DOUGLAS BARBOSA ALEXANDRE

GERAÇÃO DE CÓDIGO ORIENTADO A SERVIÇO A PARTIR DE MODELOS DE PROCESSOS DE NEGÓCIO

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. André Vital Saúde

LAVRAS
MINAS GERAIS - BRASIL
2009

DOUGLAS BARBOSA ALEXANDRE

GERAÇÃO DE CÓDIGO ORIENTADO A SERVIÇO A PARTIR DE MODELOS DE PROCESSOS DE NEGÓCIO

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 16 de junho de 2009

_____
Prof. Reginaldo Ferreira Souza

_____
Prof. Dr. Antônio Maria Pereira Resende

_____
Prof. Dr. André Vital Saúde
(Orientador)

LAVRAS
MINAS GERAIS - BRASIL
2009

Dedico esse trabalho a minha mãe e meus irmãos, Rosângela, Diego e Daniel.

Agradecimentos

Agradeço a minha mãe, Rosângela, pela educação, motivação e exemplo em minha vida.

Aos meus irmãos, Diego e Daniel, pelo apoio e conselhos.

Aos meus amigos, que além da amizade, contribuíram para minha formação acadêmica e profissional e que ajudaram a fazer esses anos de estudo mais prazerosos, entre risadas e cervejas.

Ao meu orientador André Vital Saúde, pela experiência compartilhada, pela amizade e oportunidade de trabalho em conjunto.

Geração de Código Orientado a Serviço a partir de Modelos de Processos de Negócio

Resumo

Business Process Management (BPM) consiste em uma metodologia de gestão de processos que visa torná-los mais eficientes e alinhados aos objetivos de negócio empresa, que pode ser auxiliada por ferramentas tecnológicas. Com isto o presente trabalho visa entender como identificar serviços em um modelo de processo de negócio e a partir desta identificação, como gerar o código necessário para implementação destes em uma arquitetura orientada a serviço tendo como objetivo gerar uma ferramenta que realize todo este processo de forma automática agilizando assim o processo de desenvolvimento de uma aplicação.

Para o processo de identificação de serviços nos modelos de processo de negócio foi realizado um estudo sobre a *Business Process Modeling Notation* (BPMN) e heurísticas capazes de identificar diversos padrões de *workflow* presentes nos modelos que utilizam esta notação. Com estas informações criou-se um modelo de geração de código baseado na linguagem *XML Process Definition Language* (XPDL) que através do *engine* do Velocity permitiu que transformações baseadas em modelos fossem utilizadas para gerar todo o código necessário para implementação destes serviços.

O resultado final foi uma ferramenta extensível, que permite que novos padrões de *workflow* sejam incorporados aumentando assim a capacidade de identificar serviços em modelos de processo de negócio mais complexos.

Palavras-chave: BPM, BPMN, SOA, Processos de Negócio, Geração Automática de Código.

GENERATION OF SERVICE-ORIENTED CODE FROM MODELS OF PROCESSES OF BUSINESS

Abstract

Business Process Management (BPM) is a methodology of process management that aims to make these processes more efficient and aligned to the goals of business enterprise, which may be assisted by technological tools. With this the present work aims to understand how to identify services in a model of business process and from this identification, how generate the code necessary to implement these services in a service-oriented architecture with the aim to create a tool that performs this process in an automatic way thus streamlining the process of developing an application.

In the process of identifying services in business process models, a study was conducted on the Business Process Modeling Notation (BPMN) and heuristics able to identify various workflow patterns in models that use this notation. With this information it was created a model to generate code based on XML Process Definition Language (XPDL) that through the Velocity Engine enabled transformations based models were used to generate all the code necessary to implement these services.

The end result was a extensible tool, allowing new workflow patterns to be incorporated thereby increasing the ability to identify models of services in business process more complex.

Key-words: BPM, BPMN, SOA, Business Process, Automatic Code Generation.

Sumário

Lista de Figuras

Lista de Abreviaturas

BPEL – Business Process Execution Language

BPM – Business Process Management

BPML – Business Process Modeling Language

BPMN – Business Process Modeling Notation

BPMS – Business Process Management System

OASIS – Advancing Open Standards for the Information Society

OMG – Object Management Group

SOA – Service-Oriented Architecture

TI – Tecnologia da Informação

XML – eXtensible Markup Language

XPDL – XML Process Definition Language

XSL – eXtensible Stylesheet Language

XSLT – XSL Transformations

WfMC – Workflow Management Coalition

1. Introdução

Gerenciamento de processos de negócio ou, *Business Process Management* (BPM) é um conceito relativamente novo que vem ganhando cada vez mais espaço dentro das organizações. Trata-se de uma metodologia de gestão para gerenciamento de processos, que visa gerir os negócios da empresa a fim de torná-los mais eficientes e alinhados aos objetivos de negócio das organizações (Davenport, 1994).

Ao longo dos últimos anos a área de Tecnologia da Informação (TI) disponibilizou tecnologias que formaram a base necessária para que o uso efetivo de BPM fosse possível, a exemplo temos: *Business Process Modeling Notation* (BPMN), *XML Process Definition Language* (XPDL) e *Service-Oriented Architecture* (SOA).

Neste contexto é de interesse do presente trabalho unir estas tecnologias a fim de entender como identificar serviços em um modelo de processo de negócio e a partir desta identificação, como gerar o código necessário para implementação destes em uma arquitetura orientada a serviço tendo como objetivo gerar uma ferramenta que realize todo este processo de forma automática agilizando assim o processo de desenvolvimento de uma aplicação.

A notação BPMN é um padrão para modelagem visual de processos de negócio que tem como principal objetivo fornecer uma notação de fácil entendimento pelos usuários envolvidos no negócio.

A linguagem XPDL tem como objetivo estabelecer um modelo para intercâmbio de processos de negócio entre as diversas ferramentas de modelagem existentes. Sua especificação é totalmente compatível com o padrão BPMN.

Já a arquitetura SOA é um conjunto de práticas de organização de sistemas de TI que permitem grande agilidade no atendimento das demandas geradas pelo negócio, reduzindo custos e tornando a área de TI mais dinâmica. A agilidade é resultado de um processo simplificado para criação de novas funcionalidades, através da integração de sistemas, reaproveitamento em larga escala de código e a possibilidade de intercâmbio de funcionalidades entre os diversos setores da empresa. Vale ressaltar que soluções BPM não requerem SOA, mas SOA simplifica e muito as implementações destas soluções.

Com isto para o processo de identificação de serviços nos modelos de processo de negócio foi realizado um estudo sobre a *Business Process Modeling Notation* (BPMN) e heurísticas capazes de identificar diversos padrões de *workflow* presentes nos modelos que utilizam esta notação. Com estas informações criou-se um modelo de geração de código baseado na linguagem *XML Process Definition Language* (XPDL) que através do *engine* do Velocity permitiu que transformações baseadas em modelos fossem utilizadas para gerar todo o código necessário para implementação destes serviços.

Para alcançar o objetivo proposto, este trabalho está organizado da seguinte forma:

No Capítulo 2, são explanados todos os conceitos tomados como necessários para o melhor entendimento do presente trabalho. Já no Capítulo 3, é descrito a metodologia utilizada para solucionar os problemas relacionados à identificação de serviços válidos em um modelo de processo de negócio e a partir da identificação destes serviços, como gerar o código necessário para implementação deste ou como associá-lo a um serviço já disponível na organização. O Capítulo 4 descreve os objetivos da solução proposta, abordando quais os problemas existentes e como ela os soluciona. Em seguida apresenta-se

a arquitetura e as tecnologias utilizadas, bem como o motivo para suas escolhas e por fim são detalhados os diversos módulos que compõem a arquitetura, bem como o seu funcionamento. Na conclusão são expostas as dificuldades encontradas e os resultados obtidos com a implementação da solução proposta neste trabalho. Também são propostos trabalhos futuros que visam complementar as funcionalidades implementadas, já que uma das características principais da ferramenta é que ela pode ser facilmente estendida.

2. Referencial Teórico

Neste capítulo será apresentando todas as informações necessárias para um complemento entendimento da metodologia e da solução proposta neste trabalho.

Este capítulo inicia-se falando a respeito da definição de processo de negócio e da área de gerenciamento de processos de negócio. Na seqüência são abordadas as notações e linguagens utilizadas para a modelagem de processos de negócio. No final tem-se uma visão geral da arquitetura orientada a serviços, bem como os seus objetivos.

2.1. Business Process

Business Process, ou processo de negócio, pode ser visto como a ordenação de atividades de trabalho utilizando tempo e espaço, com um início, um fim e um conjunto claramente definido de entradas e saídas (Davenport, 2004). Não necessariamente as entradas e as saídas precisam ser algo material podendo ser, por exemplo, informação.

Assim podemos entender processo de negócio como um conjunto de passos (ou atividades) que devem ser executadas em uma determinada ordem a fim de produzir um bem ou um serviço. Estas atividades podem ser executadas seqüencialmente ou paralelamente.

2.2. Business Process Management

Business Process Management (BPM), ou gerenciamento de processos de negócio, consiste em uma metodologia de gestão para gerenciamento de processos, que pode ser auxiliada por ferramentas tecnológicas. Além das

características relativas ao fluxo de controle, envolve métodos, técnicas e ferramentas para apoiar o projeto, execução, gerenciamento e análise operacional dos processos de negócio (Davenport, 1994).

Historicamente, as empresas adotam estruturas verticais em sua organização, que correspondem às funções organizacionais, onde os funcionários ficam responsáveis apenas pelos resultados de seu departamento. Ao utilizarem BPM, as empresas adotam uma gestão baseada em processos que permite integrar os esforços de um único departamento. Mas usar BPM não significa apenas mudar a estrutura organizacional de uma empresa, mas sim em aplicar técnicas para modelagem, implantação, monitoramento e melhoria contínua dos processos, visando alcançar agilidade operacional, maior confiabilidade, redução de custos, maior capacidade de se adaptar as mudanças impostas por clientes e, principalmente, estabelecer o alinhamento estratégico (Chiavenato, 1995).

O BPM ajuda as empresas a identificarem a importância estratégica de seus processos e a tirarem vantagens competitivas disso. Serve também para proporcionar ao gestor uma maior facilidade de encontrar oportunidades de melhoria para o serviço prestado ao cliente, através de indicadores de resultados.

Com o BPM, houve a possibilidade da abstração da lógica do negócio nos aplicativos, pois, segundo Delphi Group (2005), ofereceu uma das primeiras oportunidades reais para a separação de gerenciamento de negócios do gerenciamento de sistemas.

Tecnologias disponibilizadas pela área de Tecnologia da Informação (TI) nos últimos anos formaram a base necessária para a efetiva utilização de BPM, a exemplo da *Business Process Modeling Notation* (BPMN), *XML*

Process Definition Language (XPDL), *Business Process Management System* (BPMS), e *Service-Oriente Architecture* (SOA).

Uma questão importante que deve ser abordada é que soluções BPM não requerem SOA, mas ao se utilizar SOA o processo de implementação destas soluções BPM se tornam muito mais simples.

Um sistema de BPM, ou BPMS é constituído dos seguintes módulos: *Workflow* (ambiente de execução ou motor de execução); Modelagem de Processos (definição e projeto do processo); Simulação; Monitoramento de Atividade e Interface de usuário.

2.3. Business Process Modeling Notation

A notação BPMN é um padrão para modelagem visual de processos de negócio. Segundo White and Miers (2008) sua criação se deu devido à necessidade de uma representação gráfica para uma linguagem de execução de processos em *eXtensible Markup Language* (XML) a *Business Process Modeling Language* (BPML), que estava sendo desenvolvida no final de 2001 pelo *Business Process Management Initiative* (BPMI).

Após dois anos de pesquisas pelo BPMI em maio de 2004 a notação BPMN teve sua primeira especificação formal liberada para o público. Desde então cerca de 50 empresas desenvolveram implementações sobre esta notação. Desta forma em fevereiro de 2006 a *Object Management Group* (OMG) adotou a BPMN como de fato um padrão (White and Miers, 2008).

White (2004) afirma que a BPMN tem como principal objetivo fornecer uma notação de fácil entendimento pelos usuários de negócio, desde os analistas

de negócio, que começam a modelar os processos, até os desenvolvedores que os implementam, incluindo também os demais membros do negócio, que irão gerenciar e monitorar esses processos.

BPMN também deve suportar um modelo interno que permite a geração de código executável em *Business Process Execution Language* (BPEL) eliminando assim a lacuna existente entre a modelagem do processo de negócio e sua implementação (White, 2004).

Já White and Miers (2008) explica que a simbologia utilizada pelo BPMN permite a especificação dos fluxos num nível de detalhamento próximo da complexidade de um ambiente real.

Os diagramas de processos de negócio definidos através da notação BPMN como se pode ver na Figura 2.1 não passam de um conjunto de elementos gráficos aliado a técnicas de fluxograma, que já são bastante conhecidas, diminuindo assim a curva de aprendizagem da notação. Mas estes elementos gráficos não conseguem representar todas as informações necessárias para executar um processo de negócio de acordo com um dos objetivos desta notação já citados, para isto os elementos oferecem suporte a atributos que fornecem informações adicionais sobre o processo em questão. Estes atributos serão utilizados neste trabalho para que seja possível identificar durante o processo de geração de código se um serviço que já se encontra implementado na organização contempla uma atividade.

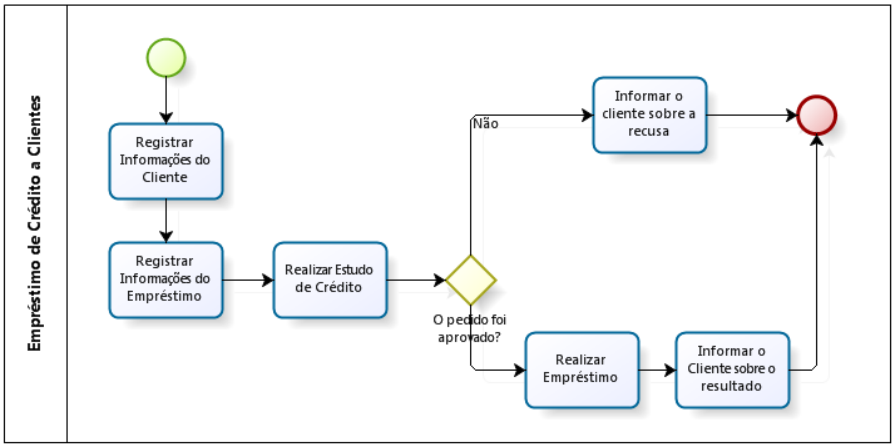

Figura 2.1: Processo de negócio utilizando a notação BPMN

Para que fosse possível atingir aos objetivos descritos acima os elementos gráficos da notação foram separados em categorias específicas. Esta organização provê um pequeno conjunto de categorias de notação que habilita o leitor de um diagrama de processos de negócio reconhecer facilmente as formas básicas de elementos e entender o diagrama (White, 2004). Dentro dessas categorias podem ser adicionadas variações e informações que possibilitem a notação suportar as complexidades envolvidas nessas modelagens sem alterar drasticamente o aspecto visual do diagrama.

As quatro categorias básicas em que os elementos da BPMN foram organizados são:

- Objetos de fluxo;
- Objetos de conexão;
- Raias;
- Artefatos.

Objetos de fluxo: São os principais elementos gráficos para definir o comportamento de um processo de negócio. Esta categoria define apenas um

conjunto de três elementos utilizados nos diagramas. Este número reduzido de elementos visa evitar que os modeladores tenham que aprender e reconhecer diversos elementos (White, 2004). Os três elementos pertencentes a esta categoria são:

- Evento: É representado por um círculo e é algo que ocorre no decorrer do processo de negócio. Eles afetam o fluxo do processo e podem ter um gatilho, ou um resultado associado. Há três tipos de eventos: inicial, intermediário e final (White, 2004).

Figura 2.2: Três tipos de eventos: inicial, intermediário e final.

- Atividade: A atividade é uma tarefa genérica executada pela organização; ela pode ser classificada como atômica ou complexa (White, 2004). Atividades consideradas atômicas estão relacionadas ao conceito de tarefa, que representa a menor unidade de trabalho a ser realizada e são representadas por retângulos arredondados. Já quando consideradas complexas estão relacionadas ao conceito de sub-processos, ou seja, a um subconjunto de tarefas pertencentes ao processo que esta atividade complexa pertence, estas são diferenciadas pela presença de um pequeno sinal de mais na parte inferior central do retângulo.

Figura 2.3: Atividades: tarefa e sub-processo

- Gateway: Gateways são usados como estruturas de controle na seqüência do fluxo (White, 2004). Eles são representados por um losângulo com uma marca interna que irá indicar o tipo do comportamento de controle.

Figura 2.4: Gateways: XOR, OR e AND.

Objetos de conexão: Os objetos de conexão conectam eventos, atividades e gateways definindo a estrutura básica de um processo de negócio (White, 2004). Há três objetos de conexão utilizados para conectar os objetos do fluxo às outras informações:

- Fluxo de Seqüência: É utilizado para mostrar a ordem que as atividades serão executadas no fluxo (White, 2004).
- Fluxo de Mensagem: É utilizado para mostrar o tráfego de mensagens entre as entidades do negócio (White, 2004).
- Associação: Uma associação conecta informação adicional aos elementos básicos como, por exemplo, uma descrição. As

associações também são utilizadas para mostrar as entradas e as saídas das atividades (White, 2004).

Figura 2.5: Objetos de conexão: fluxo de seqüência, fluxo de mensagem e associação.

Raias: São utilizadas para agrupar os elementos de modelagem, possibilitando a organização das atividades em categorias (White, 2004). Há dois tipos de raias:

- Piscinas: Uma piscina é um contêiner que agrupa um conjunto de atividades de uma organização. As piscinas podem ser do tipo caixa preta ou caixa branca. A piscina do tipo caixa preta esconde seus detalhes; a comunicação só pode ocorrer com a borda da piscina enquanto a piscina do tipo caixa branca exibe seus detalhes e permite a comunicação com os elementos em seu interior.
- Pistas: Para decompor uma organização em unidades específicas divide-se longitudinalmente o interior da piscina por meio de pistas.

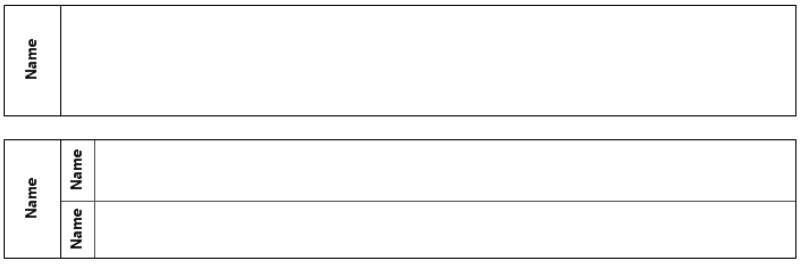

Figura 2.6: Raias: piscina e pistas.

Artefatos: Artefatos são utilizados para agregar informações ao modelo permitindo uma maior flexibilização da notação (White, 2004). Atualmente, existem três tipos de artefatos definidos:

- Objetos de dados: São mecanismos utilizados para mostrar como dados são produzidos, ou requeridos pelas atividades. São conectados as atividades por associações (White, 2004).
- Anotação: É utilizada para fornecer um texto com informações adicionais para quem estiver lendo o diagrama BPMN (White, 2004).
- Grupo: Grupos são utilizados para documentação e não afetam o fluxo do negócio (White, 2004).

Figura 2.7: Artefatos: objeto de dados, anotação e grupo.

2.4. XML Process Language Definition

A linguagem *XML Process Definition Language* (XPDL) foi proposta pela *Workflow Management Coalition* (WfMC), entidade que define padrões para a comunidade que trabalha com *workflow*. O objetivo desta linguagem é em estabelecer um modelo para intercâmbio de processos de negócio entre as diversas ferramentas de modelagem existentes (WfMC, 2005).

A especificação XPDL atualmente é aceita como um padrão para o intercâmbio de modelos de processos entre as diversas ferramentas de modelagem existentes. A representação destes modelos é feita em um arquivo que utiliza a *eXtensible Markup Language* (XML) como o mecanismo para especificar como estas informações serão armazenadas e como se dá o processo de intercâmbio destas informações (WfMC, 2005).

De acordo com WfMC (2008), a especificação do XPDL é totalmente compatível com o padrão *Business Process Modeling Notation* (BPMN) discutido anteriormente. Para os criadores do XPDL, o BPMN é o padrão ideal para modelar o processo em nível visual e o XPDL para definir suas regras em nível técnico (WfMC, 2005).

Os elementos do BPMN – Piscinas, Atividades, Transições, Artefatos, Mensagens de Fluxo, Associações entre outros, podem ser encontradas na sintaxe do XPDL sendo que para cada elemento existe um código apropriado (WfMC, 2005).

A principal diferença conceitual entre o padrão BPMN e o XPDL é referente ao conceito de atividade. Este conceito no XPDL é utilizado para modelar diversos elementos da notação BPMN entre eles:

- Tarefas – Quando o escopo da atividade é local ao processo e esta representa uma unidade de trabalho dentro do processo;
- Subprocesso – Quando a atividade representar um subfluxo, ou seja, quando ela for um contêiner para execução de outro processo, seja ele local ou remoto;
- Gateway – Quando a atividade for um roteamento, ou seja, quando for utilizada somente para dar suporte a decisões que interferiram no fluxo do processo;
- Eventos – Quando a atividade representar algum fato que ocorre durante a execução do processo e que afeta o fluxo do mesmo.

2.5. Service-Oriented Architecture

Segundo o SOA Working Group (2006), *Service-oriented Architecture* (SOA) pode ser traduzido como arquitetura orientada a serviços, sendo este um estilo de arquitetura de software cujo princípio fundamental diz que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços.

Como se pode observar SOA tem como seu conceito central a idéia de serviço. Segundo o dicionário o significado da palavra "serviço" é definido como: "Execução de trabalho ou desempenho de funções, de um para outro.". Entretanto, serviço, como o termo é entendido, também relaciona as seguintes idéias (OASIS, 2006):

- A capacidade de executar um trabalho para o outro;
- A especificação do trabalho oferecido por outro;
- A oferta da execução do trabalho por outro.

Já do ponto de vista da arquitetura SOA, podemos entender o conceito de serviço como uma representação lógica de uma atividade empresarial repetitiva que possui um resultado específico, como "verificação de crédito dos clientes", "fornecer dados meteorológicos", ou "consolidar relatórios financeiros". Este serviço deve ser auto-contido, podendo ser composto por outros serviços ou não, e deve atuar como uma "caixa preta" para os seus consumidores (SOA Working Group, 2006). Este será o conceito adotado no decorrer deste trabalho.

Mas a definição apresentada no início desta seção sobre SOA deixa muitas dúvidas em relação a qual a sua utilidade e suas funcionalidades. Por isso existem vários pontos de vista de acordo com a interpretação da definição segundo Lublinsky (2007).

- Para analistas de negócio: é um conjunto de serviços que constitui de capacidades de Tecnologia da Informação (TI) e pode ser usado para construir uma solução.
- Para um gerente de projeto é um conjunto de princípios e padrões de arquitetura que tem como princípios modularidade, encapsulamento, fraco acoplamento, separação de conceitos e reúso de software, possibilitando um grande desenvolvimento paralelo de aplicações.
- Para um desenvolvedor de *software* é um modelo completo de programação com ferramentas, padrões e tecnologias, como *Web Services*.

O *Advancing Open Standards for the Information Society* (OASIS) é o grupo que atualmente se encontra responsável pela definição e manutenção do que realmente é SOA. Em agosto de 2006 este grupo definiu um Modelo de Referência para SOA. Este modelo não diz como fazer uma aplicação SOA, mas

define seus conceitos e termos utilizados (OASIS, 2006). Nele estão definidos os principais termos da arquitetura:

- Serviço: Os meios pelos quais as necessidades de um consumidor são complementadas por um produtor de capacidades.
- Capacidade: Uma parcela do mundo real que um provedor de serviços é capaz de prover a um consumidor de serviços.
- Arquitetura Orientada a Serviços: é um paradigma para organizar e utilizar capacidades distribuídas que podem estar em diferentes domínios. Ela provê um meio uniforme de oferecer descoberta, interação e uso de capacidades.

Com a utilização de SOA, pode-se observar que o sistema que fornece o suporte necessário para as operações da empresa é organizado como um conjunto de serviços. Estes serviços, em conjunto com a infra-estrutura pela qual são suportados, fornecem o fluxo interno e externo das informações da empresa como pode ser observada na figura a seguir.

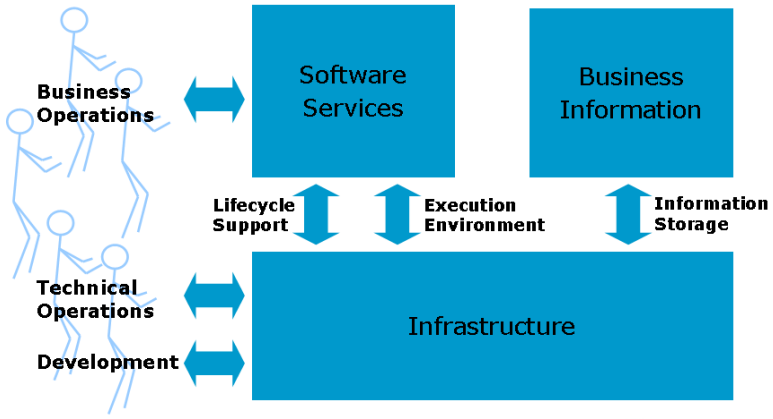

Figura 2.8: Visão geral da arquitetura orientada a serviço (SOA). Fonte: SOA Working Group (2006).

O que torna a arquitetura SOA tão interessante é que ela permite grande agilidade no atendimento das demandas geradas pelo negócio, reduzindo custos e tornando a área de TI da empresa mais dinâmica.

Isto é possível graças aos princípios básicos desta arquitetura que são o de fraco acoplamento entre seus componentes, o que possibilita que estes serviços sejam independentes de outras partes da aplicação tornando o processo de criação de novas funcionalidades mais simples através da integração de sistemas, e reusabilidade, possibilitando ao mesmo serviço ser usado em outras aplicações, ou seja, o reaproveitamento em larga escala de código.

Todos estes fatores acima mencionados contribuem para redução nos custos de treinamento pessoal, manutenção e maior agilidade no processo de desenvolvimento, uma vez que componentes podem ser trocados sem grande interferência no funcionamento geral do sistema, e permite que se utilizem componentes já desenvolvidos pela empresa independente de linguagens de programação.

3. Metodologia

Este capítulo trata sobre as técnicas que foram utilizadas para solucionar os problemas encontrados no decorrer deste trabalho para realizar geração de código orientado a serviço a partir de modelos de processo de negócio.

Primeiramente é discutida a técnica utilizada para identificar os serviços que devem ser gerados dentro do modelo de processos de negócio que utilizam a notação BPMN e em seguida será abordada a técnica utilizada no gerador de código propriamente dito.

3.1. Identificação de Serviços Válidos em um Modelo de Processo de Negócio

Para realizar a identificação de serviços válidos em um modelo de processo de negócio utilizou-se como base a etapa de identificação e classificação dos serviços candidatos da técnica proposta por Azevedo et al (2009) com algumas exceções.

A idéia central desta etapa é que as atividades devem ser analisadas dentro dos seus contextos nos modelos de processos, segundo um conjunto de heurísticas que levam em consideração tanto a semântica da atividade (regras de negócio, requisitos de negócio) quanto à estrutura do fluxo de atividades (padrões de *workflow*) e também a presença de fluxos recorrentes.

Como citado por Azevedo et al (2009) serviços em uma arquitetura orientada a serviços estão diretamente associados à implementação de regras de negócio e requisitos de negócio. A automatização de uma atividade associada a uma regra de negócio se refere à implementação desta regra em aplicações ou em bancos de dados. Já os requisitos de negócio são analisados por descreverem

funcionalidades que devem estar (ou já se encontram) implementadas em aplicações, sendo potenciais definidores de serviços.

Um padrão de *workflow* é uma abstração de uma forma concreta que se mantém repetitiva em contextos específicos. Os padrões de *workflow* são freqüentemente apresentados como um benchmark para funcionalidades de sistemas de gerenciamento de *workflow* Van der Aalst e Ter Hofstede (2002). Os serviços identificados a partir destes padrões são responsáveis por encapsular regras de negócio que determinam dependência entre atividades, explicitando o fluxo dos processos (Azevedo et al, 2009).

Uma das exceções que será aplicada nesta etapa previamente descrita é referente à análise dos padrões referentes aos fluxos recorrentes. Esta análise não foi implementada neste trabalho, mas a solução apresentada permitirá que o processo de identificação de serviços seja facilmente estendido contemplando novos padrões.

Outra exceção é referente a serviços candidatos, Azevedo et al (2009) na etapa de identificação classifica todos os serviços encontrados como serviços candidatos e em uma etapa posterior realiza a consolidação dos serviços candidatos ficando apenas com os serviços que interessam de fato, mas como a proposta deste trabalho é utilizar a geração automática de código, todo serviço candidato se torna um serviço. Como citado em Azevedo et al (2009) “Serviços candidatos são abstrações de serviços que, na fase de projeto, serão implementados como serviços físicos ou como funcionalidades de aplicações”.

Dentre as nove heurísticas propostas por Azevedo et al (2009) quatro delas foram implementadas diretamente neste trabalho e uma no caso, a heurística número um, sofreu algumas mudanças na sua implementação conforme explicação logo abaixo. As outras quatro heurísticas foram descartadas

e não foram implementadas neste trabalho, pois elas tratam de elementos da notação BPMN que não aparecem freqüentemente em modelos de processo de negócio. Desta forma, as cinco heurísticas implementadas são capazes de identificar diversos serviços a partir dos elementos mais freqüentes nestes modelos.

As definições bem como a identificação nos modelos de processos de negócio em BPMN das heurísticas utilizadas são descritas a seguir.

Heurística 1: Um serviço deve ser identificado a partir de uma regra de negócio.

Exceção: Como a notação BPMN não possui um elemento que define explicitamente uma regra de negócio, as regras de negócio são declaradas em gateways que ficam responsáveis por realizar a escolha do caminho que o processo deve tomar. Devido a este fato a solução proposta neste trabalho traz o gateway para dentro do serviço que ficará responsável por representá-lo como um todo. A regra de negócio deverá ser implementada manualmente no serviço, no corpo do método gerado especificamente para este propósito, isto devido à falta de informações presentes no modelo de processo de negócio que não sejam específicas da ferramenta utilizada para realizar a modelagem. Um exemplo desta exceção poderá ser observado quando a heurística 4 for aplicada.

Heurística 2: Um serviço deve ser identificado a partir de um requisito de negócio.

Exemplo: A partir das atividades “Identificar Método de Pagamento”, “Aceitar Dinheiro ou Cheque”, “Processar Cartão de Crédito” e “Preparar Pacote para o Cliente” descritas abaixo, deve ser identificado um serviço para cada atividade.

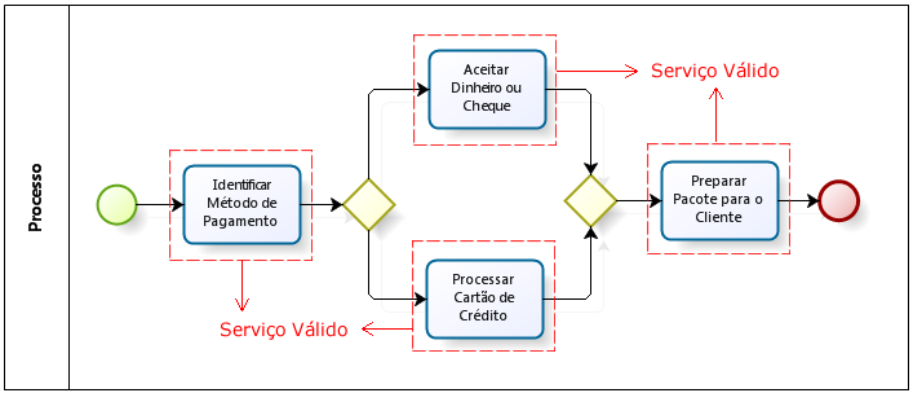

Figura 3.1: Exemplo aplicação heurística 2.

Heurística 3: Um serviço deve ser identificado a partir de um conjunto de atividades seqüenciais.

Exemplo: A partir da seqüência de atividades "Comprometer Limite de Crédito", "Determinar Taxa de Juros a ser Cobrada" e "Gerar Contrato" descritas abaixo, um serviço deve ser identificado.

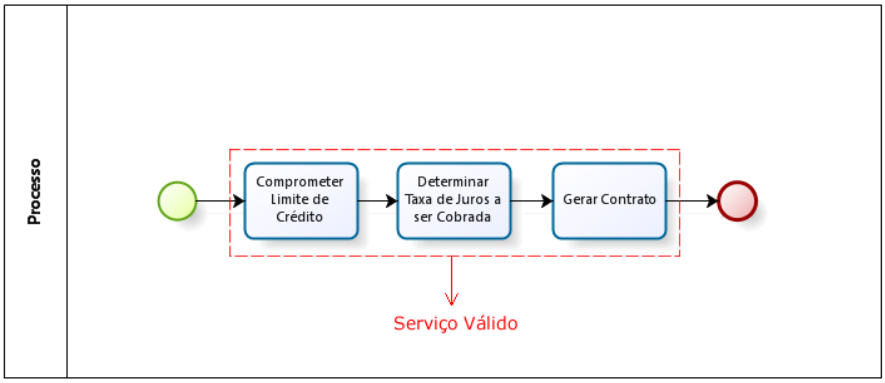

Figura 3.2: Exemplo aplicação heurística 3.

Heurística 4: Um serviço deve ser identificado a partir de um gateway paralelo onde um fluxo de controle simples divide-se em fluxos de controle múltiplos, que podem ser executados em paralelo, e finalizado em um ponto no processo onde os múltiplos fluxos que estão sendo executados em paralelo

convergem em um único fluxo de controle simples novamente, ou onde as transições terminem em evento final.

Exemplo: Um serviço é identificado para as atividades “Preparar Envio” e “Aprovar Solicitação” descritas na figura abaixo.

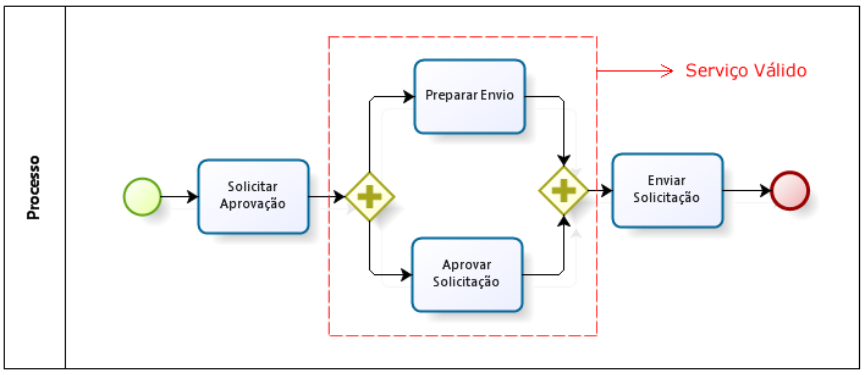

Figura 3.3: Exemplo aplicação heurística 4.

Heurística 5: Um serviço deve ser identificado a partir de um gateway exclusivo onde, baseado em uma decisão, uma, e somente uma, das várias transições do fluxo é escolhida, e finalizada em um ponto no processo onde as transições do fluxo se juntem em um único fluxo de controle simples novamente ou quando uma ou mais das transições termina em evento final.

Exemplo: Para as atividades “Aceitar Dinheiro ou Cheque” e “Processar Cartão de Crédito” descritas na figura abaixo, um serviço deve ser identificado, sendo que este deve oferecer suporte para que seja possível explicitar a regra de negócio presente no gateway exclusivo que realiza a bifurcação no fluxo.

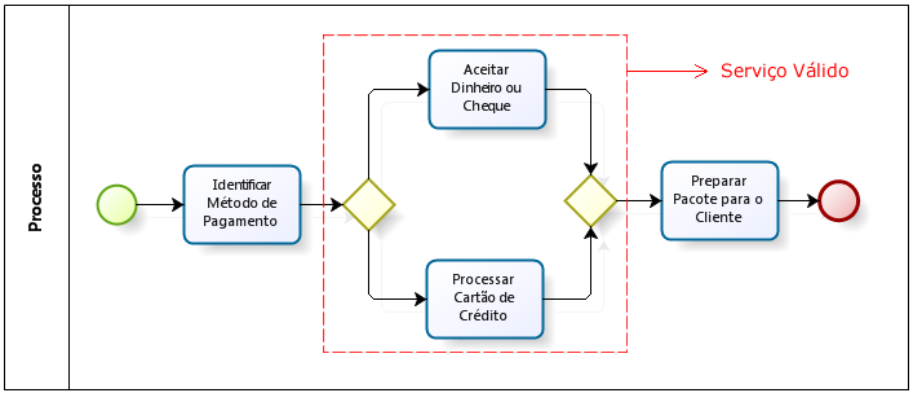

Figura 3.4: Exemplo aplicação heurística 5.

O presente trabalho utilizando apenas as cinco heurísticas descritas acima consegue identificar os serviços válidos em uma vasta gama de modelos de processo de negócio. Como será visto no capítulo seguinte, a ferramenta implementada pode ser facilmente estendida, possibilitando assim que novas heurísticas sejam adicionadas aumentando ainda mais a capacidade de identificação de serviços nos mais diversos modelos de processos de negócio.

3.2. Geração Automática de Código

Para a implementação do gerador de código foi utilizada a técnica de transformações baseadas em modelos. Esta técnica está presente em diversas partes no mundo de desenvolvimento de software desde visualização de dados à geração automática de código.

Um grande número de ferramentas utilizam modelos para transformar dados de um formato para outro. Como exemplo, podemos citar o *XSL Transformations* (XSLT) que é uma linguagem de marcação *eXtensible Markup Language* (XML) usada para realizar transformações em documento no formato XML com base em modelos que são descritos de acordo com a *eXtensible*

Stylesheet Language (XSL). A Figura 3.5 mostra os quatro componentes envolvidos em um processo de transformação baseado em modelos. Eles são:

- Modelo de dados: Contém os dados que devem ser transformados. Estes dados devem estar organizados em uma estrutura específica.
- Modelo: Formata o modelo de dados para o código de saída. Contém referências a entidades pertencentes ao modelo de dados.
- Motor: A aplicação que realiza a transformação propriamente dita. Tem como entrada o modelo de dados e o modelo que deve ser aplicado, a saída é gerada substituindo às referências internas do modelo com dados reais provenientes do modelo de dados.
- Saída: O resultado do processo de transformação.

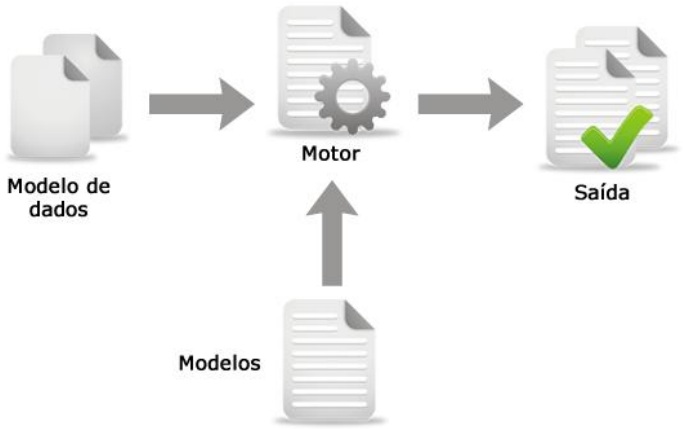

Figura 3.5: Transformações baseadas em modelos.

Como se pode observar o processo de geração de código é simplesmente um processo de transformação, no qual o modelo de dados contém as informações sobre as entidades que você deseja gerar e o modelo representa a estrutura da entidade já na sintaxe na linguagem de programação em que se deseja gerar o código.

A característica mais importante sobre as transformações baseadas em modelos é que o formato das entidades geradas pode ser alterado sem precisar que nenhuma alteração seja realizada na aplicação que executa a transformação, para isso basta que as alterações sejam realizadas diretamente no modelo.

Esta característica é interessante e de grande valor em um gerador de código, pois permite que o código gerado esteja dentro das normas e padrões do projeto que o está utilizando.

Para o caso específico deste trabalho, o modelo de dados será composto pelo conjunto de serviços que foram identificados na etapa anterior, serviços estes que dão suporte para que o modelo de processo de negócio que foi analisado possa ser implementado e disponibilizado para os seus consumidores.

Cada serviço presente neste modelo de dados possui um tipo que o identifica. Este tipo será utilizado na etapa de seleção do modelo que será aplicado a este serviço no momento da geração do seu código. Ao término do processo teremos como saída um conjunto de classes que representam os serviços identificados propriamente ditos.

A Figura 3.6 mostra o processo de transformações baseado em modelos atualizado com o modelo de dados, os modelos e saídas específicas deste trabalho.

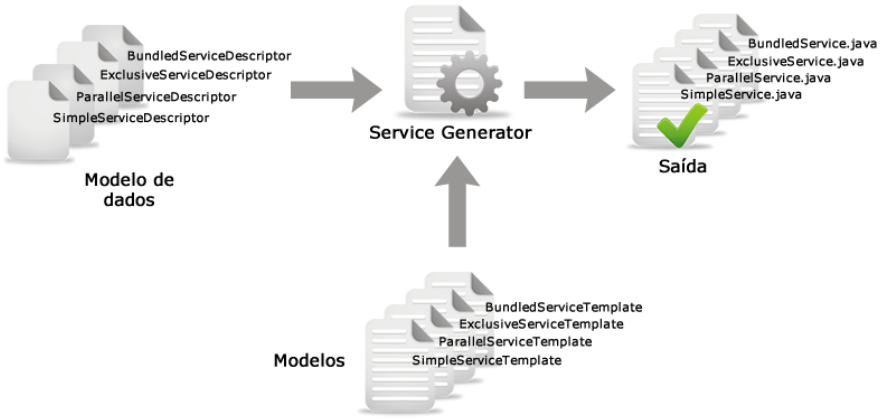

Figura 3.6: Processo de transformações baseado em modelos utilizados neste trabalho.

4. Redbox

Neste capítulo será apresentada a estrutura e funcionamento da solução proposta. No caso a ferramenta Redbox que foi implementada no decorrer deste trabalho com o detalhamento de cada um de seus componentes.

Inicia-se o capítulo discutindo os objetivos da solução proposta, abordando quais os problemas existentes e como ela os soluciona. Em seguida apresenta-se a arquitetura e as tecnologias utilizadas, bem como o motivo para suas escolhas. Por fim são detalhados os diversos módulos que compõem a arquitetura, bem como o seu funcionamento.

4.1. Objetivo

Mesmo com o atual crescimento da adoção de soluções baseadas na arquitetura orientada a serviços (SOA), algumas questões ainda se encontram em aberto na área de desenvolvimento de *software* principalmente ao que se refere à identificação do conjunto mais adequado de serviços para apoio às atividades do negócio da organização. Dentro destas questões pode-se citar: “De onde retirar estas informações?” e “A partir destas como identificar os serviços válidos que devem ser reaproveitados ou implementados?”.

Segundo Azevedo et al (2009) diversos autores concordam que o processo de desenvolvimento destas soluções deve seguir uma abordagem sistemática, tendo como ponto de partida para a identificação dos serviços os modelos de processos de negócio, já que estes representam fielmente todo o funcionamento das atividades do negócio da organização.

Na pesquisa bibliográfica realizada para elaboração deste trabalho, nenhuma ferramenta com as características que o Redbox possui foi encontrada

e poucos foram os artigos que propunham um método para a identificação destes serviços. Sendo que nenhum destes abordava como realizar este processo de forma automática o que torna este processo demorado e altamente dependente do analista SOA.

Visto os problemas acima descritos a solução proposta neste trabalho visa implementar um método automático e de fácil extensão para identificação de serviços baseado nos modelos de processo de negócio que utilizam a notação BPMN em sua modelagem.

Além da identificação dos serviços válidos, a solução proposta ficará encarregada de gerar todo o código necessário a partir de transformações baseadas em modelos para implementar estes serviços ou delegar a sua execução a serviços já implementados pela organização seguindo o padrão de arquitetura SOA, tornando assim este pr

4.2. Arquitetura

A idéia central da arquitetura proposta pela ferramenta Redbox é que ela seja simples e ao mesmo tempo extensível. Para que fosse possível alcançar estes dois objetivos a ferramena foi dividida em apenas dois componentes principais: “XPDL Reader” e o “Service Generator” (Veja Figura 4.1).

O “XPDL Reader” é responsável por ler o arquivo XPDL que contêm a definição do modelo de processo de negócio, que foi modelado através da notação BPMN, e a partir das informações disponíveis neste arquivo aplicar as heurísticas citadas no capítulo anterior a fim de identificar os serviços válidos que devem ser implementados para dar suporte a aplicação que está sendo desenvolvida. À medida que os serviços são identificados, um modelo de dados com as informações necessárias para a geração de código é construído. Este

modelo é o resultado final desta etapa e serve como entrada para o gerador de código.

Já o “Service Generator” é o componente responsável por gerar todo o código da aplicação que se deseja construir. Ele não passa de um simples gerador de código baseado em modelos que utiliza o *engine* do Velocity. Este componente tem como entrada o modelo de dados que foi construído na etapa anterior. Para cada serviço presente neste modelo de dados ele deve identificar qual o seu tipo e associá-lo a um modelo específico do Velocity. Após esta associação o modelo do Velocity é preenchido com as informações do serviço e o código é gerado segundo a estrutura definida neste modelo.

A Figura 4.1 exibe os principais componentes presentes na arquitetura implementada, bem como suas entradas e saídas.

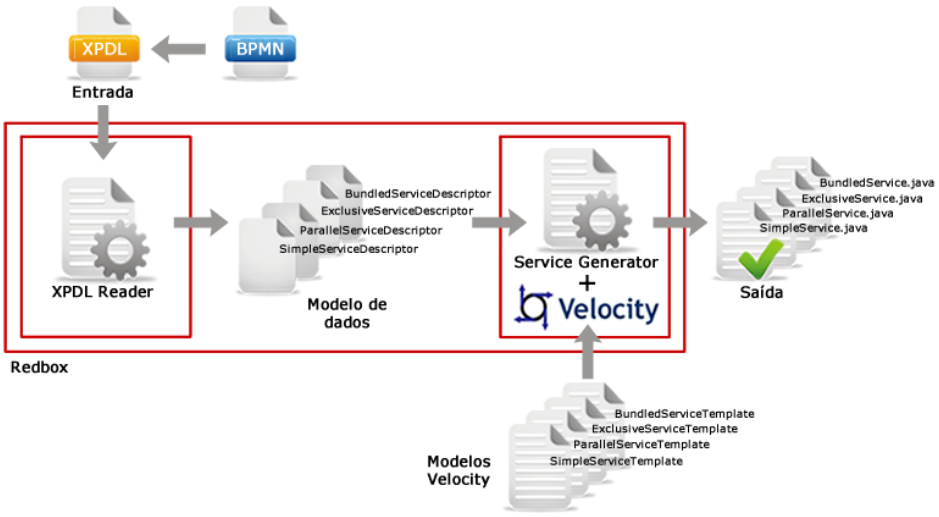

Figura 4.1: Componentes da arquitetura proposta, bem como suas entradas e saídas.

4.2.1. **Tecnologias utilizadas**

Diversas tecnologias oferecem o suporte necessário para que a arquitetura acima discutida seja implementada, mas os seguintes pontos influenciaram na escolha das tecnologias utilizadas:

- Utilização de padrões comuns entre grandes empresas para representação de processos de negócio;
- Disponibilidade de componentes licenciados de forma livre;
- Possibilidade de padronização do código gerado pela solução proposta;
- Possibilidade de utilização em diversas plataformas;
- Facilidade de integração entre os componentes.

De acordo com os pontos acima observados as seguintes tecnologias foram escolhidas:

1) XPDL – Esta linguagem foi escolhida por ser um padrão aceito e reconhecido mundialmente para definição e troca de informações referentes a processos de negócio modelados em BPMN, bem como pela necessidade de maximizar o público alvo que irá utilizar a solução proposta já que praticamente todas as ferramentas de modelagem de processos em BPMN disponíveis no mercado fornecem a opção de exportar o modelo para este formato.
2) Plataforma de desenvolvimento Java: A utilização desta plataforma foi determinada por diversos fatores, os principais são:
    a. Independência de plataforma – possibilitando a implementação e execução em diversos sistemas operacionais;
    b. Amplo suporte a arquitetura orientada a serviços (SOA);
    c. Amplo suporte dos fornecedores de TI (Sun, IBM, Oracle, etc.) – proporcionando a independência de fornecedor;
    d. Ampla disponibilidade de bibliotecas e ferramentas de desenvolvimento – facilitando a manutenção da solução e a adição de novas funcionalidades através da utilização de bibliotecas já existentes.
    e. Disponibilidade de ferramentas *Business Process Management System* (BPMS), com as quais o código gerado pode ser integrado no futuro.
3) *Engine* Velocity: A escolha deste motor, para as transformações baseadas em modelo se deu devido aos seguintes fatores:

a. Desenvolvido em Java – o que permite fácil integração com os outros componentes da arquitetura;
b. Amplamente utilizado em projetos renomados;
c. Oferece uma linguagem simples, mas poderosa para a definição dos modelos;
d. Extensível – permite que você crie ferramentas e associe ao contexto do modelo permitindo que você as referencie de dentro do modelo;
e. Projeto *open source* mantido pela fundação Apache.

4) JDOM: Esta biblioteca foi escolhida para realizar o acesso as informações presentes no arquivo XPDL, pois ela provê uma solução completa e simples para se trabalhar com arquivos no formato XML, de fácil entendimento para qualquer desenvolvedor Java e bem como por também projeto *open source*;

4.3. Funcionamento do Redbox

Nesta seção será detalhado passo a passo como funciona os algoritmos implementados no Redbox desde a leitura do arquivo XPDL até a geração automática dos códigos.

Para isto será tomado como base o modelo de processo de negócio hipotético descrito na Figura 4.2 que é composto por nove atividades sendo que destas duas, a “Atividade 6” e a “Atividade 9”, contêm o atributo estendido “Service” que nos informa que o serviço que implementa esta atividade já se encontra disponível e 2 gateways um do tipo XOR e outro do tipo AND. Este modelo não representa nenhum processo de negócio real, mas é o suficiente para que os conceitos empregados na solução proposta neste trabalho sejam facilmente compreendidos.

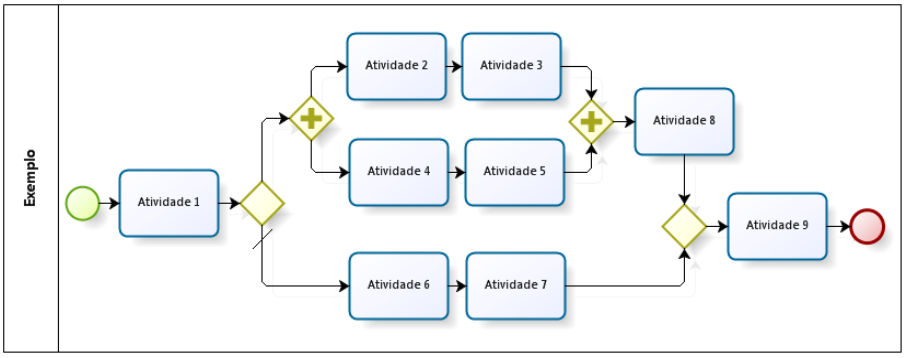

Figura 4.2: Modelo de processo de negócio hipotético.

4.3.1. Como funciona o algoritmo de identificação dos serviços válidos

Para que seja possível entender o algoritmo utilizado pelo Redbox no processo de identificação dos serviços válidos é necessário primeiramente entender a estrutura do modelo de dados que contêm todos os serviços identificados durante o processamento desta etapa e como as informações necessárias para identificar estes serviços estão representadas no arquivo XPDL.

4.3.1.1. Estrutura do modelo de dados

O modelo de dados utiliza-se de uma estrutura em árvore para representar os serviços identificados durante o processo. Cada elemento desta árvore representa um serviço identificado durante o processo sendo este elemento do tipo "ServiceDescriptor" este pode referenciar um dos quatro tipos abaixo de acordo com o diagrama de classes representado na Figura 4.3.

- "SimpleServiceDescriptor": Representa os serviços simples que correspondem as atividades do modelo de processo de negócio, são a forma mais simples de se representar um serviço, são responsáveis por executarem a lógica da atividade em si;

- “BundledServicesDescriptor”: Representa o agrupamento de serviços em seqüência ou em paralelo após a aplicação de cada heurística, ou seja, sua lista de serviços possui apenas serviços que já foram simplificados e agrupados previamente por outra heurística.
- “ExclusiveServiceDescriptor”: Representa o padrão de *workflow* XOR, possui uma lista de serviços que representa os serviços que podem ou não ser executados baseados no resultado da expressão condicional.
- “ParallelServiceDescriptor”: Representa o padrão de *workflow* AND, possui uma lista de serviços que representa todos os serviços que devem ser executados em paralelo.

Figura 4.3: Diagrama de classes referente as classes que descrevem um serviço.

O elemento presente na raiz desta árvore representa o processo como um todo e sempre será do tipo “BundledServicesDescriptor”, pois o processo é formado pela simplificação e agrupamento dos demais serviços. A lista de serviços deste elemento representa os serviços que devem ser executados para que este possa seja finalizado. Cada serviço presente nesta lista pode ser tanto a representação de um novo serviço, o qual exige que outros serviços também sejam executados para que ele possa ser finalizado, ou pode ser uma folha da árvore, ou seja, um serviço do tipo “SimpleServiceDescriptor” que irá executar a atividade em si.

Para facilitar o entendimento desta estrutura de dados, a árvore gerada no final da etapa de identificação de serviços para o modelo hipotético da Figura 4.2 está representada na Figura 4.4.

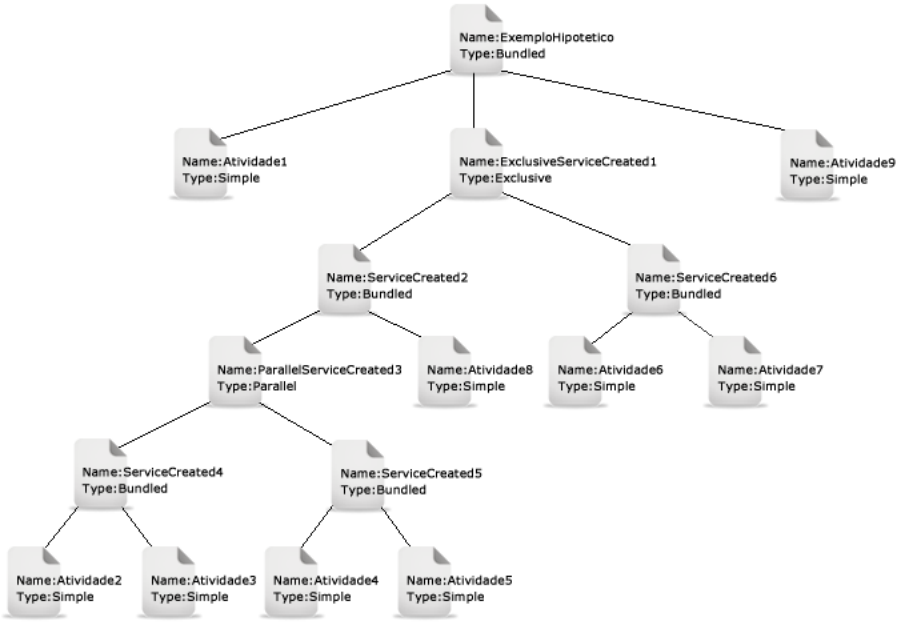

Figura 4.4: Árvore gerada no final da etapa de identificação de serviços para o modelo hipotético da Figura 4.3.

4.3.1.2. Como obter as informações necessárias a partir do arquivo XPDL

Para que seja possível identificar, simplificar e agrupar os serviços é necessário consultar algumas informações que estão presente no arquivo XPDL. Para isto o componente "XPDL Reader" conta com quinze métodos auxiliares que utilizam a biblioteca JDOM e são responsáveis por retornar estas informações para o algoritmo.

Nesta seção iremos entender qual o propósito destes quinze métodos e como as informações retornadas por eles estão presentes no arquivo XPDL do modelo representado na Figura 4.4. Serão exibidos apenas os fragmentos importantes deste arquivo para que seja possível entender como estas

informações são armazenadas no mesmo. A implementação em Java destes métodos podem ser encontrados nos anexos deste trabalho.

Método 1: private List<Element> getWorkflowProcesses()

Descrição: O arquivo XPDL pode conter a representação de um ou mais processos, o objetivo deste método é que ele retorne uma lista com todos os processos presentes no arquivo. Para isto basta que ele retorne todos os filhos do tipo "WorkflowProcess" do elemento "WorkflowProcesses" presente a partir da raiz do documento como pode ser visto na figura a seguir.

```
1  <xpdl2:WorkflowProcesses>
2
3      <xpdl2:WorkflowProcess Id="_FDL28ElxEd6rN_rG6TgLiA" Name="ExemploHipotetico">
4          ...
5      </xpdl2:WorkflowProcess>
6
7      ...
8
9    </xpdl2:WorkflowProcesses>
10 </xpdl2:Package>
```

Figura 4.5: Representação dos processos no arquivo XPDL.

Método 2: private List<Element> getActivities(Element workflowProcess)

Descrição: Este método deve retornar uma lista com todas as atividades pertencentes ao processo informado. Para isto ele deve retornar todos os filhos do tipo "Activity" do elemento "Activities" pertencente ao elemento que representa o processo desejado conforme figura a seguir.

```
<xpdl2:WorkflowProcesses>
  <xpdl2:WorkflowProcess Id="_FDL28ElxEd6rN_rG6TgLiA" Name="ExemploHipotetico">

    ...

    <xpdl2:Activities>
      <xpdl2:Activity Id="_HBjBGUlxEd6rN_rG6TgLiA" Name="StartEvent">
        <xpdl2:Event>
          <xpdl2:StartEvent Trigger="None"/>
        </xpdl2:Event>
      </xpdl2:Activity>
      <xpdl2:Activity Id="_IX03gElxEd6rN_rG6TgLiA" Name="Atividade1">
        <xpdl2:Implementation>
          <xpdl2:No/>
        </xpdl2:Implementation>
      </xpdl2:Activity>

      ...

     </xpdl2:Activities>

     ...

  </xpdl2:WorkflowProcess>
</xpdl2:WorkflowProcesses>
</xpdl2:Package>
```

Figura 4.6: Representação das atividades de um processo no arquivo XPDL.

Método 3: private Type getActivityType(Element activity)

Descrição: Como no arquivo XPDL o elemento "Activity" é utilizado para representar não só atividades, mas também os outros elementos da notação BPMN como discutido na seção 2.4 do capítulo 2. Este método deve nos retornar o tipo da atividade que estamos trabalhando. Para isto ele deve verificar se o elemento que representa a atividade possui um elemento dentre os três tipos abaixo após esta verificação deve retorna o tipo do elemento como pode ser visto na Figura 4.7.

- "Event": Se a atividade possuir este elemento como filho então ela representa um evento da notação BPMN. Neste caso outro método auxiliar será chamado para nos informar qual o tipo do evento.
- "Implementation": Se este filho estiver presente então a atividade representa uma atividade da notação BPMN.

- “Route”: Finalmente se a atividade possuir este elemento como filho então representa um gateway da notação BPMN. Neste caso outro método auxiliar será chamado para nos informar qual o tipo do gateway.

```
1
2       ...
3
4       <xpdl2:Activities>
5         <xpdl2:Activity Id="_HBjBGUlxEd6rN_rG6TgLiA" Name="StartEvent">
6           <xpdl2:Event>
7             <xpdl2:StartEvent Trigger="None"/>
8           </xpdl2:Event>
9         </xpdl2:Activity>
10        <xpdl2:Activity Id="_HBjBGklxEd6rN_rG6TgLiA" Name="EndEvent">
11          <xpdl2:Event>
12            <xpdl2:EndEvent Result="None"/>
13          </xpdl2:Event>
14        </xpdl2:Activity>
15        <xpdl2:Activity Id="_IX03gElxEd6rN_rG6TgLiA" Name="Atividade1">
16          <xpdl2:Implementation>
17            <xpdl2:No/>
18          </xpdl2:Implementation>
19        </xpdl2:Activity>
20        <xpdl2:Activity Id="_KQrPgElxEd6rN_rG6TgLiA">
21          <xpdl2:Route GatewayType="Exclusive" MarkerVisible="true" ExclusiveType="Data"/>
22
23          ...
24
25        </xpdl2:Activity>
26        <xpdl2:Activity Id="_L1tToElxEd6rN_rG6TgLiA">
27          <xpdl2:Route GatewayType="Parallel"/>
28          ...
29        </xpdl2:Activity>
30      </xpdl2:Activities>
31
32      ...
33
```

Figura 4.7: Representação dos tipos de atividade presentes em um arquivo XPDL.

Método 4: private Type getEventType(Element activity)

Descrição: Como vários tipos de eventos da notação BPMN são representados através de atividades que possuem o elemento “Event” como filho, este método deve retornar o tipo do evento que a atividade informada representa. Para isto basta verificar se o elemento “Event” desta atividade possui o elemento “StartEvent” ou “EndEvent” o como filho o que caracteriza esta atividade como o início ou o fim do modelo de processo de negócio

respectivamente. A representação desta informação no arquivo XPDL pode ser observada na Figura 4.7.

Método 5: private Type getRouteType(Element activity)

Descrição: Como vários tipos de gateways da notação BPMN são representados através de atividades que possuem o elemento "Route" como filho, este método deve retornar o tipo do gateway que a atividade informada representa. Para isto basta verificar o valor do atributo "GatewayType" do elemento "Route" desta atividade. A representação desta informação no arquivo XPDL também pode ser observada na Figura 4.7.

Método 6: private List<Element> getTransitions(Element workflowProcess)

Descrição: Este método deve retornar uma lista com todas as transições pertencentes ao processo informado. Para isto ele deve retornar todos os filhos do tipo "Transition" do elemento "Transitions" pertencente ao elemento que representa o processo desejado como demonstrado na Figura 4.8.

```
<xpdl2:WorkflowProcesses>
  <xpdl2:WorkflowProcess Id="_FDL28ElxEd6rN_rG6TgLiA" Name="ExemploHipotetico">

    ...

    <xpdl2:Transitions>
      <xpdl2:Transition Id="_IX03gUlxEd6rN_rG6TgLiA"From="_HBjBGUlxEd6rN_rG6TgLiA" To="_IX03gElxEd6rN_rG6TgLiA">
      </xpdl2:Transition>
      <xpdl2:Transition Id="_KQrPgUlxEd6rN_rG6TgLiA" From="_IX03gElxEd6rN_rG6TgLiA" To="_KQrPgElxEd6rN_rG6TgLiA">

      ...

    </xpdl2:Transitions>
  </xpdl2:WorkflowProcess>
</xpdl2:WorkflowProcesses>
```

Figura 4.8: Representação das transições de um process no arquivo XPDL.

Método 7: private Element getTransition(Element workflowProcess, String id)

Descrição: Este método retorna a transição que possui o id informado dentre todas as transições pertencentes ao processo informado. Para isto basta que verifique se o valor do atributo “Id” do elemento “Transition” é igual ao valor informado. A representação desta informação no arquivo XPDL pode ser vista na Figura 4.8.

Método 8: private Element getActivity(Element workflowProcess, String id)

Descrição: Este método retorna a atividade que possui o id informado dentre todas as atividades pertencentes ao processo informado. Para isto basta que verifique se o valor do atributo “Id” do elemento “Activity” é igual ao valor informado. A representação desta informação no arquivo XPDL pode ser vista na Figura 4.7.

Método 9: private Element getStartEvent(Element workflowProcess)

Descrição: Este método retorna a atividade que representa o evento inicial na modelagem do processo de negócio. Para isto basta verificar para todas as atividades do processo informado qual delas possui o tipo igual a “Start” através do “getActivityType” já mencionado anteriormente. A representação desta informação no arquivo XPDL também pode ser vista na Figura 4.7.

Método 10: private List<Element> getRouteTransitions(Element workflowProcess, Element route)

Descrição: Retorna uma lista contendo as transições que tem como ponto de partida o gateway informado. Para isto basta retornar a lista de filhos

do tipo “TransitionRef” do elemento “TransitionRefs” pertencente ao gateway e processo informados conforme demonstrado na Figura 4.9.

```
1
2         ...
3
4         <xpdl2:Activity Id="_KQrPgElxEd6rN_rG6TgLiA">
5           <xpdl2:Route GatewayType="Exclusive" MarkerVisible="true" ExclusiveType="Data"/>
6           <xpdl2:TransitionRestrictions>
7             <xpdl2:TransitionRestriction>
8               <xpdl2:Split Type="Exclusive" ExclusiveType="Data">
9                 <xpdl2:TransitionRefs>
10                  <xpdl2:TransitionRef Id="_cdNRYElxEd6rN_rG6TgLiA"/>
11                  <xpdl2:TransitionRef Id="_gtv5gElxEd6rN_rG6TgLiA"/>
12                </xpdl2:TransitionRefs>
13              </xpdl2:Split>
14            </xpdl2:TransitionRestriction>
15          </xpdl2:TransitionRestrictions>
16        </xpdl2:Activity>
17        <xpdl2:Activity Id="_L1tToElxEd6rN_rG6TgLiA">
18          <xpdl2:Route GatewayType="Parallel"/>
19          <xpdl2:TransitionRestrictions>
20            <xpdl2:TransitionRestriction>
21              <xpdl2:Split Type="Parallel">
22                <xpdl2:TransitionRefs>
23                  <xpdl2:TransitionRef Id="_dD8hQElxEd6rN_rG6TgLiA"/>
24                  <xpdl2:TransitionRef Id="_dtOHwElxEd6rN_rG6TgLiA"/>
25                </xpdl2:TransitionRefs>
26              </xpdl2:Split>
27            </xpdl2:TransitionRestriction>
28          </xpdl2:TransitionRestrictions>
29        </xpdl2:Activity>
30
31      ...
32
```

Figura 4.9: Representação das restrições de transições de um gateway no arquivo XPDL.

Método 11: private Element getNextActivity(Element workflowProcess, Element from)

Descrição: Retorna a próxima atividade no processo referente à atividade informada. Para isto basta percorrer a lista de transições do processo informado verificando qual delas possui o valor do atributo “From” igual ao valor do atributo “Id” da atividade informada. A representação desta informação no arquivo XPDL pode ser vista na Figura 4.8.

Método 12: private Type getJoinType(Element activity)

Descrição: Como uma atividade que representa um “Join”, ou seja, um ponto no processo onde dois ou mais se encontram são iguais para os diversos tipos de gateways existentes na notação BPMN este método deve nos retornar qual o tipo referente a este join. Para isto basta verificar o valor do atributo “Type” do elemento “Join” como pode ser visto na Figura 4.10.

```
1
2         ...
3
4         <xpdl2:Activity Id="_PqFkMElxEd6rN_rG6TgLiA">
5           <xpdl2:Route GatewayType="Parallel"/>
6           <xpdl2:TransitionRestrictions>
7             <xpdl2:TransitionRestriction>
8               <xpdl2:Join Type="Parallel"/>
9             </xpdl2:TransitionRestriction>
10          </xpdl2:TransitionRestrictions>
11        </xpdl2:Activity>
12
13        ...
14
15        </xpdl2:Activity>
16        <xpdl2:Activity Id="_Ye1-YElxEd6rN_rG6TgLiA">
17          <xpdl2:Route GatewayType="Exclusive" MarkerVisible="true" ExclusiveType="Data"/>
18          <xpdl2:TransitionRestrictions>
19            <xpdl2:TransitionRestriction>
20              <xpdl2:Join Type="Exclusive" ExclusiveType="Data"/>
21            </xpdl2:TransitionRestriction>
22          </xpdl2:TransitionRestrictions>
23        </xpdl2:Activity>
24
25        ...
26
```

Figura 4.10: Representação do tipo de um “Join” em um arquivo XPDL.

Método 13: private boolean isJoin(Element activity)

Descrição: Retorna verdadeiro caso a atividade informada represente um “Join” e falso caso contrário. Para realizar esta verificação basta verificar se a atividade informada possui o elemento “Join” como pode ser visto na Figura 4.10.

Método 14: private List<Element> getExtendedAttributes(Element activity)

Descrição: Retorna uma lista com todos os atributos estendidos da atividade informada. Para isto basta retornar a lista de filhos do tipo "ExtendedAttribute" do elemento "ExtendedAttributes" da atividade informada conforme figura abaixo.

```
1
2         ...
3
4         <xpdl2:Activity Id="_U333sElxEd6rN_rG6TgLiA" Name="Atividade6">
5           <xpdl2:Implementation>
6             <xpdl2:No/>
7           </xpdl2:Implementation>
8           <xpdl2:ExtendedAttributes>
9             <xpdl2:ExtendedAttribute Name="Service" Value="myApplication.myModule.Task6"/>
10          </xpdl2:ExtendedAttributes>
11        </xpdl2:Activity>
12
13        ...
14
```

Figura 4.11: Representação dos atributos estendidos de uma atividade no arquivo XPDL.

Método 15: private String getExtendedAttributeValue(Element activity, String name)

Descrição: Retorna o valor do atributo estendido informado da atividade em questão. Para isto basta percorrer a lista de atributos estendidos da atividade informada e retornar o valor do atributo "Value" do atributo estendido que tiver o atributo "Name" igual ao nome do atributo informado como pode ser visto na Figura 4.11.

4.3.1.3. Algoritmo de identificação de serviços válidos

O algoritmo implementado neste trabalho é dividido em cinco métodos. Para facilitar o entendimento do mesmo será fornecido uma breve descrição sobre cada um destes métodos bem como os pontos mais importantes serão ressaltados.

O primeiro método que podemos observar na Figura 4.12 é responsável por percorrer todos os processos que estão presentes no arquivo XPDL e para cada processo identificar e criar os serviços necessários para que seja possível representá-lo como um todo. É importante ressaltar duas etapas muito importantes que ocorrem dentro deste método.

```
1  public List<ServiceDescriptor> findServices() {
2
3      List<ServiceDescriptor> processServices = new ArrayList<ServiceDescriptor>();
4
5      try {
6
7          createDocument();
8
9          List<Element> workflowProcesses = getWorkflowProcesses();
10
11         for (Element workflowProcess : workflowProcesses) {
12
13             List<ServiceDescriptor> services = new LinkedList<ServiceDescriptor>();
14
15             Element activity = getNextActivity(workflowProcess, getStartEvent(workflowProcess));
16
17             while (activity != null) {
18                 activity = createService(workflowProcess, activity, services);
19             }
20
21             BundledServicesDescriptor bundledServices = new BundledServicesDescriptor();
22             bundledServices.setName(workflowProcess.getAttributeValue("Name"));
23             bundledServices.getServices().addAll(services);
24
25             processServices.add(bundledServices);
26
27         }
28
29     } catch (Exception e) {
30         e.printStackTrace();
31     }
32
33     return processServices;
34 }
```

Figura 4.12: Método responsável por identificar e criar os serviços necessário para representar cada processo descrito no arquivo XPDL.

A primeira etapa inicia na linha 15 e finaliza na linha 19, nesta etapa o algoritmo identifica e agrupa segundo as heurísticas descritas no capítulo anterior todas as atividades presentes no arquivo XPDL. Para isto ele inicia obtendo a primeira atividade presente no modelo de processo de negócio, ou seja, a próxima atividade a partir do evento inicial. Enquanto esta atividade for diferente de nulo, ou seja, ainda houver mais atividades no processo ele chama o

método responsável por identificar e criar um novo serviço a partir do tipo da atividade informada associando-o a lista de serviços do processo e retornando para este método a próxima atividade após o serviço criado como pode ser visto na figura a seguir.

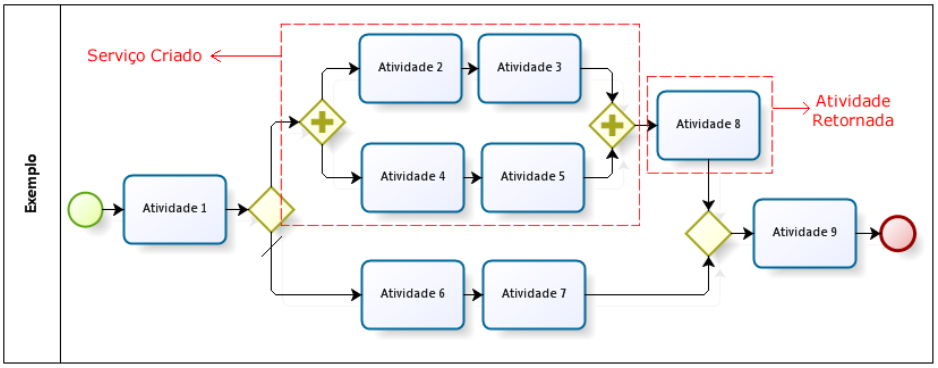

Figura 4.13: Atividade retornada após a criação de um serviço.

A segunda etapa inicia nas linhas 21 a 23 após ser realizada a identificação e o agrupamento da primeira etapa, na lista de serviços para este processo criada na linha 13 encontram-se apenas serviços simples que devem ser agrupados de forma linear para que o serviço criado na linha 21 possa representar processo como um todo como pode ser visto na figura a seguir.

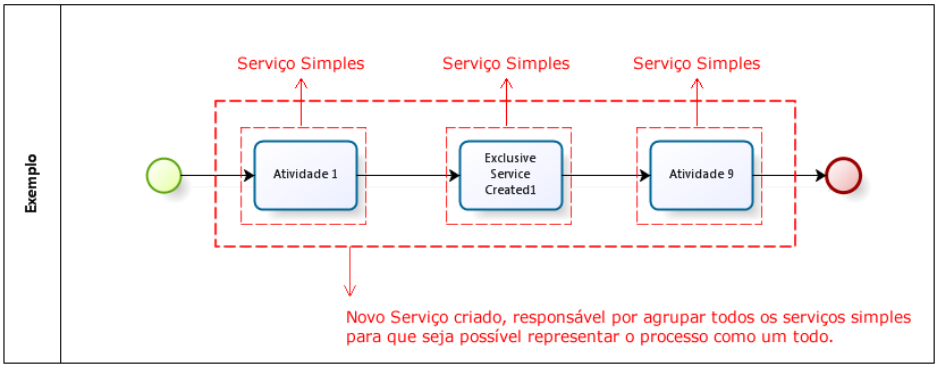

Figura 4.14: Agrupamento linear a fim de representar o serviço como um todo.

O segundo método presente no algoritmo é o método descrito na Figura 4.15 como mencionado anteriormente ele é responsável por criar um novo serviço a partir da atividade que recebe adicionando este novo serviço criado a lista de serviços informada e retornando para o método que o chamou a próxima atividade após este novo serviço criado. Para isto o método identifica qual o tipo de atividade e chama o método que trata este tipo de atividade em específico, ou seja, o método que será responsável por aplicar a heurística que irá criar o serviço.

```
1  public Element createService(Element workflowProcess, Element activity, List<ServiceDescriptor> services) {
2
3      Type activityType = getActivityType(activity);
4
5      if (activityType != Type.END) {
6
7          if (activityType == Type.TASK) {
8              return createSimpleService(workflowProcess, activity, services);
9          } else if (activityType == Type.XOR) {
10             return createExclusiveService(workflowProcess, activity, services);
11         } else if (activityType == Type.AND) {
12             return createParallelService(workflowProcess, activity, services);
13         }
14
15     }
16
17     return null;
18 }
```

Figura 4.15: Método responsável pela criação do novo serviço.

Os próximos métodos que serão discutidos a partir de agora são os métodos responsáveis por aplicar as heurísticas discutidas no capítulo anterior.

Começaremos observando o método descrito na Figura 4.16, este método é responsável por criar o serviço mais simples possível, ou seja, o serviço que irá representar cada atividade do modelo de processo de negócio. É importante observar a linha 6, onde é verificado se a atividade possui o atributo estendido "Service", que diz que o serviço que implementa esta atividade já foi desenvolvido pela organização, se este atributo existir será retornado o nome do serviço que deverá ser chamado pelo serviço recém-criado para que esta atividade possa ser executada.

```
1  public Element createSimpleService(Element workflowProcess, Element activity, List<ServiceDescriptor> services) {
2
3      SimpleServiceDescriptor simpleService = new SimpleServiceDescriptor();
4      simpleService.setId(activity.getAttributeValue("Id"));
5      simpleService.setName(activity.getAttributeValue("Name"));
6      simpleService.setService(getExtendedAttributeValue(activity, "Service"));
7
8      services.add(simpleService);
9
10     return getNextActivity(workflowProcess, activity);
11 }
```

Figura 4.16: Método responsável por aplicar a heurística 2.

Já o método descrito na Figura 4.17 é responsável por tratar as heurísticas que envolvem o padrão de *workflow* XOR. Primeiramente ele cria um novo serviço do tipo "ExclusiveServiceDescriptor" que irá representar este padrão e para cada transição que este gateway possui no arquivo XPDL ele

criará um novo serviço do tipo “BundledServicesDescriptor” que será responsável por armazenar todos os serviços presentes nesta transição.

É importante observar as linhas 13 a 17, nesta etapa é realizado o agrupamento linear das atividades presentes na transição, ou seja, enquanto não encontrar o final do padrão de *workflow* XOR, que é representado por um “Join” do tipo XOR, ele irá chamar o método responsável por criar o serviço visto anteriormente com uma diferença do método descrito na Figura 4.12, pois desta vez ele passará a lista de serviços do serviço que representa o agrupamento linear para que o novo serviço seja associado a ela. Já na linha 23 é realizado o agrupamento em paralelo dos serviços que representam cada transição do gateway.

```
1  public Element createExclusiveService(Element workflowProcess, Element activity, List<ServiceDescriptor> services) {
2
3      ExclusiveServiceDescriptor exclusiveService =
4          new ExclusiveServiceDescriptor("", "ExclusiveServiceCreated" + ++serviceCreatedCounter);
5
6      Element element = null;
7
8      for (Element transition : getRouteTransitions(workflowProcess, activity)) {
9
10         BundledServicesDescriptor bundledServices =
11             new BundledServicesDescriptor("", "ServiceCreated" + ++serviceCreatedCounter);
12
13         element = getActivity(workflowProcess, transition.getAttributeValue("To"));
14
15         while (!isJoin(element) && (getJoinType(element) != Type.XOR)) {
16             element = createService(workflowProcess, element, bundledServices.getServices());
17         }
18
19         exclusiveService.getServices().add(bundledServices);
20
21     }
22
23     services.add(exclusiveService);
24
25     return getNextActivity(workflowProcess, element);
26 }
```

Figura 4.17: Método responsável por criar um novo serviço baseado no padrão de *workflow* XOR.

Como se pode observar os métodos descritos na Figura 4.12 e na Figura 4.17 possuem a mesma idéia central, criar o serviço que irá realizar o agrupamento das atividades que estão em seqüência e em seguida criar o serviço que irá realizar o agrupamento destes serviços em paralelo. O que tornou o algoritmo simples foi a idéia de se utilizar um método recursivo que irá criar o

serviço a partir da atividade que representa um padrão de *workflow* e retornará a próxima atividade após este serviço recém criado.

Devido a esta simplicidade presente no algoritmo, podemos observar que o método descrito na Figura 4.18 segue o mesmo princípio, mas tratando agora o padrão de *workflow* AND. Este método e o método da Figura 4.17 são praticamente iguais apenas diferem na condição de parada da recursão que neste caso busca por um elemento "Join" do tipo AND.

```
1 public Element createParallelService(Element workflowProcess, Element activity, List<ServiceDescriptor> services) {
2
3     ParallelServiceDescriptor parallelService =
4         new ParallelServiceDescriptor("", "ParallelServiceCreated" + ++serviceCreatedCounter);
5
6     Element element = null;
7
8     for (Element transition : getRouteTransitions(workflowProcess, activity)) {
9
10        BundledServicesDescriptor bundledServices =
11            new BundledServicesDescriptor("", "ServiceCreated" + ++serviceCreatedCounter);
12
13        element = getActivity(workflowProcess, transition.getAttributeValue("To"));
14
15        while (!isJoin(element) && (getJoinType(element) != Type.AND)) {
16            element = createService(workflowProcess, element, bundledServices.getServices());
17        }
18
19        parallelService.getServices().add(bundledServices);
20
21    }
22
23    services.add(parallelService);
24
25    return getNextActivity(workflowProcess, element);
26 }
```

Figura 4.18: Método responsável por criar um novo serviço baseado no padrão de *workflow* AND.

Após o algoritmo percorrer todos os processos presentes no arquivo XPDL, ele irá retornar para o método que o chamou o modelo de dados que possui as informações necessárias para que os serviços sejam gerados. O modelo de dados resultante após este algoritmo ser aplicado ao modelo hipotético da Figura 4.2 está representando na Figura 4.4 como visto anteriormente.

4.3.2. **Como o código é gerado**

O componente responsável por gerar todo o código necessário para dar o suporte ao modelo de processo de negócio analisado na etapa anterior é o "Service Generator", cujo funcionamento é bem simples.

Este componente recebe como entrada o caminho no qual os serviços devem ser gerados e a lista de serviços gerados pelo componente "XPDL Reader" que representa o modelo de dados.

O método responsável por gerar o código dos serviços propriamente dito trabalha de forma recursiva, ou seja, para cada serviço presente na lista de serviços que este método recebeu como parâmetro, ele irá identificar qual o tipo deste serviço e associá-lo a um modelo do Velocity, modelo este que define a estrutura do código a ser gerado. Se este serviço possui uma lista de serviços então este método será chamado novamente, mas passando como parâmetro a lista de serviços deste serviço. Se o serviço analisado não depender de outros serviços este será associado ao contexto do Velocity, e este contexto será passado para o modelo para que desta forma o Velocity possa preenchê-lo com as informações presentes neste serviço.

Após gerar todos os serviços, o "Service Generator" gera um mapeamento deste serviço para que este se torne disponível para os seus consumidores.

A saída desta etapa com todos os serviços identificados e gerados a partir do modelo hipotético da Figura 4.2 pode ser visto na Figura 4.19.

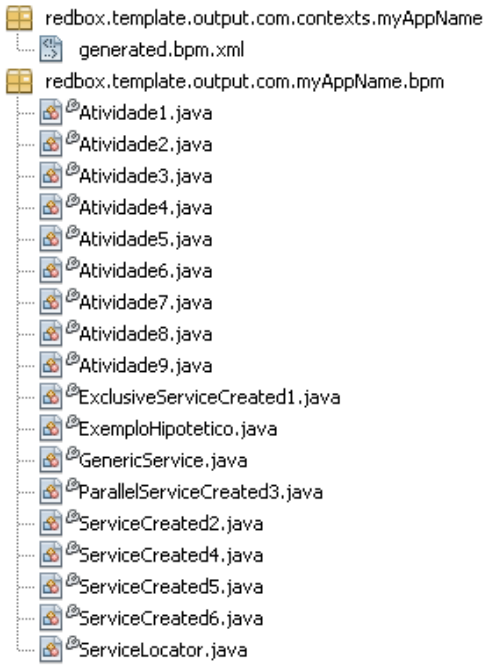

Figura 4.19: Saída do Redbox para o modelo hipotético da Figura 4.2.

4.4. Extensibilidade

Uma das características principais do Redbox é que ele não se limita as heurísticas que foram implementadas neste trabalho, devido a sua arquitetura simples e de fácil entendimento ele se tornou uma ferramenta facilmente extensível.

Novas heurísticas, para tratar diferentes padrões de *workflow*, podem ser facilmente incorporadas à ferramenta. Para isto basta que o método descrito na Figura 4.13 seja alterado para que ele consiga identificar este novo padrão e encaminhar a atividade para o método que irá implementar a heurística em si. Estendendo desta forma as capacidades da solução proposta de identificar novos serviços bem como analisar modelos de processos de negócio mais complexos.

5. Conclusão

Visto a demanda que as empresas hoje possuem por agilidade operacional, maior confiabilidade, redução de custos, maior capacidade de se adaptar as mudanças impostas por clientes e, principalmente e estabelecer o alinhamento estratégico, uma nova forma de gestão a *Business Process Management* (BPM) vem ganhando cada vez mais espaço dentro destas empresas. Com isto novas tecnologias são disponibilizadas com o intuito de prover todo o suporte necessário para que o uso efetivo de BPM seja possível.

Neste contexto o presente trabalho apresentou uma ferramenta capaz de identificar serviços em um modelo de processo de negócio que utiliza a notação *Business Process Modeling Notation* (BPMN), bem como a geração de todo o código em uma arquitetura orientada a serviço para dar o suporte necessário para que a aplicação descrita neste modelo possa ser executada. Permitindo com isto que as mudanças e melhorias que se façam necessárias nestes modelos sejam rapidamente refletidas nas aplicações que dão todo o suporte aos negócios da empresa, contribuindo assim para que a mesma alcance os seus objetivos.

Para que fosse possível realizar a implementação desta ferramenta foi necessário realizar um estudo sobre a BPMN e heurísticas capazes de identificar serviços nestes diagramas através da aplicação de padrões de *workflow*. Com base nestas informações foi criado um modelo de geração de código baseado na XML Process Definition Language (XPDL), linguagem para troca de informações entre as ferramentas de modelagem totalmente compatível com a notação BPMN. Na implementação deste modelo foi utilizado o *engine* Velocity que permitiu que toda a geração de código fosse realizada a partir de transformações baseadas em modelo, tornando assim o código gerado totalmente adaptável.

O principal problema encontrado antes de iniciar a implementação da solução proposta neste trabalho, é que um grande número de heurísticas poderiam ser implementadas para tratar as mais diversas situações que podem estar presentes em um modelo de processo de negócio. Decidiu-se então gerar uma ferramenta que fosse simples, poderosa, mas ao mesmo tempo extensível, possibilitando assim que novas heurísticas fossem adicionadas ao núcleo da ferramenta.

O resultado final foi uma ferramenta extensível, que permite que novos padrões de *workflow* sejam incorporados, aumentando assim a capacidade de identificar serviços em modelos de processo de negócio mais complexos.

Como sugestão para trabalhos futuros temos: implementar os demais padrões de *worflow*, realizar a integração com os diversos Business Process Management Systems (BPMS) presentes no mercado e implementar o processo de gerenciamento dos serviços gerados pela ferramenta.

6. Referencial Bibliográfico

Azevedo, L.G; Baião, F. A.; Santoro, F.; Souza, J.; Revoredo, K.; Pereira, V.; Erlain, I.. Identificação de Serviços a partir da Modelagem de Processos de Negócio. **V Simpósio Brasileiro de Sistemas de Informação**, 2009

Chiavenato, I.. **Manual de reengenharia: um guia para reinventar e humanizar a sua empresa com a ajuda das pessoas**. Makron Books. São Paulo, 1995.

Davenport, T.. **Reengenharia de Processo (5ª Edição)**. Editora Campus. Rio de Janeiro, 2004.

Delphi Group. **BPM 2005: market milestone report**, 2005. Disponível em http://www.delphigroup.com. Consultado 12/11/2008.

Lublinsky, B.. **Defining SOA as an architectural style**, 2007. Disponível em http://www.ibm.com/developerworks/architecture/library/ar-soastyle/. Consultado em 12/03/2009.

OASIS - **Reference Model for Service Oriented Architecture 1.0**, 2006. Disponível em http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf. Consultado em 09/01/2009.

SOA Open Group. **Whitepaper: Service-Oriented Architecture (SOA)**, 2006. Disponível em http://www.opengroup.org/projects/soa/doc.tpl?CALLER=documents.tpl&dcat=&gdid=18732. Consultado em 11/05/2009.

Van der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B., Barros, A. P.. **Workflow patterns. Distributed and Parallel Databases**, 2002. Disponível em http://www.workflowpatterns.com/documentation/documents/wfs-pat-2002.pdf. Consultado em 03/04/2009.

WfMC. **Process Definition Interface - XML Process Definition Language**, 2005. Disponível em http://www.wfmc.org/index.php?option=com_docman&task=doc_details&gid=37&Itemid=72. Consultado em 13/05/2009.

WfMC. **XPDL 2.1 - Integrating Process Interchange & BPMN**, 2008. Disponível em http://www.wfmc.org/index.php?option=com_docman&task=doc_details&gid=132&Itemid=72. Consultado em 13/05/2009.

White, S. A.. **Introduction to BPMN**, 2004. Disponível em http://www.bpmn.org/Documents/Introduction to BPMN.pdf. Consultado em 12/10/2008.

White, S. A. and Miers, D.. **BPMN Modeling and Reference Guide**. Booksurge Llc, 2008.

Anexo 1

Implementação em Java referente à solução proposta neste trabalho.

ServiceGenerator.java

```
package redbox.core;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.List;
import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.app.Velocity;
import redbox.descriptor.BundledServicesDescriptor;
import redbox.descriptor.ExclusiveServiceDescriptor;
import redbox.descriptor.ParallelServiceDescriptor;
import redbox.descriptor.ServiceDescriptor;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class ServiceGenerator {

    private final String bundledServicesTemplate =
"src/redbox/template/BundledServices.vt";
    private final String exclusiveServiceTemplate =
"src/redbox/template/ExclusiveService.vt";
    private final String parallelServiceTemplate =
"src/redbox/template/ParallelService.vt";
    private final String simpleServiceTemplate =
"src/redbox/template/SimpleService.vt";
    private final String mappingServicesTemplate =
"src/redbox/template/MappingServices.vt";

    private String appName;
    private String bpmPackage;
    private String outputPath;
    private String outputDirectoryOfServices;
    private String outputDirectoryOfMapping;

    public ServiceGenerator(String appName, String bpmPackage, String
outputPath) {
        this.appName = appName;
        this.bpmPackage = bpmPackage;
        this.outputPath = outputPath;
    }
```

```
    /**
     * Inicializa o engine do Velocity, setando algumas propriedades.
     *
     */
    public void init() throws Exception {
        Velocity.setProperty("directive.foreach.counter.initial.value",
0);
        Velocity.init();
    }

    /**
     * Método responsável por inicializar a geração dos serviços.
     *
     * @param services Lista com os serviços que devem ser gerados
     */
    public void start(List<ServiceDescriptor> services) {

        try {

            createAndSetOutputDirOfServices();
            generateServices(services);
            createAndSetOutputDirOfMapping();
            generateMapping(services);

            System.out.print

        } catch (Exception e
            ex.printStackTra...,
        }
    }

    /**
     * Função auxiliar responsável por criar um diretório incluindo os
     * diretórios pai inexistentes.
     *
     * @param dir String que representa o caminho do diretório a ser
criado
     */
    private void createDirectories(String dir) {

        File file = new File(dir);

        if (!file.exists()) {
            file.mkdirs();
        }
    }

    /**
     * Cria e seta o diretório onde será armazenado o mapeamento do
processo gerado.
     *
     */
    private void createAndSetOutputDirOfMapping() {

        String fileSeparator = System.getProperty("file.separator");

        if (!String.valueOf(outputPath.charAt(outputPath.length() -
1)).equals(fileSeparator)) {
            outputPath += fileSeparator;
```

```
        }

        outputDirectoryOfMapping = outputPath + fileSeparator + "com" +
fileSeparator +
                "contexts" + fileSeparator + appName + fileSeparator;

        createDirectories(outputDirectoryOfMapping);
    }

    /**
     * Cria e seta o diretório onde será armazenado os serviços gerados.
     *
     */
    private void createAndSetOutputDirOfServices() {

        String fileSeparator = System.getProperty("file.separator");

        if (!String.valueOf(outputPath.charAt(outputPath.length() -
1)).equals(fileSeparator)) {
            outputPath += fileSeparator;
        }

        outputDirectoryOfServices = outputPath + fileSeparator + "com" +
fileSeparator
                + appName + fileSeparator + bpmPackage + fileSeparator;

        createDirectories(outputDirectoryOfServices);
    }

    /**
     * Gera o mapeamento referente ao processo como um todo, para que os
seus
     * consumidores possam utilizá-lo.
     *
     * @param services Lista com os serviços que representam um processo
como um todo
     */
    private void generateMapping(List<ServiceDescriptor> services) throws
Exception {

        String outputFileName = outputDirectoryOfMapping + "generated." +
bpmPackage + ".xml";
        BufferedWriter writer = new BufferedWriter(new
FileWriter(outputFileName));

        VelocityContext context = new VelocityContext();

        context.put("appName", appName);
        context.put("bpmPackage", bpmPackage);
        context.put("services", services);

        Template template = Velocity.getTemplate(mappingServicesTemplate);

        template.merge(context, writer);

        writer.flush();
        writer.close();

        System.out.println("Mapping: " + outputFileName + " generated!");
```

```
    }

    /**
     * Gera os serviços necessário para dar o suporte processo como um
todo.
     *
     * @param services Lista com os serviços que representam um processo
como um todo
     */
    private void generateServices(List<ServiceDescriptor> services) throws
Exception {

        VelocityContext context = new VelocityContext();

        context.put("appName", appName);
        context.put("bpmPackage", bpmPackage);

        Template template = null;

        for (ServiceDescriptor serviceDescriptor : services) {

            String outputFileName = outputDirectoryOfServices +
serviceDescriptor.getKlassName() + ".java";
            BufferedWriter writer = new BufferedWriter(new
FileWriter(outputFileName));

            context.put("service", serviceDescriptor);

            if (serviceDescriptor.getType().equals("Simple")) {
                template = Velocity.getTemplate(simpleServiceTemplate);
            } else if ((serviceDescriptor.getType().equals("Exclusive")))
{
                template = Velocity.getTemplate(exclusiveServiceTemplate);
                generateServices(((ExclusiveServiceDescriptor)
serviceDescriptor).getServices());
            } else if ((serviceDescriptor.getType().equals("Parallel"))) {
                template = Velocity.getTemplate(parallelServiceTemplate);
                generateServices(((ParallelServiceDescriptor)
serviceDescriptor).getServices());
            } else if (serviceDescriptor.getType().equals("Bundled")) {
                template = Velocity.getTemplate(bundledServicesTemplate);
                generateServices(((BundledServicesDescriptor)
serviceDescriptor).getServices());
            }

            template.merge(context, writer);

            writer.flush();
            writer.close();

            System.out.println("Service: " + outputFileName + "
generated!");
        }
    }

}
```

XPDLReader.java

```
package redbox.core;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.Namespace;
import org.jdom.input.SAXBuilder;
import redbox.descriptor.BundledServicesDescriptor;
import redbox.descriptor.ExclusiveServiceDescriptor;
import redbox.descriptor.ParallelServiceDescriptor;
import redbox.descriptor.ServiceDescriptor;
import redbox.descriptor.SimpleServiceDescriptor;
import redbox.elements.Type;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class XPDLReader {

    private Document document;
    private Element root;
    private Namespace ns;
    private String fileName;
    private int serviceCreatedCounter = 0;

    public XPDLReader(String fileName) {
        this.fileName = fileName;
    }

    /**
     * Cria a representação do arquivo XPDL fornecido.
     *
     */
    private void createDocument() throws JDOMException, IOException {

        // Crie o documento utilizando SAX e Xerces, sem nenhuma validação
        SAXBuilder builder = new SAXBuilder(false);

        // Cria o documento
        document = builder.build(new File(fileName));

        // Obtém o elemento raiz
        root = document.getRootElement();

        // Obtém o namespace;
        ns = root.getNamespace();
    }

    /**
     * Retorna uma lista com todas as atividades do processo informado.
```

```
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @return Lista de atividades
     */
    private List<Element> getActivities(Element workflowProcess) {
        return workflowProcess.getChild("Activities",
ns).getChildren("Activity", ns);
    }


    /**
     * Retorna a atividade que representanda o evento inicial do processo
informado.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @return Evento inicial
     */
    private Element getStartEvent(Element workflowProcess) {

        List<Element> activities = getActivities(workflowProcess);

        for (Element activity : activities) {

            if (getActivityType(activity) == Type.START) {
                return activity;
            }
        }

        return null;
    }

    /**
     * Retorna uma atividade específica do processo informado. Caso não
exista
     * a atividade com o id informado retorna null.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param id Id que identifica o elemento referente a atividade
desejada no arquivo xpdl
     * @return Atividade
     */
    private Element getActivity(Element workflowProcess, String id) {

        List<Element> activities = getActivities(workflowProcess);

        for (Element activity : activities) {

            if (activity.getAttributeValue("Id").equals(id)) {
                return activity;
            }
        }

        return null;
    }

    /**
```

```
     * Retorna a próxima atividade no processo referente a atividade
informada.
     * Caso não exista mais atividades a partir da atividade informada
retorna null.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param from Elemento que representa uma atividade no arquivo XPDL
     * @return Próxima atividade
     */
    private Element getNextActivity(Element workflowProcess, Element from)
{

        if (from != null) {

            List<Element> transitions = getTransitions(workflowProcess);

            for (Element transition : transitions) {

                if
(transition.getAttributeValue("From").equals(from.getAttributeValue("Id"))
) {
                    return getActivity(workflowProcess,
transition.getAttributeValue("To"));
                }
            }
        }

        return null;
    }

    /**
     * Retorna uma transição específica do processo informado. Caso não
exista
     * a transição com o id informado retorna null.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param id Id que identifica o elemento referente a transição
desejada no arquivo xpdl
     * @return Atividade
     */
    private Element getTransition(Element workflowProcess, String id) {

        List<Element> transitions = getTransitions(workflowProcess);

        for (Element transition : transitions) {

            if (transition.getAttributeValue("Id").equals(id)) {
                return transition;
            }
        }

        return null;
    }

    /**
     * Retorna uma lista com todas as transições que tem como ponto de
origem
```

```
     * o gateway informado.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param route Elemento que represeta um gateway no arquivo XPDL
     * @return Lista de transições
     */
    private List<Element> getRouteTransitions(Element workflowProcess,
Element route) {

        List<Element> transitions = new ArrayList<Element>();

        List<Element> transitionRefs =
route.getChild("TransitionRestrictions",
ns).getChild("TransitionRestriction", ns).getChild("Split",
ns).getChild("TransitionRefs", ns).getChildren("TransitionRef", ns);

        for (Element transition :  transitionRefs) {

            transitions.add(getTransition(workflowProcess,
transition.getAttributeValue("Id")));
        }

        return transitions;
    }

    /**
     * Retorna o tipo referente a notação BPMN da atividade informada.
     *
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
     * @return Tipo da atividade
     */
    private Type getActivityType(Element activity) {

        if (activity.getChild("Event", ns) != null) {
            return getEventType(activity);
        } else if (activity.getChild("Implementation", ns) != null) {
            return Type.TASK;
        } else if (activity.getChild("Route", ns) != null) {
            return getRouteType(activity);
        }

        return null;
    }
```

```
    /**
     * Retorna o tipo referente a notação BPMN do evento informado.
     *
     * @param activity Elemento que representa um evento no arquivo XPDL
     * @return Tipo do gateway
     */
    private Type getEventType(Element activity) {

        if (activity.getChild("Event", ns).getChild("StartEvent", ns) !=
null) {
            return Type.START;
        } else if (activity.getChild("Event", ns).getChild("EndEvent", ns)
!= null) {
            return Type.END;
        }

        return null;
    }

    /**
     * Retorna o tipo referente a notação BPMN do gateway informado.
     *
     * @param activity Elemento que representa um gateway no arquivo XPDL
     * @return Tipo do gateway
     */
    private Type getRouteType(Element activity) {

        String gatewayType = activity.getChild("Route",
ns).getAttributeValue("GatewayType");

        if (gatewayType.equals("XOR") || gatewayType.equals("Exclusive"))
{
            return Type.XOR;
        } else if (gatewayType.equals("AND") ||
gatewayType.equals("Parallel")) {
            return Type.AND;
        }

        return null;
    }

    /**
     * Verifica se a atividade informada é um Join no processo.
     *
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
     * @return True caso afirmativo, false caso contrário.
     */
    private boolean isJoin(Element activity) {

        Element transitionRestrictions =
activity.getChild("TransitionRestrictions", ns);

        if (transitionRestrictions == null) {
            return false;
        }
```

```
        Element join =
transitionRestrictions.getChild("TransitionRestriction",
ns).getChild("Join", ns);

        if (join == null) {
            return false;
        }

        return true;
    }

    /**
     * Retorna o tipo referente a notação BPMN do join informado.
     *
     * @param activity Elemento que representa um join no arquivo XPDL
     * @return Tipo do join
     */
    private Type getJoinType(Element activity) {

        Element transitionRestrictions =
activity.getChild("TransitionRestrictions", ns);

        if (transitionRestrictions == null) {
            return Type.BLANK;
        }

        Element join =
transitionRestrictions.getChild("TransitionRestriction",
ns).getChild("Join", ns);

        if (join == null) {
            return Type.BLANK;
        }

        String type = join.getAttributeValue("Type");

        if (type.equals("XOR") || type.equals("Exclusive")) {
            return Type.XOR;
        } else if (type.equals("AND") || type.equals("Parallel")) {
            return Type.AND;
        }

        return null;
    }

    /**
     * Retorna o valor do atributo extendido informado da atividade em
questão.
     *
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
     * @param name Nome do atributo extendido que se deseja obter o valor
     * @return Valor do atribute
     */
    private String getExtendedAttributeValue(Element activity, String
name) {

        for (Element extendedAttribute : getExtendedAttributes(activity))
{
```

```
            if
(extendedAttribute.getAttributeValue("Name").equalsIgnoreCase(name)) {
                return extendedAttribute.getAttributeValue("Value");
            }
        }

        return null;
    }

    /**
     * Retorna uma lista com todos os atributos extendidos de uma
atividade,
     * ou uma lista vazia caso não exista nenhum.
     *
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
     * @return Lista de atributos extendidos.
     */
    private List<Element> getExtendedAttributes(Element activity) {

        if (activity.getChild("ExtendedAttributes", ns) == null) {
            return new ArrayList<Element>();
        }

        return activity.getChild("ExtendedAttributes",
ns).getChildren("ExtendedAttribute", ns);
    }

    /**
     * Retorna uma lista com todas as transições do processo informado.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @return Lista de transições
     */
    private List<Element> getTransitions(Element workflowProcess) {
        return workflowProcess.getChild("Transitions",
ns).getChildren("Transition", ns);
    }

    /**
     * Retorna uma lista com todos os processos existentes no arquivo
XPDL.
     *
     * @return Lista de processos
     */
    private List<Element> getWorkflowProcesses() {
        return root.getChild("WorkflowProcesses",
ns).getChildren("WorkflowProcess", ns);
    }

    /**
     * Cria um serviço simples baseado em uma única atividade do processo.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
```

```
     * @param services Lista de serviços na qual o serviço criado deve ser
incluído
     * @return Próxima atividade após o serviço criado
     */
    public Element createSimpleService(Element workflowProcess, Element
activity, List<ServiceDescriptor> services) {

        SimpleServiceDescriptor simpleService = new
SimpleServiceDescriptor();

        simpleService.setId(activity.getAttributeValue("Id"));
        simpleService.setName(activity.getAttributeValue("Name"));
        simpleService.setService(getExtendedAttributeValue(activity,
"Service"));

        services.add(simpleService);

        return getNextActivity(workflowProcess, activity);

    }

    /**
     * Cria um serviço baseado no padrão de workflow XOR. Este processo
atua
     * de forma recursiva realizando todo o agrupamento linear das
atividades
     * presentes em cada transição do gateway como também realiza o
agrupamento
     * paralelo entre todas as transições.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param activity Elemento que representa um gateway XOR no arquivo
XPDL
     * @param services Lista de serviços na qual o serviço criado deve ser
incluído
     * @return Próxima atividade após o serviço criado
     */
    public Element createExclusiveService(Element workflowProcess, Element
activity, List<ServiceDescriptor> services) {

        ExclusiveServiceDescriptor exclusiveService = new
ExclusiveServiceDescriptor("", "ExclusiveServiceCreated" +
++serviceCreatedCounter);

        Element element = null;

        for (Element transition : getRouteTransitions(workflowProcess,
activity)) {

            BundledServicesDescriptor bundledServices = new
BundledServicesDescriptor("", "ServiceCreated" + ++serviceCreatedCounter);

            element = getActivity(workflowProcess,
transition.getAttributeValue("To"));

            while (!isJoin(element) && (getJoinType(element) != Type.XOR))
{
```

```
                element = createService(workflowProcess, element,
bundledServices.getServices());
            }

            exclusiveService.getServices().add(bundledServices);
        }

        services.add(exclusiveService);

        return getNextActivity(workflowProcess, element);
    }

    /**
     * Cria um serviço baseado no padrão de workflow AND. Este processo
atua
     * de forma recursiva realizando todo o agrupamento linear das
atividades
     * presentes em cada transição do gateway como também realiza o
agrupamento
     * paralelo entre todas as transições.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param activity Elemento que representa um gateway AND no arquivo
XPDL
     * @param services Lista de serviços na qual o serviço criado deve ser
incluído
     * @return Próxima atividade após o serviço criado
     */
    public Element createParallelService(Element workflowProcess, Element
activity, List<ServiceDescriptor> services) {

        ParallelServiceDescriptor parallelService = new
ParallelServiceDescriptor("", "ParallelServiceCreated" +
++serviceCreatedCounter);

        Element element = null;

        for (Element transition : getRouteTransitions(workflowProcess,
activity)) {

            BundledServicesDescriptor bundledServices = new
BundledServicesDescriptor("", "ServiceCreated" + ++serviceCreatedCounter);

            element = getActivity(workflowProcess,
transition.getAttributeValue("To"));

            while (!isJoin(element) && (getJoinType(element) != Type.AND))
{
                element = createService(workflowProcess, element,
bundledServices.getServices());
            }

            parallelService.getServices().add(bundledServices);
        }

        services.add(parallelService);

        return getNextActivity(workflowProcess, element);
```

```
    }

    /**
     * Cria um novo serviço baseado em um padrão de workflow identificado
através
     * do tipo da atividade informada.
     *
     * @param workflowProcess Elemento que representa um processo no
arquivo XPDL
     * @param activity Elemento que representa uma atividade no arquivo
XPDL
     * @param services Lista de serviços na qual o serviço criado deve ser
incluído
     * @return Próxima atividade após o serviço criado
     */
    public Element createService(Element workflowProcess, Element
activity, List<ServiceDescriptor> services) {

        Type activityType = getActivityType(activity);

        if (activityType != Type.END) {
            if (activityType == Type.TASK) {
                return createSimpleService(workflowProcess, activity,
services);
            } else if (activityType == Type.XOR) {
                return createExclusiveService(workflowProcess, activity,
services);
            } else if (activityType == Type.AND) {
                return createParallelService(workflowProcess, activity,
services);
            }
        }

        return null;
    }

    /**
     * Para cada processo presente no XPDL identifica e cria os serviços
     * necessários para que seja possível representa-lo como um todo.
     *
     * @return Lista dos serviços já processados e agrupados
     */
    public List<ServiceDescriptor> findServices() {

        List<ServiceDescriptor> processServices = new
ArrayList<ServiceDescriptor>();

        try {

            createDocument();

            List<Element> workflowProcesses = getWorkflowProcesses();

            for (Element workflowProcess : workflowProcesses) {

                System.out.println("Workflow Process Name: " +
workflowProcess.getAttributeValue("Name"));
```

```
                List<ServiceDescriptor> services = new
LinkedList<ServiceDescriptor>();

                /*
                 *
                 * Identifica e agrupa segundo os padrões de workflow
todas as
                 * atividades presentes no XPDL
                 *
                 */

                Element activity = getNextActivity(workflowProcess,
getStartEvent(workflowProcess));

                while (activity != null) {
                    activity = createService(workflowProcess, activity,
services);
                }

                /*
                 *
                 * Após realizar a identificação e o agrupamento na etapa
acima
                 * encontra-se na lista de serviços apenas serviços
simples
                 * que devem ser agrupados de forma linear para
representar
                 * o serviço que representa o processo como um todo.
                 *
                 */

                BundledServicesDescriptor bundledServices = new
BundledServicesDescriptor();

bundledServices.setName(workflowProcess.getAttributeValue("Name"));
                bundledServices.getServices().addAll(services);

                processServices.add(bundledServices);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return processServices;
    }

}
```

BundledServicesDescriptor.java

```
package redbox.descriptor;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class BundledServicesDescriptor extends ServiceDescriptor {

    List<ServiceDescriptor> services;

    public BundledServicesDescriptor() {
        this.type = "Bundled";
        this.services = new LinkedList<ServiceDescriptor>();
    }

    public BundledServicesDescriptor(String id) {
        this();
        this.id = id;
    }

    public BundledServicesDescriptor(String id, String name) {
        this(id);
        this.name = name;
    }

    public List<ServiceDescriptor> getServices() {
        return services;
    }

    public void setServices(List<ServiceDescriptor> services) {
        this.services = services;
    }

}
```

ExclusiveServiceDescriptor.java

```
package redbox.descriptor;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class ExclusiveServiceDescriptor extends ServiceDescriptor {

    List<ServiceDescriptor> services;

    public ExclusiveServiceDescriptor() {
        this.type = "Exclusive";
```

```
        this.services = new LinkedList<ServiceDescriptor>();
    }

    public ExclusiveServiceDescriptor(String id) {
        this();
        this.id = id;
    }

    public ExclusiveServiceDescriptor(String id, String name) {
        this(id);
        this.name = name;
    }

    public List<ServiceDescriptor> getServices() {
        return services;
    }

    public void setServices(List<ServiceDescriptor> services) {
        this.services = services;
    }

}
```

ParallelServiceDescriptor.java

```
package redbox.descriptor;

import java.util.LinkedList;
import java.util.List;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class ParallelServiceDescriptor extends ServiceDescriptor {

    List<ServiceDescriptor> services;

    public ParallelServiceDescriptor() {
        this.type = "Parallel";
        this.services = new LinkedList<ServiceDescriptor>();
    }

    public ParallelServiceDescriptor(String id) {
        this();
        this.id = id;
    }

    public ParallelServiceDescriptor(String id, String name) {
        this(id);
        this.name = name;
    }

    public List<ServiceDescriptor> getServices() {
        return services;
    }

    public void setServices(List<ServiceDescriptor> services) {
```

```
        this.services = services;
    }

}
```

ServiceDescriptor.java

```
package redbox.descriptor;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.WordUtils;
import redbox.util.EscapeChars;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public abstract class ServiceDescriptor {

    protected String id;
    protected String name;
    protected String type;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    public String getKlassName() {
        return
EscapeChars.removeIllegalCharacters(StringUtils.deleteWhitespace(WordUtils
.capitalize(name)));
    }

    public String getObjectName() {
        return WordUtils.uncapitalize(getKlassName());
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

}
```

SimpleServiceDescriptor.java

```
package redbox.descriptor;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class SimpleServiceDescriptor extends ServiceDescriptor {

    private String service;

    public SimpleServiceDescriptor() {
        this.type = "Simple";
    }

    public SimpleServiceDescriptor(String id) {
        this();
        this.id = id;
    }

    public SimpleServiceDescriptor(String id, String name) {
        this(id);
        this.name = name;
    }

    public SimpleServiceDescriptor(String id, String name, String service)
{
        this(id, name);
        this.service = service;
    }

    public String getService() {
        return service;
    }

    public void setService(String service) {
        this.service = service;
    }

}
```

Type.java

```
package redbox.elements;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public enum Type {

    START ("StartEvent"), END ("EndEvent"), JOIN ("Join"), TASK ("Task"),
        ROUTE ("Route"), SPLIT ("Split"), AND ("AND"), XOR ("XOR"),
BLANK("");

    private String type;
```

```
    Type(String type) {
        this.type = type;
    }

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

}
```

EscapeChars.java

```
package redbox.util;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class EscapeChars {

    /**
     * Função auxiliar responsável por subtituir caracteres inválidos para
o
     * nome de uma classe em Java para gerar o nome da classe que
representa um
     * serviço criado.
     *
     * @param s String contendo o nome da classe que representa um serviço
criado
     */
    public static String removeIllegalCharacters(String s) {

        s = s.replaceAll("[èéêë]","e");
        s = s.replaceAll("[ûù]","u");
        s = s.replaceAll("[ïî]","i");
        s = s.replaceAll("[àâã]","a");
        s = s.replaceAll("[ôõ]","o");

        s = s.replaceAll("[ÈÉÊË]","E");
        s = s.replaceAll("[ÛÙ]","U");
        s = s.replaceAll("[ÏÎ]","I");
        s = s.replaceAll("[ÀÂÃ]","A");
        s = s.replaceAll("[ÔÕ]","O");

        s = s.replaceAll("[?!.,;<>{}]", "");

        return s;
    }

}
```

GenericService.java

```
package redbox.util;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public interface GenericService {

    public Object call(Object args);

}
```

ServiceLocator.java

```
package redbox.util;

/**
 *
 * @author Douglas Barbosa Alexandre
 */
public class ServiceLocator {

    public static GenericService getService(String name) {
        // TODO Auto-generated stub
        return null;
    }

}
```

BundledServices.vt

```
## BundledService.vt in redbox.template packpage

package com.${appName}.${bpmPackage};

import redbox.util.GenericService;

/**
 *
 * @author Redbox Service Generator
 */
public class ${service.KlassName} implements GenericService {

    @Override
    public Object call(Object args) {

        Object result = null;
#foreach ( $s in ${service.Services} )

#if ( ${velocityCount} eq 0 )
        ${s.KlassName} ${s.ObjectName} = new ${s.KlassName}();
        result = ${s.ObjectName}.call(args);
#else
        ${s.KlassName} ${s.ObjectName} = new ${s.KlassName}();
```

```
        result = ${s.ObjectName}.call(result);
#end
#end

        return result;

    }

}
```

ExclusiveService.vt

```
## ExclusiveService.vt in redbox.template packpage

package com.${appName}.${bpmPackage};

import redbox.util.GenericService;

/**
 *
 * @author Redbox Service Generator
 */
public class ${service.KlassName} implements GenericService {

    @Override
    public Object call(Object args) {

        Object result = null;

        int branch = testBranch(args);

        switch (branch) {

#foreach ( $s in $service.Services )
            case ${velocityCount}:
                ${s.KlassName} ${s.ObjectName} = new ${s.KlassName}();
                result = ${s.ObjectName}.call(args);
                break;

#end
            default:

        }

        return result;

    }

    private int testBranch(Object args) {
        // TODO Auto-generated stub
        return 0;
    }

}
```

ParallelService.vt

```
## ParallelService.vt in redbox.template packpage

package com.${appName}.${bpmPackage};

import redbox.util.GenericService;

/**
 *
 * @author Redbox Service Generator
 */
public class ${service.KlassName} implements GenericService {

    @Override
    public Object call(Object args) {

        Object result = null;

#foreach ( $s in $service.Services )
        ${s.KlassName} ${s.ObjectName} = new ${s.KlassName}();
        result = ${s.ObjectName}.call(args);

#end
        return result;

    }

}
```

SimpleService.vt

```
## SimpleService.vt in redbox.template packpage

package com.${appName}.${bpmPackage};

import redbox.util.GenericService;
#if ( ${service.Service} )
import redbox.util.ServiceLocator;
#end

/**
 *
 * @author Redbox Service Generator
 */
public class ${service.KlassName} implements GenericService {

    @Override
    public Object call(Object args) {

#if ( !${service.Service} )
        Object result = null;

        // TODO Auto-generated stub

        return result;
#else
```

```
        GenericService service =
ServiceLocator.getService("${service.Service}");
        return service.call(args);
#end

    }

}
```

MappingServices.vt

```
## MappingServices.vt in redbox.template packpage
<?xml version="1.0"?>
<config>
    <!-- Map config file for module: ${bpmPackage} -->
    <map>
        <!-- Processes: -->
#foreach ( $service in $services )
        <link source="${appName}.${bpmPackage}.${service.KlassName}"
target="com.${appName}.${bpmPackage}.${service.KlassName}" />
#end
    </map>
</config>
```