- Desenvolvimento de aplicações de regras de negócio - esta atividade consiste na criação de um *Rule Project* no Rule Studio, um tipo de projeto do Eclipse dedicado ao desenvolvimento de aplicações de regras de negócio. Além disso, também é desenvolvido o vocabulário que será utilizado na criação das regras. Esse vocabulário é baseado no modelo de objeto de negócio (BOM – *Business Object Model*).
- Orquestração da execução de regras – corresponde à definição de como as regras serão executadas (definindo um *Ruleflow*). Dentro do projeto, o Arquiteto e o Desenvolvedor organizam as regras de negócio em *packages* e definem como será feita a persistência dessas regras dentro de um sistema SCC (*Source Code Control*).
- Criação de regras de negócio – os Desenvolvedores e Arquitetos criam maneiras de facilitar a criação das regras pelos autores de regras, escrevendo regras mais complexas, criando *templates* e definindo categorias de vocabulários.

- Revisão dos artefatos de regras e análise da consistência e completude das regras – os Desenvolvedores utilizam o Rule Studio para realizarem uma análise das regras, executando consultas para certificar que a regra está consistente. Além disso, o Rule Studio permite a criação de relatórios.

- Teste e validação das regras – os Desenvolvedores utilizam o Rule Studio para realizar o *troubleshoot* dos Serviços de Validação de Decisão (DVS - *Decision Validation Services*), o qual é apresentado em mais detalhes na seção 4.2.4.

- Depuração das aplicações de regras de negócio – os Desenvolvedores utilizam o Rule Studio para depurarem as regras em um ambiente de testes.

- Implantação das regras – pode ser feita no Rule Studio de duas formas: tornar as regras disponíveis imediatamente para o propósito de testes ou implantação em estágios para executar as regras em ambientes controlados.

No módulo de Desenvolvimento de Aplicações de Regras de Negócio, o Rule Studio é integrado ao Eclipse para o desenvolvimento de aplicações de regras de negócio. Isto pode beneficiar o desenvolver, que pode desenvolver em paralelo seus projetos de regras de negócio e de aplicações em Java. Além disso, o Rule Studio fornece ferramentas para sincronização com o Rule Team Server, que é a ferramenta para gestão de regras voltada ao usuário do negócio. O Rule Team Server é utilizado no módulo de Gestão e Autoria de Regras de Negócio pelo Usuário do Negócio.

No módulo de Desenvolvimento de Aplicações de Regras de Negócio, além do Rule Studio, o Rule Execution Server e o *Decision Validation Services* também são utilizados, respectivamente para configurar métodos de execução/otimização das regras e para testes e simulações das regras.

Além das atividades descritas anteriormente, [Ilog, 2010] propõe a criação dos seguintes papéis no contexto do módulo de Desenvolvimento de Aplicações de Regras de Negócio:

- Arquiteto: atua no mapeamento dos pacotes lógicos do Rule Team Server para pacotes físicos no Rule Studio e na definição de *rulesets* (pacote executável de artefatos de regras e outros artefatos) dentro da aplicação de regras de negócio.

- Desenvolvedor: cria a implementação do vocabulário de regras, faz a depuração da execução das regras, testa e valida os *rulesets*, disponibliza no Rule Team Server os testes e simulações para os usuários do negócio, otimiza as regras, cria regras complexas que o usuário do negócio não poderia criar e integra o motor de execução das regras na aplicação para poder testar a execução dos *rulesets*. Essa integração pode ser feita com a aplicação utilizando a API do Rule Engine ou se a escolha for por utilizar o Rule Execution Server, a integração pode ser feita de diversas formas: geração de código (POJOs), criação de serviços web (SAAJ[36], JAX-WS[37]), geração de SCA (*Service Component Architecture*[38]) e geração de SSP (*Scenario Service Provider*) Setup.

- O Analista de negócio define o vocabulário que será utilizado nas regras, escreve e organiza as políticas do negócio em regras, para que o gestor de políticas possa mantê-las. Além disso, o Analista de negócio também inspeciona a execução das regras (por exemplo, que valores elas estão assumindo, que decisões estão sendo tomadas etc).

4.2.2 Gestão e Autoria de Regras de Negócio pelo Usuário do Negócio

O módulo de Gestão e Autoria de Regras de Negócio pelo Usuário do Negócio é utilizado pelos Analistas de negócio e Gestores de políticas. Através do Rule Team Server, eles mantêm os projetos de regras, criam, analisam, validam e implementam as regras de negócio.

---

[36] http://java.sun.com/webservices/docs/2.0/tutorial/doc/
[37] http://java.sun.com/webservices/docs/2.0/tutorial/doc/
[38] http://www.oasis-opencsa.org/sca

Neste módulo, a principal ferramenta é o Rule Team Server, um ambiente de trabalho colaborativo onde os usuários do negócio podem trabalhar no projeto de regras. Este ambiente possui controle de acesso e suporta múltiplos usuários. Além do Rule Team Server, o Rule Solutions for Office é utilizado neste módulo para que os usuários possam trabalhar nas regras de negócio utilizando ferramentas do Microsoft Office 2007.

As atividades executadas neste módulo são:

- Sincronização – Quando os Desenvolvedores tornam a aplicação de regras disponível, eles realizam a sincronização dos projetos de regras com o Rule Team Server, para que os Gestores de políticas e Analistas de negócio possam realizar modificações e manter essas regras. Os usuários de negócio e Desenvolvedores passarão então a compartilhar os projetos de regras. A sincronização também possibilita aos usuários do negócio trabalhar no Rule Solutions for Office (*plug-in* para o Microsoft Office 2007), e depois sincronizarem seus projetos com o Rule Team Server.

- Criação de regras – Gestores de políticas e Analistas de negócio criam e editam regras de negócio tanto no Rule Team Server quanto no Rule Solutions for Office. O trabalho deles é armazenado em um repositório que controla as versões, históricos e acessos múltiplos.

- Validação das regras – O Rule Team Server e os Serviços de Validação de Decisão (DVS) atuam junto na validação das políticas. Com isso, é possível criar suítes de testes que podem rodar no Rule Team Server, seus resultados podem ser inspecionados e, se necessário, pode se fazer mudanças nos testes e nos dados dentro do mesmo.

- Revisão do projeto de regras – Os Gestores de políticas e Analistas de negócio podem acompanhar o andamento do projeto, verificar as regras que já foram escritas, gerar relatórios, revisar modificações feitas por todos os participantes do projeto e fazer consultas das regras.

- Implantação das regras – Administradores, Gestores de políticas e Analistas de negócio podem realizar a implantação das regras de negócio do Rule Team Server para o Rule Execution Server, dentro de um ambiente de testes. Não é recomendável que esses usuários realizem a implantação no ambiente de produção.

No contexto do módulo de Gestão e Autoria de Regras de Negócio pelo Usuário do Negócio:

- O Gestor de Políticas cria e atualiza as regras, revisa a orquestração da execução das regras dentro do fluxo de regras, reporta o estado das políticas de negócio, testa as regras para certificar que elas estão escritas de maneira correta e as valida para saber se elas estão atingindo o objetivo esperado.
- O Administrador instala e configura o Rule Team Server e os *Decision Validation Services* (DVS) e gerencia o acesso aos serviços de regras.
- O Analista de negócio é o responsável pela sincronização das regras do Rule Solution for Office para o Rule Team Server e também pode atuar na criação, edição, revisão, teste e implantação das regras.

4.2.3 Integração, Monitoramento e Auditoria nas Aplicações da Organização

O módulo Integração, Monitoramento e Auditoria nas Aplicações da Organização diz respeito à forma como as regras implementadas irão se integrar dentro do ambiente da organização. Nesta etapa é definida a execução das regras, como elas estarão disponíveis e como estarão disponíveis os *Decision Validation Services* (DVS).

O Rule Execution Server e os *Decision Validation Services* são as ferramentas utilizadas neste módulo. O Rule Execution Server é um ambiente controlado e monitorado para a execução e implantação de regras de negócio. Os *Decision Validation Services* permitem que os Administradores gerenciem, façam

backup das regras e removam as que são desnecessárias. Além disso, os *Decision Validation Services* armazenam e geram relatórios detalhados da execução.

As atividades executadas neste modulo são:

- Implantação das regras – Os Desenvolvedores realizam a implantação através do Rule Studio de duas formas: implantação "quente", quando as regras tornam-se diretamente disponíveis, e implantação em estágios, quando as regras são implantadas em um ambiente controlado. Os Administradores também podem realizar a implantação das regras através do Rule Team Server ou do console do Rule Execution Server.

- Integração das regras – Desenvolvedores e Arquitetos criam arquivos executáveis, chamados *rulesets*, contendo os itens do projeto de regras. Além disso, eles escrevem o código necessário para executar esses *rulesets* no Rule Execution Server. Eles utilizam a sessão apropriada e podem usar *Decision Validation Services* (DVS) no Rule Execution Server.

- Monitoramento da execução das regras – Administradores utilizam o console do Rule Execution Server, Ant *scripts*, Enterprise Management Tools (Ex.: IBM Tivoli ou HP OpenView) e JMX MBeans para monitorar a execução de *rulesets* dentro do Rule Execution Server. Eles podem criar versões de Backup para possíveis restaurações e podem coletar informações estatísticas sobre o desempenho. A execução dos *Decision Validation Services* pode ser monitorada e arquivada em um *Decision Warehouse*.

- Auditoria – Os *Decision Warehouse* armazenam informações que podem ser usadas para consultas *ad-hoc* e auditoria das políticas do negócio. Auditores podem consultar os resultados armazenados e até mesmo abrir regras que estão na produção através do Rule Team Server.

No contexto do módulo de Integração, Monitoramento e Auditoria nas Aplicações da Organização, [Ilog, 2010] propõe os seguintes papéis:

- O Administrador instala e configura o Rule Execution Server, implanta e re-implanta as regras, gerencia o acesso do usuário aos servidores de execução, resolve problemas de transações passadas, provê os dados para o rastreamento das políticas que estão sendo aplicadas, gerencia o *Decision Warehouse* e gera relatórios detalhados de execução.
- Os Desenvolvedores são os responsáveis por implementar o código que chamará a execução das regras.
- Os Arquitetos são responsáveis por assegurar a implantação correta das regras nas aplicações da organização e por incentivar o reuso das regras.

4.2.4 Validação de Regras de Negócio

É necessário testar as regras de negócio sem que uma aplicação tenha que invocar as regras. Essa necessidade torna-se ainda mais evidente quando as regras serão utilizadas em um contexto de uma arquitetura orientada a serviços, onde essas regras serão utilizadas em múltiplas instâncias dos processos da organização. Para isso a ferramenta provê os *Decision Validation Services* (DVS).

Decision Validation Services (DVS) é o módulo do JRules que permite o teste das regras pelos desenvolvedores, usuários do negócio e engenheiros de controle de qualidade. Diferentemente dos outros módulos, o DVS não é um módulo que funciona separadamente: ele é integrado ao Rule Studio, Rule Team Server e Rule Execution Server.

A validação de regras de negócio envolve a:

- Criação de um projeto de regras;
- Criação de cenários que representem os casos de uso para validar o comportamento das regras;
- Criação de *test suites* que comparem os resultados esperados com os resultados obtidos na aplicação dos cenários;

- Criação de simulações que permitam a geração de relatórios com informações relevantes ao negócio, baseado em indicadores de desempenho (*Key Performance Indicators* – KPIs);
- Customização de testes para:
  - Apresentar dados nos diferentes cenários;
  - Destacar os resultados mais importantes nos relatórios gerados.

A Figura 13 mostra como o DVS é integrado aos diferentes módulos do BRMS. No Rule Studio os desenvolvedores habilitam e testam os artefatos que os usuários de negócio necessitam para poderem validar as regras. Utilizando uma conexão com o Rule Execution Server os usuários de negócio validam as regras executando os cenários. Usuários do Rule Studio e Rule Team Server podem sincronizar os projetos para poder facilitar o desenvolvimento e depuração das regras. Um arquivo DVS é projetado para passar informações do Rule Team Server para o Rule Studio, permitindo a depuração pelos desenvolvedores.

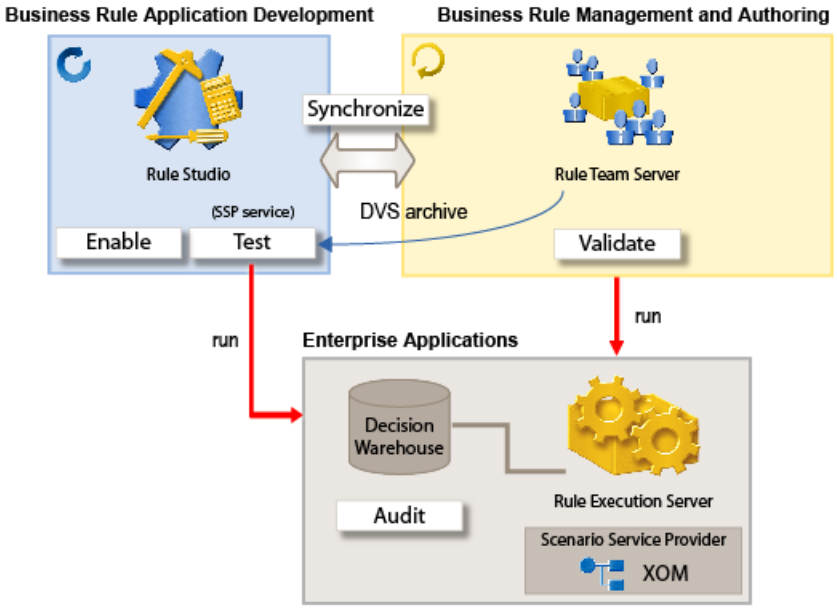


Figura 13 - Decision Validation Service integrado ao BRMS

4.3 Disponibização de regras como serviços

Após as regras serem desenvolvidas e definidas dentro da ferramenta Ilog Jrules, é necessário integrar a aplicação ao motor de execução das regras. Um importante componente para execução das regras é o Rule Execution Server, que é o

módulo da Ilog que permite que as regras sejam executadas, controladas e monitoradas. A ferramenta Ilog possibilita que o motor de execução das regras seja integrado às aplicações das seguintes formas:

- Dentro de um cenário de teste no Rule Studio: com a possibilidade de criar um *Java Project for Rules* e com os templates fornecidos é possível instanciar o motor de execução;
- Utilizando a API do motor de execução: em uma aplicação Java, é possível utilizar chamadas da API para executar as regras;
- Trabalhar com o Rule Execution Server, utilizando os seguintes templates de geração de código:
  - POJO (*Plain Old JavaObject*): provê as classes Java necessárias para executar um ou mais *rulesets* no Rule Execution Server;
  - Web Service: permite gerar um *web decision service* ou um *monitored transparent decision service*. Com esse *template,* as regras podem ser disponibilizadas como *web services*, podendo ser incorporadas em uma arquitetura orientada a serviço. Esse gerador utiliza a API JAX-WS RI 2.1[39];
  - *Service Component Architecture* (SCA) *generator*: provê os arquivos necessários para declarar e executar um componente SCA utilizando uma RuleApp tanto em JAVA SE quanto JAVA EE. Esta implementação é baseada no Tuscany[40];
  - *Scenario Service Provider* (SSP) *Setup generator*: permite a geração de arquivos Ant para atualizar uma aplicação SSP com os XOMs utilizados pelo RuleApp.

A Figura 14 demonstra as possibilidades de geração de código para integração das regras.

[39] https://jax-ws.dev.java.net/
[40] http://tuscany.apache.org/

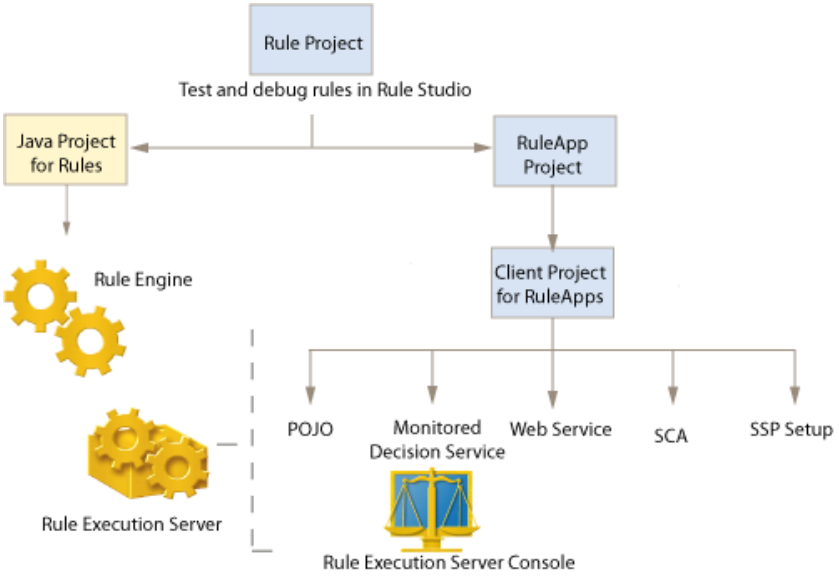


Figura 14 – Possibilidades de geração de código para integração das regras

Neste trabalho, iremos exemplificar a disponibilização de regras como *web service*. Por isso utilizaremos o *template* de geração de código de *web service*.

A ferramenta WebSphete Ilog JRules [ILOG, 2010] apresenta 3 possibilidades para disponibilização das regras como serviço: *web decision service*, *monitored transparent decision service* e *hosted transparent decision service*.

4.3.1 *Transparent Decision Services*

Um serviço de decisão transparente (*transparent decision service*) é tecnicamente um *web service* com capacidade de gerenciamento que utiliza JMX MBeans[41]. Um serviço de decisão transparente guia a execução das regras. Com os serviços de decisão transparentes os usuários podem acessar o Rule Execution Server através de um *web service*, ao invés de acessar diretamente. Os usuários não precisam saber como o serviço foi implementado, tudo o que eles precisam saber é que o serviço pode ser acessado por HTTP e formatos XML (SOAP) [Ilog, 2010].

A implementação de serviços de decisão transparentes baseia-se no conceito de separar a atividade de tomada de decisão e deixá-lo como um serviço *standalone* que pode ser acessado por outros serviços. Quando um processo ou aplicação deve realizar uma decisão, ele acessa o serviço de decisão e especifica qual *ruleset* deve

[41] http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

ser executado. O motor de execução executa o *ruleset* e retorna a decisão ou valor apropriado.

Um passo importante no desenvolvimento de um serviço de decisão transparente consiste na definição da interface entre a regra de negócio e do processo ou aplicação que realiza a chamada. É necessário saber as entradas requisitadas para se executar a regra e os dados retornados como resultado. A interface é definida pelas decisões do negócio que serão automatizados pelas regras de negócio, onde cada decisão corresponde a um *ruleset*, e pelos parâmetros de entrada e saída de cada decisão, que podem ser de dois tipos JAVA ou XML. É importante que um serviço de decisão não seja dependente de um único processo. e então, é preciso pensar nos parâmetros de forma que eles possam ser executados por diversos processos.

A Ilog JRules apresenta dois tipos de serviços de decisão transparente:

- *Hosted transparent decision service:* Essencialmente um *ruleset* publicado como um *web service*. Ele é integrado ao Rule Execution Server e instalado no mesmo servidor de aplicação;
- *Monitored transparent decision service*: Gerado pelo Rule Studio, ele pode ser publicado no mesmo servidor de aplicação do Rule Execution Server, mas não é integrado a ele. O serviço de decisão é independente do Rule Execution Server, mas o acessa para executar as regras.

Além desses dois tipos de serviços, a Ilog JRules permite a disponibilização dos *rulesets* como *web decision services*, que são praticamente iguais ao *monitored decision services*, porém com a utilização do mesmo o Rule Execution Server não consegue armazenar informações sobre sua execução.

4.3.1.1 Hosted Transparent Decision Service

Um serviço de decisão transparente hospedado (*Hosted Transparent Decision Service*) é um componente de execução ligado a um caminho de *ruleset* com um JMX *management bean*. Para utilizar o serviço de decisão transparente hospedado é necessário instalar o Rule Execution Server no servidor de aplicação, e então

publicar o arquivo do serviço de decisão no mesmo servidor. A Figura 15 mostra a arquitetura de um serviço de decisão transparente hospedado.

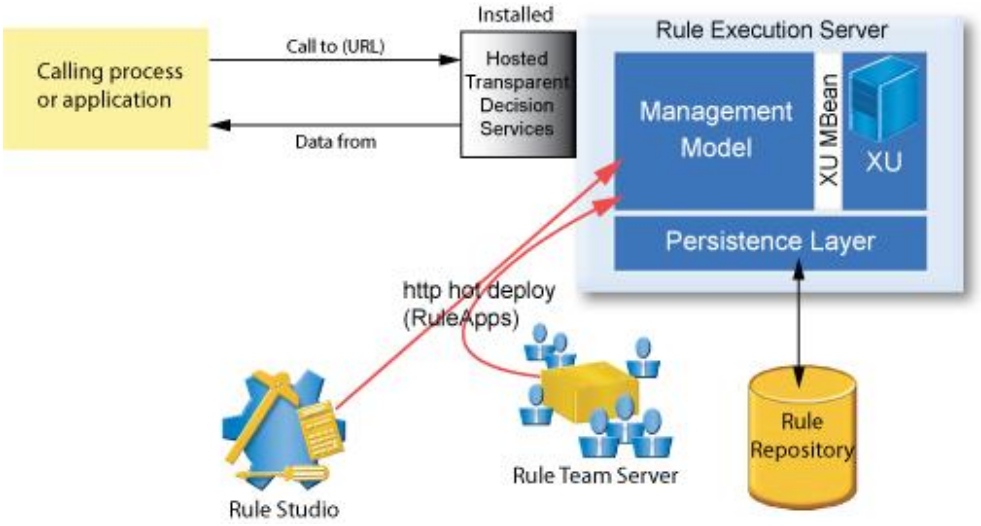


Figura 15 – Arquitetura para um serviço de decisão transparente hospedado

Os serviços de decisão transparentes hospedados provêem: a declaração do serviço de decisão, o caminho do *ruleset* que ele utiliza, estatísticas do serviço de decisão (número de invocações bem sucedidas e com erros, tempo médio de processamento, o último tempo de processamento e data de inicialização) e metadados (autor,versão, nome e descrição do serviço, nome do computador e endereço IP e número de versão dos componentes de execução do JRules).

Com o serviço de decisão transparente hospedado, o Rule Execution Server automaticamente expõe todos os *rulesets,* que utilizam XML Schema (.xsd) ou JAVA XOM com apenas tipo simples, como *web services*. O Serviço de decisão automaticamente gera um arquivo WSDL (*Web Service Description Language*) para cada *ruleset* disponibilizado. O WSDL e os arquivos do *schema* são utilizados pelo mecanimso de XML *binding*. O serviço de decisão transparente hospedado suporta os seguintes parâmetros XML e tipos Java (Tabela 2):

Tabela 2 – Tipos suportados pelos serviços de decisão transparentes

Parâmetros XML	Tipos JAVA
`int`	`int`
`short`	`short`
`long`	`long`

Parâmetros XML	Tipos JAVA
double	double
boolean	boolean
int	java.lang.Integer
short	java.lang.Short
long	java.lang.Long
double	java.lang.Double
string	java.lang.String
boolean	java.lang.Boolean
dateTime	java.util.Date
integer	java.math.BigInteger
decimal	java.math.BigDecimal

Basicamente os serviços de decisão transparentes hospedados compreendem as seguintes etapas: geração do arquivo WSDL, que pode ser utilizado para geração do cliente do *web service*, execução do *web service* e monitoramento do serviço de decisão transparente.

4.3.1.2 Monitored Transparent Decision Service

Um serviço de decisão transparente monitorado expõe as informações de gerenciamento e execução do *web service* ao Rule Execution Server. Ele é criado através do wizard do Rule Studio (*Client Project for RuleApps*) e é instalado no servidor de aplicação. A Figura 16 apresenta a arquitetura da ferramenta para os serviços de decisão transparentes monitorados.

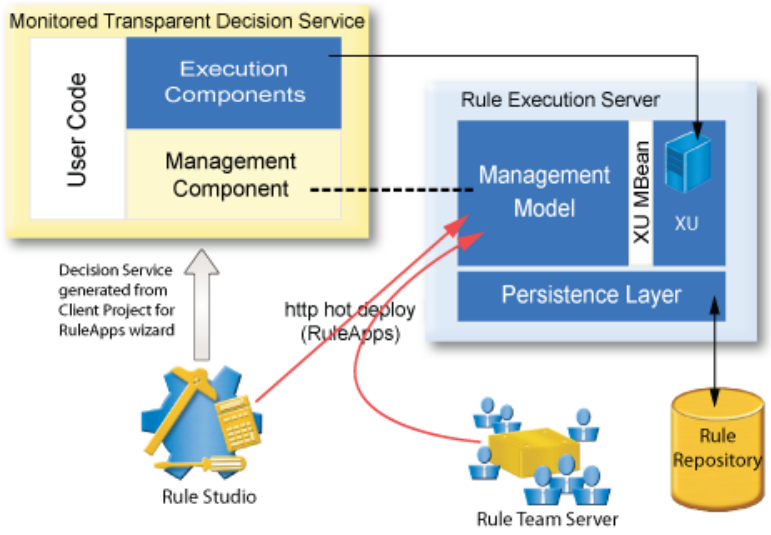


Figura 16 – Arquitetura da Ilog JRules para os serviços de decisão transparentes monitorados

Diferentemente do serviço de decisão transparente hospedado, o serviço de decisão transparente monitorado é independente do Rule Execution Server. Os componentes de execução do serviço de decisão utilizam a unidade de execução (XU – *Execution Unit*) do Rule Execution Server e o componente de gerência é compartilhado com o Rule Execution Server.

O Rule Execution Server provê as seguintes informações sobre a gestão e execução dos serviços de decisão transparentes monitorados: caminho do *ruleset*, estatísticas de execução (número de invocações bem sucedidas e com erros, tempo médio de processamento, o último tempo de processamento e data de inicialização), metadados (autor,versão, nome e descrição do serviço, nome do computador e endereço IP e número de versão dos componentes de execução do JRules), metadados de *rulesets* e RuleApps compátiveis, nível de depuração das regras, auditoria e *tracing* e nome/valor de atributos customizáveis.

Além dos tipos de dados suportados pelos serviços de decisão transparentes hospedados (XML e tipos simples de JAVA), o serviço de decisão transparente monitorado permite a utilização de objetos Java como parâmetros.

Utilizando o wizard de *Client Project for RuleApps* é possível criar tanto os serviços de decisão transparentes monitorados como os *web decision services*. Após a execução do wizard são criados dois projetos: o projeto do *web service*, com as classes Java e um arquivo de build para publicar o serviço, o RuleApp e o projeto do cliente que contém as classes necessárias para consumir o serviço.

Por não restringir o uso de objetos como parâmetro para as regras, permitir monitorar os serviços, mesmo que esses estejam independentes do Rule Execution Server, e por disponibilizar um wizard para a criação do mesmo, decidiu-se por implementar os serviços do estudo de caso como serviços de decisão transparentes monitorados (*Monitored Transparent Decision Services*).

4.4 Resumo

Este capítulo apresentou os principais conceitos e arquitetura utilizados pela ferramenta WebSphere Ilog JRules BRMS, além de apresentar como as regras de negócio são disponibilizadas como serviços.

A ferramenta utiliza os conceitos de *Business Object Model* (BOM) e *Execution Object Model* (XOM) para construir o modelo de dados que é utilizado na criação das regras de negócio. Além disso, são apresentados os conceitos de *Rule Model, Template, Ruleset, RuleApp* e *RuleFlow*. O *Rule Model* define os artefatos que serão gerenciados num projeto de regras e permite que esses artefatos tenham suas propriedades estendidas. O *Template* é uma regra de negócio ou tabela de decisão parcialmente escrita, utilizada como ponto de partida para a criação de múltiplas regras ou tabelas de decisão com estruturas e conteúdos semelhantes. O *ruleset* é um conjunto de regras que pode ser executado pelo motor de execução. Um RuleApp é uma unidade de publicação e gerenciamento para o Rule Execution Server, que pode conter um ou mais *rulesets.* Já o *RuleFlow* define a ordem de execução dos artefatos de regras dentro de uma decisão maior. Para as aplicações o *ruleflow* é chamado como se fosse apenas uma decisão. Foram apresentados também os modos de execução utilizados pela ferramenta, no caso o RetePlus, *Sequential* e FastPath.

Os módulos que compõem a arquitetura da ferramenta também foram apresentados, juntamente com as atividades e papéis que estão presentes neles.

O módulo de Desenvolvimento de Aplicações de Regras de Negócio é utilizado pelos Desenvolvedores, Arquitetos e Analistas de negócio para desenvolver aplicações de regras de negócio, criar, orquestrar, testar e validar as regras de negócio e também para depurar as aplicações de regras, rever os artefatos e implantar as regras.

O módulo de Gestão e Autoria de Regras de Negócio pelo usuário do Negócio é utilizado pelos Analistas de negócio e Gestores de políticas. Basicamente, eles executam as atividades de criação, validação, implantação das regras, sincronização e revisão do projeto de regras.

O módulo Integração, Monitoramento e Auditoria nas Aplicações da Organização diz respeito à forma como as regras implementadas irão se integrar dentro do ambiente da organização, nesta etapa são definidos a execução das regras, como elas estarão disponíveis e como estarão disponíveis os *Decision Validation Services* (DVS).

Decision Validation Services (DVS) é o módulo do JRules que permite o teste das regras pelos desenvolvedores, usuários do negócio e engenheiros de controle de qualidade. Diferentemente dos outros módulos, o DVS não é um módulo que funciona separadamente: ele é integrado ao Rule Studio, Rule Team Server e Rule Execution Server.

A ferramenta Ilog JRules possibilita a disponibilização das regras de negócio como serviços. Essa disponibilização pode ser feita de 3 maneiras, através de serviços de decisão transparentes, hospedados ou monitorados, e *web decision service*. Os serviços de decisão transparentes são tecnicamente um *web service* com capacidade de gerenciamento, que utiliza JMX MBeans. Quando um processo ou aplicação deve realizar uma decisão, ele acessa o serviço de decisão e especifica qual ruleset deve ser executado.

As diferenças entre os serviços de decisão transparente hospedado e monitorado são basicamente a dependência do Rule Execution Server, no qual o serviço de decisão transparente monitorado não depende do Rule Execution Server, além da aceitação de objetos Java como parâmetro dos rulesets. Além disso, para a criação dos serviços de decisão transparentes monitorados é possível utilizar o *wizard* de Client Project for RuleApps do RuleStudio. O *web decision service* é similar ao serviço de decisão transparente monitorado, porém ele não permite o armazenamento das informações de execução pelo Rule Execution Server.

Capítulo 5: ESTUDO DE CASO

O objetivo deste capítulo é demonstrar o uso da ferramenta WebSphere Ilog JRules para desenvolver regras de negócio e disponibilizá-las como serviços em uma arquitetura orientada a serviços. Dessa forma, permitindo avaliar a viabilidade do uso de uma BRMS para execução de regras de negócio em SOA. Além disso, um ESB (*Enterprise Service Bus*) foi utilizado para fazer o roteamento entre o cliente (consumidor) e os serviços de regras de negócio (provedor).

Para isso foi utilizado um processo de negócio fictício [Diirr *et al.*, 2010], do qual foram extraídas algumas regras de negócio que foram especificadas na ferramenta e posteriormente disponibilizadas como serviços. Todo o processo de desenvolvimento e disponibilização das regras de negócio é detalhado nas seções deste capítulo.

As ferramentas que apoiaram o desenvolvimento desse estudo de caso foram:

- WebSphere Ilog JRules BRMS[42]: BRMS que utilizado para criação, publicação e disponibilização das regras de negócio como serviço;
- Oracle Service Bus 10gR3[43]: Barramento utilizado para mediar a execução dos serviços de regras;
- Apache Tomcat 6.0[44]: Servidor de aplicação onde foi instalado o Rule Execution Server e onde os serviços foram publicados;
- WebLogic Server 10gR3[45]: Servidor de aplicação onde foi instalado o Oracle Service Bus;
- PostgreSQL 8.3[46]: SGBD utilizado para armazenar o cadastro dos clientes;
- Netbeans IDE 6.9[47]: IDE utilizada para desenvolvimento da interface gráfica do cliente consumidor dos serviços.

[42] http://www-01.ibm.com/software/websphere/products/business-rule-management/
[43] http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html
[44] http://tomcat.apache.org/
[45] http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html
[46] http://www.postgresql.org/
[47] http://netbeans.org/

5.1 Levantamento de regras

A partir de uma análise feita no modelo de processo "Analisar pedido de crédito" [Diirr *et al*., 2010], foram levantadas as seguintes regras de negócio:

- Cadastro de cliente desatualizado: O cadastro do cliente está desatualizado se o telefone ou endereço ou renda informados na proposta de crédito forem diferentes das informações do cliente existentes na base de dados.
- Cliente novo: Um cliente é novo se não existir cadastro do cliente com mesmo CPF que o informado na proposta de crédito.
- Limite de crédito: O limite de crédito do cliente é igual a 20% da sua renda menos o valor mensal referente às parcelas ainda em aberto dos créditos anteriormente concedidos ao cliente.
- Aprovação de crédito: Se limite de crédito do cliente for maior ou igual ao valor a ser pago para cada parcela da proposta de contrato, então o crédito é aprovado; senão, o crédito não é aprovado.
- Comprometimento do limite de crédito: No comprometimento do crédito para o cliente, a situação de crédito concedido deve ser igual a comprometido.
- Cálculo da taxa de alteração do contrato: O valor da taxa de alteração do contrato é de R$0,33 × número de alterações feitas no contrato.
- Cálculo da taxa de entrega: A taxa de entrega do contrato é calculada da seguinte forma: distância em km × R$0,01, onde distância em km significa a quantidade de quilômetros que o CEP do destinatário está de distância do centro de distribuição.
- Cálculo da taxa de impressão: A taxa de impressão é igual a R$3,17 por contrato impresso.
- Cálculo do IOF: A valor do IOF é igual 0,3814% do valor do contrato.
- Determinação da taxa de juros: A taxa de juros deve ser determinada de acordo com o valor total solicitado para financiamento e o número de parcelas. Isto é feito através da comparação do valor total e do

número de parcelas com limites estabelecidos nas taxas de juros cadastradas.

- Valor limite máximo para taxa de juros: O valor limite máximo para a taxa de juros é de 12%.

Essas regras de negócio abrangem o fluxo inicial do processo de Analisar pedido de crédito, como pode ser observado na Figura 17, onde as atividades destacadas são as atividades que estão sendo contempladas. O projeto dos serviços definidos para disponibilização das regras foi organizado em três serviços: (i) serviço para tratar o cadastro do cliente; (ii) serviço para tratar limite de crédito; e (iii) serviço para tratar taxas, juros e cálculo do valor total do contrato.

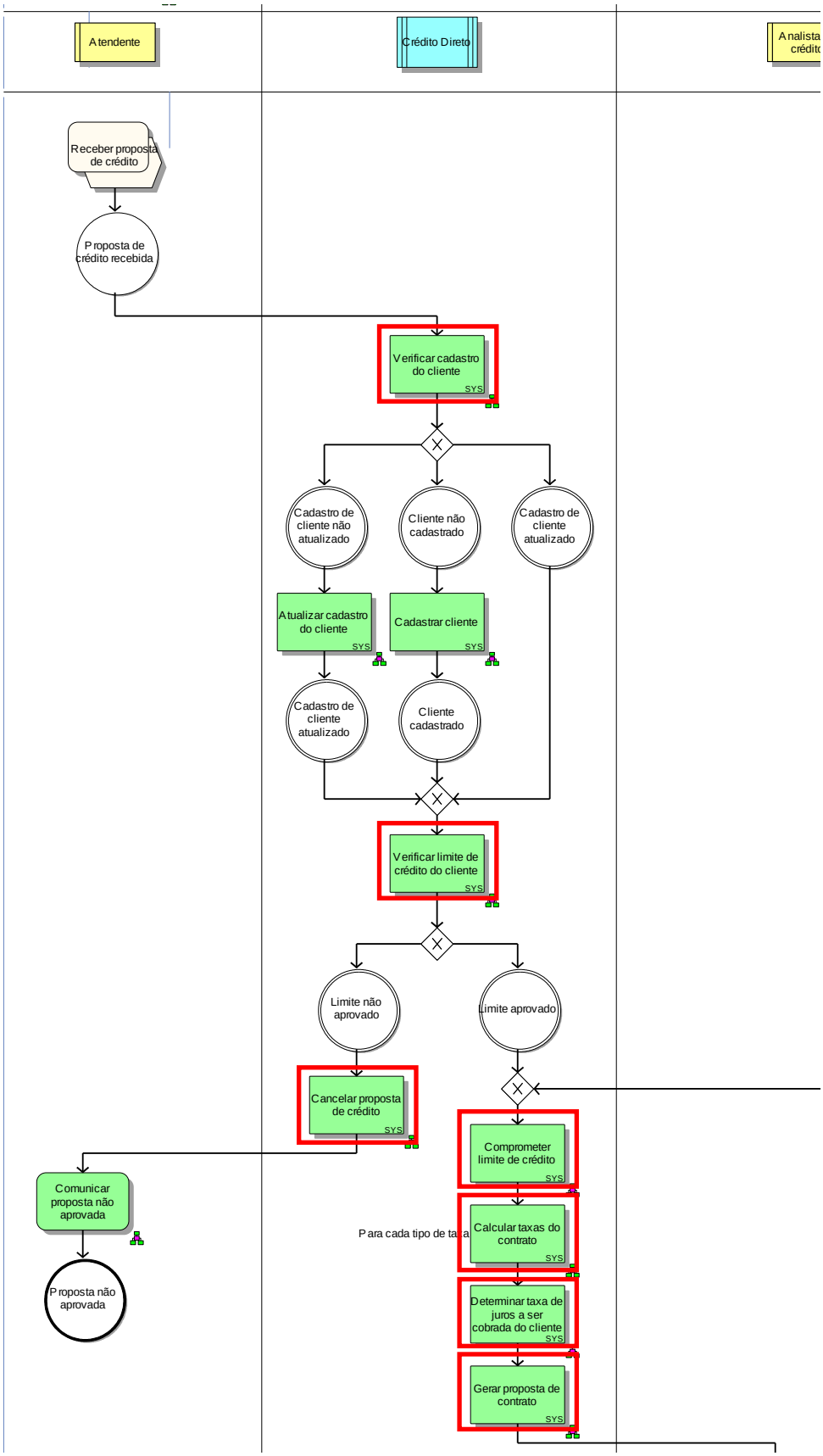


Figura 17 – Fluxo inicial do processo de Analisar pedido de crédito

5.2 Criação do modelo que representa os objetos

Para que as regras possam ser escritas é necessária a criação do modelo de objetos do negócio (BOM) e o mapeamento deste para o modelo de objetos de execução (XOM), conforme descrito nas seções 4.1.1 e 4.1.2. Este modelo foi criado a partir de classes JAVA. Neste estudo de caso foram criadas 3 classes que representam as entidades envolvidas no processo:

- Cliente, cujos atributos são cpf, nome, idade, endereco, telefone, creditoAberto, parcelasAberto, renda, estadoCadastro, limiteCredito e categoria;
- CreditoConcedido, cujos atributos são valor, valorTotal, parcelasAprovadas, juros, taxas, situação, codProposta, distanciaKM, alterações e impressões;
- Proposta, cujos atributos são código, valorFinanciamento, parcelas, estadoProposta e cpfCliente.

Todas essas classes foram criadas dentro do projeto analisarpedidocredito-xom (Figura 18) que serviu como base para a criação do *Business Object Model* (BOM) que será utilizado nos projetos de regras.

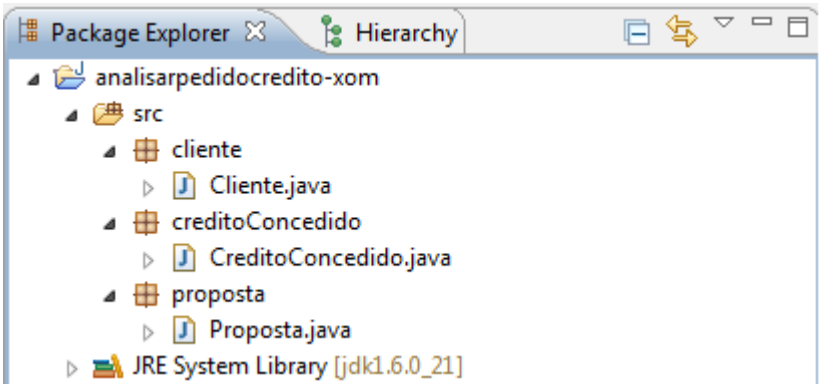


Figura 18 – Classes que compõem o modelo de objetos de execução

Após a criação dessas classes foram criados os projetos de regras, nos quais foram elaborados os BOMs utilizados na criação das regras. Foram criados 6 projetos de regras (Figura 19) que podem conter uma ou mais regras. Os projetos são os seguintes:

- checkCadastro: contém as regras responsáveis por analisar o cadastro do cliente e atualizar o estado do cadastro. Os estados do cliente podem ser “Cliente Novo”, “Atualizado” ou “Desatualizado”;
- calcularLimiteCredito: contém a regra responsável por calcular o limite de crédito do cliente;
- verificarLimiteCredito: contém a regra que verifica se o limite do cliente é suficiente para a proposta de crédito solicitada;
- comprometerLimiteCredito: contém a regra que compromete o limite de crédito ajustando a situação do creditoConcedido;
- calcularJuros: contém a regra, em formato de tabela de decisão, que calcula os juros do contrato. Para a especificação desta regra foi utilizada tabela de decisão, não só para porque apresenta a melhor notação para este tipo de regra, como também para testar regra descrita dessa forma;
- calcularValorTotal: contém as regras que calculam as taxas do contrato e o valor total, com as taxas e juros aplicados.

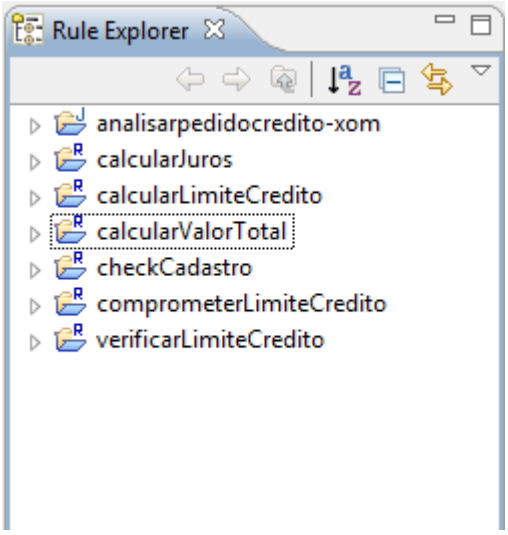


Figura 19 – Projetos de regras criados

Com os projetos criados é necessário associar o XOM aos projetos e criar o BOM. Cada projeto precisa ter somente o BOM da entidade que será usada na regra. Então, por exemplo, o projeto checkCadastro só necessita ter o BOM Cliente. A Figura 20 ilustra o BOM da classe cliente. Os outros BOMs, de proposta e creditoConcedido, foram criados da mesma forma, porém com os atributos

respectivos a essas classes. O BOM é criado automaticamente, bastando apenas dar um nome e apontar uma classe Java ou *Schema* XML, que ele é gerado automaticamente. Com os BOMs criados, é possível escrever de forma textual a regra, referenciando o BOM e seus atributos.

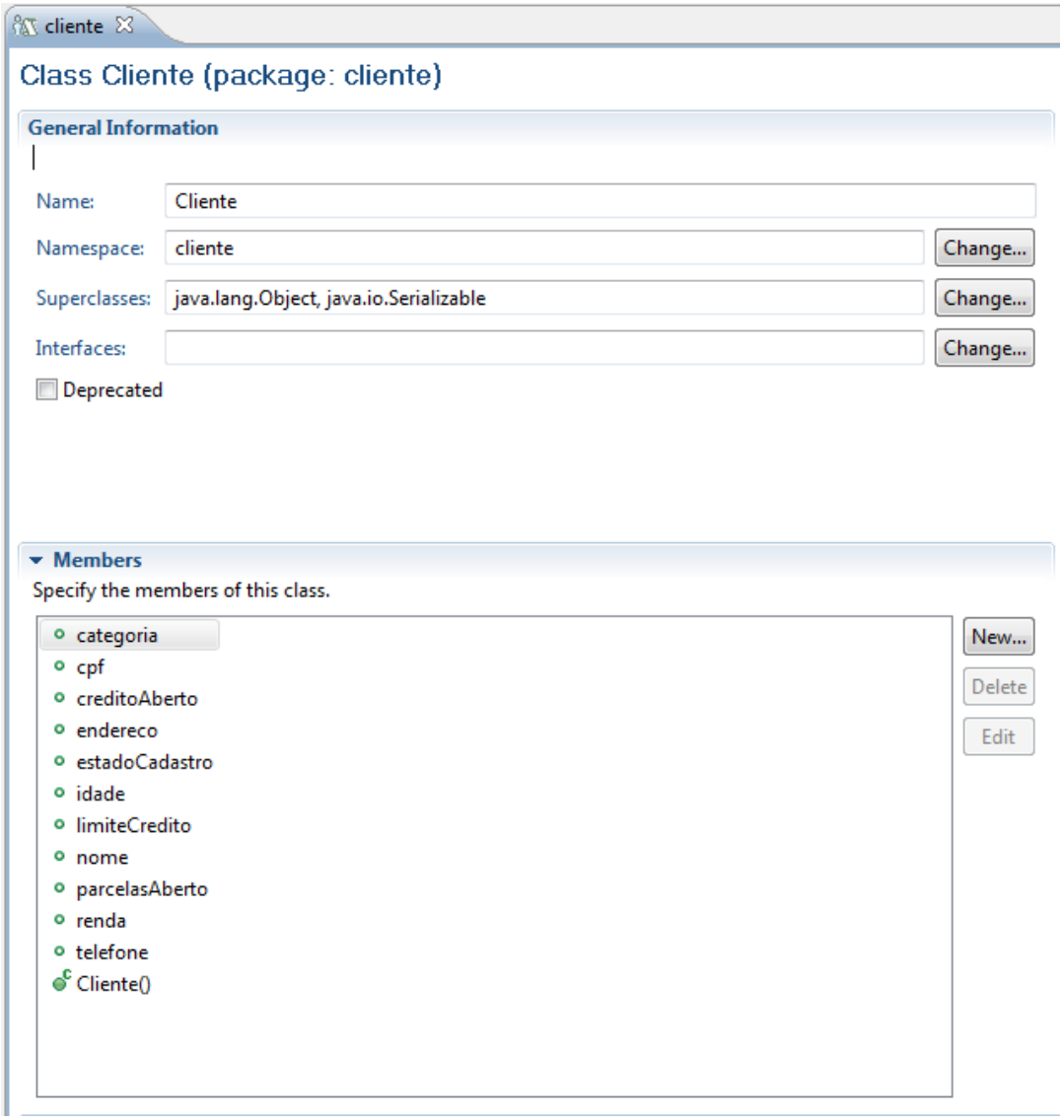


Figura 20 – BOM do Cliente

5.3 Desenvolvimento do projeto de regras

Antes de desenvolver efetivamente as regras é preciso definir os parâmetros de entrada e saída do projeto. Esses parâmetros de entrada e saída podem ser utilizados dentro das regras. Ou seja, é possível passar um Cliente como parâmetro para a regra, e a mesma ler e editar as informações do objeto e retorná-lo, caso este parâmetro esteja definido como IN_OUT. As direções dos parâmetros são: IN (só

entrada), IN_OUT (entrada e saída) e OUT (só saída). O parâmetro, além da direção, deve possuir um nome, um tipo de dados (pode ser um objeto do BOM ou um tipo de dado Java) e um nome verbalizado que será utilizado no texto da regra. Para cada projeto foram definidos os seguintes parâmetros:

- checkCadastro (Figura 21)

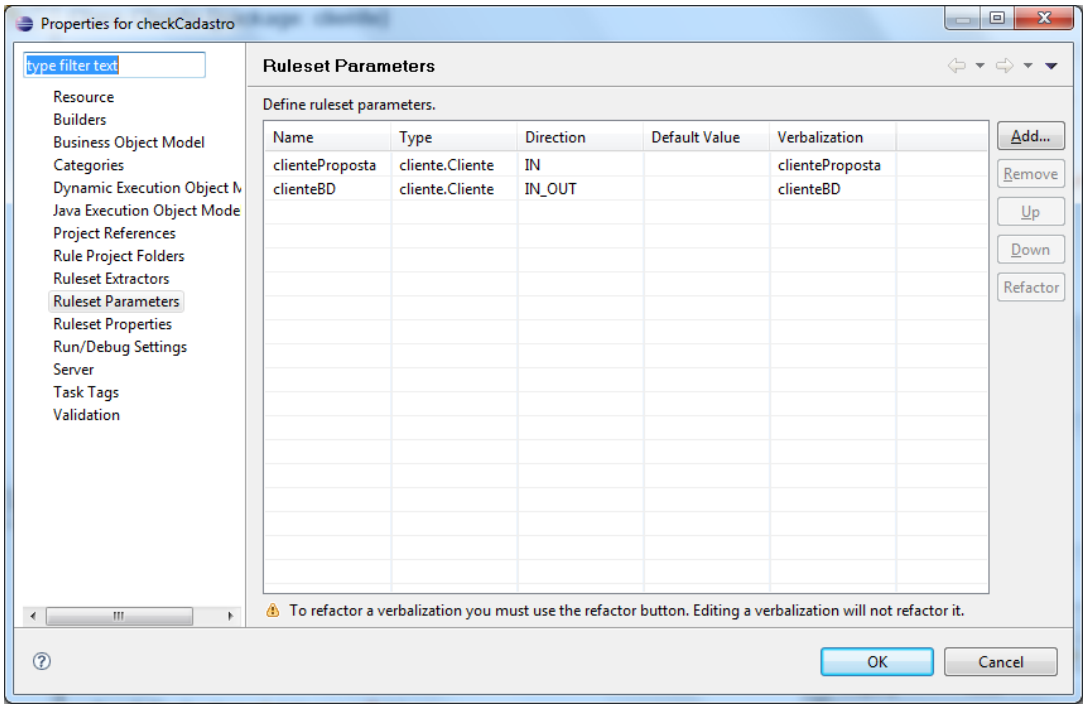


Figura 21 – Parâmetros do projeto checkCadastro

- calcularLimiteCredito (Figura 22)

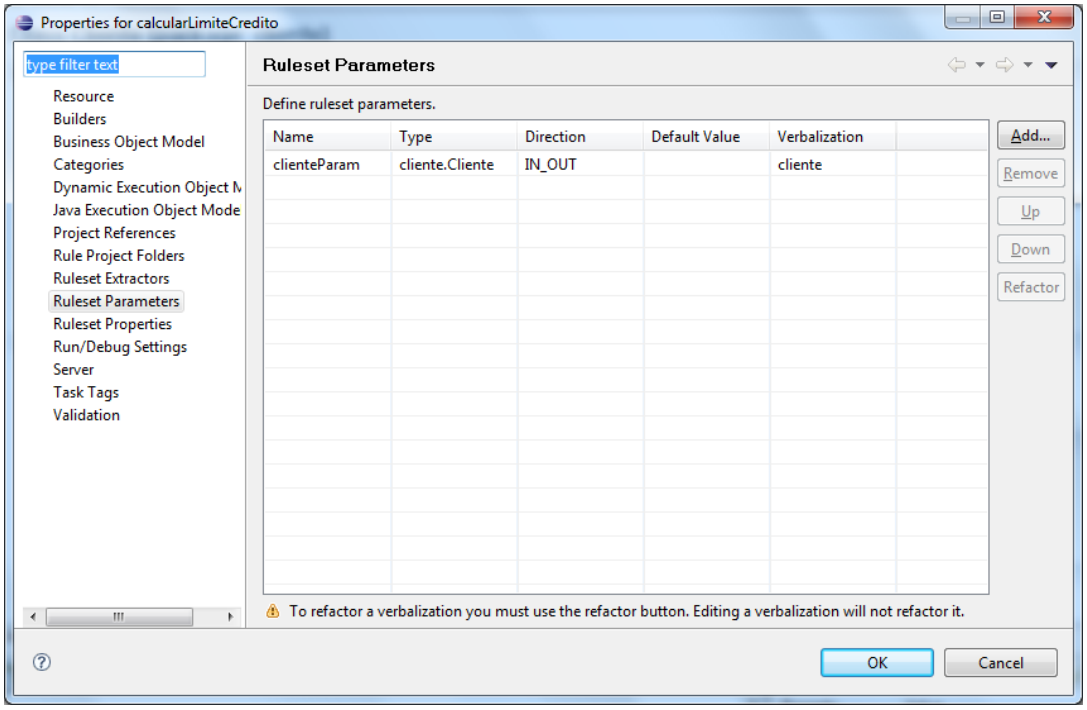


Figura 22 – Parâmetros do projeto calcularLimiteCredito

- verificarLimiteCredito (Figura 23)

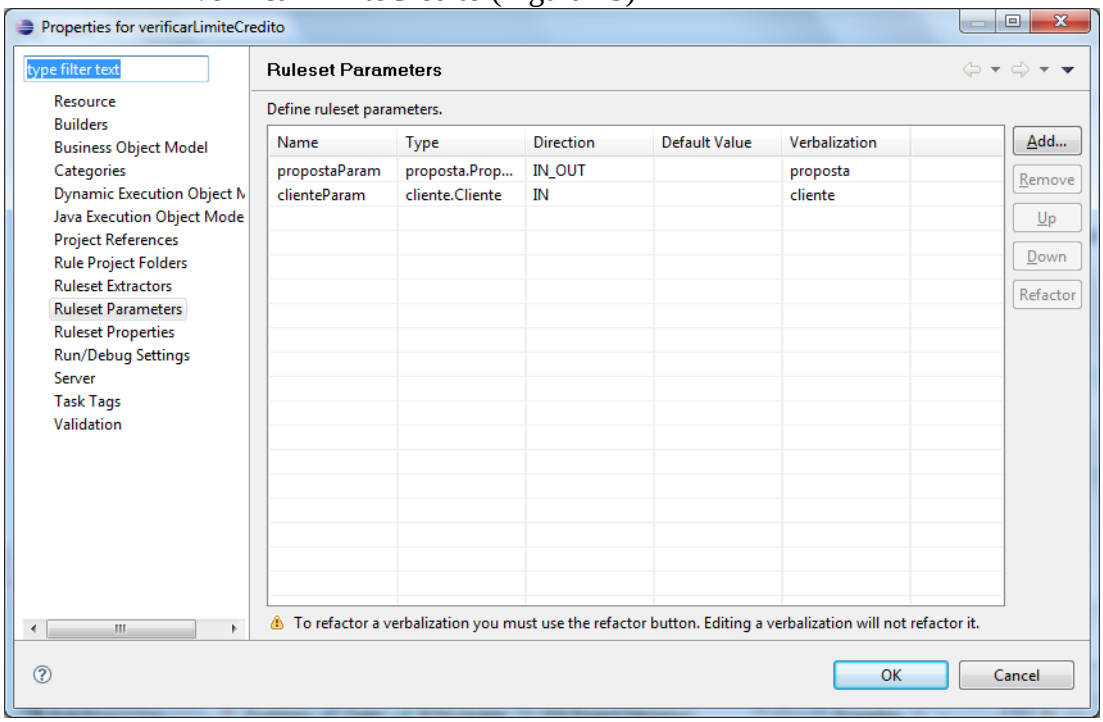


Figura 23 – Parâmetros do projeto verificarLimiteCredito

- comprometerLimiteCredito (Figura 24)

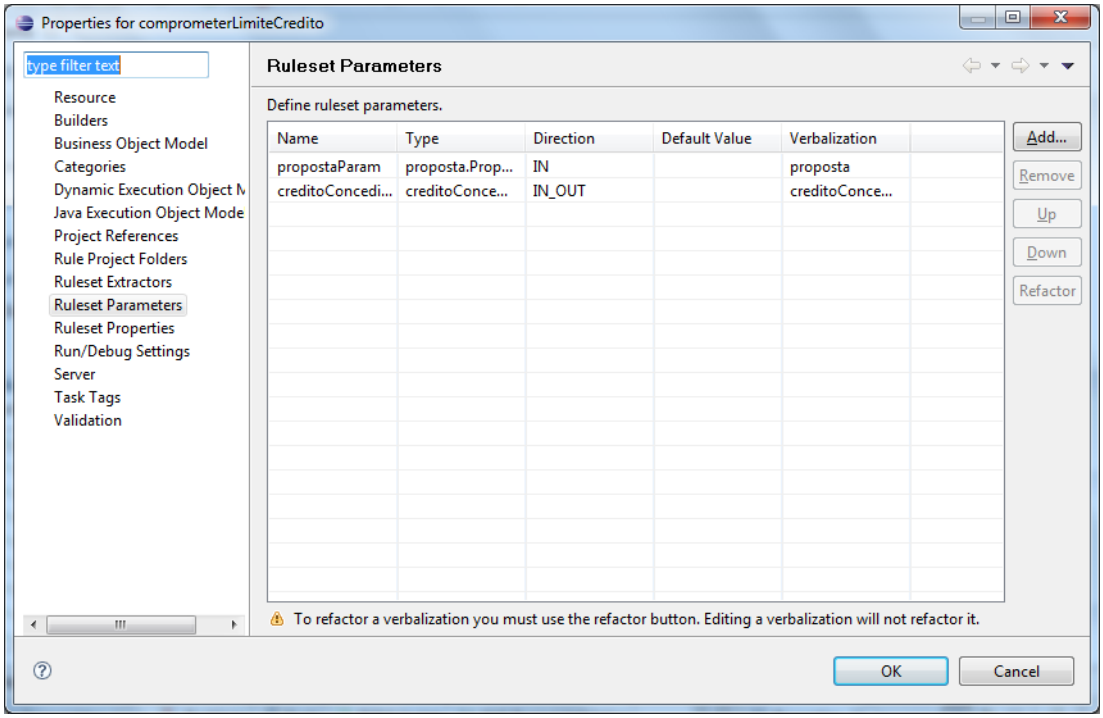


Figura 24 – Parâmetros do projeto comprometerLimiteCredito

- calcularJuros (Figura 25)

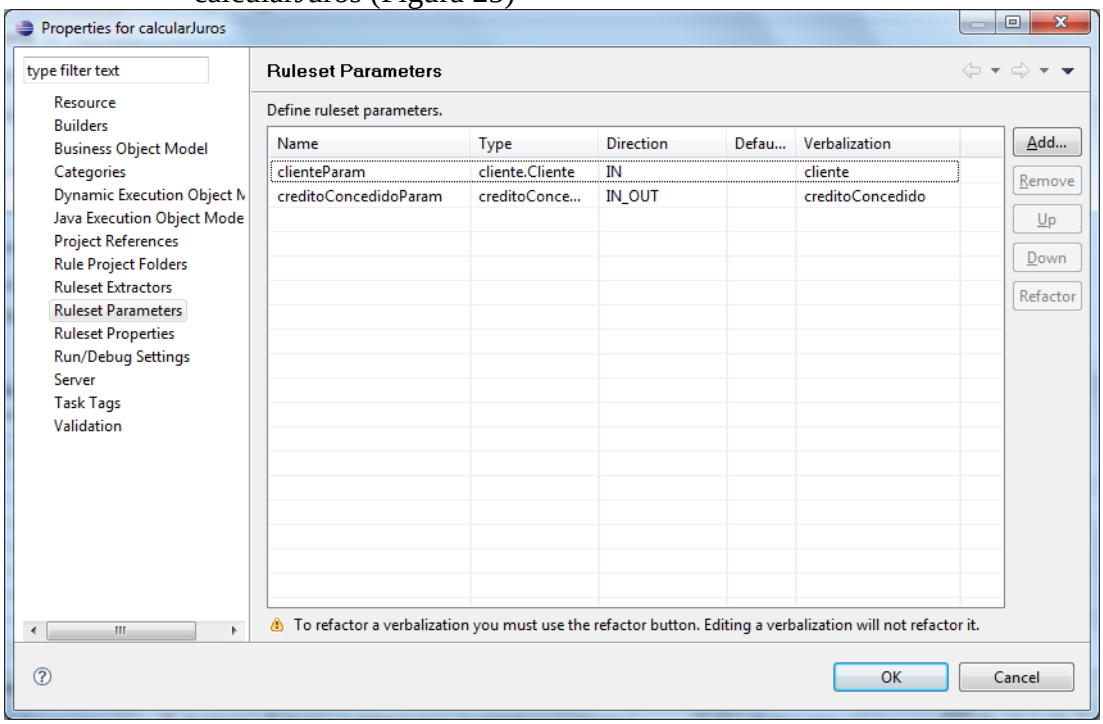


Figura 25 – Parâmetros do projeto calcularJuros

- calcularValorTotal (Figura 26)

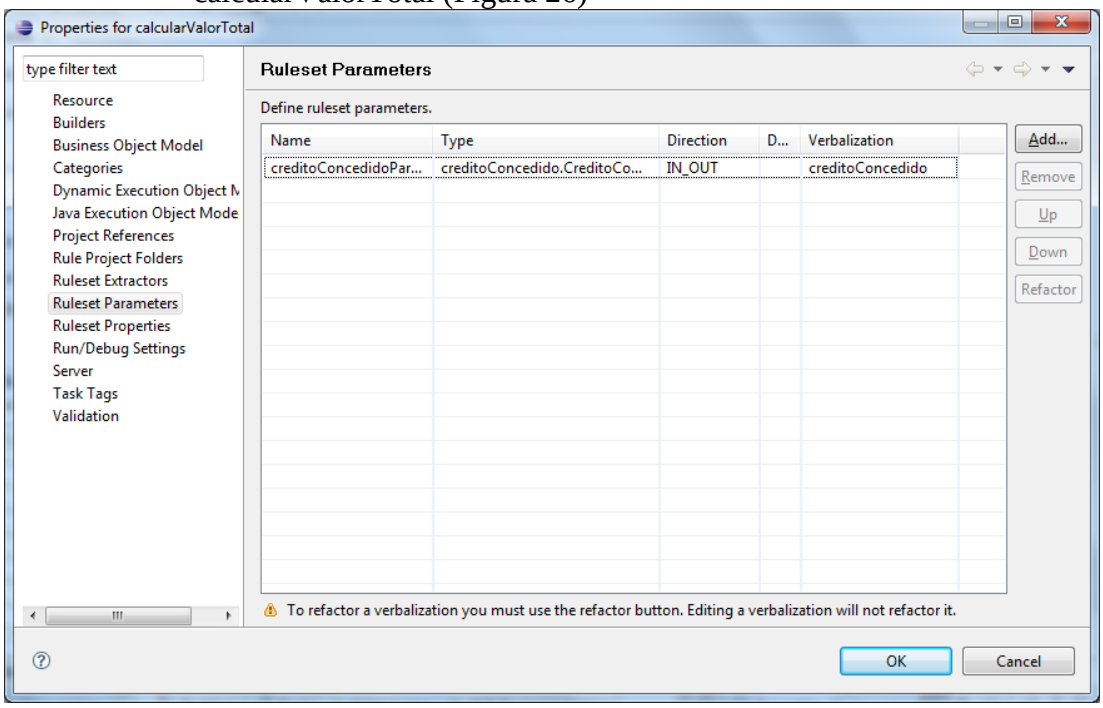


Figura 26 – Parâmetro do projeto calcularValorTotal

Após os parâmetros terem sido definidos foram criadas as regras de negócio de acordo com o que está especificado em [Diirr *et al*., 2010]. Além da criação das regras, para que as mesmas sejam disponibilizadas como serviço é necessário que seja criado um fluxo de execução das regras. Para todos os projetos os fluxos são simples, executando de forma seqüencial as regras. Apenas no projeto checkCadastro o fluxo definido possui uma condição. Nas subseções seguintes serão detalhados os projetos de regras, assim como será feita a relação entre o projeto e a regra definida no processo.

5.3.1 checkCadastro

O projeto checkCadastro visa atender as regras de negócio “Cadastro de cliente desatualizado” e “Cliente novo”. Para isso foram criadas duas regras de negócio e um fluxo de regras. As regras de negócio são checkClienteNovo (Figura 27) e checkAtualização (Figura 28). De acordo com o fluxo (Figura 29), a regra checkClienteNovo é executada. Caso o estado de cadastro do ClienteBD seja “Cliente Atualizado”, a regra checkAtualização é executada; caso contrário a execução do *ruleset* termina. Ao final da execução do *ruleset*, o estado de cadastro do ClienteBD é definido. A regra pode ser simplificada com a utilização de

templates, que permite criar frases prontas para apenas algumas partes das regras serem definidas pelos usuários.

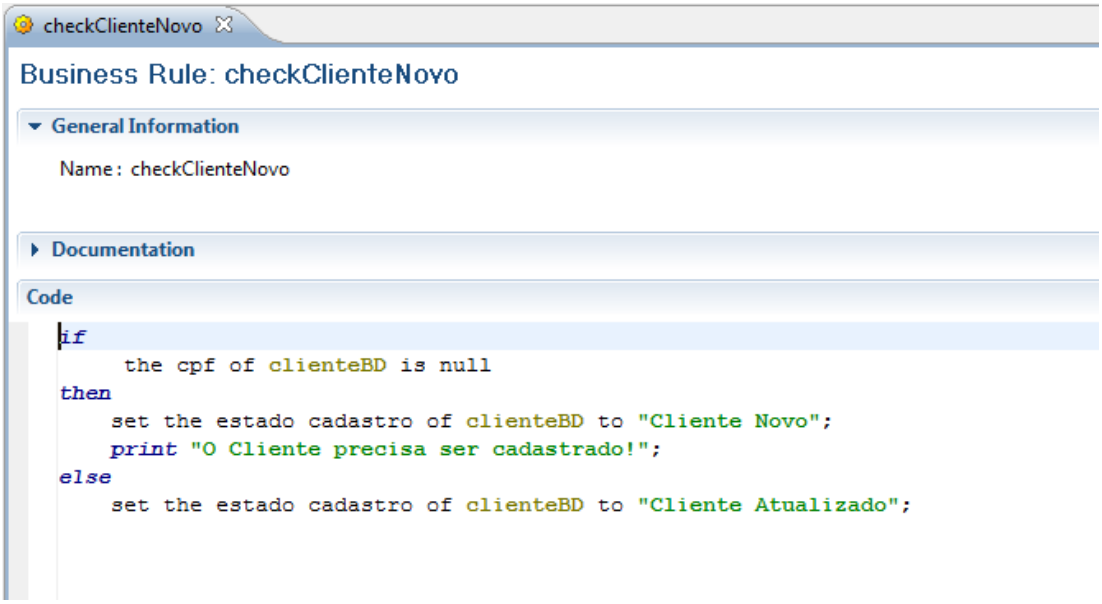


Figura 27 – Regra checkClienteNovo

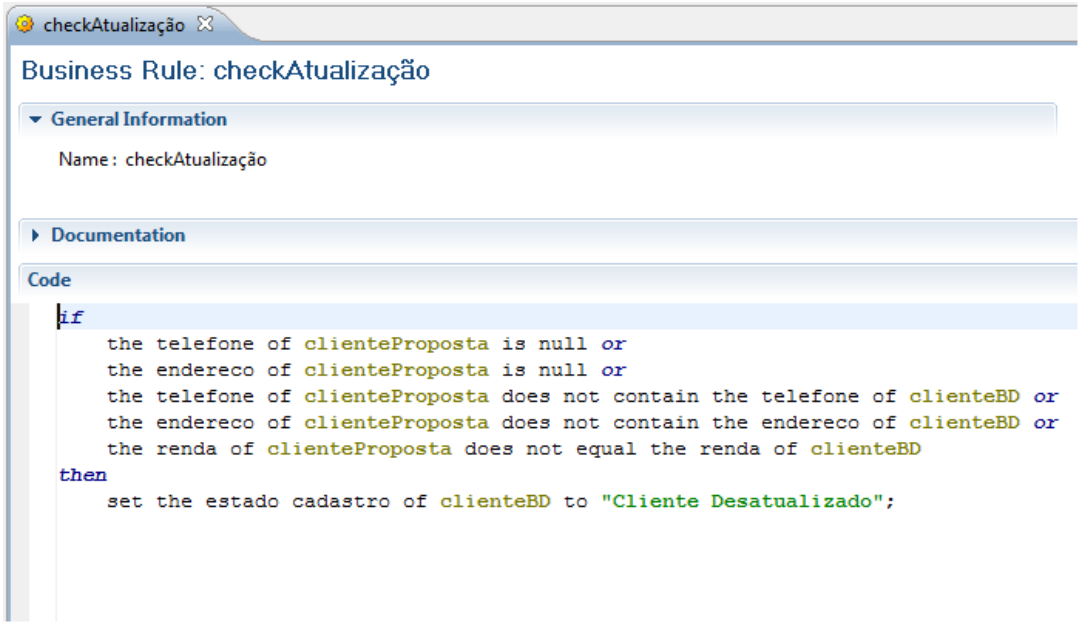


Figura 28 – Regra checkAtualização

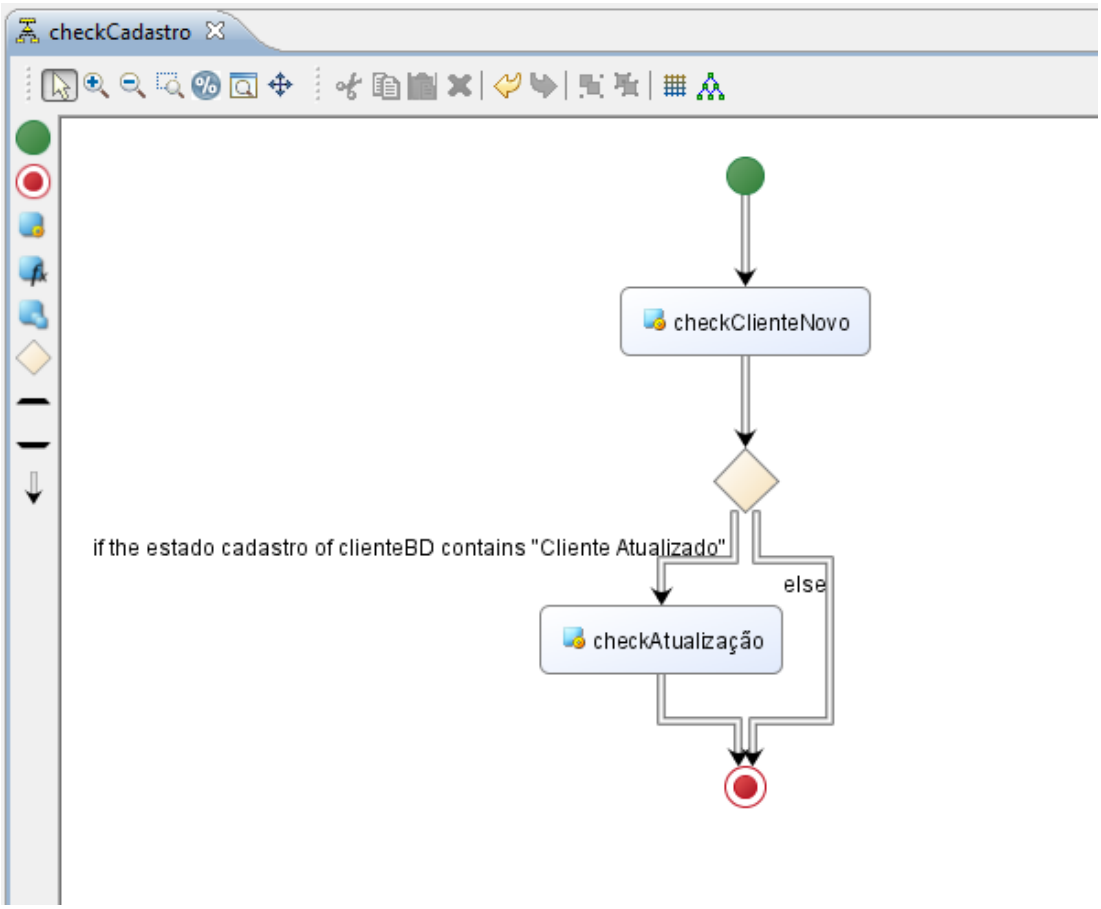


Figura 29 – Fluxo de execução do projeto checkCadastro

5.3.2 calcularLimiteCredito

O projeto de regras calcularLimiteCredito tem como objetivo atender a regra de “Limite de crédito”. Esse projeto contém apenas uma regra (Figura 30) que calcula o limite de crédito do cliente baseado na sua renda, no crédito aberto e nas parcelas em aberto. O fluxo de execução desse projeto é apenas a execução dessa regra.

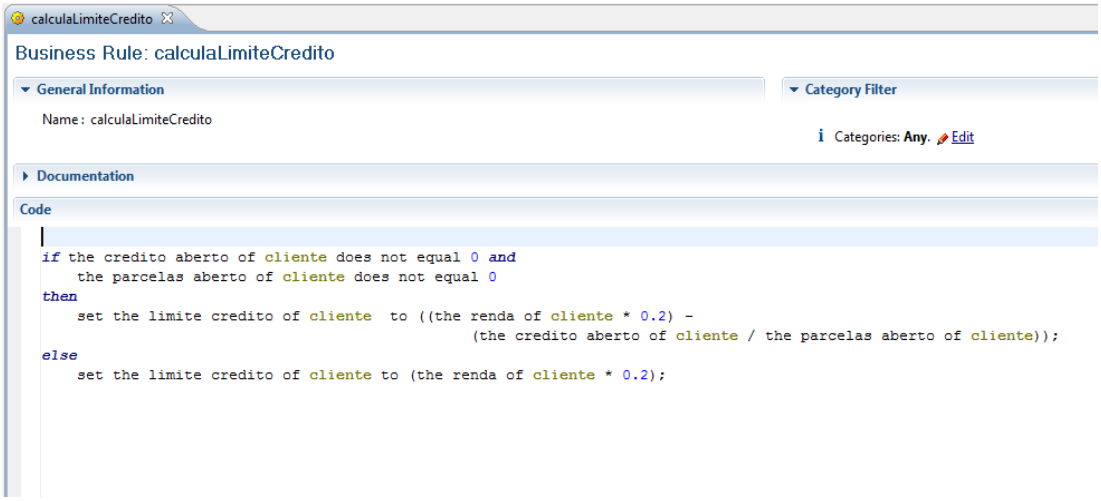


Figura 30 – Regra calculaLimiteCredito

5.3.3 verificarLimiteCredito

O projeto de regras verificaLimiteCredito atende a regra de "Aprovação de crédito". A regra (Figura 31) desse projeto verifica se o limite de crédito do cliente é maior que o valor da proposta dividido pelo número de parcelas. O fluxo de regras desse projeto é a execução desta regra.

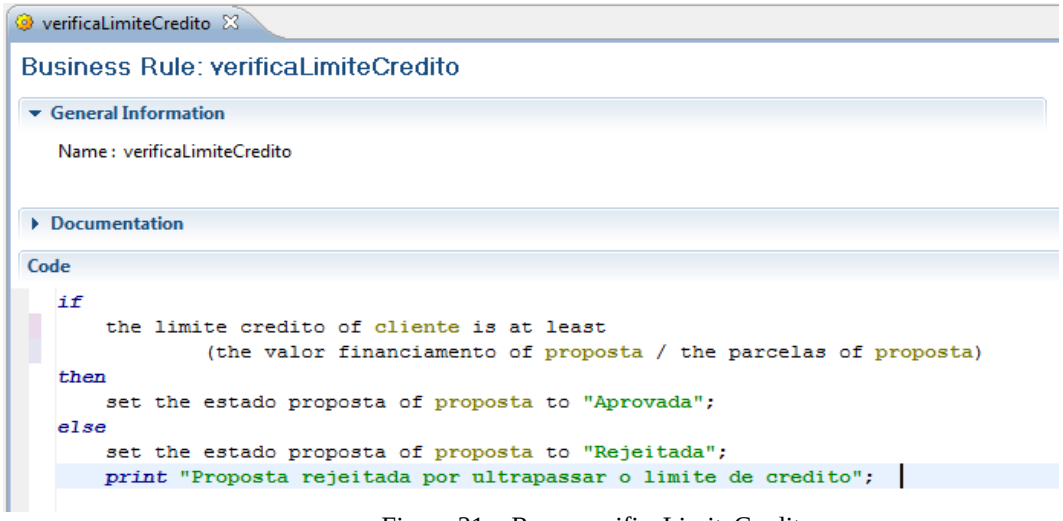


Figura 31 – Regra verificaLimiteCredito

5.3.4 comprometerLimiteCredito

O projeto de regras comprometerLimiteCredito visa atender a regra de "Comprometimento do limite de crédito". A regra (Figura 32) verifica se a proposta realmente foi aprovada e, caso verdadeiro, ela modifica a situação do

creditoConcedido para comprometido e insere as informações da proposta. Seu fluxo de execução é a execução da própria regra.

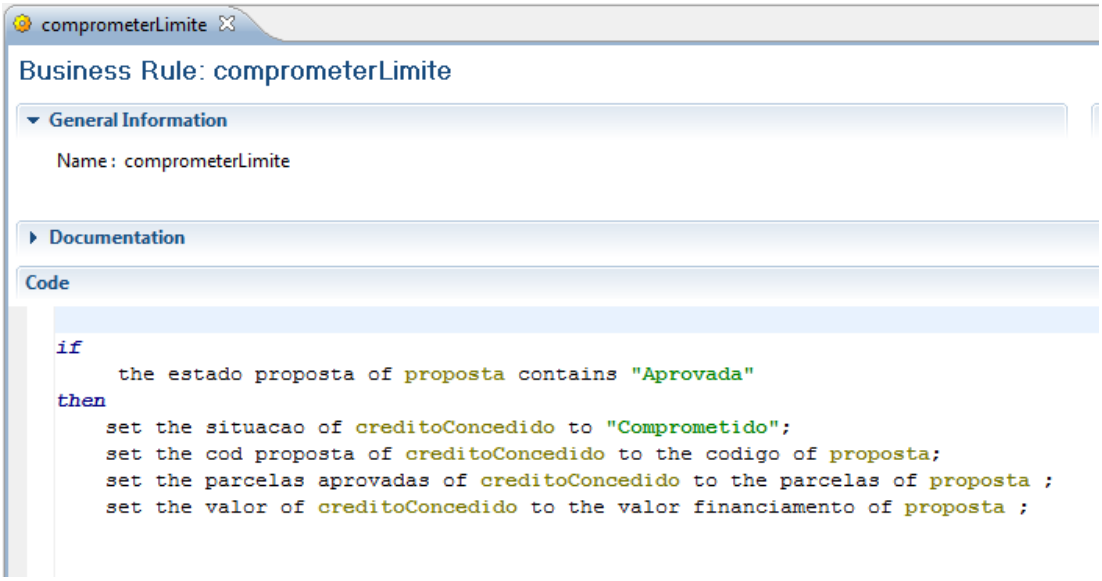
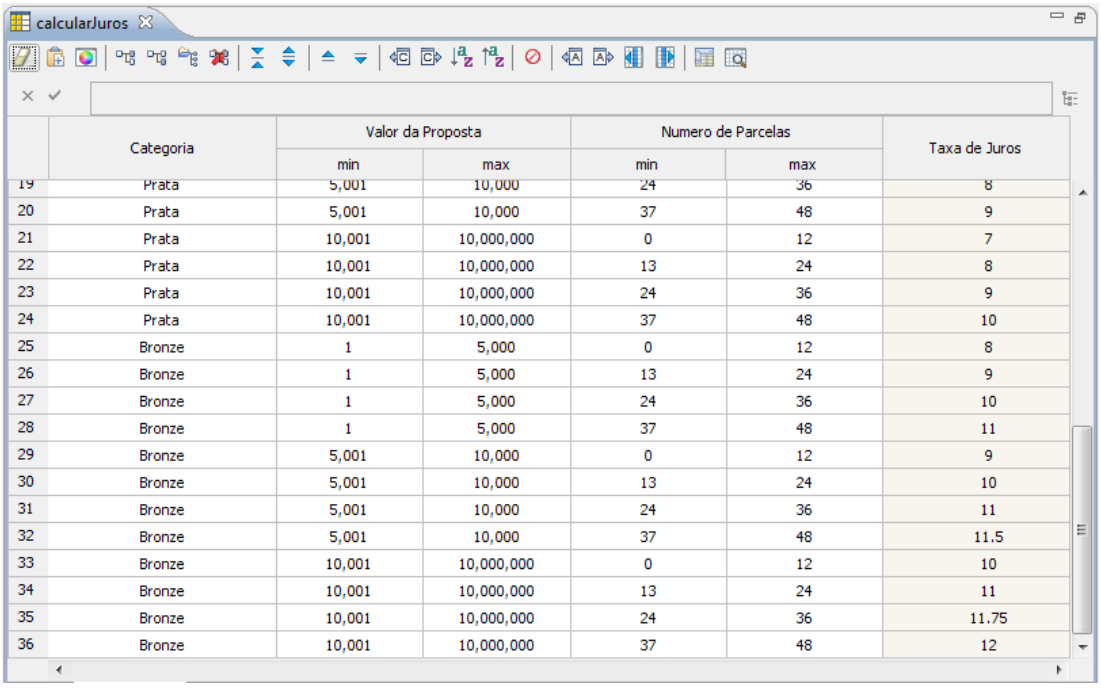


Figura 32 – Regra comprometerLimite

5.3.5 calcularJuros

O projeto de regras calcularJuros tem como objetivo atender às regras de "Determinação da taxa de juros" e "Valor limite máximo para taxa de juros". Para essas regras foi criada uma tabela de decisão (Figura 33), onde 3 colunas representam as condições (categoria do cliente, valor da proposta e número de parcelas) e, de acordo com os possíveis valores para cada condição, é estabelecida a taxa de juros, a qual não ultrapassa 12%. O fluxo de decisão desse projeto é a execução da tabela de decisão.



calcularJuros

	Categoria	Valor da Proposta		Numero de Parcelas		Taxa de Juros
		min	max	min	max	
19	Prata	5,001	10,000	24	36	8
20	Prata	5,001	10,000	37	48	9
21	Prata	10,001	10,000,000	0	12	7
22	Prata	10,001	10,000,000	13	24	8
23	Prata	10,001	10,000,000	24	36	9
24	Prata	10,001	10,000,000	37	48	10
25	Bronze	1	5,000	0	12	8
26	Bronze	1	5,000	13	24	9
27	Bronze	1	5,000	24	36	10
28	Bronze	1	5,000	37	48	11
29	Bronze	5,001	10,000	0	12	9
30	Bronze	5,001	10,000	13	24	10
31	Bronze	5,001	10,000	24	36	11
32	Bronze	5,001	10,000	37	48	11.5
33	Bronze	10,001	10,000,000	0	12	10
34	Bronze	10,001	10,000,000	13	24	11
35	Bronze	10,001	10,000,000	24	36	11.75
36	Bronze	10,001	10,000,000	37	48	12

Figura 33 – Tabela de decisão calcularJuros

5.3.6 calcularValorTotal

Este projeto de regras visa atender as seguintes regras: “Cálculo da taxa de alteração do contrato”, “Cálculo da taxa de entrega”, “Cálculo da taxa de impressão” e “Cálculo do IOF”. Além disso, esse projeto de regras calcula o valor final e total do creditoConcedido, já com os juros aplicados. O projeto consiste em três regras: uma regra (Figura 34) para definir a porcentagens de taxa (como o IOF), outra regra (Figura 35) para calcular o valor das taxas e somar ao valor da proposta e, por fim, uma terceira regra (Figura 36) para calcular o valor dos juros e somar ao valor total do creditoConcedido. O fluxo de execução (Figura 37) desse projeto é a execução seqüencial dessas 3 regras.

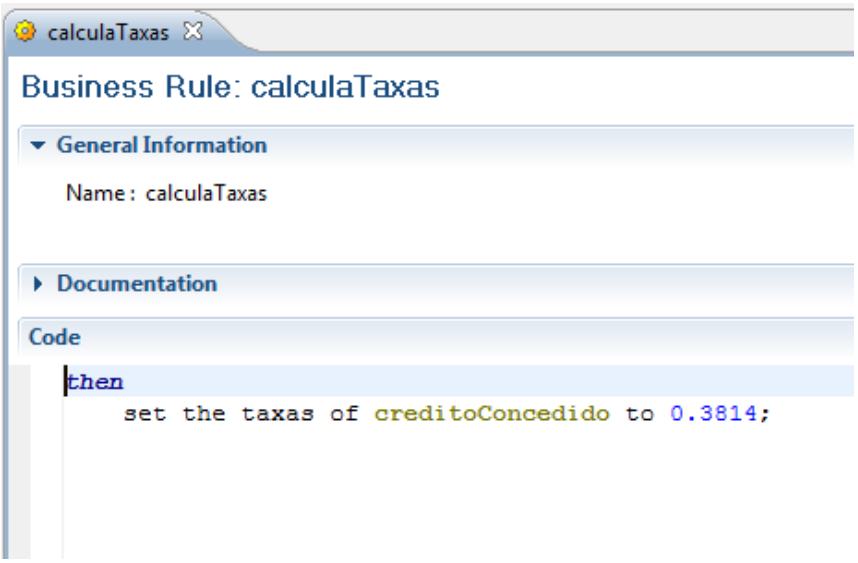


Figura 34 – Regra calculaTaxas

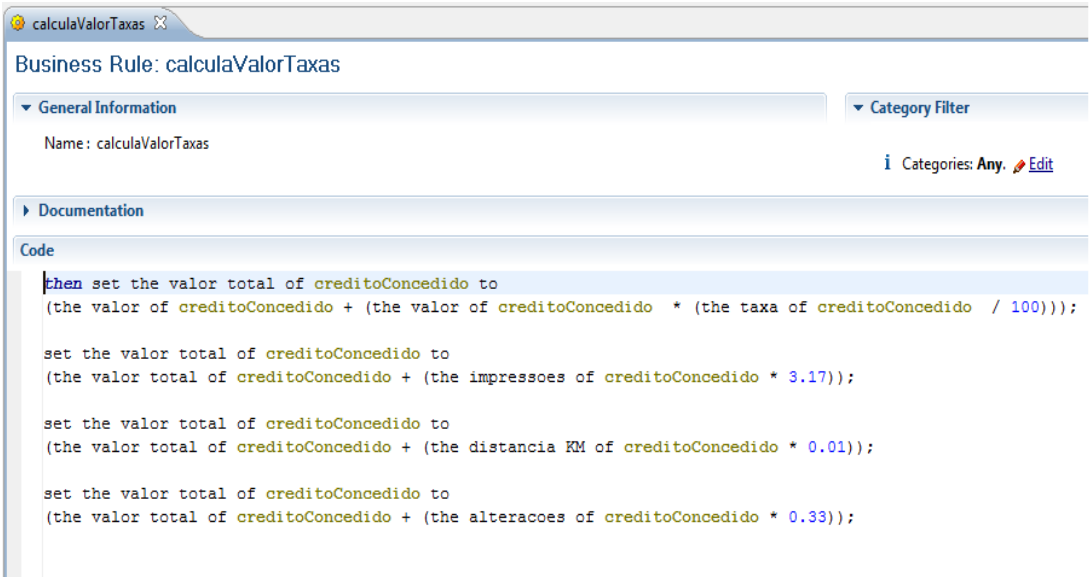


Figura 35 – Regra calculaValorTaxas

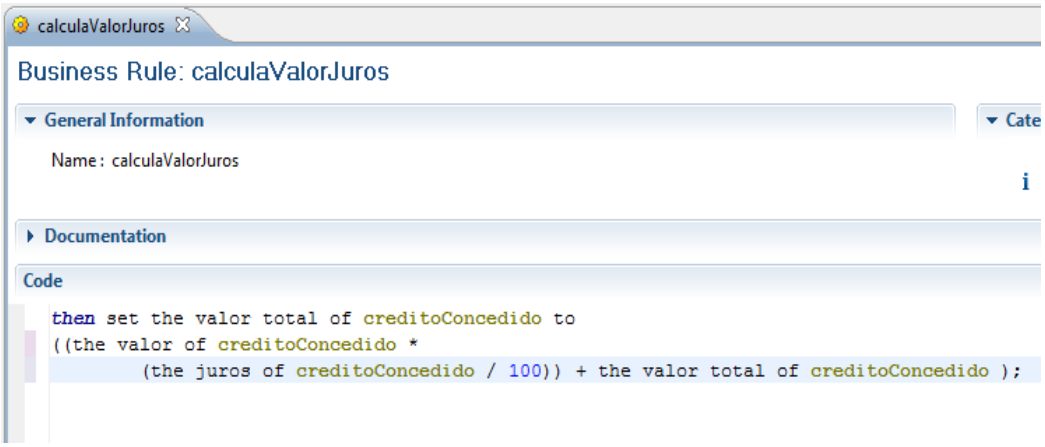


Figura 36 – Regra calculaValorJuros

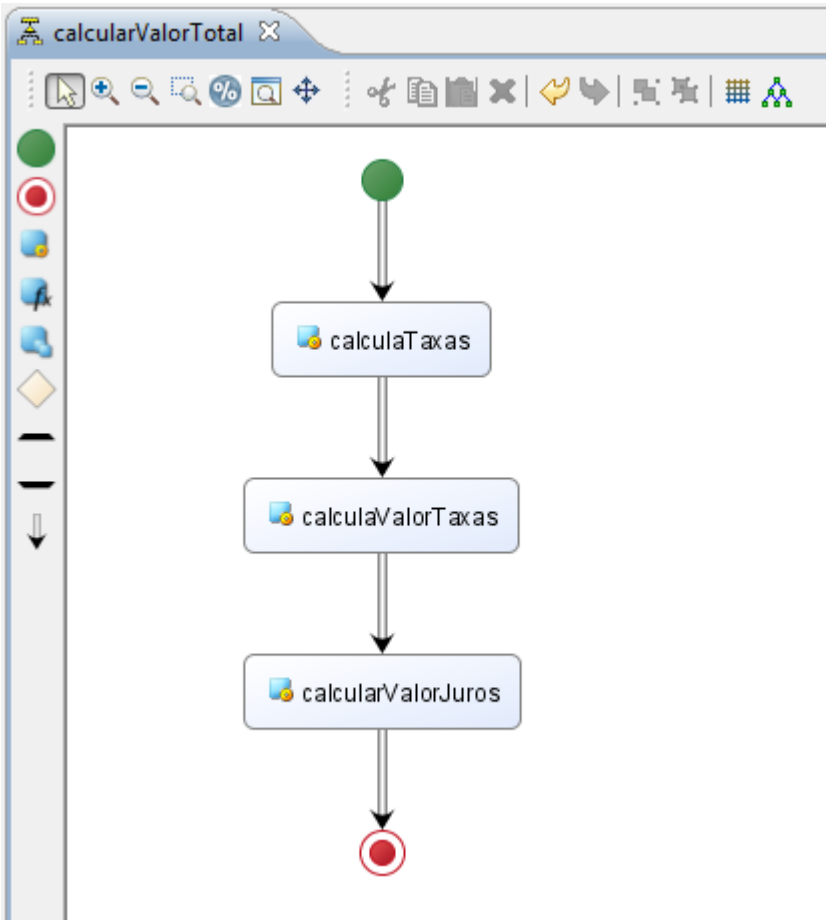


Figura 37 – Fluxo de execução calcularValorTotal

5.4 Disponibilização dos projetos de regras como *Monitored Transparent Decision Service*

Com os projetos de regras criados e prontos para serem executados é necessária a disponibilização das regras de negócio como serviços. Para isso foram criados três RuleApps que representam os serviços disponibilizados. O primeiro RuleApp é o ServiçoClienteApp, que contém apenas o projeto checkCadastro. O segundo RuleApp é o ServiçoLimiteCreditoApp que contém os projetos calcularLimiteCredito, verificarLimiteCredito e comprometerLimiteCredito. Por fim, o último RuleApp contém os projetos de regras relacionados ao contrato, o calcularJuros e calcularValorTotal, como pode ser observado na Figura 38. O RuleApp contém um arquivo .XML e os *rulesets* (.jar) que representam os projetos de regras.

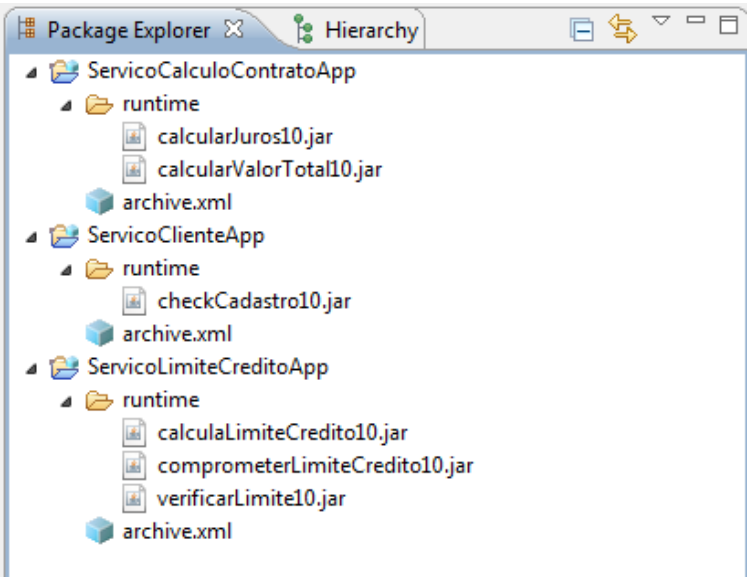


Figura 38 - RuleApps

Cada RuleApp será disponibilizado como um *web service* e cada projeto de regra será um método dentro desse serviço. Cada método receberá como entrada um objeto do tipo *Request* e retornará como saída um do tipo *Response*. Esses objetos são instâncias de classes customizadas, contendo como atributos os objetos do modelo de dados dos projetos de regras. Por exemplo, o método checkCadastro receberá como entrada um objeto do tipo checkCadastroRequest que tem como atributo um cliente do tipo Cliente. Essas interfaces podem ser observadas na Figura 39.

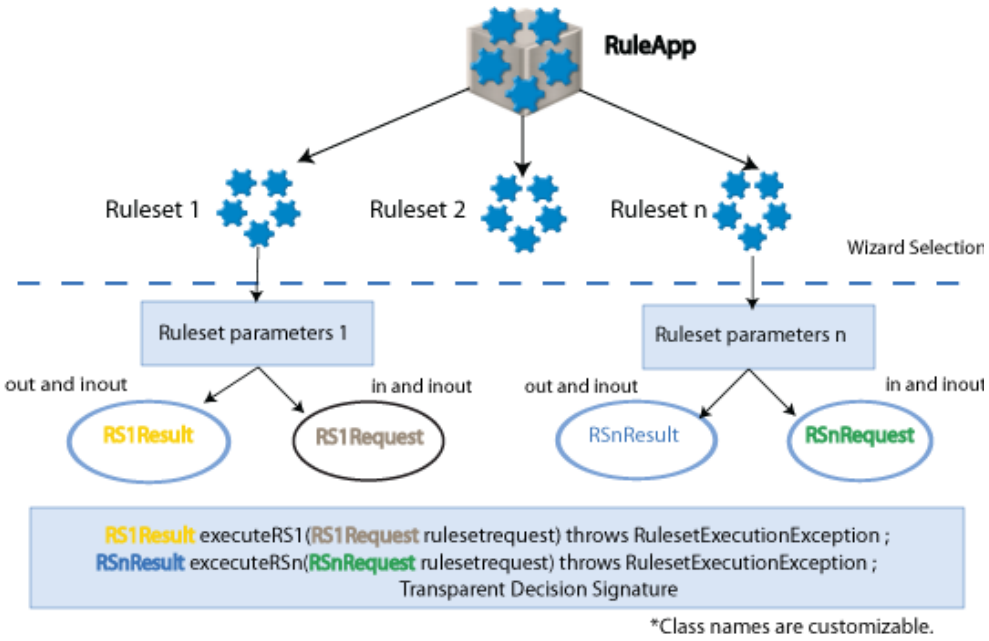


Figura 39 – Entrada e saída dos métodos dos serviços

Após a criação dos RuleApps, é possível executar um *wizard* para geração dos *web services*. O *wizard* em questão é o *Client for RuleApp Project*. Ele é executado utilizando o módulo Rule Studio. Na sua execução, o *wizard* possibilita a escolha de diversos *templates* para a geração de código. No estudo de caso, o *template* utilizado foi o *template* para geração de *web services*. O *wizard* requisita o nome do projeto e o RuleApp que será a base do serviço. Além disso, ele permite selecionar quais rulesets e quais fluxos de execução farão parte do serviço. No *wizard* deve-se especificar o nome que será dado ao projeto do *web service* e do cliente, além de escolher o servidor aonde ele será publicado. Antes de finalizar o *wizard*, é possível ainda definir quais informações de gerenciamento sobre a execução devem ser geradas. Após a execução do *wizard*, são criados 2 projetos, um com o *web service* em si, que deve ser publicado no servidor de aplicação, e outro projeto cliente, como ilustrado na Figura 40. A figura demonstra a arquitetura de um serviço de decisão transparente monitorado. O *wizard* cria as classes do cliente e do *web service* e através das classes criadas a partir de *ant tasks* é possível realizar a comunicação do cliente com o serviço de decisão, o qual envia notificações ao Rule Execution Server.

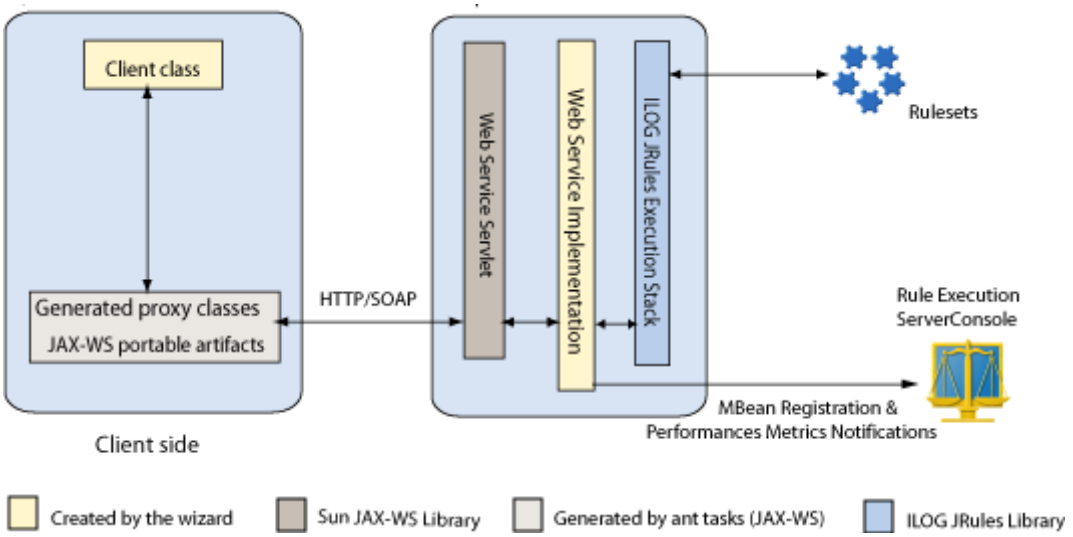


Figura 40 - Componentes criados a partir do *wizard*

No projeto do *web service* é preciso apenas executar o arquivo build.xml com as *targets* deploywar e deployruleapp. Em seguida, o serviço é publicado no servidor de aplicação e o RuleApp é publicado no Rule Exectuion Server e pode ser visualizado dentro da ferramenta, como demonstrado na Figura 41. O WSDL que representa o serviço publicado pode ser visualizado através da seguinte URL

http://hostname:porta/jaxws-NomeDoRuleApp/NomeDoWebService?WSDL. Neste estudo de caso, o WSDL do ServicoClienteApp ficou com a seguinte URL: http://localhost:8080/jaxws-ServicoClienteApp/ServicoClienteWS?wsdl. O WSDL pode ser observado na Figura 42.

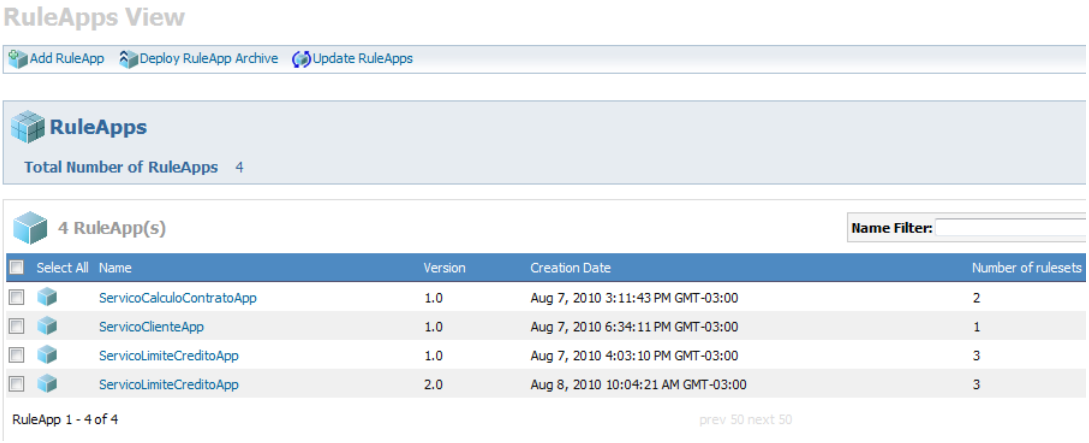


Figura 41 - RuleApps publicados no Rule Execution Server

```
−<definitions targetNamespace="http://ServicoClienteApp/" name="ServicoClienteWSRunnerImplService">
   <import namespace="http://ServicoClienteApp" location="http://localhost:8080/jaxws-ServicoClienteApp/ServicoClienteWS?wsdl=1"/>
  −<binding name="ServicoClienteWSRunnerImplPortBinding" type="ns1:ServicoClienteWSRunner">
     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    −<operation name="checkCadastro">
       <soap:operation soapAction="urn:executecheckCadastro"/>
      −<input>
         <soap:body use="literal"/>
       </input>
      −<output>
         <soap:body use="literal"/>
       </output>
      −<fault name="RulesetExecutionException">
         <soap:fault name="RulesetExecutionException" use="literal"/>
       </fault>
     </operation>
   </binding>
  −<service name="ServicoClienteWSRunnerImplService">
    −<port name="ServicoClienteWSRunnerImplPort" binding="tns:ServicoClienteWSRunnerImplPortBinding">
       <soap:address location="http://localhost:8080/jaxws-ServicoClienteApp/ServicoClienteWS"/>
     </port>
   </service>
 </definitions>
```

Figura 42 – WSDL do ServicoClienteWS

Além da criação do projeto dos *web services*, o *wizard* cria um projeto cliente que já está pronto para executar, necessitando apenas inicializar os parâmetros que serão passados ao serviço, como pode ser visto na Figura 43. A classe deixa marcado com TODO as linhas que devem ser modificadas com a inclusão do parâmetro. Estes métodos de execução das regras podem ser customizados para receberem parâmetros e retornarem os resultados, como foi feito no estudo de caso.

Figura 43 - Classe cliente do ServicoClienteWS

5.5 Publicação dos serviços no OSB

Com os serviços já publicados e rodando em um servidor de aplicação, é necessário fazer com que esses serviços sejam publicados no ESB (Enterprise Service Bus). Com o uso do ESB, os serviços continuam sendo executados no servidor de aplicação de origem, mas seus clientes enviam mensagens a eles via o ESB, com o qual se conectam. Com o serviço publicado no ESB, diferentes plataformas podem se conectar ao serviço via mensagens SOAP ou conectores disponibilizados pelo ESB. Além disso, o ESB permite o monitoramento da execução do serviço, roteamento e balanceamento de carga, entre outras características.

Neste estudo de caso, foi utilizado o OSB (Oracle Service Bus). O OSB foi instalado no servidor Weblogic, através do sbconsole (http://localhost:7001/sbconsole), e os serviços executam no servidor Tomcat.

Para disponibilização dos serviços no OSB, foram criados três projetos no OSB, com os nomes de ServicoCadastroCliente, ServicoCalculaContrato e

ServicoLimiteCredito. Dentro de cada projeto foram criadas três pastas: WSDL, BusinessService e ProxyService. Na pasta WSDL foi criado um recurso a partir de uma URL: no caso, a URL é a do WSDL do respectivo serviço que está publicada no Tomcat. Por exemplo, no caso do ServicoCadastroCliente a url passada foi: http://localhost:8080/jaxws-ServicoClienteApp/ServicoClienteWS?wsdl. Após a criação desses recursos a partir do WSDL, foram criados os BusinessServices tendo como base os WSDLs importados, como pode ser observado na Figura 44. Maiores detalhes sobre publicação de serviços no OSB são apresentados por Sousa *et al.* [2010].

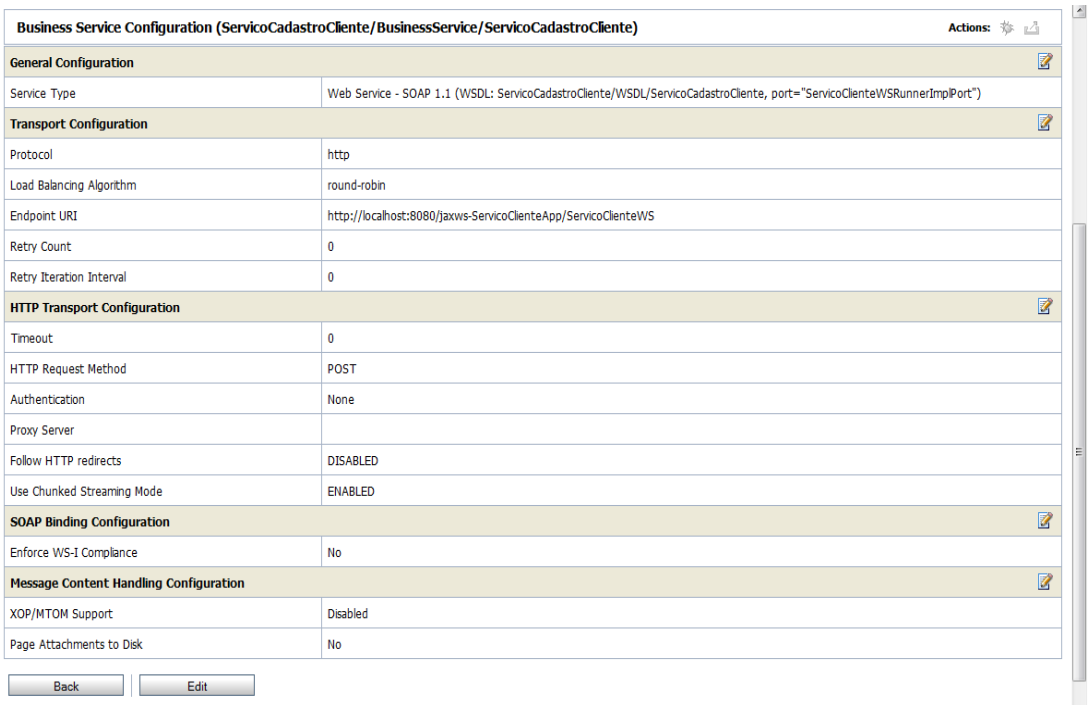


Figura 44 – Configuração do Business Service

Com o Business Service criado foi possível criar um Proxy Service que apontasse para o Business Service. Sendo assim o Proxy Service recebe as chamadas e repassa ao Business Service que na verdade invoca o *web service* que está publicado em outro servidor de aplicação, no caso no Tomcat. Com a criação do Proxy Service o WSDL do serviço agora pode ser acessado através da seguinte URL: http://localhost:7001/ServicoCadastroCliente/ProxyService/ServicoCadastroClienteProxy?WSDL. A nova arquitetura para execução dos *web services* é apresentada na Figura 45.

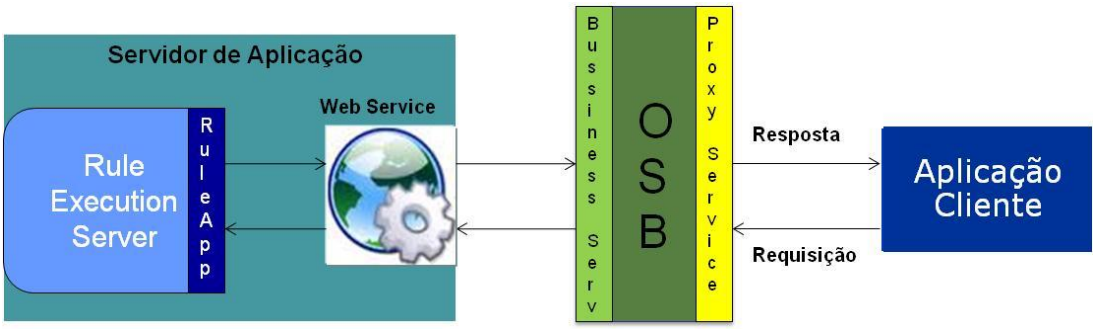


Figura 45 - Arquitetura de execução dos *web services* utilizando o OSB

5.6 Criação da aplicação cliente

Antes de criar a aplicação cliente, foi criada uma base de dados no PostgreSQL com uma tabela de clientes para ser acessada. Para contemplar as regras de cadastro, foi criada a tabela clientes com os atributos da Classe cliente. Ao iniciar a aplicação cliente, um formulário é aberto, pedindo as informações do cliente (CPF, Telefone, Endereco e Renda). A partir do CPF é feita uma consulta ao banco e são enviados dois objetos para a regra: um objeto cliente que é retornado pelo banco a partir da consulta e outro objeto cliente com as informações fornecidas pelo usuário. Um exemplo da tabela cliente e os seus registros podem ser visualizados Figura 46.

	cpf [PK] characte	nome character var	idade integer	renda real	telefone character var	endereco character var	categoria character var	estadoCadast character var	creditoAberto real	parcelasAber integer	limiteCredito real
1	11111	clienteC	18	2000	1111-1111	BA, Brasil	Ouro				
2	55555	clienteD	23	3000	5555-5555	RJ, Brasil	Bronze		500	2	
3	12345	clienteA	55	15000	1234-5678	SP, Brasil	Ouro		1000	5	
4	22222	clienteB	30	5000	9999-9999	RS, Brasil	Prata		2500	5	
5	77777	clienteE	60	30000	7777-7777	AM, Brasil	Bronze		5000	5	
6	88888	clienteF	40	5000	8888-8888	PE, Brasil	Prata		3000	3	
*	99999	clienteG	25	1000	9999-9999	MT, Brasil	Bronze				

Figura 46 – Tabela de clientes no banco de dados

Utilizando os projetos clientes gerados pelo *wizard Client Project For RuleApp* foi criada uma aplicação cliente, que apresenta a interface descrita e invoca os 3 serviços, realizando a orquestração dos mesmos. Para isso, foi criada a classe Principal.Java, que contém o método executarServicos que realiza a chamada aos serviços, como pode ser observado na Figura 47. Esta aplicação envia a requisição ao serviço de Proxy no Oracle Service Bus, que por sua vez invoca o Business Service e retorna a resposta para a aplicação cliente.

```
public CreditoConcedido executarServicos(Cliente cliente, Proposta
proposta) {
        Cliente clienteBanco;
        Contrato contrato = new Contrato();
        CreditoConcedido creditoConcedido = new CreditoConcedido();
        ClienteDAO clienteDAO = new ClienteDAO();


        clienteBanco = clienteDAO.getCliente(cliente.getCpf());
        cliente = ServicoClienteClient.checkCadastro(cliente,
clienteBanco);
        if (cliente.getEstadoCadastro().contains("Desatualizado")) {
            JOptionPane.showMessageDialog(null, "O cliente precisa ter
seu cadastro atualizado");
            return null;
        }
        if(cliente.getEstadoCadastro().contains("Cliente Novo")){
            JOptionPane.showMessageDialog(null, "O cliente precisa ser
cadastrado");
            return null;
        }
        cliente =
ServicoLimiteCreditoClient.calculaLimiteCredito(cliente);
        System.out.println(clienteBanco.getLimiteCredito());

        proposta = ServicoLimiteCreditoClient.verificarLimite(cliente,
proposta);
        if(proposta.getEstadoProposta().contains("Rejeitada")){
            JOptionPane.showMessageDialog(null, "O limite de crédito do
cliente não é suficiente");
            return null;
        }


        creditoConcedido =
ServicoLimiteCreditoClient.comprometerLimiteCredito(proposta,
creditoConcedido);

        creditoConcedido =
ServicoCalculoContratoClient.calcularJuros(cliente,creditoConcedido);
        creditoConcedido =
ServicoCalculoContratoClient.calcularValorTotal(creditoConcedido);


cliente.setCreditoAberto(cliente.getCreditoAberto()+creditoConcedido.get
ValorTotal());

cliente.setParcelasAberto(cliente.getParcelasAberto()+creditoConcedido.g
etParcelasAprovadas());
```

Figura 47 – Método executarServicos da classe Principal

Foi criada uma interface gráfica para entrada dos dados do Cliente e da Proposta. Essa interface se comunica com a classe Principal, chamando o método executarServico. A interface, com um exemplo de execução da aplicação, é apresentada na Figura 48 e na Figura 49. Caso o CPF passado dentro do formulário de requisição não esteja cadastro no banco, a aplicação exibe uma mensagem informando que o cliente precisa ser cadastrado (Figura 50). Entretanto, caso o CPF passado esteja cadastrado no banco, mas o telefone, renda ou endereço sejam

diferentes do que está no banco, é exibido um aviso de cliente desatualizado (Figura 51). As funcionalidades de cadastrar e atualizar cliente não foram implementadas, pois as mesmas estão fora do escopo do trabalho.

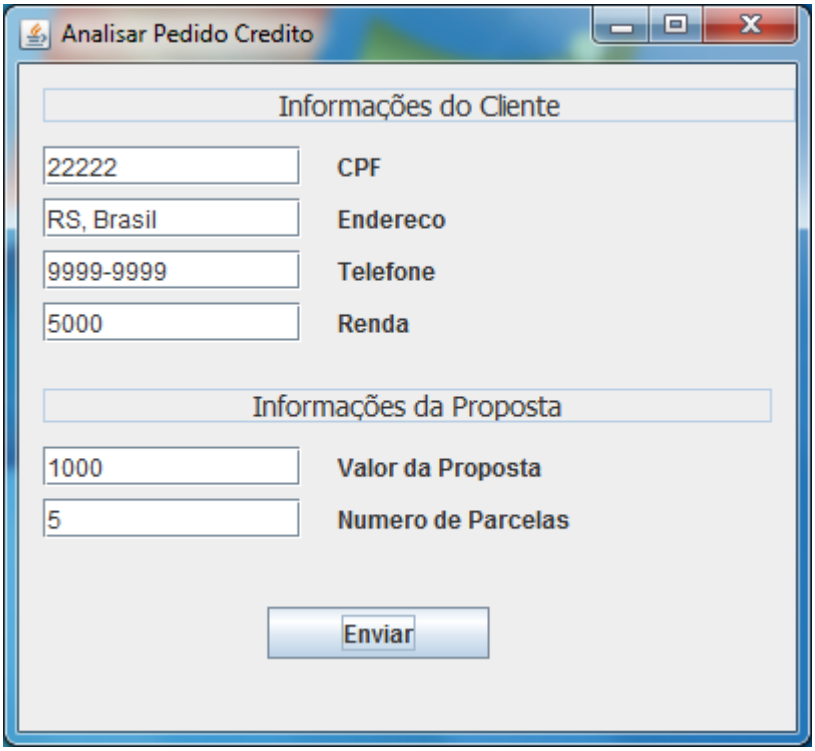


Figura 48 - Tela principal da aplicação

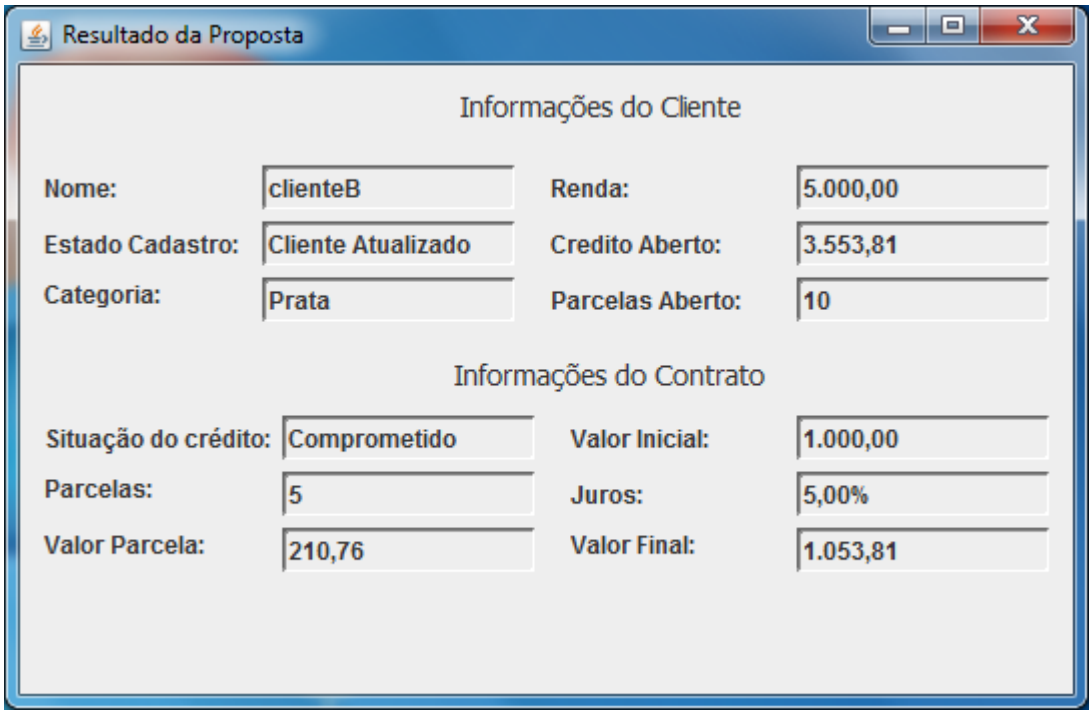


Figura 49 - Tela de resposta da aplicação

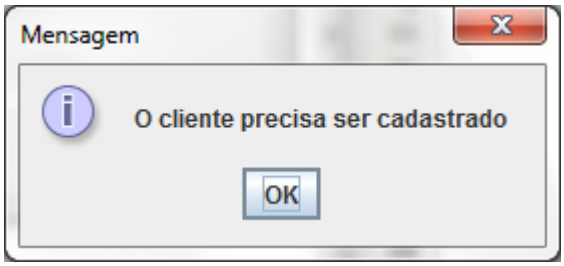


Figura 50 – Mensagem de erro de cliente novo

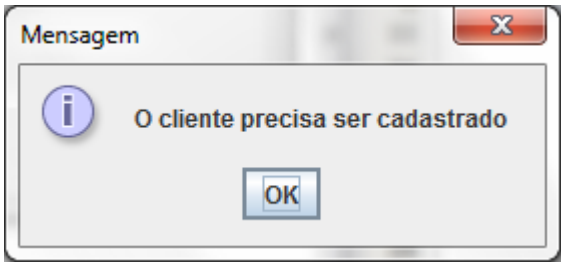


Figura 51 - Mensagem de erro de cliente desatualizado

5.7 Execução da aplicação

Com a aplicação criada e rodando é possível executar os *web services* que foram gerados pelo *Client For RuleApp Project wizard*. Quando a aplicação é executada, o formulário para entrada dos dados do cliente e da proposta é apresentado e após apertar o botão de "Enviar" o método de ação do botão (Figura 52) é chamado. Este método faz uma chamada ao método sendProposta (Figura 53) da classe FormProposta.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
        CreditoConcedido creditoC = new CreditoConcedido();
        sendProposta();

    }
```

Figura 52 – Método de ação do botão "Enviar"

```
public void sendProposta() {
        Proposta proposta = new Proposta();
        CreditoConcedido creditoC = new CreditoConcedido();
        Cliente cliente = new Cliente();

        cliente.setCpf(jTextFieldCPF.getText());
        cliente.setEndereco(jTextFieldEndereco.getText());
        cliente.setTelefone(jTextFieldTelefone.getText());
        if(!jTextFieldRenda.getText().isEmpty())

cliente.setRenda(Float.parseFloat(jTextFieldRenda.getText()));


        if(!jTextFieldValor.getText().isEmpty())

proposta.setValorFinanciamento(Float.parseFloat(jTextFieldValor.g
etText()));
        if(!jTextFieldParcelas.getText().isEmpty())

proposta.setParcelas(Integer.parseInt(jTextFieldParcelas.getText(
)));


        creditoC = new
Principal().executarServicos(cliente,proposta);
        if(creditoC != null){
        FormResposta resp = new FormResposta(creditoC);
        resp.setVisible(true);
        }
    }
```

Figura 53 – Método sendProposta

O método sendProposta passa as informações para o método executarServicos (Figura 47) e o mesmo aplica as regras. O método tem como retorno um objeto da classe creditoConcedido, mas se o cliente for novo ou estiver desatualizado, ou o limite de crédito não for suficiente, o método para de executar e é exibida uma mensagem de erro como apresentado na Figura 54.

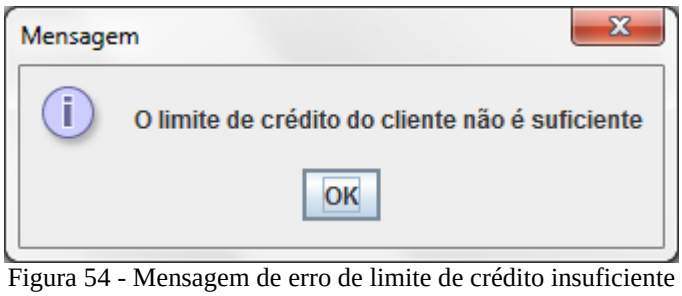


Figura 54 - Mensagem de erro de limite de crédito insuficiente

Para cada serviço dentro da aplicação há um pacote que contém todas as classes geradas necessárias para a execução do serviço. As mais importantes são a {NomeDoServico}RunnerImplService.Java (Figura 56) e {NomeDoServico}Client.Java (Figura 55). A classe RunnerImplService define como é feita a chamada do serviço e a URL do WSDL que será utilizado. A classe Client

apenas realiza a chamada do serviço. Ela oferece todos os métodos que são gerados a partir dos projetos de regra.

```
public class ServicoLimiteCreditoClient {

	public static CreditoConcedido comprometerLimiteCredito(Proposta
proposta, CreditoConcedido creditoC) {
		ServicoLimiteCreditoWSRunner port = new
ServicoLimiteCreditoWSRunnerImplService()
				.getServicoLimiteCreditoWSRunnerImplPort();

		Proposta propostaParam = proposta;
		CreditoConcedido creditoConcedidoParam = creditoC;


		Holder<CreditoConcedido> hcreditoConcedidoParam = new
Holder<CreditoConcedido>(
				creditoConcedidoParam);
		String outputString = null;
		Holder<String> houtputString = new
Holder<String>(outputString);
		String userdata = null;
		Holder<String> huserdata = new Holder<String>(userdata);
		huserdata = new Holder<String>(

	"ServicoLimiteCreditoApp.ServicoLimiteCreditoWSRunnerClient.exec
utecomprometerLimiteCredito");
		try {

	port.comprometerLimiteCredito(hcreditoConcedidoParam,
					propostaParam, houtputString,
huserdata);
		} catch (RulesetExecutionException_Exception e) {
			e.printStackTrace();
		}
		return hcreditoConcedidoParam.value;

	}
```

Figura 55 - Classe ServicoLimiteCreditoClient

```
@WebServiceClient(name = "ServicoClienteWSRunnerImplService",
targetNamespace = "http://ServicoClienteApp/",
wsdlLocation = "http://localhost:8080/jaxws-
ServicoClienteApp/ServicoClienteWS?wsdl")
public class ServicoClienteWSRunnerImplService
    extends Service
{

    private final static URL
SERVICOCLIENTEWSRUNNERIMPLSERVICE_WSDL_LOCATION;

    static {
        URL url = null;
        try {
            url = new
URL("http://localhost:7001/ServicoCadastroCliente/ProxyService/ServicoCa
dastroClienteProxy?wsdl");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        SERVICOCLIENTEWSRUNNERIMPLSERVICE_WSDL_LOCATION = url;
    }

    public ServicoClienteWSRunnerImplService(URL wsdlLocation, QName
serviceName) {
        super(wsdlLocation, serviceName);
    }

    public ServicoClienteWSRunnerImplService() {
        super(SERVICOCLIENTEWSRUNNERIMPLSERVICE_WSDL_LOCATION, new
QName("http://ServicoClienteApp/",
"ServicoClienteWSRunnerImplService"));
    }
```

Figura 56 – Classe ServicoLimiteCreditoRunnerImplService

Após a execução dos serviços, a classe FormResposta inicializa a classe FormResposta passando as informações que serão mostradas na tela de resposta.

5.8 Análise dos resultados do estudo de caso

Este estudo de caso apresentou o uso da ferramenta WebSphere Ilog JRules para criar regras de negócio descritas em um modelo de processo de negócio e disponibilizar as mesmas como serviços em um ambiente SOA.

Foi possível observar com este estudo de caso que a disponibilização das regras de negócio como serviço traz ganho na gestão das mesmas, através da descrição das regras em um nível mais alto do que o de codificação. A manutenção das regras não é feita diretamente no código, mas sim em um ponto auto-contido que é disponibilizado como serviço. Logo, não é necessário alterar as aplicações cliente. Além disso, aumenta-se o reuso das regras por diferentes aplicações, que podem estar escritas em qualquer linguagem de programação e disponibilizadas em

diferentes plataformas. Isto é possível porque as regras são disponibilizadas de forma padronizada e de acordo com o princípio de SOA de interoperabilidade.

Portanto, é possível utilizar uma ferramenta BRMS para desenvolver regras e utilizá-las em um contexto SOA. As ferramentas BRMS oferecem funcionalidades para a criação de regras de forma textual, possibilitando que usuários que não tenham conhecimentos profundos de programação criem suas próprias regras, e com isso a gestão das regras torna-se mais rápida e eficiente, permitindo que o usuário do negócio modifique as de forma imediata, diferentemente do que aconteceria se as mesmas estivessem embutidas nas aplicações, onde seria necessário uma equipe de TI para gerar uma nova versão do software. Com a disponibilização das regras em um ambiente SOA, essas regras podem ser reutilizadas e podem ser executadas por aplicações de diferentes plataformas.

A ferramenta apresentou algumas limitações, como a impossibilidade de disponibilizar o *web service* gerado pela ferramenta em outro servidor de aplicação, diferente do Rule Execution Server. Além disso, a ferramenta apresenta 3 opções de geração de *web services*, onde todas tem limitações, como apresentado na Figura 57.

Features or constraints	Hosted transparent decision service	Monitored transparent decision service	JRules Web service
Ready to use (one artifact to deploy)	✓	⊘	⊘
Manage complete XML parameter signature	✓	⊘	⊘
All supported platforms	✓	⊘	⊘
Ruleset executions monitored	✓	✓	⊘
Manage Java XOM signature	⊘	✓	✓
Rule Studio generation wizard	⊘	✓	✓
Tomcat 6.0 / JDK 1.5 / JAX-WS 2.1.1 / JBoss	✓	✓ on JBoss4.2 , JBoss 5 and Tomcat only **Important** On JBoss, hosted decision services and monitored decision services are mutually exclusive: you can run one or the other, but not both.	✓
XML parameters represented as a String (not as a business signature)	⊘	✓	✓

Figura 57 - Restrições e funcionalidades das implementações de *web services*

A publicação dos serviços que implementam as regras no OSB e o wizard que gera o código do *web service* e do cliente são pontos positivos observados neste

estudo de caso. A publicação dos serviços não é complexa, e com pouco trabalho é possível realizar a chamada do serviço através do ProxyService implementado pelo OSB. O *wizard* de criação do *web service* da regra e de seu cliente possibilitam o desenvolvedor, com pouco trabalho, executar a regra publicada no servidor de aplicação.

5.9 Resumo

Este capítulo apresentou um estudo de caso, onde o objetivo foi demonstrar o uso da ferramenta WebSphere Ilog JRules para desenvolver regras de negócio e disponibilizá-las como serviços em uma arquitetura orientada a serviços.

O estudo de caso utilizou as regras de negócio apresentadas no modelo de processo "Analisar pedido de crédito" [Diirr *et al.*, 2010]. Foram selecionadas 11 regras de negócio, que compreendiam 7 atividades do processo, para serem desenvolvidas nesse estudo de caso.

Para o desenvolvimento das regras de negócio foi criado o modelo que representa os objetos das regras: no caso, o XOM (*Execution Object Model*) e o BOM (*Business Object Model*). O XOM é composto por 3 classes Java: Cliente, Proposta e CreditoConcedido. A partir do XOM foram criadas as entidades do BOM, com os mesmos nomes das classes, as quais foram utilizadas para a criação das regras de negócio. As regras de negócio foram criadas dentro de 6 projetos de regras diferentes. Cada projeto contendo seu próprio fluxo de execução e suas regras. A partir desses projetos foi gerado o RuleApp, que é necessário para a disponibilização das regras como serviço.

Com os projetos de regras criados e os RuleApps gerados, foi executado o wizard *Client Project For RuleApp*, utilizando o *template* de geração de *web services.* Após a execução do wizard, 2 projetos foram criados automaticamente: um projeto que contém o código do *web service* e que é publicado no servidor de aplicação e outro projeto que contém o código da aplicação cliente que invoca o *web service*.

Após a publicação dos serviços no servidor de aplicação, esses serviços foram publicados no Oracle Service Bus, e foram criados um *Business Service*, que é o

serviço que efetivamente executa o *web service* que está no outro servidor de aplicação, no caso o Tomcat, e foi criado um *Proxy Service* que é o serviço que fará o roteamento das chamadas no OSB até o *Business Service.*

Em seguida foi criada uma aplicação cliente, que contém uma interface gráfica que permite a entrada dos dados do cliente e da proposta e retorna, em outra tela, a resposta com os dados do contrato e do cliente atualizados. Essa aplicação se conecta com um banco de dados, que contém uma tabela cliente, para obter as informações do cliente e poder enviar para as regras. Essa aplicação cliente chama os métodos dos *Proxy services* que foram disponibilizados no OSB.

No final do capítulo foi apresentada uma análise dos resultados obtidos através da execução do estudo de caso. Foram destacados pontos positivos, como a facilidade em se criar *web services* a partir das regras, e pontos negativos, como as restições para cada tipo de *web service* gerado pela ferramenta.

Capítulo 6: CONCLUSÃO

Regras de negócio são declarações que definem ou restringem alguns aspectos do negócio. Para que a divulgação do conhecimento organizacional através das regras de negócio seja efetiva, é necessário que estas sejam organizadas e gerenciadas. Esse gerenciamento de regras de negócio necessita de um apoio computacional, que é fornecido pelos BRMS. Os sistemas gerenciadores de regras de negócio (BRMS) têm como objetivo prover um conjunto de funcionalidades capaz de apoiar uma estratégia de gestão de regras de negócio completa. Além disso, é importante que essas regras sejam integradas às aplicações sem necessidade de grandes mudanças nas mesmas. Uma das abordagens para atender este requisito é a disponibilização das regras de negócio como serviços em uma arquitetura orientada a serviços (SOA).

A arquitetura orientada a serviços (SOA) é um paradigma para a realização e a manutenção dos processos corporativos que se encontram em grandes sistemas distribuídos. Esses processos são executados em diferentes passos (atividades ou tarefas) em diferentes sistemas. Um serviço tem como objetivo primário representar uma funcionalidade do negócio. A arquitetura de *web services* é a mais utilizada para implementar serviços em uma arquitetura SOA, já que é baseada em um conjunto padrões. SOA demanda suporte computacional e infra-estrutura para integrar diferentes plataformas e linguagens de programação. A principal infra-estrutura de SOA é o *Enterprise Service Bus* (ESB). O ESB é um barramento de mensagens baseado em padrões e aberto, que permite a implementação, disponibilização e gerenciamento de soluções SOA.

Neste trabalho foi realizada uma avaliação de ferramentas BRMS para utilização em uma arquitetura SOA. Foram listadas 16 ferramentas BRMS, entre as quais 10 possibilitam a disponibilização das regras de negócio como serviço. A avaliação de ferramentas não é trivial, devido a diversos problemas, como o grande número de ferramentas presentes e a falta da definição de requisitos. Para atender a essa demanda, torna-se necessária a aplicação de um método para avaliação e seleção

de ferramentas. A prospecção de ferramentas realizada neste trabalho segue o processo de avaliação de ferramentas proposto por Azevedo *et al.* [2010b].

Após a avaliação das ferramentas constatou-se que a ferramenta que seria mais adequada para ser utilizada no estudo de caso, apresentado no Capítulo 5: deste trabalho, foi a WebSphere Ilog BRMS. A ferramenta Ilog é composta por 4 módulos que executam atividades distintas e possibilita a disponibilização das regras de negócio como serviços. Essa disponibilização pode ser feita de 3 maneiras, através de serviços de decisão transparentes hospedados ou monitorados e *web decision service*. No estudo de caso as regras de negócio desenvolvidas foram implementadas como serviços de decisão transparentes monitorados.

O estudo de caso utilizou as regras de negócio apresentadas no modelo de processo "Analisar pedido de crédito" [Diirr *et al.*, 2010]. Foram selecionadas 11 regras de negócio, que compreendiam 7 atividades do processo, para serem desenvolvidas nesse estudo de caso. Essas regras foram desenvolvidas utilizando a ferramenta selecionada e as mesmas foram disponibilizadas em 3 serviços, que atendiam às regras de negócio relacionadas ao cadastro do cliente, ao limite de crédito do cliente e a confecção e cálculo do contrato. Esses serviços foram publicados no Oracle Service Bus. Para o teste da solução, foi desenvolvida uma aplicação cliente que era responsável por realizar as chamadas aos serviços e passar as informações necessárias para a execução das regras.

Com isso conclui-se que o uso de ferramentas BRMS para gestão de regras e disponibilização destas como serviços em uma arquitetura orientada a serviços (SOA) é possível. Além disso, deve se pensar que a utilização de uma ferramenta BRMS demanda uma integração com as aplicações que utilizam as regras de negócio de forma transparente, o que SOA atende.

Este trabalho contempla as seguintes contribuições: descrição dos conceitos de regras de negócio, ferramentas de gestão de regras de negócio, SOA e ESB; avaliação de ferramentas BRMS para utilização em um contexto SOA; descrição da ferramenta escolhida na avaliação, envolvendo a arquitetura, os principais conceitos da ferramenta e a disponibilização das regras como serviços; execução de um estudo de caso de uso de ferramentas BRMS em uma abordagem SOA; e a análise dos resultados obtidos no estudo de caso.

A importância da gestão de regras de negócio nas organizações é cada vez maior e com isso o uso de ferramentas BRMS deve se intensificar. Como extensão deste trabalho estamos definindo novos critérios específicos para avaliação de ferramentas em um ambiente SOA. Como trabalhos futuros pode-se abordar o uso dessas ferramentas em diferentes domínios, ou mesmo utilizar a ferramenta para guiar a execução de um processo de negócio por completo. É possível também integrar as regras de negócio utilizando outras abordagens para a integração das regras de negócio com as aplicações como, por exemplo, utilizando SCA [Chapel, 2009; Azevedo *et al.*, 2010d]. Além disso, é preciso definir como pode ser feita a governança da gestão de regras disponibilizadas como serviços em uma arquitetura orientada a serviços. Outras possibilidades de trabalhos futuros são:

- A adoção de metodologias (heurísticas) para gestão de regras de negócio em SOA;
- Realizar uma avaliação do desempenho dos serviços de regras de negócio e a integração com as ferramentas;
- Realizar uma avaliação e comparação das abordagens de implementação de sentenças estruturais no SGBD (*triggers*, *storage procedures* e esquema do SGBD) em relação ao uso de uma BRMS para implementação deste tipo de regra;
- Trabalhar no mapeamento conceitual, lógico do XOM/BOM para os conceitos do negócio.

Capítulo 7: REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, L. G., SOUZA, J., LOPES, M., SIQUEIRA, S., CAPPELLI, C., BAIAO, F.A., *et al*. "Inspeção de Ferramentas de Ontologias". Relatório Técnico 0003/2008, Departamento de Informática Aplicada da UNIRIO, Rio de Janeiro, 2008.

AZEVEDO, L; DUARTE, D.; PUNTAR, S.; ROMEIRO, C.; BAIAO, F.; CAPPELLI, C. **Conceituação em BRMS**. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0022/2009, 2009. Disponível (também) em: http://seer.unirio.br/index.php/monografiasppgi.

AZEVEDO, L.; DUARTE, D.; BAIAO, F.; CAPPELI, C., **Avaliação de Ferramentas para Gestão e Execução de Regras de Autorização.** In: VI Simpósio Brasileiro de Sistemas de Informação (SBSI 2010), Marabá, PA, Brasil, 2010a.

AZEVEDO, L.; ROMEIRO, C.; CAPPELLI, C.; BAIAO, F. **Passo-a-Passo para Avaliação de Software para Aquisição em Modelos de Processos de Negócio**. Relatórios Técnicos do DIA/UNIRIO, RT-0002/2010, 2010b. Disponível (também) em: http://np2tec.uniriotec.br:9093/np2tec/publicacoes/RT-NP2TEC-2010-002-Passo-a-Passo%20para%20Avaliacao%20de%20Software%20para%20Aquisicao%20em%20Modelos%20de%20Processos%20de%20Negocio.pdf

AZEVEDO, L. G., Puntar, S., Thiago, R., Baião, F., Cappelli, C., **A Flexible Framework for Applying Data Access Authorization Business Rules**. In: 12th International Conference on Enterprise Information Systems (ICEIS 2010), Funchal, Madeira, Portugal, 2010c.

AZEVEDO, L. G., SANTORO, F., BAIÃO, F., **Estudo dos Principais Conceitos sobre *Service Component Architecture* (SCA).** Relatórios Técnico NP2Tec/UNIRIO, 2010. Disponível em <http://np2tec.uniriotec.br:9093/np2tec/publicacoes/relatorios-tecnicos>. Acessado em agosto de 2010.

BOTTO, R.: *Arquitetura Corporativa de Tecnologia da Informação*. Ed. Brasport, 2004.

BRG. Business Rules Group, **Defining Business Rules – What are they really?**. Julho de 2000 . Disponível em: http://www.businessrulesgroup.org/first_paper/br01c0.htm.

CHAPPELL, D. **Introducing SCA**. Chappell & Associates., 2007. Disponível em <http://www.davidchappell.com/articles/introducing_sca.pdf>. Acessado em 11 Dez. 2009.

COMELLA-DORA, S., DEAN, J., LEWIS, G., MORRIS, E., OBERNDORF, P., HARPER, E. **A Process for COTS Software**. Technical Report CMU/SEI-2003-TR-017. Carnegie Mellon Software Engineering Institute, 2004.

CRERIE, R., **Um Método Para Descoberta De Regras De Negócio Através De Mineração**. Dissertação de Mestrado – PPGI, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

CHAPPELL, D., **Enterprise Service Bus**. O'Reilly Media, Inc., 2004.

DEITERT, E., MCCOY, D., **The Anatomy of a Business Rule Management System**, Gartner Research, 2007.

DIIRR, T., SOUZA, A, AZEVEDO, L. G., SANTORO, F. **Modelo do Processo Analisar pedido de crédito**. (2010). Relatórios Técnico NP2Tec/UNIRIO, 2010. Disponível em <http://np2tec.uniriotec.br:9093/np2tec/publicacoes/relatorios-tecnicos>. Acessado em agosto de 2010.

ERICKSON, J., SIAU, K., **Web Services, Service-Oriented Computing, and Service-Oriented Architecture: Separating Hype from Reality,** Journal of Database Management, 19(3): 42-54, 2008

ERL, T., Service-Oriented Architecture: concepts, technology, and Design. Prentice Hall, 2005.

EVERNDEN, R., EVERNDEN, E.: "Third-generation information architecture". *Communications of the ACM* 46.3 (March 2003): pp. 95-98.

FORGY, C. **Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem**. Artificial Intelligence 19(1): 17-37 (1982)

FURTADO, C. ; PEREIRA ; AZEVEDO, L. G. ; BAIAO, F. ; SANTORO, F. . **Arquitetura Orientada a Serviço: Conceituação**. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0012/2009, 2009. Disponível (também) em: http://seer.unirio.br/index.php/monografiasppgi.

GRAHAM, I., **Business Rules Management and Service Oriented Architecture: A Pattern Language**, Wiley, 2007.

HALLE, B. V. **Business Rules Applied: Building Better Systems Using the Business Rules Approach**, Wiley, 2002.

HEWITT, E., **Java SOA Cookbook**, O'Reilly Media, Inc., 2009.

ILOG. **IBM WebSphere ILOG JRules Version 7.2 documentation**. Disponível em http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m2/index.jsp. Acesso em: 26 Jul. 2010.

JOSUTTIS, N. M., **SOA in Practice: The Art of Distributed System Design**, O'Reilly, 2007.

MARKS, E. A.; BELL, M., **Service-Oriented Architecture: a planning and implementation guide for business and techonology**, John Willey & Sons Inc, 2006.

MORGADO, G. P., GESSER, I, SILVEIRA, D. S., *et al.*, **Práticas do CMMI como Regras de Negócio**, *Revista Produção*, vol. 17, n. 2, pp.383-394, 2007.

NELSON, M., RARIDEN, R., SEN, R., **A Lifecycle Approach towards Business Rules Management**, Proceedings of the 41st Hawaii International Conference on System Sciences, pp 113-123, 2008.

NEWCOMER, E. and LOMOW, G., **Understanding SOA with Web Services** (Independent Technology Guides). Addison-Wesley Professional, 2004.

PAPAZOGLOU, MIKE P.; HEUVEL, WILLEM-JAN, **Service oriented architectures: approaches, technologies and research issues**, VLDB Journal, Springer-Verlag, 2007.

RETEPLUS. **The RetePlus algorithm.** Disponível em http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m2/index.jsp?topic=/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global990.html Acesso em: 09 Ago. 2010.

SCHULTE, R., **Predicts 2003: Enterprise service buses emerge**. Report, Gartner, December 2002

SEQUENTIAL. **The Sequential algotithm.** Disponível em http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m2/index.jsp?topic=/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global990.html Acesso em: 09 Ago. 2010.

SINUR, J., MCCOY, D. W., **Business Rule Engines: 10 Questions to Ask Before Buying**, Gartner Research, 2007.

SOAP, GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J., NILSEN, H., KARMARKAR, A., LAFON, Y., **SOAP Version 1.2 specification**, World Wide Web Consortium Note 27 April 2007. Disponível em < http://www.w3.org/TR/soap12-part1>. Acesso em: 20 Jul. 2010.

SOUSA, H. P., AZEVEDO, L. G., SANTORO, F., BAIÃO, F. "Estudos de ESB e OSB". Relatórios Técnico NP2Tec/UNIRIO, 2010. Disponível em <http://np2tec.uniriotec.br:9093/np2tec/publicacoes/relatorios-tecnicos>. Acessado em agosto de 2010.

UDDI, **Universal Description, Discovery & Integration Specification version 3**, 2004. Disponível em <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3>. Acesso em: 20 Jul. 2010.

W3C, BOOTH, D., HAAS, H., McCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., ORCHARD, D., **Web Services Architecture**, W3C Working Group Note 11 February 2004. Disponível em <http://www.w3.org/TR/ws-arch/#engaging>. Acesso em: 20 jul. 2010.

WSDL, CHINNICI, R., MOREAU, J., RYMAN, A., WEERAWARANA, S., **Web Services Description Language (WSDL) 2.0**, World Wide Web Consortium Note 26 June 2007. Disponível em < http://www.w3.org/TR/wsdl20>. Acesso em: 20 Jul. 2010.

WSDL, CHRISTENSEN, E., CURBERA, F., MEREDITH, G., WEERAWARANA, S., **Web Services Description Language (WSDL) 1.1**, World Wide Web Consortium Note 15 March 2001. Disponível em <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 20 Jul. 2010.

XML, BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C., MALER, E., YERGEAU, F., **Extensible Markup Language (XML) 1.0 (Fifth Edition)**, World Wide Web Consortium Recommendation 26 November 2008. Disponível em <

http://www.w3.org/TR/2008/REC-xml-20081126/>. Acesso em: 23 Jul. 2010.

Apêndice 1 Critérios Genéricos

Os critérios genéricos utilizados para a avaliação são apresentados nas tabelas a seguir.

Tabela 3 – Distribuição

Critérios genéricos para distribuição		
Critério	**Peso**	**Forma de pontuação**
Como são entregues as novas versões da ferramenta?	1	0: Instalação no local pelo fabricante em prazo maior que 1 semana 0,5: Instalação no local pelo fabricante em prazo máximo de 1 semana 0,5: Via midia (CD, DVD) 1: Internet
A empresa está em algum processo de venda ou junção?		1: Não 0: Sim
A empresa foi adquirida recentemente por outra empresa?		0,00: t ≤ 2 anos ou empresa tem menos do que 2 anos 0,50: 2 anos < t ≤ 5 anos 1,00: t > 5 anos ou não foi adquirida por outra empresa
Quantos clientes para o produto o fornecedor possui?		0: Menos do que 20 clientes 0,25: 20 a 50 clientes 0,5: 50 a 100 clientes 1: Mais do que 100 clientes
Há quanto tempo a empresa está no mercado?		0: Menos de 1 ano 0,25: Entre 1 e 5 anos 0,5: Entre 5 e 10 anos 1: Mais do que 10 anos
Há quanto tempo a empresa disponibiliza o produto para o mercado?		0: Menos de 1 ano 0,5: Entre 1 e 5 anos 1: Mais do que 5 anos
A empresa vende o produto de gestão de regras, como um produto único?		0: Não 1: Sim
Novas versões são integráveis com a versão corrente?		0: Não 1: Sim
Total (8 critérios)	**8**	

Tabela 4 – Escalabilidade

Critérios genéricos para escalabilidade		
Critério	**Peso**	**Forma de pontuação**
Número máximo de regras que a ferramenta permite incluir/executar?	5	0: < 1.000 0,5: >= 1.000 e < 10.000 1: >= 10.000
Tempo para execução de consultas?	0	0: t > 0,25 s 0,5: 0,01 s ≤ t ≤ 0,25 s 1: t < 0,01 s
Total (2 critérios)	**5**	

Tabela 5 – Flexibilidade

Critérios genéricos para flexibilidade		
Critério	**Peso**	**Forma de pontuação**

A ferramenta permite a adição de novas funcionalidades (extensível)?	4	0: Não 1: Sim
Quais são os requisitos para realizar a programação/criação de novas funcionalidades (requisitos de linguagem de programação e software)?		0: Não é possível solicitar criação de novas funcionalidades para atender requisitos da Petrobras 0,25: Novos requisitos da Petrobras podem ser implementados pela contratação de equipe de customização pelo fornecedor. 0,5: É possível realizar a implementação de novas funcionalidades por equipe própria da Petrobras. Caso necessário, o fornecedor pode dar suporte para este trabalho via contrato específico. 1: É possível realizar a implementação de novas funcionalidades por equipe própria da Petrobras, e o fornecedor dá apoio a este desenvolvimento sem acréscimo ao contrato.
É possível customizar a ferramenta de acordo com as características específicas da empresa compradora?		0: Não 1: Sim
Total (3 critérios)	**12**	

Tabela 6 - Integração com outros sistemas

Critérios genéricos para integração com outros sistemas		
Critério	**Peso**	**Forma de pontuação**
Permite relacionar itens de regras de negócio com elementos de banco de dados (tabelas, relacionamentos, atributos etc)?	5	0: Não 1: Sim
Permite integração com a ferramenta de modelagem de processos (ARIS) ?	2	
Permite integração com SGDB (Oracle) ?	2	
Total (3 critérios)	**9**	

Tabela 7 - Plataforma tecnológica

Critérios genéricos para plataforma tecnológica		
Critério	**Peso**	**Forma de pontuação**
Instalação no Linux?	3	0: Não 1: Sim
Instalação no Windows?		
Total (2 critérios)	**6**	

Tabela 8 - Qualidade da documentação

Critérios genéricos para qualidade da documentação		
Critério	**Peso**	**Forma de pontuação**
Documentação da uma visão clara dos objetivos e propostas da ferramenta?	2	0: Não 1: Sim
Documentação aborda instalação e configuração da ferramenta?		
Documentação aborda o uso das funcionalidades da ferramenta?		
Documentação contém tutoriais para aprendizagem da ferramenta?		
Documentação aborda questões como escalabilidade e resultados de testes com a ferramenta?		

Existe fórum de discussão da ferramenta e o fórum é utilizado intensamente?		0: Não existe forum ou existe mas possui um volume menor que 10 mensagens por semana. 1: Possui um volume maior que 10 mensagens por semana.
Total (6 critérios)	**12**	

Tabela 9 – Suporte

Critérios genéricos para suporte		
Critério	**Peso**	**Forma de pontuação**
Existe suporte a novas versões da ferramenta?	3 (código fechado)	0: Não 1: Sim
	2 (código aberto)	
Possui suporte por email?	2	
Possui suporte por telefone?		
Possui suporte in loco?		
Fornecedor possui parceiro no Brasil para venda e suporte?		
Total (5 critérios)	**10 ou 11**	

Apêndice 2 Critérios específicos para gestão

Os critérios específicos para avaliação de ferramentas para gestão de regras de autorização são apresentados nas tabelas a seguir.

Tabela 10 – Edição de regras

Critérios genéricos para edição de regras		
Critério	**Peso**	**Forma de pontuação**
Permite uso por múltiplos usuários ao mesmo tempo durante o cadastro das regras?	4	0: usuários simultâneos < 10 0,5: 10 ≤ usuários simultâneos ≤ 20 1: usuários simultâneos > 20
Possui um Ambiente Integrado de Desenvolvimento (IDE) de regras?	2	0: Não 1: Sim
Possui capacidades de inclusão, alteração e remoção de elementos de uma regra?	4	0: Não 1: Sim
Realiza verificação de conflito entre regras durante edição?	3	0: Não 1: Sim
Realiza verificação de sobreposição entre regras ?	2	0: Não 1: Sim
Realiza verificação de redundância entre regras?	2	0: Não 1: Sim
Realiza verificação de regras que nunca serão executadas?	2	0: Não 1: Sim
Realiza a criação de novas regras a partir de regras existentes utilizando mecanismo de dedução? (IA)	2	0: Não 1: Sim
Suporta a criação de vocabulários para as regras?	2	0: Não 1: Sim
Realiza verificações sobre o dicionário de dados? Por exemplo, "Tipos que não são usados por nenhum atributo", "Atributos que não são utilizados por nenhuma regra", "Regras que não foram criadas, mas ainda não foram disponibilizadas", "Regras que são independentes de outras regras"	2	0: Não 1: Sim
Suporta linguagens para descrição de regras? Quais?	2	0,0: Não suporta nenhuma linguagem para descrição de regra. 0,5: Suporta apenas linguagens proprietárias. 1,0: Suporta linguagem padrão além da proprietária.
Permite cadastrar/importar informações de usuários e associá-los às regras indicando quais usuários terão seu acesso controlado pelas regras durante a execução das mesmas?	5	0: permite gerenciar apenas regras e não usuários (resposta que desqualifica a ferramenta) 0,25: permite cadastrar usuários e perfis/grupos de usuários e associá-los às regras 0,75: permite importação de dados de usuários da base dados corporativa, além de permitir cadastrar perfis/grupos de

		usuários e associá-los às regras 1: permite associação de dados de usuários da base dados corporativa com as regras sem necessidade de importar as informações, além de permitir cadastrar perfis/grupos de usuários e associá-los às regras
Permite importar modelo de dados a partir de um banco de dados para construção do vocabulário (termos do negócio)?	5	0: Não 0,5: Permite importar o modelo do banco de dados a partir das tabelas, que é um modelo lógico, mas não permite elaborar um modelo mais abstrato associado ao modelo do banco. 1,0: Permite importar o modelo do banco de dados a partir das tabelas, que é um modelo lógico, e permite elaborar um modelo mais abstrato associado ao modelo do banco.
Permite importar o modelo de dados a partir de modelos gerados por ferramentas de modelagem (por exemplo, modelo Entidade-Relacionamento, modelo de classes?	2	0: Não 1: Sim
Armazena o mapeamento realizado entre modelo de dados?	2	0: Não 0,5: Armazena o mapeamento quando o modelo conceitual é gerado a partir do banco de dados ou (exclusivo) quando o modelo do banco de dados é gerado a partir do conceitual (criado na ferramenta). 1: Armazena o mapeamento quando o modelo conceitual é gerado a partir do banco de dados e quando o modelo do banco de dados é gerado a partir do conceitual (criado na ferramenta).
Permite criação de elementos de regras de negócio utilizando diretamente o modo de edição gráfica?	3	0: Não 1: Sim
Tipos de regras de negócio suportadas (Segundo a classificação BRG)	5	0,00: Não suporta todos os tipos de regras propostos pela [BRG,2000] 1,00: Suporta pelo menos os tipos de regras termos, relacionamento entre termos e autorização propostos pela [BRG,2000]
Oferece template para acelerar o processo de criação de regras?	2	0,0: Não oferece 0,25: possui templates previamente cadastrados, mas não permite editá-los 0,5: Permite criar templates 1,0: Permite criar templates e possui templates previamente cadastrados
Possui wizards para auxiliar usuários do negócio na criação das regras?	4	0,00: Não oferece wizards 0,25: Oferece wizard para regras de "Definição de termos de negócio"

		0,50: Oferece wizard para regras de "Definição de termos de negócio" e "Fatos relacionando termos entre si". 0,75: Oferece wizard para regras de "Definição de termos de negócio" e "Fatos relacionando termos entre si" e "Restrição". 1,00: Oferece wizard para os quatro tipos de regras.
Permite criar wizards para auxiliar usuários do negócio na criação das regras?	2	0: Não 1: Sim
Permite realizar as operações de edição de regras através da API para integração?	5	0: Não 1: Sim
Qual é o número máximo de regras que a ferramenta permite incluir?	3	0: < 1.000 0,5: >= 1.000 e < 10.000 1: >= 10.000
Total (22 critérios)	**65**	

Tabela 11 - Consulta às regras

Critérios genéricos para consulta às regras		
Critério	**Peso**	**Forma de pontuação**
Permite realizar consultas nas linguagens de consultas para regras?	2	0,0: Não permite executar a funcionalidade em nenhuma linguagem. 0,5: Permite apenas em linguagem proprietária 1,0: Permite em linguagem padrão além da proprietária.
Permite consultas restritas a domínios?		0: não permite restringir as consultas 0,25: permite restringir as consultas apenas por projeto 0,50: permite restringir as consultas por projeto e por conceito(regras que se referem a um ou mais conceitos) 1,00: permite restringir as consultas por projeto, por conceito(regras que se referem a um ou mais conceitos) e por domínio.
Possui mecanismos de inferência?		0: Não 1: Sim
Permite utilização dos mecanismos de inferências durante a consulta? Um exemplo de consulta que poderia ser respondida seria o usuário perguntar "Coca-cola engorda?" e dado que na base de regras existem as regras "Coca-cola é refrigerante" e "Refrigerante engorda", a ferramenta, utilizando inferência, responderia sim para esta pergunta.		0: Não 1: Sim
Permite criação de explicações em linguagem natural para as regras definidas pelo usuário?		0: Não 1: Sim
Permite a geração de relatórios		0: Não

derivados dos resultados de consultas?		1: Sim
Permite realizar as operações de consulta a regras através da API para integração?	5	0: Não 1: Sim
Permite gerar relatórios do dicionário de dados das regras?	2	0: Não 1: Sim
Total (7 critérios)	**17**	

Tabela 12 - Visualização de regras

Critérios genéricos para visualização de regras		
Critério	**Peso**	**Forma de pontuação**
Possui mecanismos para visualização de regras?	2	0: Não permite visualização (Atribuir Zero a todas as questões deste macro-critério) 0,5: Permite visualização apenas do texto da regra 1: Permite visualização do texto da regra e também visualização gráfica, explicitando dependências entre regras.
Possui formas de acesso às propriedades dos itens de regras durante sua visualização?		0,00: Não permite visualização (Atribuir Zero a todas as questões deste macro-critério) 0,50: Permite visualização apenas do texto da regra 0,75: Permite visualização do texto da regra e também visualização gráfica, explicitando dependências entre regras. 1,00: Permite visualização do texto da regra e também visualização gráfica, explicitando dependências entre regras e possui formas de acesso às propriedades dos itens de regras durante sua visualização.
É possível imprimir a visão exibida?		0: Não 1: Sim
Permite gerar relatórios das regras de acordo com o que se deseja visualizar?		0: Não 1: Sim
Permite utilizar os mecanismos de visualização de regras através da API para integração?	5	0: Não 1: Sim
Total (5 critérios)	**13**	

Tabela 13 - Exportação / Importação

Critérios genéricos para exportação/importação		
Critério	**Peso**	**Forma de pontuação**
Possui conversores de formatos que apóiam a tradução da regra para outros formatos?	1	0: Não 1: Sim
Importa e exporta regras em diferentes formatos de representa-ção?		0: Não 1: Sim
Posssui ferramentas para derivação de regras (semi) automaticamente utilizando textos de linguagem natural?		0: Não 1: Sim

Total (4 critérios)	4	

Tabela 14 - Repositório e versionamento de regras

Critérios genéricos para repositório e versionamento de regras		
Critério	**Peso**	**Forma de pontuação**
Permite armazenar diferentes versões da regra?	4	0: Não 0,25: Permite armazenar versões do projeto, mas não de uma regra específica. 1: Permite armazenar versões de regras específicas.
Permite comparar e fazer merge entre diferentes versões da regra?	2	0: Não permite comparação e merge entre versões da regra 0,50: Permite comparar diferentes versões da regras 1,00: Permite comparar e fazer merge entre diferentes versões da regra
É possível verificar conflitos e inconsistências entre versões?	4	0: Não 1: Sim
O controle de versões se dá em nível de regras e não somente em nível físico (arquivos)?	2	0: Não 1: Sim
Permite o repositóro ser armazenado em um banco de dados externo à ferramenta, tal como, Oracle, SQL-Server, PostgreSQL?	1	0: Não (resposta que desqualifica a ferramenta, caso ela seja escolhida como ferramenta somente para gestão) 1: Sim
É possível estender o metamodelo do repositório?	3	0: Não permite extensão 0,25: Permite extensão dos atributos da regra 0,75: Permite extensão dos atributos da regra e dos relacionamentos entre regras (composição de regras) 1: Permite extensão dos atributos da regra e dos relacionamentos entre regras (composição de regras) e dos atributos de visualização da regra
O repositório de regras é independente do componente de execução?	0	0: Não 1: Sim
É possível armazenar histórico das alterações ocorridas em cada versão da regra?	4	0: Não 1: Sim
Total (8 critérios)	**20**	

Tabela 15 - Integração de regras

Critérios genéricos para integração de regras		
Critério	**Peso**	**Forma de pontuação**
Permite realizar combinação entre regras?	4	0: Não permite combinação 0,25: permite definir um grupo de regras as quais devem ser válidas para as ações poderem ser executadas. 1,00: permite compor regras com diferentes operadores para realizar a composição.
Permite definir regras de exceção? Por exemplo, em um conjunto de	3	0: Não 1: Sim

regras incluir uma nova regra que se não for satisfeita, faz com que todas as demais regras não sejam executadas ou conjunto de regras retorne falso.		
Total (2 critérios)	**7**	

Tabela 16 - Validação de regras

Critérios genéricos para validação de regras		
Critério	**Peso**	**Forma de pontuação**
Possui ferramentas para teste e depuração?	2	0: Não 1: Sim
Permite geração de relatórios para validação de regras?		0: Não 1: Sim
Permite execução de testes de consistência (ex: regras incomple-tas)?		0: Não 1: Sim
Total (3 critérios)	**6**	

Tabela 17 – Simulação de regras

Critérios genéricos para simulação de regras		
Critério	**Peso**	**Forma de pontuação**
Possui um modelo de simulação de regras?	2	0: Não permite executar simulação. 0,25: Permite executar a simulação de uma única regra independente das demais. 0,5: Permite executar a simulação de um fluxo (fixo) definido. 1: Permite executar a simulação de regras utilizando mecanismo de inferência, ou seja, após a execução de uma regra, infere-se as outras regras que devem ser executas.
Permite visualizar graficamente a sequência de execução da regra e suas dependências?		0: Não 1: Sim
Permite simular a execução da regra utilizando dados novos bem como dados antigos?		0: Não 1: Sim
Total (3 critérios)	**6**	

Tabela 18 – Segurança

Critérios genéricos para segurança		
Critério	**Peso**	**Forma de pontuação**
Permite criação de contas de usuários?	5	0: Não 1: Sim
Possui controle de concorrência?		0: Não 1: Sim
Permite criação de perfis de acesso para usuários?		0: Não 1: Sim
É possível associar os perfis de acesso ao perfil do usuário da Petrobras (associando a chave)?		0: Não 0,25: Sim, mas apenas criando usuários manualmente com a chave/senha igual a da Petrobras 1: Sim, pela importação das informações de um repositório de usuários.
É possível definir as permissões de		0: Não

acordo com os tipos de regras? (Ex.: Regras do Poço, Regras de bloco).		1: Sim
Possui controle de acesso aos elementos das regras?		0: Não 1: Sim
Total (6 critérios)	**30**	

Tabela 19 – Qualidade de regras

Critérios genéricos para qualidade de regras		
Critério	**Peso**	**Forma de pontuação**
Permite a definição de padrões organizacionais para descrição de regras (por exemplo, regras de nomenclatura, etc.)?	1	0: Não 0,5: Permite estender elementos da regra 1,0: Permite definir padrões para tratar elementos de regras
Permite a validação de padrões organizacionais pela ferramenta?		0: Não 1: Sim
Total (2 critérios)	**2**	

Tabela 20 – Administração

Critérios genéricos para administração		
Critério	**Peso**	**Forma de pontuação**
Provê ferramentas para publicação das regras de negócio nos ambientes alvo? Ou seja, possuir ferramentas para publicação de regras em ambientes de desenvolvimento, testes, homologação e produção, por exemplo.	3	0: Não 1: Sim
Permite promover regras entre ambientes?	2	0: Não 1: Sim
Total (2 critérios)	**5**	