UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA A DECOMPOSIÇÃO DE PROCESSOS DE NEGÓCIO PARA EXECUÇÃO EM NUVENS COMPUTACIONAIS

LUCAS VENEZIAN POVOA

ORIENTADOR: PROF. DR. WANDERLEY LOPES DE SOUZA

São Carlos - SP
Novembro / 2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA A DECOMPOSIÇÃO DE PROCESSOS DE NEGÓCIO PARA EXECUÇÃO EM NUVENS COMPUTACIONAIS

LUCAS VENEZIAN POVOA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores.
Orientador: Dr. Wanderley Lopes de Souza

São Carlos - SP
Novembro / 2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P879ad

Povoa, Lucas Venezian.
Uma abordagem para a decomposição de processos de negócio para execução em nuvens computacionais / Lucas Venezian Povoa. -- São Carlos : UFSCar, 2015.
143 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Ciência da computação. 2. Gerenciamento de processos de negócio. 3. Computação em nuvem. 4. Decomposição de processos de negócio. 5. Modelo baseado em grafos. I. Título.

CDD: 004 (20ª)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA A DECOMPOSIÇÃO DE PROCESSOS DE NEGÓCIO PARA EXECUÇÃO EM NUVENS COMPUTACIONAIS

LUCAS VENEZIAN POVOA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores.

Aprovado em 17 de novembro de 2014.

Membros da Banca:

Prof. Dr. Wanderley Lopes de Souza

(Orientador – DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado

(DC/UFSCar)

Prof. Dr. Luís Ferreira Pires

(Faculty of EEMCS – University of Twente)

Prof. Dr. Ciro de Barros Barbosa

(ICE/UFJF)

São Carlos - SP

Novembro / 2014

Dedico este trabalho ao meu amado filho João Pedro Rossinholi Venezian e a minha amada mulher Ana Carla Rossinholi Pinto que tornam minha vida completamente feliz.

AGRADECIMENTOS

Esta pesquisa não seria possível sem a ajuda de várias pessoas. Em primeiro lugar, agradeço a minha amada e encantadora companheira e amiga Ana Carla Rossinholi Pinto por todo o seu apoio e compreensão. Gostaria de agradecer a minha família materna, em especial minha mãe, por seu esforço sobre humano para me fornecer o bem mais precioso da vida, o amor, e também pelos exemplos que influenciaram e influenciarão a minha vida toda da forma mais positiva possível.

Ao meu orientador Dr. Wanderley Lopes de Souza, agradeço a grande oportunidade de desenvolver o meu mestrado, por me fazer crescer a cada conversa e principalmente por nunca desistir de mim, mesmo quando eu não conseguia determinar se seria possível continuar. Gostaria de agradecer todo suporte dado pelo Dr. Luís Ferreira Pires, que conseguiu pacientemente me ajudar a encontrar soluções para problemas que pareciam insolúveis. Ao Dr. Antonio Francisco do Prado gostaria de agradecer por suas palavras de motivação no início do mestrado, as quais me ajudaram muito.

Gostaria de agradecer a Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP) e ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) por me fornecerem tempo para o desenvolvimento do meu mestrado.

Por fim, gostaria de agradecer a todos que direta ou indiretamente contribuíram para o desenvolvimento desta pesquisa.

“Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.”

- Albert Einstein

RESUMO

Gerenciamento de Processos de Negócio emergiu como um meio para gerenciar e aperfeiçoar processos de negócio continuamente. Entretanto, fornecer tais processos de maneira eficiente pode demandar altos investimentos devido às necessidades de software, hardware e suporte técnico. Computação em Nuvem emergiu como um meio para prover uma forma rápida e barata de adquirir recursos computacionais com pagamento sob demanda, e pode ser aplicada para disponibilizar processos de negócio eficientes e com baixo custo. Entretanto, devido a requisitos de segurança, certos dados ou atividades de um processo de negócio devem ser mantidos na premissa do usuário, enquanto outros podem ser alocados em uma nuvem computacional. Esta dissertação de mestrado apresenta uma abordagem genérica para a decomposição de processos de negócio, que leva em conta custos, desempenho e requisitos de segurança, outorgando assim uma maior confiabilidade no uso de recursos de nuvens computacionais. Essa abordagem é ilustrada através de um estudo de caso no domínio da Saúde.

Palavras-chave: Processos de Negócio, Computação em Nuvem, Grafos, BPMN, WS-BPEL.

ABSTRACT

Business Process Management emerged as a means for managing and improving business processes. However providing efficient business processes can demand high costs due to the need of software, hardware, and technical support. Cloud Computing emerged as a means for providing a fast and cheap way of acquiring computational resources in a pay-per-use manner, and can be employed for achieving efficient business processes with low costs. However, due to safety requirements, certain data or activities of a business process should be kept within the user premises, while others can be allocated to a cloud. This master dissertation presents a generic approach for decomposing business process taking into account costs, performance and safety requirements, thus granting greater reliability in the use of cloud resources. This approach is illustrated by means of a case study in the Healthcare domain.

Keywords: Business Processes, Cloud Computing, Graph, BPMN, WS-BPEL.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta a introdução ao tema desta dissertação de mestrado, suas motivações e objetivos. A Seção 1.1 aborda o contexto desta pesquisa; a Seção 1.2 trata das motivações e objetivos; e a Seção 1.3 descreve a estrutura deste documento.

1.1 Contexto

Atualmente várias organizações dispõem de grandes sistemas computacionais para que a crescente demanda por processamento e armazenamento de um volume cada vez maior de dados seja atendida. Enquanto na indústria grandes companhias constroem centros de dados em larga escala para fornecer serviços Web rápidos e confiáveis, na academia muitos projetos de pesquisa envolvem grandes conjuntos de dados e alto poder de processamento, geralmente providos por supercomputadores. Dessa demanda por grandes centros de dados emergiu o conceito de Computação em Nuvem (ARMBRUST, FOX, *et al.*, 2009), onde Tecnologias da Informação e Comunicação (TICs) são oferecidas como serviços via Internet. Google App Engine, Amazon Elastic Compute Cloud (EC2), Manjrasoft Aneka, UOL Cloud, Rackspace e Microsoft Azure são alguns exemplos de nuvens computacionais (BUYYA, YEO, *et al.*, 2009).

A Computação em Nuvem lida com problemas de escalabilidade e custo, oferecendo um conjunto aparentemente ilimitado de recursos computacionais com tarifação baseada no uso (*pay-per-use*). Embora a Computação em Nuvem esteja tornando a Computação o quinto utilitário (BUYYA, YEO, *et al.*, 2009), similar a água, eletricidade, gás e telefonia, essa área ainda possui problemas de confiança

que podem limitar sua aplicação em certos domínios. A Saúde é um desses, dado que a divulgação de informações deve satisfazer requisitos legais, tais como os presentes no *Health Insurance Portability and Accountability Act* (*HIPAA*) (BANKS, 2006).

A falta de confiança em provedores de nuvem ocorre principalmente porque estes não conseguem garantir os níveis de confidencialidade requeridos, uma vez que normalmente não revelam como os dados são manipulados no seu ambiente de nuvem (YU, WANG, *et al.*, 2010).

Empresas vêm empregando Gerenciamento de Processos de Negócio (*BPM*) (WESKE, 2007) para gerenciar e aperfeiçoar seus processos. Um processo de negócio consiste de atividades exercidas por humanos ou sistemas e um Sistema de Gerenciamento de Processos de Negócio (*BPMS*) dispõe principalmente de um mecanismo (*engine*), no qual instâncias de um processo são coordenadas e monitoradas.

A compra de um BPMS pode ser um alto investimento para uma empresa, devido aos custos envolvidos na aquisição de software e hardware e na contratação de profissionais qualificados. Escalabilidade também pode ser um problema, já que um executor é capaz de coordenar simultaneamente um número limitado de instâncias de um processo. Esse fato implica na necessidade de aquisição de hardware adicional para lidar com situações de pico de carga, além de mais licenças de software para atender expansões no número de servidores.

BPMSs baseados em nuvens computacionais e oferecidos como *Software-as-a-Service* (*SaaS*) por meio da Internet podem ser uma solução para o problema de escalabilidade. Entretanto, o medo de perder ou expor dados confidenciais é um dos maiores obstáculos para a implantação desse tipo de software em nuvens computacionais, além do que há atividades em um processo de negócio que não se beneficiam dessa prática. Por exemplo, uma atividade com pouca demanda computacional pode se tornar mais onerosa se colocada na nuvem. Essa situação ocorre devido à necessidade de transferir para a nuvem os dados a serem processados por essa atividade. Isso demanda mais tempo para a execução dessa atividade, aumentando o seu custo uma vez que a transferência de dados também é um dos fatores de faturamento dos provedores (HAN, SUN, *et al.*, 2010).

O modelo *Process-as-a-Service*, onde um processo de negócio é executado parcial ou totalmente em uma nuvem computacional, foi identificado em

(LINTHICUM, 2010). Devido a requisitos de segurança, nesse modelo há determinados dados ou atividades de um processo são mantidos na premissa do usuário, enquanto outros podem ser alocados numa nuvem, o que requer uma decomposição desse processo.

1.2 Motivações e Objetivos

Nas últimas décadas, aplicações baseadas em workflows têm sido utilizadas com sucesso na solução de problemas científicos e empresariais (DEELMAN, 2010). BPM é considerado uma extensão das abordagens clássicas de workflow, sendo empregado para projetar, promulgar, gerenciar e analisar processos de negócio (VAN DER AALST, HOFSTEDE e WESKE, 2003). *Service-Oriented Architecture* (*SOA*) (PAPAZOGLOU e HEUVEL, 2007) tem facilitado o emprego de BPMSs, ajudando empresas a alcançar seus objetivos de negócio através da orquestração de seus serviços, além de tornar possível a criação rápida de serviços e produtos complexos (SPECTOR, NORVIG e PETROV, 2012).

A combinação de BPM, SOA, Computação em Nuvem e tecnologias tais como a Web 2.0, vem sendo a maior responsável pelo surgimento de inovações no provimento e gerenciamento de serviços envolvendo TICs (JIANG, LE, *et al.*, 2011). Segundo dados do relatório (BRASSCOM, 2012), a previsão da taxa de crescimento anual médio de uso de serviços em Computação em Nuvem no Brasil é de 9,6% até 2015. Outra pesquisa aponta que o total dos investimentos na Europa em serviços de nuvens públicas ultrapassará um trilhão de dólares até 2020. Em um cenário mais concreto, 33% das companhias globais já possuem ou estão planejando a implementação de camadas de serviços na nuvem no modelo SaaS, sendo que mais de 23% das empresas de alto desempenho já possuem esse tipo de serviço em seu ambiente de TICs (DUTTA e BILBAO-OSORIO, 2012).

De acordo com (CURTISS e EUSTIS, 2012), atualmente uma das áreas que mais influenciam o BPM é a Computação em Nuvem. A automatização de processos de negócio através de BPMSs tem criado novas oportunidades de negócio, sendo que BPMSs são fundamentais para disponibilizar soluções inovadoras e eficientes. Esses fatos motivaram vários fabricantes de BPMSs a estender as funcionalidades

de seus produtos para dar suporte à Computação em Nuvem. Os softwares IBM WebSphere e Oracle WebLogic (IBM SOFTWARE, 2011), softwares proprietários, e jBPM (JBOSS COMMUNITY, 2014), software livre, são alguns exemplos. Além disso, há uma previsão que o mercado de BPM, atualmente no valor de 2,6 bilhões de dólares, atingirá 7 bilhões de dólares em 2018 (CURTISS e EUSTIS, 2012).

Resultados obtidos via uma pesquisa efetuada em três grandes bases de dados de bibliografias internacionais, apontam que o número de publicações científicas envolvendo conjuntamente as áreas de Computação em Nuvem e BPM cresceu significativamente entre 2009 e 2012. A Figura 1.1 ilustra a relação entre o número de publicações e os anos das mesmas, sendo que a taxa média de crescimento anual é de 177,14% no período estudado.

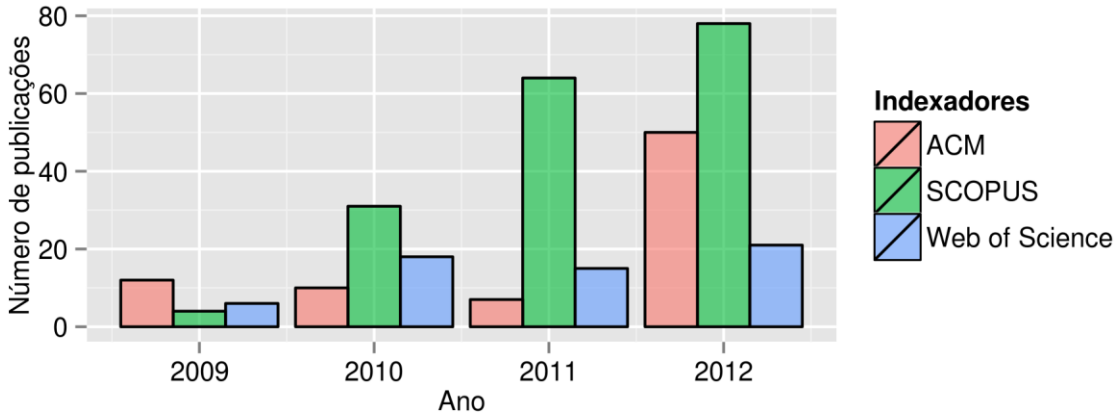


Figura 1.1 – Número de publicações sobre BPM e Computação em Nuvem.

Neste sentido, o objetivo deste trabalho foi propor uma abordagem para a decomposição de processos de negócio para execução em nuvens computacionais, considerando restrições de segurança e visando aumento de desempenho e menores custos dos provedores de nuvem. Essa abordagem foi inspirada nos resultados apresentados em (DUIPMANS, PIRES e SANTOS, 2013) e baseada na arquitetura proposta em (HAN, SUN, *et al.*, 2010) e estendida em (DUIPMANS, PIRES e SANTOS, 2012). Para o desenvolvimento dessa abordagem foram consideradas as seguintes questões de pesquisa:

QP1: Como modelar processos de negócio independentemente da linguagem utilizada para a especificação dos mesmos?

QP2: Como decompor um processo de negócio para executar parte na nuvem e parte na premissa do usuário considerando restrições de segurança, desempenho e custo?

Para responder a essas questões de pesquisa, a metodologia empregada constituiu-se dos seguintes passos:

1. Efetuar levantamento bibliográfico sobre abordagens de descentralização e decomposição de processos de negócio;
2. Definir um modelo intermediário, independente de linguagens de especificação, para representar processos de negócio;
3. Definir um mapeamento e as transformações entre o modelo intermediário e linguagens de especificação de processos de negócio;
4. Definir regras de decomposição baseadas no modelo intermediário;
5. Definir um framework para selecionar a localização, na nuvem ou na premissa do usuário, de cada atividade e dos dados associados, levando em consideração restrições de dados, desempenho e custos;
6. Definir formalmente a sintaxe e semântica do modelo intermediário;
7. Desenvolver um protótipo orientado a objetos da abordagem proposta; e
8. Realizar um estudo de caso no domínio da Saúde empregando o protótipo desenvolvido.

1.3 Estrutura da Dissertação

O restante desta dissertação está organizado em sete capítulos:

- **Capítulo 2** apresenta os principais conceitos e definições inerentes a BPM e Computação em Nuvem;
- **Capítulo 3** discorre sobre a abordagem proposta, detalhando suas etapas e a relação entre as mesmas. São descritos o modelo intermediário, as regras de decomposição, o framework para seleção de localização de atividades e os dados associados. As transformações entre o modelo intermediário e as

linguagens de especificação de processos de negócio *Business Process Model and Notation* (*BPMN*) e *Web Service Business Process Execution Language* (*WS-BPEL*) também são descritas nesse capítulo;

- **Capítulo 4** trata da definição formal da estrutura e da semântica do modelo intermediário;
- **Capítulo 5** lida com a implementação da abordagem proposta. Neste capítulo a arquitetura e os meta-modelos orientados a objeto são descritos;
- **Capítulo 6** apresenta um estudo de caso. Neste capítulo a abordagem proposta é aplicada a um processo de negócio na Saúde, onde são ilustradas todas as etapas da abordagem e também é comparado o desempenho do processo monolítico e sua forma decomposta;
- **Capítulo 7** discute e compara trabalhos correlatos encontrados na literatura; e
- **Capítulo 8** tece as considerações finais e aponta para alguns trabalhos futuros.

Capítulo 2

CONHECIMENTOS PRELIMINARES

Para o desenvolvimento deste trabalho, foi fundamental a aquisição de conhecimentos relativos às áreas BPM e Computação em Nuvem, os quais são descritos respectivamente na Seção 2.1 e Seção 2.2 deste capítulo.

2.1 Gerenciamento de Processos de Negócio

Workflow é definido como a automação total ou parcial de processos de negócio, durante o qual documentos, informações ou tarefas são transmitidos de um participante a outro para a execução de uma determinada ação de acordo com um conjunto de regras procedurais (VAN DER AALST, HOFSTEDE e WESKE, 2003).

Segundo (KO, 2009) um processo de negócio pode ser definido como um conjunto ou rede de atividades com certo valor agregado, as quais são executadas por papeis específicos ou colaboradores do processo, com a finalidade de atingir um objetivo de negócio comum. Um processo de negócio pode ser privado, quando acessível somente no âmbito da empresa ou organização que o detém, ou colaborativo, também denominado público, quando há a participação de colaboradores externos.

Processos de negócio podem também ser classificados em relação à perspectiva do seu nível de aplicação e em relação as suas principais competências. No primeiro caso a classificação é baseada na funcionalidade fundamental do processo de negócio (KO, 2009), ou seja:

- **Controle Operacional**, cuja finalidade é assegurar uma execução eficiente das tarefas e atividades de uma empresa ou organização;
- **Controle Administrativo**, cuja finalidade é garantir a obtenção, alocação e uso racional de recursos por parte dos administradores, no acompanhamento dos objetivos da empresa ou organização; ou
- **Planejamento Estratégico**, cuja finalidade é definir os objetivos, as modificações desses, os recursos necessários para atingi-los e as políticas de aquisição e utilização de tais recursos.

No segundo caso, a classificação é baseada nas principais competências do processo de negócio, ou seja:

- **Processos de Negócio Núcleo**, cuja finalidade é gerar receita para a empresa que o detém (e.g., processos de negócio de desenvolvimento de software de uma empresa de TICs);
- **Processos de Negócio Administrativos**, cuja finalidade é assegurar eficiência, conformidade corporativa e governança (e.g., processos de requisição e notificações); e
- **Processo de Negócio de Suporte**, cuja finalidade é dar suporte aos processos inerentes às áreas geradoras de receita e que exercem um papel fundamental no cumprimento dos objetivos de negócio (e.g., processos de TICs de uma empresa de varejo).

BPM pode ser definido como um suporte a processos de negócio através de métodos, técnicas e softwares para projetar, promulgar, controlar e analisar processos operacionais envolvendo humanos, organizações, aplicações, documentos e outras fontes de informação (KO, 2009).

A adoção de BPM pode trazer os seguintes benefícios: aumento da visibilidade e conhecimento das atividades da empresa; melhoria na habilidade de identificação de gargalos nos processos de negócios; identificação mais eficiente das áreas com potenciais melhorias; redução do tempo de espera nos processos de negócio; melhor identificação dos deveres e papeis na empresa; e aumento da habilidade de prevenção de fraudes, auditoria e avaliação do cumprimento das regras e políticas do negócio (KO, 2009).

2.1.1 Ciclo de Vida do Gerenciamento de Processos de Negócio

BPM é uma área multidisciplinar, que lida com várias áreas do conhecimento, tais como Gerenciamento Organizacional, Linguística, Ciência da Computação e Matemática (KO, 2009), acarretando em diversas visões e definições para BPM. Segundo (VAN DER AALST, HOFSTEDE e WESKE, 2003) o ciclo de vida do BPM, possui quatro fases, as quais constituem o conjunto o qual o ciclo de vida do Gerenciamento de Workflow *(WfM)* está inserido. A Figura 2.1 ilustra as interações entre essas fases:

- **Projeto:** fase na qual o processo de negócio é modelado ou remodelado via uma ferramenta de modelagem, que é geralmente integrada a um BPMS;
- **Configuração:** fase na qual o processo de negócio é configurado para um mecanismo de processo, geralmente também integrado a um BPMS, onde é realizada a sincronização dos papéis envolvidos e dos organogramas da empresa com o processo;
- **Promulgação:** fase na qual o processo configurado é implantado no mecanismo de processo e instanciado. Tais instâncias são gerenciadas e acompanhadas via um monitor, o qual fornece um quadro das instâncias em execução e das que terminaram com sucesso, falha ou erro, auxiliando na detecção de eventuais problemas que podem ocorrer com essas instâncias; e
- **Diagnóstico:** fase na qual são averiguados possíveis pontos de gargalo e potenciais brechas para fraudes no processo de negócio. Geralmente essa fase possui o suporte de um componente do BPMS, que contém recursos para análise e verificação de processos de negócio.

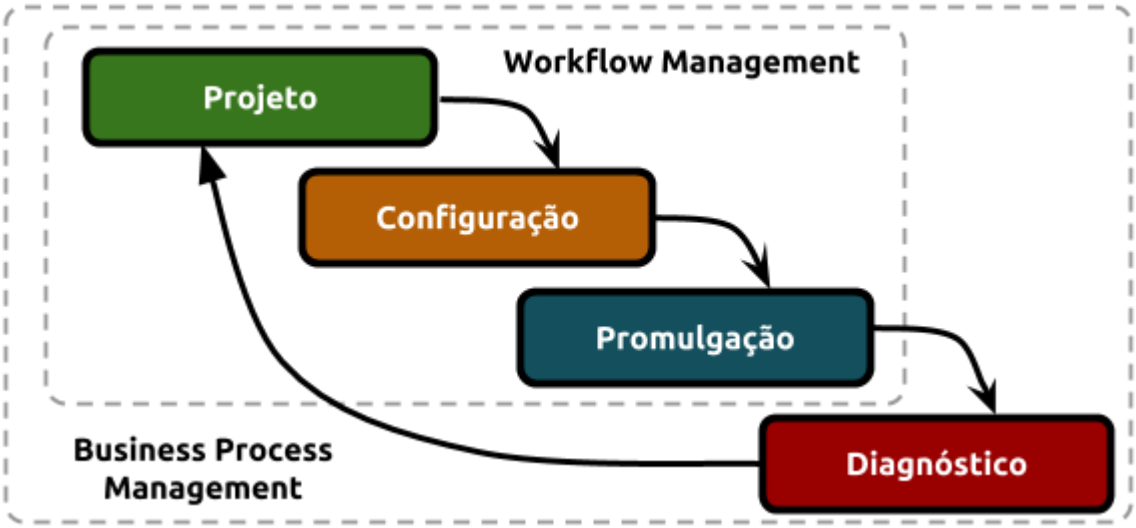


Figura 2.1 – Ciclo de vida do BPM comparado com os recursos do WfM (VAN DER AALST, HOFSTEDE e WESKE, 2003).

Segundo um relatório da Gartner (HILL, 2006 apud KO, 2009, p. 14), WfM é uma tecnologia de fluxo, a qual é encontrada em BPMSs, diferente do BPM definido como uma área para gerenciamento de processos de negócio que não se preocupa somente com o fluxo dos processos, mas com todas as fases do seu ciclo de vida. Reengenharia de Processos de Negócio (*BPR*) é outro termo usado nessa área, empregado quando uma completa reestruturação do processo em foco é realizada, diferentemente de BPM empregado quando uma atividade iterativa é realizada para produzir modificações suaves no processo em foco.

2.1.2 Linguagens de Processos de Negócio

Diversas organizações internacionais buscam definir padrões de suporte ao BPM, em particular linguagens para modelagem e execução de processos de negócio. O Quadro 2.1 apresenta algumas dessas organizações acompanhadas de alguns padrões já estabelecidos pelas mesmas.

Duas das principais linguagens de processos de negócio são BPMN e WS-BPEL (OASIS STANDARD, 2007), sendo a primeira orientada a grafos e a segunda uma linguagem estruturada em blocos (MAZANEK e HANUS, 2011).

Quadro 2.1 - Padrões para BPM (JIANG, LE, *et al.*, 2011).

Organização	Padrões para BPM
BPMI	BPMN 1.0, BPML, BPQL
OMG	UML, BPMN 2.0, BPMI, BPDM, BSBR, OSM
WfMG	Workflow Reference Model, XPDL, Wf-XML, WAPI
OASIS	ebXML, WS-BPEL 2.0
W3C	WS-CDL, WSDL

2.1.3 Sistemas de Gerenciamento de Processos de Negócio

BPMSs têm a finalidade de dar suporte a todo ciclo de vida do BPM e, segundo (WESKE, 2007), a arquitetura de um BPMS possui cinco componentes essenciais (

Figura 2.2):

- **Ferramentas de Modelagem** é o módulo composto por ferramentas usadas para criar modelos que contenham informações sobre atividades, suas operações e estruturas do processo de negócio;
- **Repositório de Modelos de Processos de Negócio** é responsável por armazenar os modelos gerados pelas ferramentas de modelagem de processo de negócio;
- **Ambiente do Processo de Negócio** é o componente responsável por disparar instâncias dos processos de negócio com base em seus modelos armazenados no repositório de modelos;
- **Mecanismo de Processo de Negócio** é responsável por controlar as instâncias disparadas e acessar os parceiros externos com a finalidade de executar as atividades do processo; e
- **Provedores de Serviço** são os detentores das atividades de um processo, as quais podem ser serviços automatizados ou humanos.

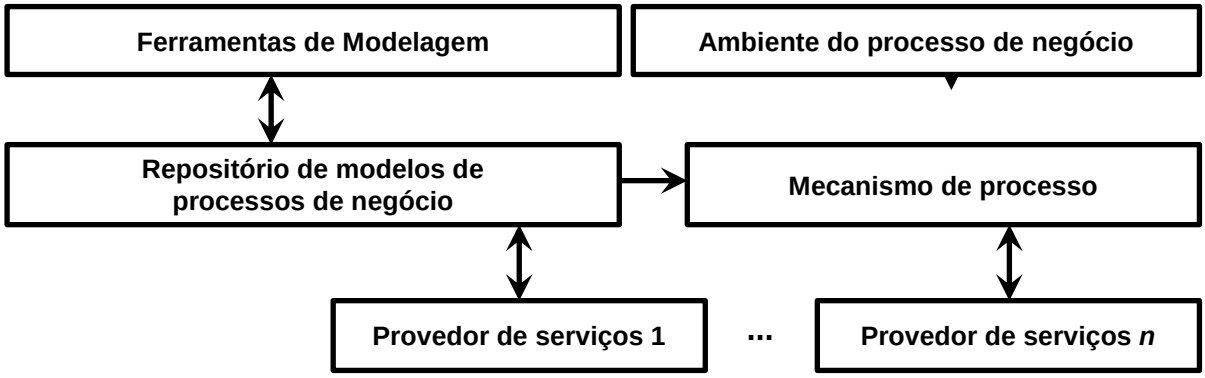


Figura 2.2 – Arquitetura abstrata de um BPMS (WESKE, 2007, p. 120).

2.2 Computação em Nuvem

Computação em Nuvem pode ser definida como uma área que estuda a forma de prover, a um público com interesses heterogêneos, um conjunto de recursos virtualizados, tais como software, hardware e plataformas de implantação. Tais recursos podem ser dinamicamente alocados e configurados em função da variação da carga de processamento e armazenamento, buscando assim uma melhor utilização dos recursos disponíveis. O modelo de tarifação empregado é geralmente o *pay-per-use*, onde os custos são determinados sob a demanda de uso dos recursos contratados. Nesse modelo as garantias aos clientes são dadas pelos provedores de nuvem via *Service Level Agreements (SLAs)* personalizados. (VAQUERO, RODERO-MERINO e CACERES, 2009).

De acordo com (FURHT, 2010) as eras da Computação podem ser classificadas de acordo com seis paradigmas (Figura 2.3): Mainframes com seus terminais burros; Computação Local com seus computadores pessoais; Computação em Rede com seus desktops, laptops e servidores conectados via rede local; Computação sobre a Internet com a interconexão de diversas redes locais visando ao uso remoto de recursos e aplicações; Computação em Grade com seu poder computacional e armazenamento compartilhados via computação distribuída; e Computação em Nuvem.

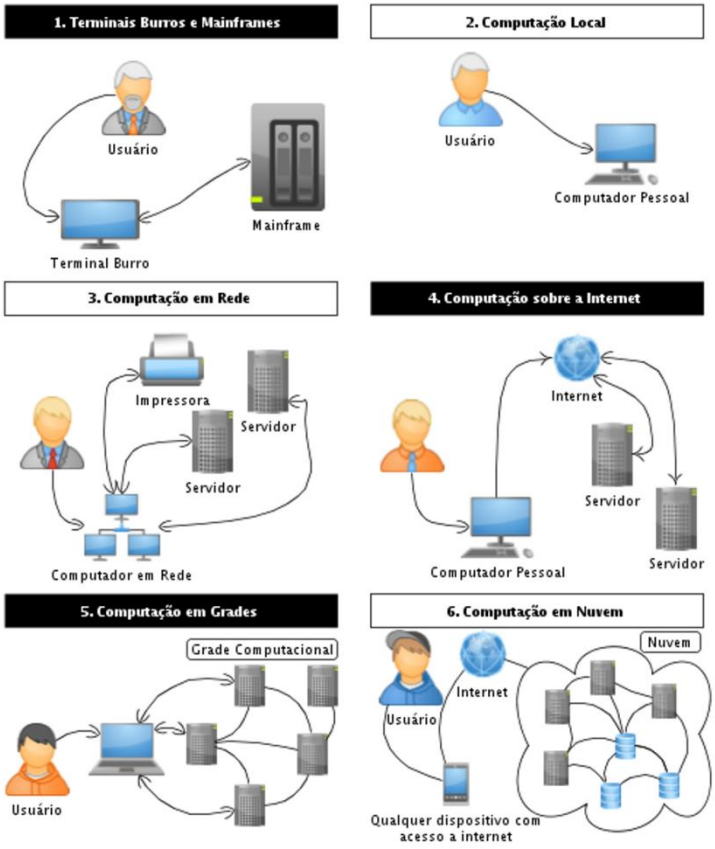


Figura 2.3 – Os seis paradigmas computacionais.

A Computação em Nuvem fornece um novo e excitante modelo de negócio para o mundo da tecnologia, sendo que várias empresas globais investem freneticamente nesse novo paradigma. Em (MARSTON, LI, *et al.*, 2011) é citado o seguinte comentário de um diretor executivo de um provedor de nuvem: "uma empresa na área de Saúde investir dois bilhões de dólares por ano em TICs é tão insano quanto uma grande empresa de varejo online decidir quebrar todos os contratos com transportadoras e correios e decidir comprar frotas de caminhões e alocar aviões para efetuar suas próprias entregas".

Essa nova área também possibilita às empresas terceirar parte da sua infraestrutura de TIC, o que viabiliza o emprego dos seus recursos financeiros remanescentes nos seus processos primários.

2.2.1 Modelos de Serviço

Em (MELL e GRANCE, 2011) são identificados três principais modelos de serviço na Computação em Nuvem (Figura 2.4):

- *Software-as-a-Service (SaaS)*, onde um software hospedado numa infraestrutura robusta é oferecido e usuários acessam-no via interfaces, tais como navegadores Web (e.g., Facebook, Gmail, Dropbox);
- *Platform-as-a-Service* (*PaaS*) provê uma plataforma computacional onde usuários implantam suas aplicações e esta oferece recursos, tais como servidores Web, banco de dados, balanceamento e Map Reduce (e.g., Windows Azure e Google AppEngine); e
- *Infrastructure-as-a-Service* (*IaaS*), onde máquinas virtuais, com certa capacidade de armazenamento, processamento e banda, são oferecidas aos usuários (e.g., UOL Cloud, Amazon EC2 e Google Compute Engine, Rackspace).

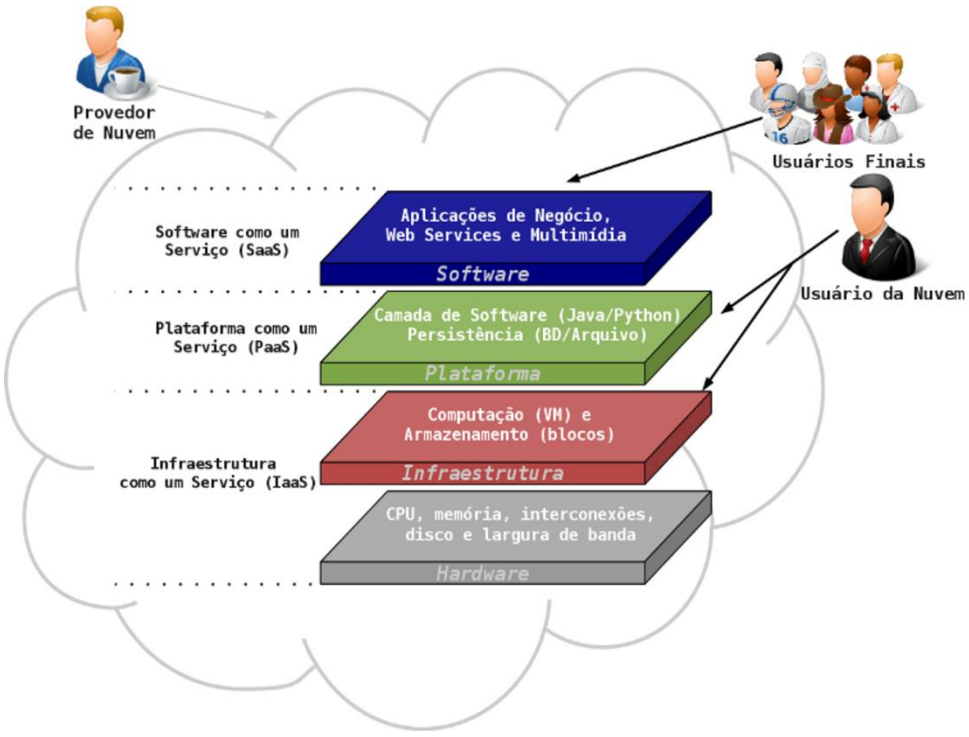


Figura 2.4 – Modelos de serviço, arquitetura e papeis da Computação em Nuvem.

2.2.2 Tipos de Nuvem

Há diversos fatores que podem influenciar na decisão de mover uma aplicação ou parte desta para uma nuvem, dentre os quais se destacam: o custo de uma infraestrutura de hardware; o custo da mão de obra para a manutenção dessa

infraestrutura; poucos recursos computacionais disponíveis; e o custo computacional da aplicação. Tais fatores influenciam também na escolha do tipo de nuvem que melhor atende às necessidades de um usuário específico. De acordo com (ZHANG; CHENG; JIN et al., 2010), os quatro principais tipos de nuvem são:

- **Nuvem pública**, também conhecida como nuvem externa, é caracterizada pelo provimento de recursos computacionais a todo e qualquer usuário e geralmente emprega o modelo de tarifação *pay-per-use*. Dentre os benefícios destacam-se: investimento inicial em infraestrutura desnecessário; transferência de risco aos provedores de infraestrutura; e garantia mínima de conectividade. O principal problema inerente a esse tipo de nuvem é a falta de garantia de sigilo das informações;
- **Nuvem privada**, também conhecida como nuvem interna, é usada por empresas ou organizações que desejam um maior controle sobre os recursos implantados na nuvem. A infraestrutura desse tipo de nuvem pode ser construída e administrada por terceiros ou pela própria empresa ou organização que necessita da nuvem. Dentre os benefícios destaca-se a garantia de sigilo sobre os dados e/ou processos. O principal problema é o custo elevado, quando comparado ao da nuvem pública, requerendo contratos específicos para os serviços prestados;
- **Nuvem comunitária** é composta pelas infraestruturas de empresas e/ou organizações que possuem objetivos em comum e que desejam compartilhar tais recursos a fim de aumentar a escalabilidade e dividir custos. Esse tipo de nuvem pode ser mantido por terceiros ou pelos seus próprios interessados, sendo que num âmbito mais específico pode tornar-se uma *Virtual Private Cloud (VPC)*. Essa especificidade é atendida via uma plataforma sobre várias nuvens, geralmente públicas, empregando-se tecnologias de *Virtual Private Network* (*VPN*). Seu objetivo é possibilitar aos provedores de serviço projetar suas próprias topologias e definir seus próprios requisitos de segurança. Uma VPC também proporciona uma fácil migração de uma infraestrutura de serviço proprietária para uma infraestrutura de serviço em nuvem; e
- **Nuvem híbrida** é a combinação de dois ou mais tipos de nuvem (privada, pública e/ou comunitária). O desfio inerente a esse tipo de nuvem é

determinar a distribuição de aplicações e dados entre as partes da arquitetura. Numa nuvem híbrida é possível e conveniente manter aplicações e dados sigilosos na parte privada da nuvem, usando restrições específicas para protegê-los, e recursos menos críticos na parte pública da nuvem.

2.2.3 Principais Vantagens e Desvantagens da Computação em Nuvem

Segundo (ZISSIS e LEKKAS, 2012), as principais vantagens da Computação em Nuvem são:

- **Elasticidade**: num ambiente de nuvem alguns recursos (e.g., número de processadores, quantidade de memória, capacidade de armazenamento, banda) podem ser facilmente adicionados ou removidos, tornando fácil a redução de custos e/ou a melhoria dos serviços prestados em função da demanda. Em alguns casos, os provedores fornecem serviços que tornam a elasticidade de seus recursos automática;
- **Escalabilidade da Infraestrutura**: numa arquitetura construída em ambiente de nuvem podem ser facilmente acrescidos ou removidos servidores (nós) através de poucas modificações no software e na infraestrutura;
- **Acesso a uma Ampla Rede**: a Computação em Nuvem provê recursos via redes de computadores, os quais são acessados através de mecanismos padronizados, tornando viável o uso de plataformas heterogêneas tanto em relação ao software (e.g., diferentes sistemas operacionais) quanto ao hardware (e.g., telefones, notebooks e arquiteturas embarcadas);
- **Independência de Localização**: a localização dos recursos computacionais, num ambiente de nuvem, não é controlada ou conhecida pelos seus usuários, sendo esta definida, se existir essa opção, via abstrações descritas através de dados tais como país, unidade federativa ou datacenter específico;
- **Redundância dos Dados**: um ambiente de nuvem conta com sistemas sofisticados de redundância e estratégias de recuperação, tornando seus recursos confiáveis às aplicações que necessitam recuperar de eventuais desastres ou garantias de que seus dados não serão facilmente perdidos; e

- **Sustentabilidade e Baixo Custo**: os datacenters de grandes arquiteturas de nuvem tendem a serem estabelecidos em ambientes propícios à redução de gastos, geralmente em locais de baixo custo energético e com recursos naturais que auxiliam no resfriamento dos equipamentos. Essa redução de gastos também é refletida nos preços dos serviços prestados, tornando todo o ciclo de vida de serviços de TICs mais econômico e sustentável.

Apesar de todo o entusiasmo com a Computação em Nuvem, esta possui algumas limitações que podem torná-la inaceitável em alguns contextos, ou impedir a sua rápida expansão. Como os serviços oferecidos pelas nuvens computacionais são acessados via Internet, problemas inerentes à rede global podem afetar esses serviços. Segundo (LIU, 2012), dois desses maiores problemas são:

- **Segurança**: vulnerabilidade, vírus e ataques em larga escala podem causar enormes prejuízos aos usuários, sendo que a arbitrariedade dos recursos alocados nas nuvens é a principal causa desse problema, o qual é de difícil solução já que uma política de segurança genérica será menos efetiva do que políticas de segurança específicos; e
- **Sigilo**: a falta de confidencialidade e integridade também podem causar enormes prejuízos, sendo que a primeira refere-se à garantia de que somente usuários autorizados podem acessar dados sigilosos, enquanto a segunda à garantia de que somente usuários autorizados podem alterar tais dados. Já que provedores de nuvem geralmente ocultam os detalhes de implementação dos serviços, as tecnologias empregadas e a forma de gerenciamento dos recursos torna-se impraticável prever um determinado comportamento ou garantir qualquer sigilo de dados e processos hospedados na nuvem.

Além desses problemas, a heterogeneidade existente nos provedores de nuvens tem gerado novos desafios que estão sendo investigados. Como consequência, novas arquiteturas (BENSON, ANAND, et al., 2011; MOSHREF, YU, et al., 2012) e protocolos de comunicação (MINERAUD, BALASUBRAMANIAM, *et al.*, 2012) para datacenters de nuvens têm sido propostos na literatura. Há também pesquisas relativas aos dados, a serem fornecidos pelos usuários em função do tipo de aplicação a ser alocada na nuvem, para que arquiteturas mais estáveis possam ser providas (BALLANI, COSTA, *et al.*, 2011). Todos esses estudos visam ao

desenvolvimento de novas abordagens e tecnologias para melhorar a vazão dos provedores de nuvens e garantir a qualidade de serviço (*QoS*) aos seus clientes.

Capítulo 3

ABORDAGEM

Este capítulo descreve a abordagem desenvolvida neste trabalho. A Seção 3.1 fornece uma visão geral da abordagem; a Seção 3.2 trata do modelo intermediário empregado para a decomposição de um processo de negócio; a Seção 3.3 apresenta o framework para a seleção de localização das atividades e dos dados associados do processo de negócio decomposto; a Seção 3.4 aborda as regras e algoritmos para a decomposição do processo de negócio; e a Seção 3.5 discorre sobre os algoritmos de mapeamento das linguagens BPMN e WS-BPEL para o modelo intermediário e vice-versa.

3.1 Visão Geral

A abordagem proposta tem a finalidade de decompor processos de negócio monolíticos em subprocessos colaborativos a serem distribuídos na nuvem ou na premissa do usuário[1], com a finalidade de aumentar o desempenho de tais processos.

A distribuição de atividades e dados de processos de negócio, entre a premissa e ambientes de BPM baseados em nuvens computacionais, foi investigada inicialmente em (HAN, SUN, *et al.*, 2010), onde o modelo denominado *Processes Enactment Engine, Activities, and Data Storage (PAD)* foi definido. Tal modelo considera a distribuição de atividades e dados de um processo de negócio, mas ignora a distribuição do mecanismo de processo. O PAD foi estendido em (DUIPMANS, PIRES e SANTOS, 2012) para incluir a distribuição do mecanismo de

[1] A partir deste ponto a palavra premissa irá se referir ao termo premissa do usuário.

processo, a fim de que a comunicação entre os subprocessos pudesse ser reduzida. A Figura 3.1 ilustra o PAD estendido.

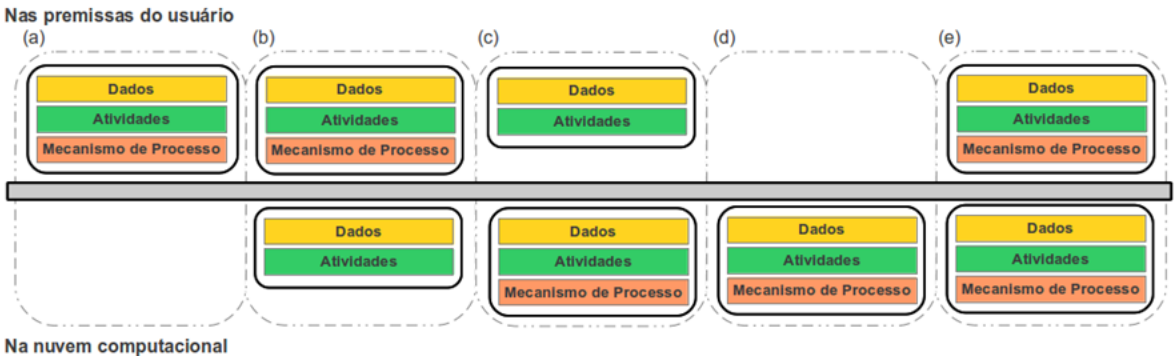


Figura 3.1 – Possibilidades de distribuição de um processo de negócio entre a premissa e a nuvem (DUIPMANS; PIRES; SANTOS, 2012; HAN et al., 2010).

Para a definição da abordagem foi considerada uma arquitetura composta por dois mecanismos de processo para a execução dos subprocessos colaborativos, como definido no PAD estendido. Essa solução visa garantir melhor desempenho na execução do processo de negócio decomposto, já que há uma redução na troca de mensagens entre a premissa e a nuvem em alguns casos.

Considere um processo de negócio no qual a saída de uma atividade é a entrada da próxima. A Figura 3.2(a) ilustra um cenário onde um processo de negócio é executado por um único mecanismo de processo alocado na premissa, sendo que algumas de suas atividades estão alocadas na nuvem. Uma vez que a coordenação do processo é feita pelo mecanismo de processo, as atividades não trocam dados diretamente, mas sim enviam-nos primeiro para o mecanismo de processo o qual os repassam para a próxima atividade. Dessa forma, o cenário descrito lida com uma troca de dados excessiva entre o mecanismo de processo na premissa e as atividades na nuvem.

A Figura 3.2(b) ilustra um cenário com um processo de negócio decomposto executado com dois mecanismos de processo, um na nuvem e outro na premissa. Nesse caso o problema descrito anteriormente é evitado já que as atividades adjacentes alocadas na nuvem não precisam enviar seus dados para a premissa uma vez que a coordenação é feita pelo mecanismo de processo na própria nuvem.

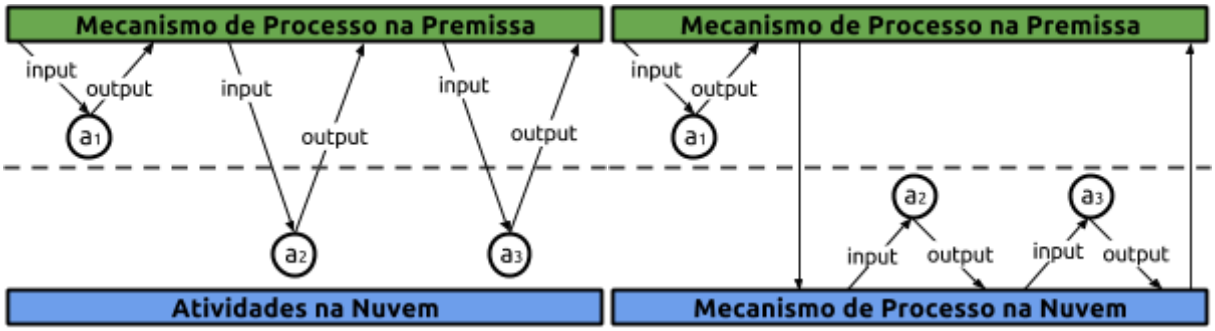


Figura 3.2 – Mecanismo de processo único na premissa (a) e distribuído (b).

A abordagem proposta também tem a finalidade de não ser uma solução direcionada a uma determinada linguagem de especificação de processos de negócio. Portanto, foi decidido empregar um modelo intermediário baseado em padrões de workflow e comunicação para a captura da estrutura e semântica de processos de negócio. Tal solução traz duas principais vantagens: (*i*) a abordagem pode ser facilmente estendida para outras linguagens de especificação de processos de negócio; e (*ii*) os algoritmos de decomposição são reaproveitados para todas as estruturas e semânticas capturadas pelo modelo intermediário.

A abordagem também deve selecionar a localização das atividades e dados associados visando baixo custo financeiro adicional por conta da contratação da nuvem e aumento relevante de desempenho do processo de negócio. Além disso, deve levar em consideração restrições de distribuição de dados, a fim de garantir que dados sigilosos não sejam expostos em ambientes não controlados.

Para atender todos esses objetivos a abordagem proposta consiste de uma cadeia de transformações composta por quatro etapas, como ilustrado na Figura 3.3:

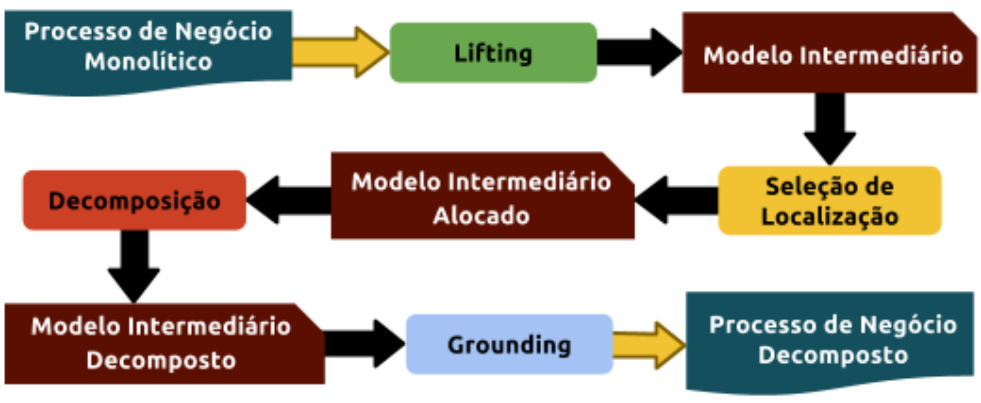


Figura 3.3 – Etapas da abordagem proposta.

1. *Lifting*, onde um processo de negócio monolítico, descrito numa linguagem de especificação de processos de negócio, é transformado para um modelo intermediário considerando o fluxo de dados entre as atividades;
2. *Seleção de localização*, onde a distribuição ótima das atividades de um processo de negócio, representado no modelo intermediário, é calculada e a localização, na premissa ou nuvem, de cada atividade e dados associados é definida;
3. *Decomposição*, onde o processo de negócio, representado no modelo intermediário, é decomposto com base nos resultados obtidos na etapa precedente e se aplicando um conjunto de regras de decomposição; e
4. *Grounding*, onde os subprocessos resultantes, representados no modelo intermediário, são transformados na linguagem de especificação de processos de negócio.

3.2 Modelo Intermediário

Nesta seção são discutidos os padrões de workflow e comunicação requeridos para o modelo intermediário usado na decomposição de processos de negócio. A Seção 3.2.1 descreve tais padrões, a Seção 3.2.2 apresenta outros modelos usados para representar processos de negócio e justifica a definição de um novo modelo e a Seção 3.2.3 apresenta o modelo intermediário propriamente dito.

3.2.1 Requisitos

Os requisitos do modelo intermediário são definidos em termos de padrões de fluxo de controle (RUSSELL, HOFSTEDE, *et al.*, 2006), fluxo de dados (RUSSELL, TER HOFSTEDE, *et al.*, 2005), tratamento de exceções (RUSSELL, VAN DER AALST e TER HOFSTEDE, 2006) e de comunicação (RUH, MAGINNIS e BROWN, 2001). Um padrão de workflow refere-se a uma capacidade específica independente de linguagens ou modelos e recorrente na modelagem de processos de negócio, ao

passo que um padrão de comunicação refere-se às estratégias para a troca de mensagens entre qualquer tipo de sistema, incluindo processos de negócio.

Para que um modelo intermediário seja expressivo e não se torne demasiadamente complexo e pouco eficiente deve existir um balanceamento do número de padrões de workflow e comunicação cobertos por seu escopo. Portanto, foram selecionados dezoito Padrões de Fluxo de Controle de Workflow (WCP), nove Padrões de Fluxo de Dados de Workflow (WDP), seis Padrões de Tratamento de Exceção de Workflow (WEP) e dois Padrões de Comunicação (CP).

A seleção de tais padrões tornou o modelo intermediário apto a capturar a maioria das construções das principais linguagens de especificação de processos de negócio, tais como *WS-BPEL 2.0* (OASIS STANDARD, 2007), *BPMN 2.0* (OBJECT MANAGEMENT GROUP, 2011), *Yet Another Workflow Language (YAWL)* (VAN DER AALST e TER HOFSTEDE, 2005), *Web Services Choreography Description Language (WS-CDL)* (W3C, 2005) e *Architectural Modelling Box for Enterprise Redesign (AMBER)* (EERTINK, JANSSEN, *et al.*, 1999).

A seleção dos WCPs e WEPs teve como finalidade cobrir as construções comuns entre as linguagens BPMN 2.0 e WS-BPEL 2.0. Os WDPs foram selecionados visando que o modelo intermediário fosse capaz de representar a troca explicíta de dados entre atividades e subprocessos. Essa característica possibilita considerar restrições de dados no processo de decomposição e também é avaliada como suficiente para um modelo intermediário expressivo (DUIPAMANS, 2012). Finalmente, os CPs foram selecionados visando que o modelo intermediário fosse capaz de representar mecanismos de comunicação para descrever a invoção de um processo a outro.

3.2.1.1 Padrões de Fluxo de Controle

Os padrões de fluxo de controle, no total de quarenta e três, estão divididos em seis categorias: (a) padrões de fluxo de controle básicos; (b) padrões avançados de ramificação e sincronização; (c) padrões estruturais; (d) padrões de instâncias múltiplas; (e) padrões baseados em estado; (f) padrões de cancelamento.

O modelo intermediário deve suportar todos os padrões de fluxo de controle básicos que se referem aos seguintes aspectos para a modelagem de processos de negócio:

- *Sequence (WCP1)* – execução sequencial de atividades;
- *Parallel Split (WCP2)* – divisão de um fluxo de controle em dois ou mais fluxos a serem executados em paralelo;
- *Synchronization (WCP3)* – junção de fluxos de controle paralelos em um único fluxo após a finalização de todos;
- *Exclusive Choice (WCP4)* – divisão de um fluxo de controle em dois ou mais fluxos, seguida da execução exclusiva de um destes; e
- *Simple Merge (WCP5)* – junção de múltiplos fluxos de controle alternativos em um único fluxo, seguida da execução deste.

Os padrões avançados de ramificação e sincronização são responsáveis pela junção e divisão de fluxos de controle, sendo que o modelo intermediário deve suportar três desses padrões:

- *Multi-choice (WCP6)* – divisão de um fluxo de controle em dois ou mais fluxos, seguida da execução de um ou mais destes;
- *Structured Synchronization Merge (WCP7)* – junção de um ou mais fluxos de controle em um único fluxo, seguida da execução deste;
- *Structured Discriminator (WCP9)* – junção de dois ou mais fluxos de controle paralelos em um único fluxo, seguida da execução deste após a finalização de qualquer um dos fluxos paralelos e do aborto da execução dos fluxos restantes; e
- *Acyclic Synchronization Merge (WCP37)* – junção de um subconjunto não vazio de fluxos de controle paralelos em um único fluxo, sendo que a atividade subsequente é ativada após a finalização de todos os elementos do subconjunto de fluxos.

Os padrões estruturais definem a finalização implícita e o comportamento iterativo (arbitrário, estruturado e recursivo) de processos de negócio. O modelo intermediário deve suportar dois padrões estruturais:

- *Implicit Termination (WCP11)* – finalização implícita da instância de um workflow, quando não houver mais atividades em execução ou aguardando algum tipo de mensagem ou sinal; e

- *Structured Loop (WCP21)* – execução iterativa estruturada de um conjunto de atividades.

Os padrões de instâncias múltiplas são inerentes à execução de múltiplas instâncias em paralelo de um mesmo conjunto de atividades. O modelo intermediário deve suportar três desses padrões:

- *Multiple instances without synchronization (WCP12)* – execução de múltiplas instâncias em paralelo de uma atividade de forma independente e sem sincronização entre as mesmas antes da continuação do fluxo de controle;
- *Multiple instances with a priori design-time knowledge (WCP13)* – execução de múltiplas instâncias em paralelo de uma atividade com sincronização entre as mesmas antes da continuação do fluxo de controle, sendo que o número de instâncias é fixado no projeto do workflow; e
- *Multiple instances with a priori run-time knowledge (WCP14)* – similar ao WCP13, exceto que o número de instâncias pode ser definido em tempo de execução, sendo que tal número pode ser proveniente, por exemplo, de valores assumidos por variáveis ou pela disponibilidade de recursos.

Os padrões baseados em estado definem a execução de atividades com base em sinais ou mensagens oriundos do próprio processo de negócio ou de alguma entidade externa. O modelo intermediário deve suportar dois desses padrões:

- *Deferred Choice (WCP16)* – escolha de um dentre vários fluxos de controle com base na interação com parceiros externos ou com o ambiente de execução; e
- *Persistent Trigger (WCP24)* – ativação de uma atividade por meio de um sinal de outra parte do workflow ou por meio de um parceiro ou ambiente externo.

Os padrões de cancelamento são responsáveis pela finalização de uma atividade específica, ou de um conjunto de atividades ou de todo o processo de negócio, sendo que o modelo intermediário deve suportar dois desses padrões:

- *Cancel Activity (WCP19)* – finalização de uma atividade antes ou durante sua execução; e
- *Cancel Case (WCP20)* – finalização explicita da instância de um workflow.

3.2.1.2 Padrões de Fluxo de Dados

Os padrões de fluxo de dados, no total de trinta e nove padrões, estão divididos em quatro categorias: (a) visibilidade de dados; (b) interação de dados; (c) mecanismos de transferência de dados; (d) desvio de fluxo baseado em dados.

Os padrões de visibilidade de dados referem-se à forma na qual os dados são vistos pelos vários componentes de um processo de negócio, sendo que o modelo intermediário deve suportar dois desses padrões:

- *Scope data (WDP3)* – determinados dados podem ser acessados somente por um determinado conjunto de atividades; e
- *Case data (WDP5)* – determinados dados podem ser acessados por qualquer atividade em um workflow.

Os padrões de interação de dados são inerentes à forma na qual os dados são comunicados entre os componentes ativos de um processo de negócio, sendo que o modelo intermediário deve suportar três desses padrões:

- *Task to Task (WDP8)* – transferência de dados entre atividades num mesmo workflow;
- *Task to Environment/Push-Oriented (WDP14)* – envio de dados a partir de uma atividade num workflow para uma entidade externa; e
- *Environment to Task/Push-Oriented (WDP16)* – envio de dados a partir de uma entidade externa para uma atividade num workflow.

Os padrões de mecanismos de transferência de dados focam no meio usado para a transferência de dados entre os componentes de um processo de negócio, sendo que o modelo intermediário deve suportar quatro desses padrões:

- *Data Transfer by Value/Incoming (WDP26)* – quando atividades de um workflow recebem dados por valor;
- *Data Transfer by Value/Outgoing (WDP27)* – quando atividades de um workflow enviam dados por valor;
- *Data Transfer by Reference/Unlocked (WDP29)* – quando atividades enviam e recebem dados usando uma referência a uma variável; e

- *Data Transformation/Input (WDP31)* – quando dados podem sofrer transformações antes de serem transferidos para outros componentes do processo de negócio.

Os padrões de desvio de fluxo baseado em dados capturam as formas nas quais dados podem interagir com controle de fluxo, influenciando no comportamento do processo de negócio. O modelo intermediário deve suportar dois desses padrões:

- *Event-Based Task Trigger (WDP37)* – execução de uma atividade quando uma expressão lógica envolver dados em sua asserção lógica; e
- *Data-based routing* (WDP39) – desvio do fluxo de controle baseado na avaliação de expressões lógicas que envolvem dados e sua asserção lógica.

3.2.1.3 Padrões de Comunicação

Os padrões de comunicação são inerentes à forma na qual os componentes de um sistema interagem entre si, sendo classificados de acordo com duas dicotomias: síncrona ou assíncrona, ponto-a-ponto ou multicast. O modelo intermediário deve suportar dois desses padrões de comunicação:

- *Request/Reply (CP1)* – envio/recepção de mensagens para/de parceiros externos com resposta; e
- *One-Way (CP2)* – envio/recepção de mensagens para/de parceiros externos sem resposta.

3.2.1.4 Padrões de Exceção

As construções do modelo intermediário para tratamento de exceções são baseadas no framework conceitual definido em (RUSSELL, VAN DER AALST e TER HOFSTEDE, 2006). Tal framework assume que uma exceção é um evento identificável e distinto dos usuais, que ocorre num instante especifico durante a exceção de um processo de negócio, sendo relacionado a uma única atividade.

No tratamento de exceções são consideradas: (a) como a atividade que disparou a exceção será tratada; (b) como as atividades que não dispararam a exceção serão tratadas; e (c) quais ações serão tomadas para sanar os efeitos da exceção. Consequentemente, um padrão típico de tratamento de exceção é divido em: *Exception Handling at Work Item Level* (*EWL*), refere-se ao momento no qual

uma exceção é disparada por uma atividade, sendo relacionado ao ciclo de vida das atividades ilustrado na Figura 3.4; *Exception Handling at Case Level* (*ECL*), refere-se ao tratamento no âmbito da instância do processo de negócio que teve uma exceção; e *Recovery Action* (*RC*), refere-se à ação a ser tomada para a recuperação dos efeitos da exceção.

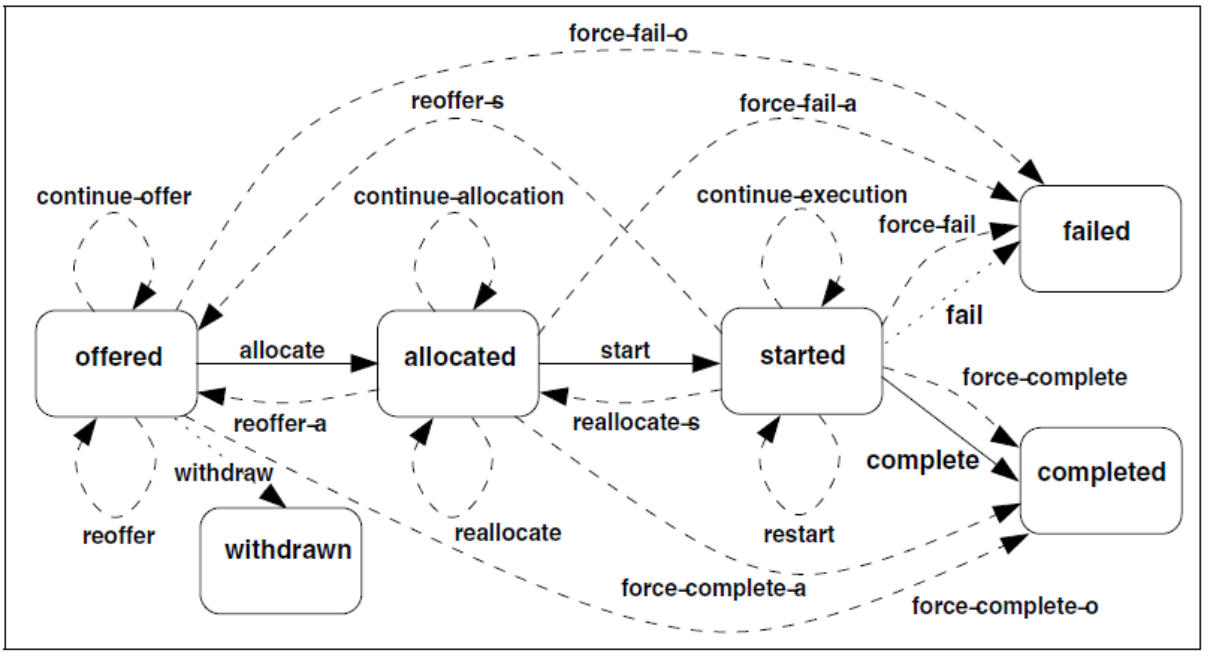


Figura 3.4 – Ciclo de vida de uma atividade de um processo de negócio (RUSSELL, VAN DER AALST e TER HOFSTEDE, 2006).

O modelo intermediário deve lidar com duas das possibilidades inerentes à EWL:

- *Continue-Execution (SCE)* – a atividade foi inicializada e não há mudança em seu estado como consequência da exceção; e
- *Force-Fail (SFF)* – a atividade está sendo executada, qualquer avanço em seu progresso é abortado e seu estado é alterado para *failed* como consequência da exceção. Nenhuma atividade subsequente é iniciada.

O modelo intermediário deve lidar com duas das possibilidades inerentes à ECL:

- *Continue Workflow Case (CWC)* – a execução do processo de negócio não é interrompida após o disparo de uma exceção; e

- *Remove Current Case (RCC)* – como consequência de uma exceção, um determinado conjunto de atividades ou todas as atividades em exceção ou aguardando para serem executadas podem ser removidas.

O modelo intermediário deve lidar com duas das possibilidades inerentes à RC:

- *No Action (NIL)* – não executa qualquer atividade em função do disparo de um determinado tipo de exceção; e
- *Compensate (COM)* – executa um conjunto de atividade em função do disparo de um determinado tipo de exceção.

Uma vez que os padrões de tratamento de exceção são definidos via uma tripla EWL-ECL-RC e tendo em vista as possibilidades inerentes aos elementos dessa tripla que o modelo intermediário deve lidar, os seguintes padrões de tratamento de exceção devem ser suportados por esse modelo:

- *SCE–CWC–COM* – tratamento de exceções que não alteram o estado de uma instância do processo de negócio e permitem que sua execução continue;
- *SCE–CWC–NIL* – captura de exceções que não alteram o estado de uma instância do processo de negócio sem tratamento das mesmas;
- *SFF–CWC–COM* – tratamento de exceções que causam a falha de uma atividade;
- *SFF–CWC–NIL* – captura de exceções que causam a falha de uma atividade sem tratamento das mesmas.
- *SFF–RCC–COM* – tratamento de exceções que causam uma falha global de uma instância do processo de negócio; e
- *SFF–RCC–NIL* – captura de exceções que causam uma falha global de uma instância do workflow sem tratamento das mesmas.

3.2.2 Seleção do Modelo

Em (DUIPMANS, PIRES e SANTOS, 2013) são discutidas as seguintes limitações dos modelos *Program Dependency Graphs (PDG)* (FERRANTE,

OTTENSTEIN e WARREN, 1987), *Control Flow Graph (CFG)*, *Protocol Tree (PT)* (PEREZ, 2012), *Petri Nets (PN)* e *Colored Petri Nets (CPN)* (MURATA, 1989), em relação à modelagem de processos de negócio:

- PDG é capaz de representar a dependência de dados, mas a dependência de controle é descrita indiretamente, o que dificulta a definição de comportamentos mais complexos tais como paralelismo e laço;
- CFG é capaz de representar fluxos de controle, mas não considera os fluxos de dados e a comunicação separadamente;
- PT é capaz de capturar somente construções baseadas em bloco, limitando o seu emprego em algumas linguagens de especificação de processos de negócio, sobretudo as que possuem estruturas baseadas em grafos;
- PN não é capaz de lidar com fluxos de dados entre atividades e emprega vários elementos para representar um processo de negócio, o que sobrecarrega a sua modelagem; e
- CPN é capaz de lidar com fluxos de dados, mas também emprega vários elementos para representar um processo de negócio .

Além das limitações acima descritas, esses modelos, assim como o definido em (DUIPMANS, PIRES e SANTOS, 2013), não levam em consideração todos os requisitos importantes para a representação de processos de negócios, tais como a captura e o tratamento de exceções. Visando a superar tais limitações, neste trabalho foi desenvolvido um novo modelo intermediário, baseado em grafos direcionados (BANG-JENSEN e GUTIN, 2009) e reutilizando as construções descritas em (DUIPMANS, PIRES e SANTOS, 2013), o qual foi denominado *Graph-based Workflow Model (GWM)*.

3.2.3 Definição do Modelo

No GWM as atividades são representadas por nós e as relações entre as mesmas por arestas, sendo que diferentes tipos de nós e arestas são definidos, em particular as arestas de controle, dados, comunicação e exceção, que permitem a representação de diferentes estruturas de controle e dados.

O GWM também permite a validação de restrições de dados em processos de negócio distribuídos e o cálculo do custo de atividades com base nos dados de entrada e saída de cada atividade. Eventualmente um comportamento inesperado de uma atividade interrompe o fluxo normal de um processo de negócio e dispara uma exceção, a qual pode ser capturada por uma aresta de exceção de saída.

Na sequência são apresentadas as construções do GWM em termos de quatro aspectos fundamentais: Fluxo de Controle, Fluxo de Dados, Comunicação e Tratamento de Exceção. Cada um desses aspectos é discutido em termos de aspectos elementares, que correspondem a um ou mais padrões WCP, WDP, WEP ou CP, os quais são indicados entre parênteses.

3.2.3.1 Fluxo de Controle

O aspecto Sequência (WCP1), ilustrado na Figura 3.5, representa a execução sequencial de atividades, sendo que a ordem de execução é determinada pelas arestas de controle direcionadas que conectam os nós.

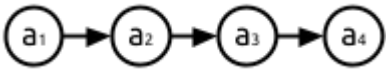


Figura 3.5 – Sequência.

O aspecto Ramo Condicional (WCP2, WCP3 e WDP40), ilustrado na Figura 3.6, é modelado pelos nós *if* e *eif* e representa uma escolha entre dois ramos alternativos. O *if* possui duas arestas de controle de saída, uma rotulada *true* e a outra *false*, e uma condição agregada ao mesmo, sendo que após a avaliação dessa condição somente uma das arestas é tomada. O *eif* junta os ramos condicionais convertendo-os num único ramo de saída.

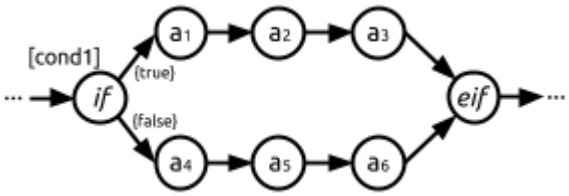


Figura 3.6 – Ramo Condicional.

O aspecto Ramos Paralelos (WCP4 e WCP5), ilustrado na Figura 3.7, é modelado pelos nós *par* e *epar* e representa a execução de múltiplos ramos em paralelo. O *par* divide um ramo de entrada em vários ramos paralelos de saída. O *epar* junta vários ramos paralelos de entrada num único ramo da saída;

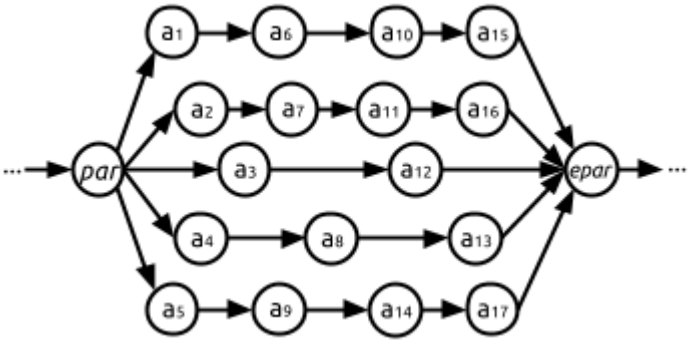


Figura 3.7 – Ramos Paralelos.

O aspecto Junção Parcial (WCP9, WCP19, WCP37), ilustrado na Figura 3.8, é modelado pelos nós *and* e *or* e permite a sincronização de subconjuntos de ramos paralelos. O ramo de saída de um *and* é executado somente após a finalização de todos os seus ramos de entrada, enquanto o ramo de saída de um *or* é executado após a finalização de um dos seus ramos de entrada, levando ao aborto das execuções dos demais ramos de entrada.

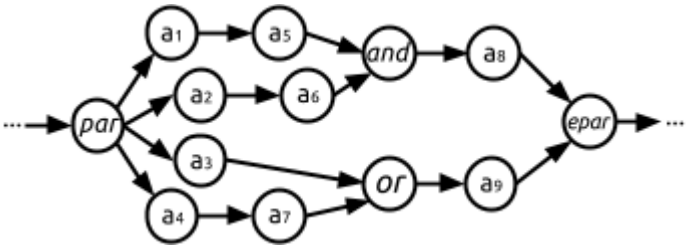
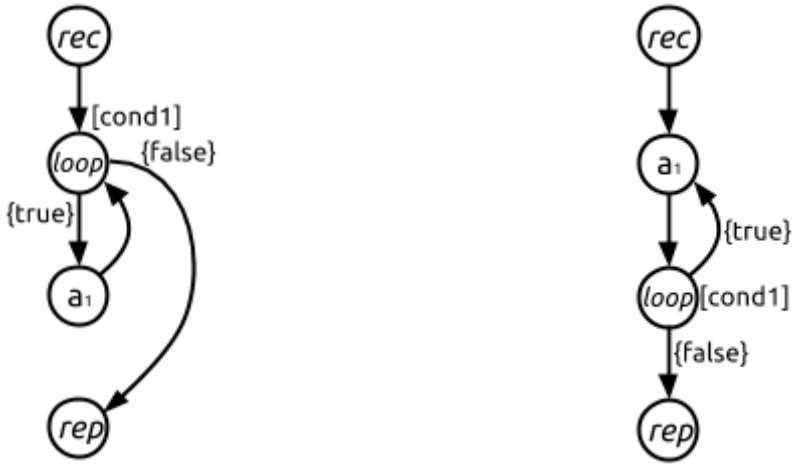


Figura 3.8 – Junção Parcial.

O aspecto Laço de Repetição (WCP21 e WDP40), ilustrado na Figura 3.9, é modelado pelo nó *loop* com uma condição agregada ao mesmo e representa iteração. Uma vez avaliada essa condição o ramo iterativo é tomado ou abandonado. Esse nó pode estar localizado antes ou depois do ramo iterativo, sendo que no primeiro caso o ramo pode ser executado zero ou mais vezes, enquanto no segundo caso o ramo é executado pelo menos uma vez.



(a) Nó *loop* antes do ramo iterativo. **(b) Nó *loop* depois do ramo iterativo.**

Figura 3.9 – Laço de Repetição.

O aspecto Escolha Exclusiva Externa (WCP16, WCP24, CP2 e WDP38), ilustrado na Figura 3.10, é modelado pelos nós *xor* e *exor* e representa a escolha de um ramo via interação com um parceiro externo. O *xor* divide um ramo de entrada em vários ramos de saída, os quais são relacionados a operações que devem ser escolhidas nas mensagens recebidas pelos parceiros externos. O *exor* junta vários ramos de entrada num único ramo de saída.

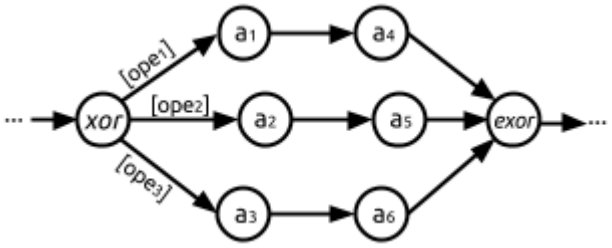


Figura 3.10 – Escolha Exclusiva Externa.

O aspecto Finalização Implícita (WCP11) não é modelado por nós específicos e finaliza um workflow implicitamente quando não há atividades em execução. O aspecto Finalização Explícita (WCP19), ilustrado na Figura 3.11, é modelado pelo nó *exit* e finaliza um workflow explicitamente.

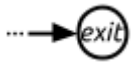


Figura 3.11 – Finalização Explícita.

O aspecto Instâncias Múltiplas (WCP12, WCP13 e WCP14), ilustrado na Figura 3.12, define uma regra de replicação via uma 3-tupla *(s, e, w)*, para gerar múltiplas instâncias paralelas de uma atividade. O número de instâncias paralelas é definido por (*e – s + 1*), sendo que não há instancias se *s* > *e*, e *w* é uma variável Booleana que define se as instâncias paralelas são sincronizadas ou não antes de disparar a execução do próximo nó.

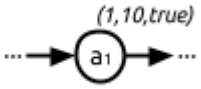


Figura 3.12 – Instâncias Múltiplas.

3.2.3.2 Fluxo de Dados

O aspecto Variável Local (WDP3) não é modelado por nós específicos e define variáveis que são visíveis somente a um determinado conjunto de atividades. O aspecto Variável Global (WDP5) também não é modelado por nós específicos e define variáveis que são visíveis a todo o processo de negócio.

O aspecto Troca de Dados (WDP9, WDP15, WDP16, WDP27, e WDP28), ilustrado na Figura 3.13, é modelo por uma aresta de dados rotulada, a qual representa a troca de dados entre duas atividades internas do processo de negócio, ou entre este e um parceiro externo.

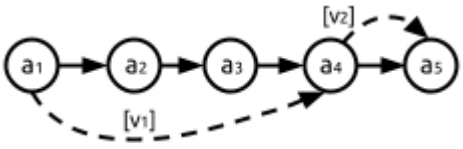


Figura 3.13 – Troca de Dados.

3.2.3.3 Comunicação

O aspecto Receber Mensagem (CP2 e WDP38), ilustrado na Figura 3.14, é modelado pelo nó *rec* rotulado com uma operação e permite a recepção de mensagens provenientes de parceiros externos.

Figura 3.14 – Receber Mensagem.

O aspecto Responder Mensagem (CP1), ilustrado na Figura 3.15, é modelado pelo nó *rep* e permite uma resposta a um parceiro externo, relativa à mensagem recebida do mesmo.

Figura 3.15 – Responder Mensagem.

O aspecto Requisitar Serviço (CP1 e CP2) é modelado pelo nó *req* quando uma mensagem é enviada a um parceiro externo e não há espera de uma resposta do mesmo, conforme ilustrado na Figura 3.16(a), ou pelos nós *req* e *get* quando há essa espera, conforme ilustrado na Figura 3.16(b).

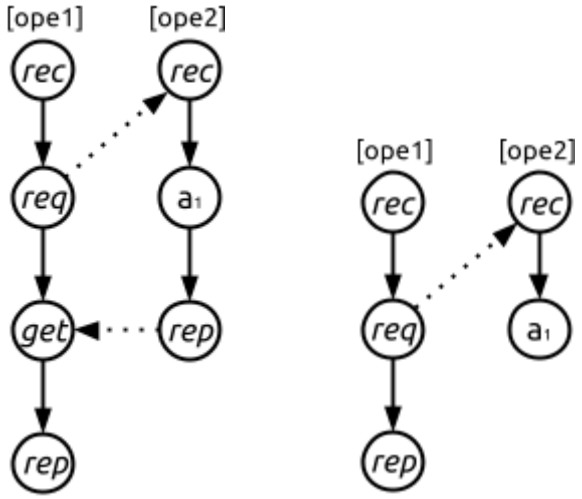


Figura 3.16 – Requisitar Serviço com (a) e sem (b) uma resposta.

3.2.3.4 Tratamento de Exceções

O aspecto Tratamento de Exceções (SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM e SFF-RCC-NIL), ilustrado na Figura 3.17, é modelado por nós *exp* e *eexp* e permite o tratamento de exceções com base nos tipos das mesmas. Ao receber uma exceção de certo tipo em sua aresta de entrada, o nó *exp* encaminha o seu tratamento via a aresta de saída rotulada no tipo dessa exceção. Se nenhuma das arestas de saída corresponde à exceção, uma aresta opcional de saída rotulada *otherwise* pode ser tomada. O nó *eexp* junta os ramos de exceção num único ramo de saída, permitindo o retorno às atividades normais do processo de negócio.

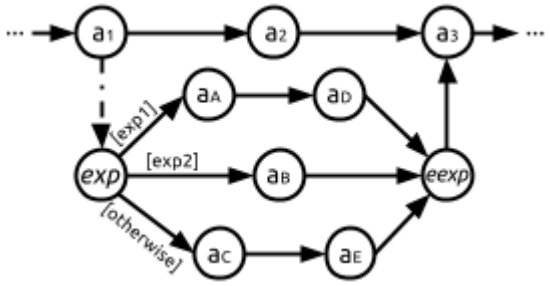


Figura 3.17 – Tratamento de Exceções.

O aspecto Tratamento de Tempo Limite (SCE-CWC-COM, SCE-CWC-NIL, SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM e SFF-RCC-NIL), ilustrado na Figura 3.18, é modelado pelos nós *ddl* e *eddl* e permite o tratamento de exceções com base em tempos limites. Ao receber uma exceção de tempo limite em sua aresta de entrada rotulada nesse tempo, a qual é proveniente de um nó *xor* ou *rec*, o nó *ddl* encaminha o seu tratamento. O nó *eddl* encerra o tratamento da exceção de tempo limite, permitindo o retorno às atividades normais do processo de negócio.

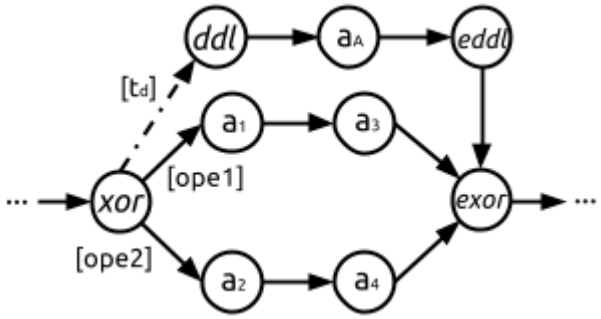


Figura 3.18 – Tratamento de Tempo Limite.

O aspecto Tratamento de Finalização (SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM e SFF-RCC-NIL), ilustrado na Figura 3.19, é modelado pelos nós *fin* e *efin* e permite que um ramo seja tomado depois da execução de um conjunto de atividades, mesmo que alguma delas dispare uma exceção.

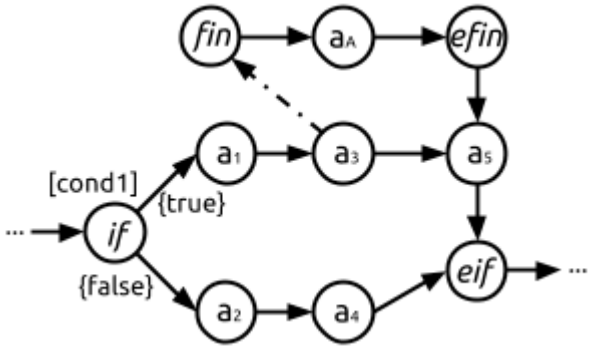


Figura 3.19 – Tratamento de Finalização.

3.3 Seleção de Localização

Com a finalidade de determinar de forma semiautomática a localização de cada atividade e dado associado, foi construído um *framework* de seleção de localização baseado em políticas de privacidade, custos monetários e métricas de desempenho (e.g., tempo de resposta, vazão). Tal *framework* assume implicitamente que se um processo na nuvem coordena a execução de uma atividade essa é executada na nuvem.

Utilizando o GWM gerado na etapa de *lifting* e com base no modelo de otimização inteiro definido em (HAN, SUN, *et al.*, 2010), o *framework* de seleção calcula o custo do processo de negócio para cada combinação possível de localização (na premissa ou na nuvem) dos nós como segue:

$$cost_d = w_e \times cost_e + w_m \times cost_m + w_p \times cost_p \quad (1)$$

onde $cost_e$, $cost_m$, e $cost_p$ são, respectivamente, o custo de execução, o custo monetário e o custo de privacidade, e w_e, w_m, e w_p são seus pesos definidos para

representar o impacto que cada um desses fatores possui no cálculo do custo total. Para determinar tais custos é assumido que:

- seja $\boldsymbol{A} = \{a_1, a_2, \ldots, a_n\}$ o conjunto de atividades, e $\boldsymbol{D} = \{d_1, d_2, \ldots, d_m\}$ o conjunto de item de dados no GWM emitido pela etapa de *lifting*;
- seja $\boldsymbol{s} = (s_1\ s_2\ \ldots\ s_n)$ o vetor de localização de atividades, onde $s_i = 1$ significa que a atividade a_i está localizada na nuvem e $s_i = 0$ na premissa;
- seja $\boldsymbol{R}$ a matriz esparsa de relação entre atividades e item de dados, onde $R(i,j) = 1$ significa que a atividade a_i possui uma relação direta com o item de dado d_j e $R(i,j) = 0$ que a atividade e o item de dado não se relacionam;
- seja $\boldsymbol{V}$ a matriz esparsa 3-dimensional para troca de dados, onde $V(i,j,k) = 1$ significa que a atividade a_i envia o item de dado d_j para a atividade a_k, e $V(i,j,k) = 0$ significa que esse item de dado não é enviado pela a atividade a_i para a atividade a_k;
- seja $\boldsymbol{Q}$ a matriz esparsa para localização de dados com

$$Q(o,p) = \sum_{i=1}^{m} [V(i,o,p) \times s_i] \tag{2}$$

  onde o item de dado d_j está localizado na premissa antes de ser enviado para a atividade a_i se $Q(i,j) = 0$, e na nuvem se $Q(i,j) = 1$;
- seja $C_{premise} = (ram_p, hdd_p, cpu_p, f_p, b)$ a 5-tupla que representa a configuração do servidor na premissa. ram_p é a quantidade de RAM em MiB, hdd_p é a quantidade de disco em GiB, cpu_p é o número de vCPUs, f_p é a frequência de cada vCPU em GHz, e b é a largura de banda em Bps entre a premissa e a nuvem;
- seja $C_{cloud} = (ram_c, hdd_c, cpu_c, f_c, cost_t, cost_h, cost_s)$ a 7-tupla que representa a configuração do servidor na nuvem. ram_c é a quantidade de RAM em MB, hdd_c é a quantidade de disco em GiB, cpu_c é o número de vCPUs, f_c é a frequência de cada vCPU em GHz, $cost_t$ é o custo em US$ por byte transferido para a nuvem, $cost_h$ é o custo em US$ por hora do servidor na nuvem, e $cost_s$ é o custo em US$ por byte armazenado na nuvem; e

- seja $exec_c(a_i)$ o tempo de execução da atividade a_i na nuvem definido como

$$exec_c(a_i) = \frac{1}{2} \cdot \left(\frac{cpu_p \cdot f_p}{cpu_c \cdot f_c} + \frac{ram_p}{ram_c} \right) \times \ exec_p(a_i) \quad (3)$$

onde $exec_p(a_i)$ é o tempo de execução da atividade a_i na premissa que deve ser fornecido por um entidade externa (e.g., usuário ou componente de BPMS).

A função $exec_c(a_i)$ define que o aumento de RAM ou o aumento do poder de processamento causam redução no tempo de execução das atividades na nuvem. Essa suposição é aceitável porque não há conhecimento prévio se uma atividade é memória intensiva, CPU intensiva ou disco intensivo. Consequentemente, $exec_c(a_i) < exec_p(a_i)$, caso a configuração do servidor na nuvem tenha maior poder de processamento do que a configuração do servidor na premissa, o que é o esperado. No entanto, reduzir o tempo de execução das atividades de um processo de negócio pode não ser um benefício real. Isso ocorre caso o tempo de transferência de dados entre a nuvem e a premissa for maior ou igual do que a redução obtida no tempo de execução das atividades. O tempo requerido para transferir dados entre a nuvem e a premissa é definido como

$$trans_t = \sum_{i=1}^{n} \sum_{j-1}^{m} \frac{size(d_j)}{b} \times \ R(i,j) \times \ |s_i - \ Q(i,j)| \quad (4)$$

onde $size(d_j)$ é o tamanho do item de dados d_j em bytes, fornecido por uma entidade externa, e o módulo $|s_i - Q(i,j)|$ é igual a *1*, caso o item de dado d_j precise ser transferido entre a nuvem e a premissa para ser utilizado pela atividade a_i, e a *0* caso contrário. Dessa forma, o custo de execução de um processo de negócio é definido como

$$cost_e = \sum_{i=1}^{n} [exec_c(a_i) \times s_i + exec_p(a_i) \times (1 - s_i) + trans_t] \quad (5)$$

Geralmente, o provedor de nuvem tarifa o tempo ativo do servidor, a quantidade de dados transferidos em um intervalo de tempo e a quantidade de dados armazenada. O custo monetário para a execução de atividades de processos de negócio na nuvem é definido como

$$monet_c = \sum_{i=1}^{n} [cost_h \times exec_c(a_i) \times s_i] \quad (6)$$

Seja ***P*** a matriz esparsa, fornecida por uma entidade externa, onde *P(i,j) = 1* se o item de dado d_j é armazenada pela atividade a_i e *0* caso contrário. O custo monetário para armazenar dados de um processo de negócio na nuvem é definido como

$$monet_s = \sum_{i=1}^{n} \sum_{j=1}^{m} [cost_s \times size(d_j) \times s_i \times P(i,j)] \quad (7)$$

O custo monetário para transferir dados entre a nuvem e a premissa é definido como

$$monet_t = \sum_{i=1}^{n} \sum_{j=1}^{m} [cost_t \times size(d_j) \times |s_i - Q(i,j)|] \quad (8)$$

Dessa forma, o custo monetário de um processo de negócio é definido como

$$cost_m = monet_c + monet_s + monet_t \quad (9)$$

Privacidade de dados é outra questão a ser considerada ao executar processos de negócio na nuvem. Seja ***c*** = *(c_1 c_2 … c_m)* um vetor de restrição de item

de dados, fornecido por uma entidade externa, onde $c_j = 1$ caso o item de dados d_j seja sensível e *0* caso contrário. Dessa forma, o custo de privacidade de um processo de negócio é definido como

$$cost_p = \sum_{i=1}^{n} \sum_{j=1}^{m} [c_j \cdot R(i,j) \cdot s_i] \quad (10)$$

Durante o cálculo do custo, o *framework* de seleção emprega algumas restrições à localização dos nós com a finalidade de tornar o processo de seleção mais eficiente:

- todos os nós empregados para juntar múltiplos ramos (e.g., *epar*, *eif*, *exor*) são sempre movidos, para a premissa ou para a nuvem, juntamente com seu nó de divisão correspondente (e.g., *par*, *if*, *xor*);
- todos os nós de comunicação (*req*, *rec*, *rep*, *get*) utilizados para acessar atividades em um parceiro externo são sempre movidos para o mesmo local que tais atividades;
- todas as atividades sequenciais, que não persistem dados e não interagem com dados sensíveis, são movidas em conjunto para reduzir a transferência de dados entre a nuvem e a premissa; e
- todos os nós *rec* e *rep* sempre são mantidos em sua localização original para que a decomposição seja transparente para os parceiros externos.

Após calcular o custo de cada possível combinação de localização dos nós (na premissa ou na nuvem), o menor custo é selecionado. O vetor ***s*** associado a esse custo é utilizado para marcar a localização de cada nó do GWM gerado na etapa de *lifting*. O GWM marcado é utilizado como entrada na etapa de decomposição.

3.4 Decomposição

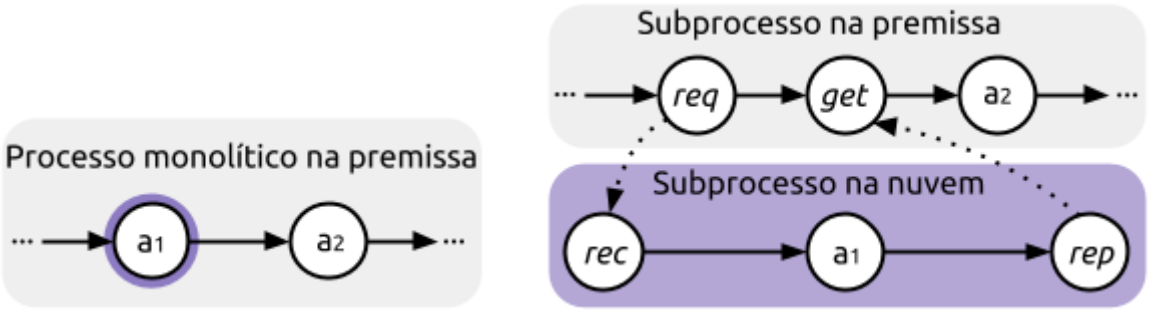
Esta seção descreve as regras de decomposição aplicadas na abordagem proposta e também as etapas necessárias para aplicar tais regras. A Seção 3.4.1 apresenta as regras de decomposição e as exemplifica e a Seção 3.4.2 apresenta as etapas de decomposição junto aos seus algoritmos.

3.4.1 Regras de Decomposição

Com a finalidade de executar a decomposição de processos de negócio, foram definidas seis regras de decomposição considerando que um processo é alocado na premissa e possui atividades e dados associados a serem alocadas na nuvem, ou vice-versa.

Para a definição de tais regras foram consideradas as seguintes restrições: (*i*) todo nó de controle de devisão deve ser alocado juntamente com o seu nó de junção correspondente; (*ii*) atividades adjancentes alocadas para o mesmo local devem ser movidas como um bloco; (*iii*) aspectos relacionados a tratamento de exceções não podem ter seus nós de controle movidos; (*iv*) a comunicação entre os subprocessos colocaborativos é definida por pares de nós *req/rec* e *rep/get* para que atividades alocadas em diferentes subprocessos continuem conectadas via troca de mensagens; e (*v*) sempre deverá haver um subprocesso no mesmo local do processo monolítico original, com a finalidade de tornar a decomposição transparente para o usuário e independente de outras tecnologias (e.g., Domain Name System (DNS), Firewall).

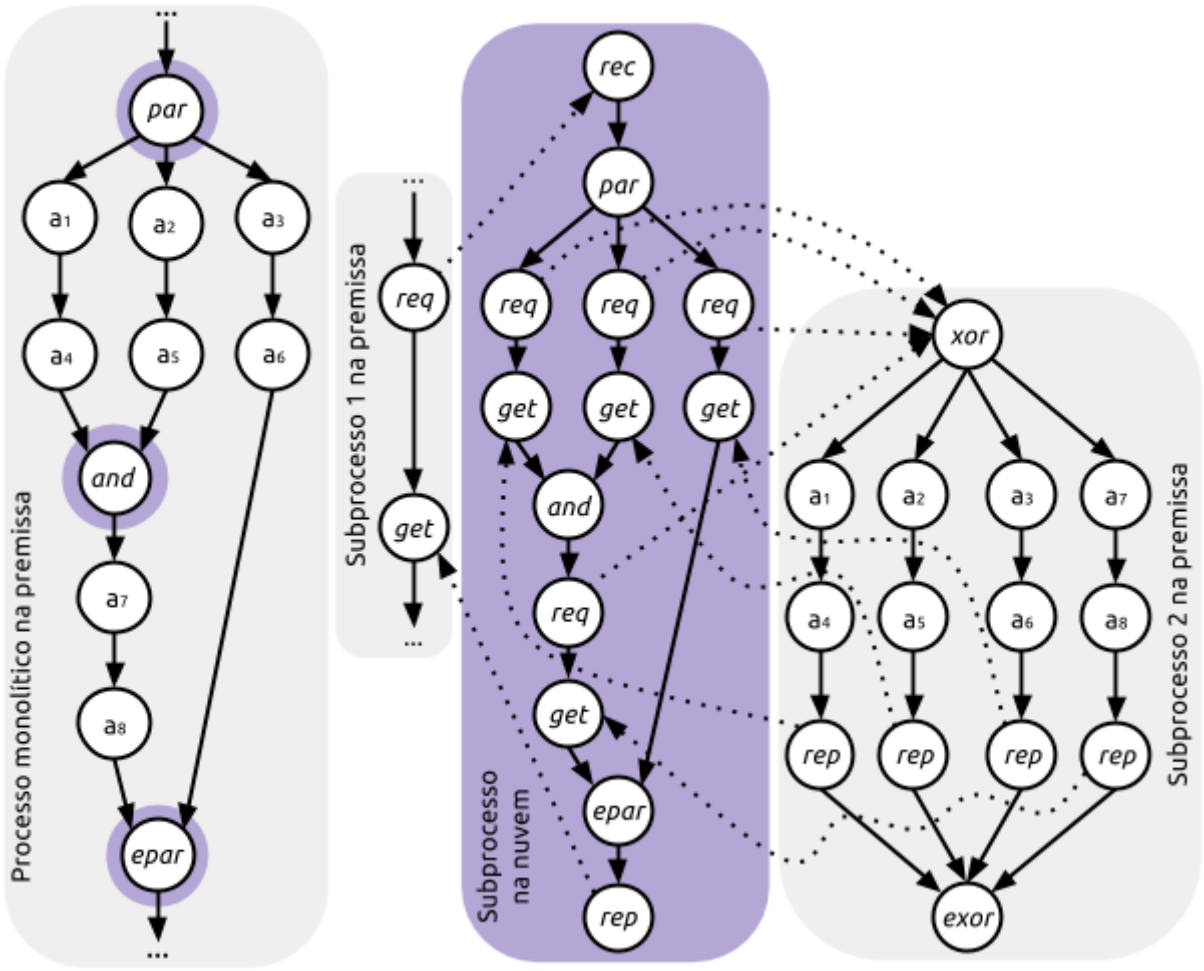
A primeira regra de decomposição aloca os aspectos Sequência, Ramo Condicional, Ramos Paralelos ou Laço de Repetição como um todo em um novo subprocesso. O aspecto selecionado no processo monolítico é substituído pelos nós *req* e *get* conectados via uma aresta de controle no subprocesso na premissa. O subprocesso da nuvem inicia com um nó *rec* e termina com um nó *rep*. Os nós *rec* e *req*, e os nós *rep* e *get* são conectados por arestas de comunicação. A Figura 3.20 ilustra essa regra aplicada ao aspecto Sequência composto por um nó de atividade.



(a) Monolítico **(b) Decomposto**

Figura 3.20 – Movendo um nó de atividade.

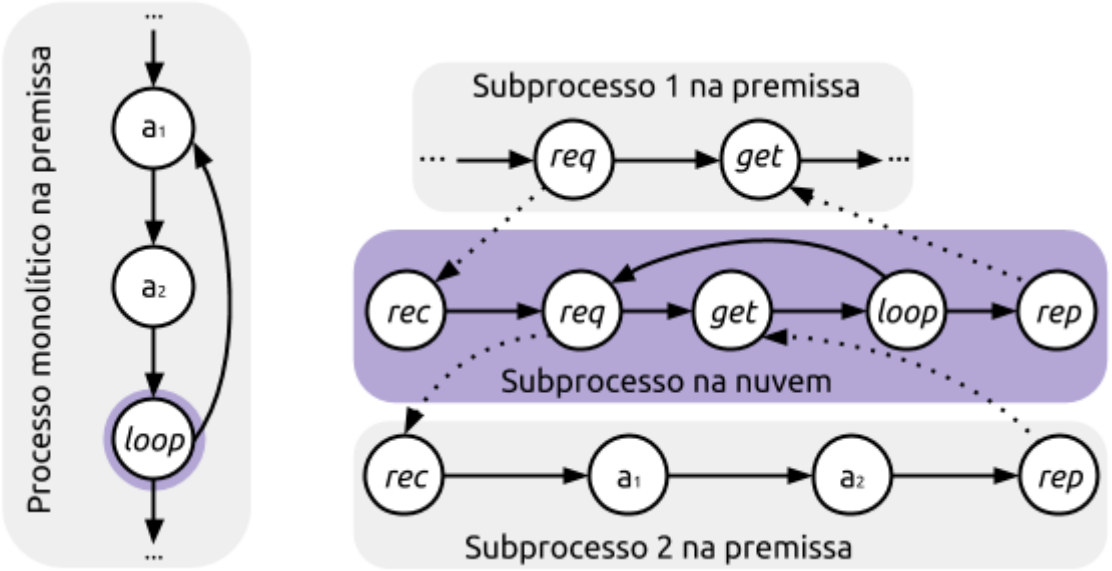
A segunda regra gera três subprocessos, onde dois são alocados na premissa e o outro na nuvem, para alocar os nós *if* e *eif* do aspecto Ramo Condicional ou os nós *par*, *epar*, *and* e *or* do aspecto Ramos Paralelos no subprocesso da nuvem. A Figura 3.21 ilustra essa regra aplicada ao aspecto Ramos Paralelos. Na primeira fase da decomposição o aspecto é alocado como um todo no subprocesso na nuvem e substituído por nós *req* e *get* no subprocesso da premissa. Então, os ramos aninhados aos nós *par*, *epar* e *and* são substituídos por nós *req* e *get* no subprocesso na nuvem, e alocados aninhos a um aspecto Escolha Exclusiva Externa em um novo subprocesso na premissa.



(a) Monolítico **(b) Decomposto**

Figura 3.21 – Movendo os nós *par*, *epar* e *and* do aspecto Ramos Paralelos.

A terceira regra de decomposição também gera três subprocessos para alocar o nó *loop* do aspecto Laço de Repetição no subprocesso na nuvem. A Figura 3.22 ilustra essa regra, onde a primeira fase é exatamente igual a da segunda regra. Então, o ramo iterativo do aspecto Laço de Repetição é substituído por nós *req* e *get*, no subprocesso na nuvem, e alocado entre os nós *rec* e *rep* no novo subprocesso na premissa.

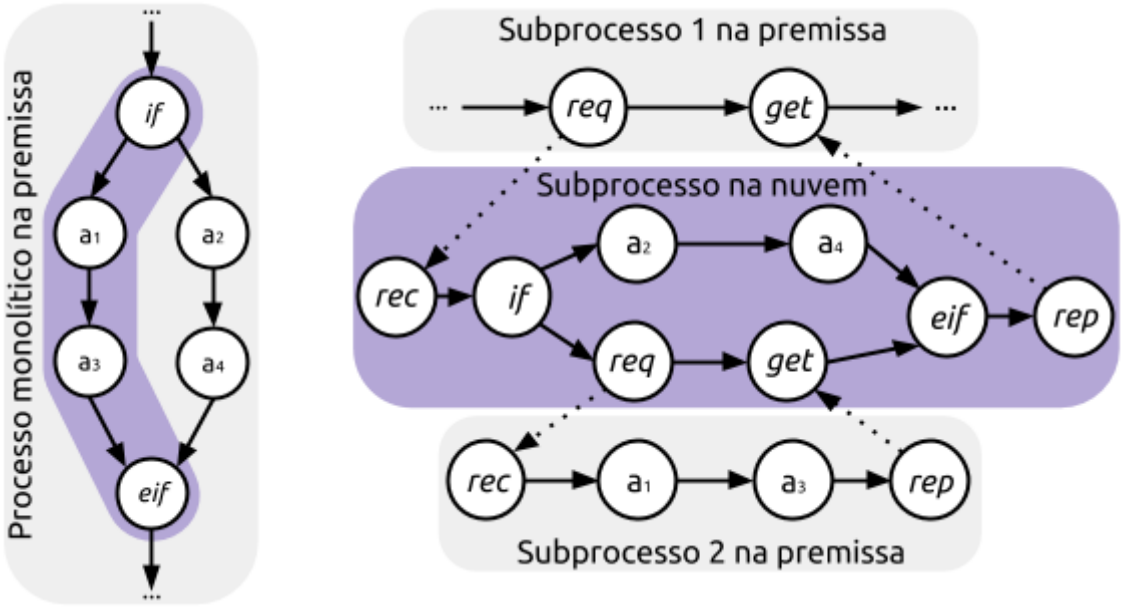


(a) Monolítico **(b) Decomposto**

Figura 3.22 – Movendo o nó *loop* de um aspecto Laço de Repetição.

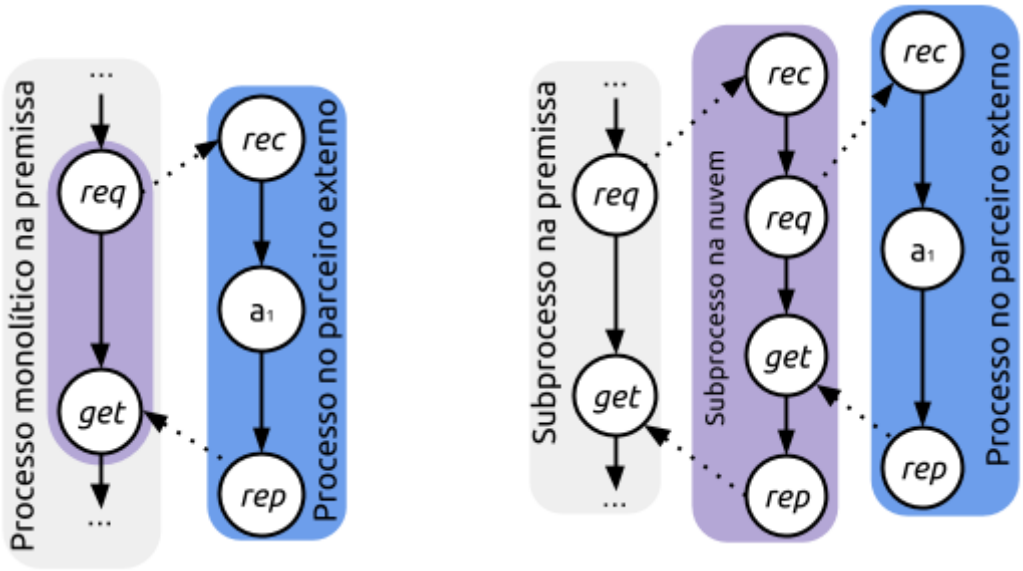
A quarta regra de decomposição combina as regas um e dois e é aplicada sobre os aspectos Ramo Condicional e Ramos Paralelos. Tal regra gera três subprocessos, onde dois são hospedados na premissa e o outro na nuvem. Essa regra é necessária para alocar no subprocesso da nuvem: (*i*) os nós *if* e *eif* e um ramo aninhado do aspecto Ramo Paralelo; ou (*ii*) os nós *par*, *epar*, *and* e *or* e um subconjunto de ramos aninhos do aspecto Ramos Paralelos. A Figura 3.23 ilustra essa regra aplicada ao aspecto Ramo Condicional.

A quinta regra de decomposição é aplicada ao aspecto Requisitar Serviço, alocando os nós *req* e *get* desse aspecto em um subprocesso na nuvem. A Figura 3.24 ilustra essa regra, onde os nós *req* e *get* no subprocesso na premissa são substituídos por nós *req* e *get* e hospedados no novo subprocesso na nuvem aninhados aos nós *rec* e *rep*. Essa regra de decomposição reduz a comunicação entre subprocessos quando requisitando um serviço em um parceiro externo que necessita de dados movidos para a nuvem.



(a) Monolítico **(b) Decomposto**

Figura 3.23 – Movendo os nós *if* e *eif* e um ramo do aspecto Ramo Condicional.

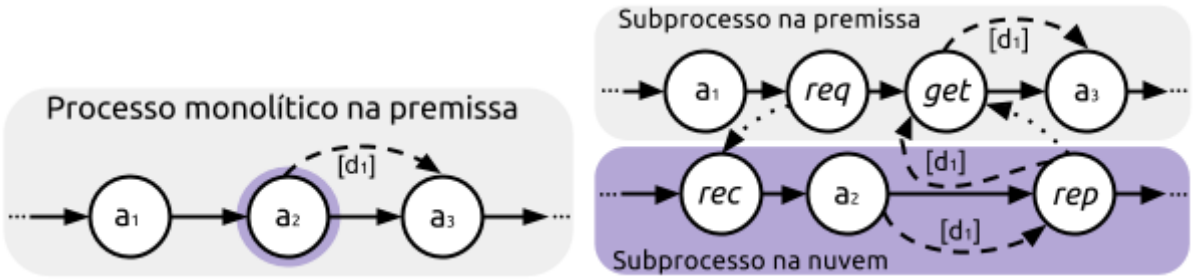


(a) Monolítico **(b) Decomposto**

Figura 3.24 – Movendo o aspecto Requisitar Serviço.

A sexta regra de decomposição move dados associados a atividades que originalmente estavam em um lado (i.e., nuvem ou premissa) e foram alocados para outro lado. A Figura 3.25 ilustra essa regra aplicada ao dado d_1 transmitido entre as atividades a_2 e a_3. A Figura 3.25(a) mostra um processo monolítico na premissa onde a atividade a_2, a qual foi alocada para a nuvem, envia o dado d_1 para a

atividade a_3. A Figura 3.25(b) mostra o processo decomposto onde a atividade a_2, agora no subprocesso na nuvem, envia o mesmo dado d_1 para a atividade a_3 no subprocesso na premissa. Nesse exemplo, a aresta de dados da atividade a_2 é conectada no nó *rep*, e duas novas arestas de dados são criadas para trocar o dado d_1: uma entre os nós *rep* e *get*, e a outra entre o nó *get* e a atividade a_3.

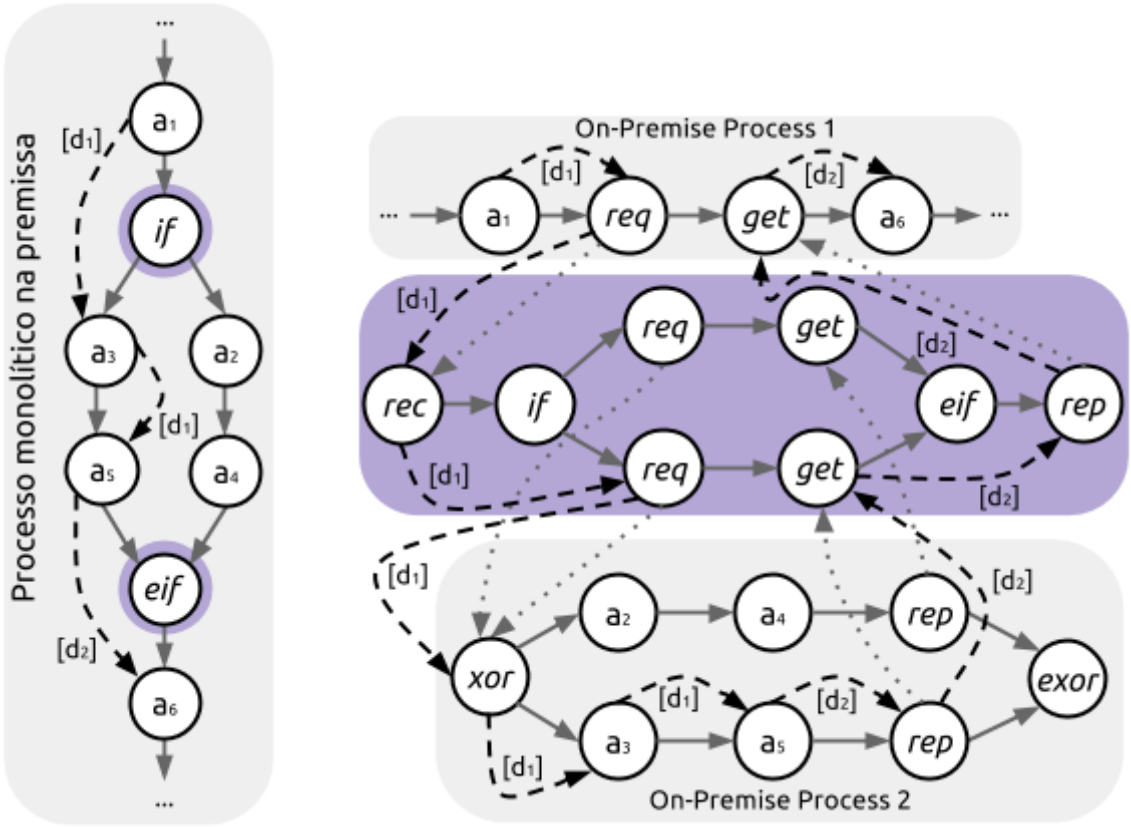


(a) Monolítico **(b) Decomposto**

Figura 3.25 – Movendo uma atividade enviando um dado.

Mover dados recebidos ou enviados para atividades alocadas no subprocesso na premissa gerado através da aplicação da segunda, terceiro ou quarta regra de decomposição requer que a sexta regra seja aplicada duas vezes: uma para enviar o dado entre os subprocessos da premissa e da nuvem, e a segunda para enviar o dado desse último subprocesso ao outro subprocesso na premissa. A Figura 3.26 ilustra esse caso aplicando a segunda regra de decomposição sobre o aspecto Ramo Condicional, onde a_3 recebe d_1 de a_1 e a_4 envia d_2 para a_6.

As regras de decomposição não podem ser aplicadas aos nós de exceção *exp*, *eexp*, *fin*, *efin*, *ddl* e *eddl* para que o tratamento adequado às exceções no processo original seja mantido, sendo que as exceções nos subprocessos restantes são propagadas através das arestas de comunicação quando não tratadas. Contudo, as regras de decomposição podem ser aplicadas sobre os ramos aninhados aos nós de exceção.



(a) Monolítico **(b) Decomposto**

Figura 3.26 – Movendo atividades para o segundo subprocesso na premissa que recebem dados de atividades no primeiro subprocesso na premissa.

3.4.2 Transformações para Decomposição

As regras de decomposição não podem ser aplicadas através de uma única fase. Portanto, a etapa de decomposição é divida em quatro fases denominadas fragmentação, agrupamento, coreografia e fluxo de dados.

A fase de fragmentação tem como entrada o GWM emitido pela etapa de seleção de localização. Tal fase emite o GWM de entrada com nós de comunicação no local dos nós adjacentes movidos para a nuvem. Também são emitidas duas listas contendo os nós substituídos. A fase de agrupamento gera um GWM para cada uma das listas emitida pela fase antecedente. A fase de coreografia tem como entrada o GWM fragmentado emitido pela primeira fase e os GWMs gerados pela fase precedente. A partir desses a fase de coreografia cria as arestas e nós necessários para a comunicação entre os GWMs. A última fase, a fase de fluxo de dados, cria as arestas requeridas para que o novo fluxo seja equivalente ao original.

A Figura 3.27 ilustra a sequência, entradas e saídas de cada uma das fases da etapa de decomposição.

Figura 3.27 – Fases da etapa de decomposição.

3.4.2.1 Fragmentação

O objetivo da fase de fragmentação é substituir nós adjacentes com localização distinta do subprocesso que os aninha por pares de nós de comunicação *req* e *get*. Cada conjunto de nós substituído recebe um identificador único, o qual também é atribuído ao par de nós de comunicação que o substituiu. Tal conjunto é adicionado a uma lista referente ao seu local de distribuição. Esse procedimento é executado recursivamente para cada ramo aninhado a um aspecto.

A entrada da etapa de fragmentação é composta pelo GWM referente ao processo monolítico emitido pela etapa de seleção de localização e a saída por três subprocessos em GWM, dois referentes aos subprocessos da premissa e um referente ao subprocess da nuvem ou vice-versa. O Algoritmo 3.1 mostra o pseudocódigo da função *PerformFragmentation* que desempenha a etapa de fragmentação.

Algoritmo 3.1 – Algoritmo de fragmentação.

```
cloudList  ← {}
premiseList ← {}

function PerformFragmentation(mainProcess)
   Fragmentation(mainProcess, mainProcess.isOnPremise)
   fragment ← new Fragment
   fragment.mainSubprocess ← mainSubprocess
   fragment.cloudList ← cloudList
   fragment.premiseList ← premiseList
   return fragment
end function
```

Note que o algoritmo mostrado anteriormente é composto pelo procedimento denominado *Fragmentation*, o qual de fato fragmenta os subprocessos e gera as listas com os conjuntos de nós adjacentes alocados para um novo subprocesso. O Algoritmo 3.2 mostra o pseudocódigo de tal procedimento. Sua entrada é composta por um subprocesso e a localização do processo que o aninha. Caso o subprocesso não seja aninhado, a localização considerada é a do próprio subprocesso. Esse procedimento percorre os nós pela ordem do fluxo de controle e para cada um deles identifica se sua localização é diferente a do seu subprocesso.

Todos os nós marcados para um local diferente do seu subprocesso são removidos junto as suas arestas de controle e adicionados ao conjunto de nós adjacentes, o qual possui um identificador único. O algoritmo também identifica as arestas de exceção de cada nó adjacente e seus ramos aninhados e as adiciona ao mesmo conjunto do respectivo nó. O Algoritmo 3.2 emprega o procedimento *Replace* para realizar a substituição dos nós adjacentes alocados para um novo subprocesso, sendo que o Algoritmo 3.3 mostra o pseudocódigo de tal procedimento.

O procedimento *Replace* recebe como entrada o subprocesso que será fragmentado, o conjunto de nós adjacentes removidos, a localização do subprocesso que aninha o subprocesso de entrada, o nó imediatamente subsequente ao último nó a ser movido e as arestas de controle de entrada do primeiro nó adjacente removido. Os nós de comunicação *req* e *get* são criados e as arestas de controle de entrada do primeiro nó adjacente são conectadas no nó *req*. As arestas de exceção identificadas no procedimento *Fragmentation* são conectadas nos nós *req* e *get*. Caso o nó subsequente ao último nó adjacente não seja vazio suas arestas de entrada são conectadas ao nó *get*.

Algoritmo 3.2 – Algoritmo do procedimento *Fragmentation*.

```
procedure Fragmentation(subprocess, location)
 n ← subprocess.startNode
 while n ≠ {} do
  if n.location ≠ location then
   adjacentNodes ← new AdjacentNodes
   iControlEdges ← n.incomingControlEdges
   repeat
    if n of type ReqNode then
     adjacentNodes.nodes ← adjacentNodes.commEdges ∪
                           {n.outgoingCommunicationEdge}
     subprocess.commEdges ← subprocess.commEdges −
                           {n.outgoingCommunicationEdge}
     ns ← n.outgoingCommunicationEdge.destinationNode
     while ns ≠ {} do
      adjacentNodes.nodes ← adjacentNodes.nodes ∪ {ns}
      subprocess.nodes    ← subprocess.nodes − {ns}
      ns ← ns.nextNode
     end while
    else if n of type GetNode then
     adjacentNodes.nodes  ← adjacentNodes.commEdges ∪
                            {n.outgoingCommunicationEdge}
     subprocess.commEdges ← subprocess.commEdges −
                            {n.outgoingCommunicationEdge}
    else if n of type Construction then
     for all b ∈ n.branches do
      Fragmentation(b, n.location)
     end for
     adjacentNodes.exceptionEdges ←
      adjacentNodes.exceptionEdges ∪ n.innerExceptionEdges
    end if
    if |n.outgoingExceptionEdges| > 0 then
     for all x ∈ n.outgoingExceptionEdges do
      if not x.destinationExceptionNode.isFragmented then
       x.destinationNode.fragmented ← true
       for all y ∈ x.destinationExceptionNode.branches do
        Fragmentation(y, location)
       end for
      end if
     end for
    end if
    adjacentNodes.nodes ← adjacentNodes.nodes ∪ {n}
    adjacentNodes.exceptionEdges ←
     adjacentNodes.exceptionEdges ∪ n.outgoingExceptionEdges
    subprocess.nodes ← subprocess.nodes − {n}
    if n.nextNode ≠ {} and n.nextNode.location ≠ location then
     subprocess.controlEdges ←
      subprocess.controlEdges − n.outgoingControlEdges
     n.outgoingControlEdges ← {}
    end if
    n ← n.nextNode
   until n ≠ {} and n.location ≠ location
   Replace(subprocess, adjacentNodes, location, n, iControlEdges)
  else
   n ← n.nextNode
  end if
 end while
end procedure
```

Algoritmo 3.3 – Algoritmo procedimento *Replace*.

```
procedure Replace(subprocess, adjacentNodes, location, n, iControlEdges)
 reqNode ← new ReqNode
 getNode ← new GetNode
 reqNode.operation ← adjacentNodes.id
 for all i ∈ iControlEdges do
  i.destinationNode ← reqNode
 end for
 for all e ∈ adjacentNodes.exceptionEdges do
  e.sourceNode ← reqNode
  subprocess.exceptionEdges ← subprocess.exceptionEdges ∪
    {new ExceptionEdge(getNode, e.destinationNode)}
 end for
 if n ≠ {} then
  for all o ∈ n.outgoingControlEdges do
   o.sourceNode ← getNode
  end for
 end if
 subprocess.addNode(reqNode, getNode)
 subprocess.addEdge(new ControlEdge(reqNode, getNode))
 if (location = onPremise) then
  cloudList ← cloudList ∪ {adjacentNodes}
 else
  premiseList ← premiseList ∪ {adjacentNodes}
 end if
end procedure
```

3.4.2.2 Agrupamento

A fase de agrupamento tem a finalidade de criar um subprocesso em GWM para cada lista criada pela fase antecedente. O Algoritmo 3.4 mostra o pseudocódigo referente à função *PerformClustering* que desempenha a tarefa da fase de agrupamento.

Para cada lista é verificada a quantidade de conjuntos de nós que a compõe. Caso exista somente um conjunto, o subprocesso criado terá um único ramo composto por um nó inicial *rec* com o nome da operação de acesso definida pelo identificador do conjunto de nós, por um nó final *rep* e pelos nós do conjunto da lista de entrada. Se a lista for composta por mais de um conjunto de nós é criado um subprocesso composto por um aspecto Escolha Exclusiva Externa com um ramo aninhado para cada conjunto de nós. Cada ramo é atrelado ao identificador do conjunto de nós adjacentes que o compõe, sendo que tal identificador corresponde ao nome da operação utilizada para acessar o subprocesso criado. Cada ramo é finalizado com um nó de comunicação *rep*.

Algoritmo 3.4 – Algoritmo da função *PerformClustering*.

```
function PerformClustering(premiseList, cloudList)
 p ← []
 i ← 0
 for all l ∈ [premiseList, cloudList]  do
  if |l| > 0 then
   p[i] ← new Graph
   if |l| = 1 then
    recNode ← new RecNode
    recNode.operation ← l[0].id
    p[i].startNode ← recNode
    p[i].controlEdges ←
     p[i].controlEdges ∪
       {new ControlEdge(recNode, l[0].firstNode)}
    for all n ∈ l[0] do
     p[i].nodes ← p[i].nodes ∪ {n}
     p[i].controlEdges ← p[i].controlEdges ∪ {n.controlEdges}
    end for
    repNode ← new RepNode
    p[i].nodes ← p[i].nodes ∪ {RepNode}
    p[i].controlEdges ← p[i].controlEdges ∪
       {new ControlEdge(l[0].lastNode, repNode)}
   else if |l| > 1 then
    x ← new ExternalExclusiveChoice
    p[i].startNode ← x
    for all p ∈ l do
     b ← new Graph
     for all n ∈ p do
      p[i].nodes ← p[i].nodes ∪ {n}
      p[i].controlEdges ←
       p[i].controlEdges ∪ {n.controlEdges}
     end for
     repNode ← new RepNode
     b.nodes ← b.nodes ∪ {repNode}
     b.controlEdges ← b.controlEdges ∪
      {new ControlEdge(p.lastNode, repNode)}
     x.branches ← x.branches ∪ {(p.id, b)}
    end for
   end if
  end if
 i ← i + 1
 end for
 cluster ← new Cluster(l[0], l[1])
 return cluster
end function
```

3.4.2.3 Coreografia

A fase de coreografia tem a finalidade de criar as arestas de comunicação necessárias para o GWM emitido pela fase de fragmentação e para os GWMs emitidos pela fase de agrupamento. O Algoritmo 3.5 mostra o pseudocódigo da função *CreateCoreography* que desempenha a tarefa da fase em questão.

Algoritmo 3.5 – Algoritmo da fase de coreografia.

```
function CreateCoreography(ms, cs, ps)
 src ← ms
 dst ← {}
 c ← new Coreography
 if ms.location = OnPremise then
  dst ← cs
 else
  dst ← ps
 end if
 for i ← 1 to 2 do
  for all r ∈ ms.nodesByType(ReqNode) do
   if r.outgoingCommunicationEdges = {} then
    edge ← new CommunicationEdge(r, dst.startNode)
    src.edges ← src.edges ∪ {edge}
    dst.edges ← dst.edges ∪ {edge}
    c.commEdges ← c ∪ {edge}
    if r.nextNode of type GetNode then
     b ← {}
     if dst.startNode of type ExclusiveExternalChoice then
      b ← dst.startNode.branchByOperation(r.operation)
     else
      b ← dst
     end if
     edge ← new CommunicationEdge(
      r.nextNode, b.lastNodeOfType(RepNode))
     c.commEdges ← c ∪ {edge}
    end if
   end if
  end for
  if dst = cs then
   dst ← ps
  else
   dst ← cs
  end if
 end for
 c.mainSubprocess ← ms
 c.cloudSubprocess ← cs
 c.premiseSubprocess ← ps
 return c
end function
```

O subprocesso emitido pela fase de fragmentação é referente ao subprocesso que é acessado diretamente pelos usuários, denominado a partir deste ponto de subprocesso principal. O algoritmo seleciona o subprocesso com a localização contrária a do subprocesso principal e cria a coreografia entre esses. A coreografia parte da seleção de todos os nós do tipo *req* dentro do subprocesso principal que não possuem nós de comunicação de saída. Com base no nome da operação associada ao nó *req* selecionado são criadas as arestas de comunicação necessárias para a definição da coreografia. O mesmo procedimento é realizado entre os subprocesso da nuvem e o outro subprocesso da premissa.

3.4.2.4 Fluxo de Dados

A fase de fluxo de dados tem a finalidade de criar novas arestas de dados para que o fluxo de dados do processo decomposto seja equivalente ao do processo monolítico. O Algoritmo 3.6 mostra o pseudocódigo da função *RefactoryDataFlow* que executa a tarefa da etapa de fluxo de dados.

Algoritmo 3.6 – Algoritmo da fase de fluxo de dados.

```
newDataEdges <- {}

function RefactoryDataFlow(c)
 for all n ∈ c.mainGraph.nodes do
  oldDataEdges <- {}
  for all e ∈ n.outgoingDataEdges - newDataEdges do
   p <- c.DataPathBetween(n, e.destinationNode)
   oldDataEdges <- oldDataEdges ∪ {e}
   BuildNewPath(p.nodes
  end for
  newDataEdges <- {}
  for all e ∈ n.incomingDataEdges - newDataEdges do
   p <- c.DataPathBetween(e.sourceNode, n)
   oldDataEdges <- oldDataEdges ∪ {e}

    BuildNewPath(
     p.nodes ∪ {n}) – {p.startNode}, p.startNode, e)
  end for
  c.mainSubprocess.dataEdges <-
   c.mainSubprocess.dataEdges – {oldDataEdges}
 end for
 return c
end function

procedure BuildNewPath(nodes, ns, nd, e)
 for all nd ∈ nodes do
  d <- new DataEdge
  d.sourceNode <- ns
  d.destinationNode <- nd
  d.dataItems <- e.dataItems
  newDataEdges <- newDataEdges ∪ {d}
  ns <- nd
 end for
end procedure
```

Tal função tem como entrada a coreografia criada pela fase antecedente. Essa entrada é composta pelos GWMs que representam os subprocessos e pelas arestas de comunicação que definem a coreografia entre esses. Apesar do subprocesso principal ser uma fragmentação do processo monolítico, as arestas de dados não foram modificadas em nenhuma das fases anteriores. Portanto, para cada nó e para cada aresta de dados que parte ou chega nele é criado um conjunto

de arestas que parte do mesmo nó de origem, passa pelos novos nós de comunicação e chega ao mesmo nó de destino do fluxo de dados original. A função *DataPathBetween(from, to)* em destaque no Algoritmo 3.6 tem a finalidade de retornar os nós de comunicação necessários para que o fluxo de dados seja criado entre o nó *from* e o nó *to*.

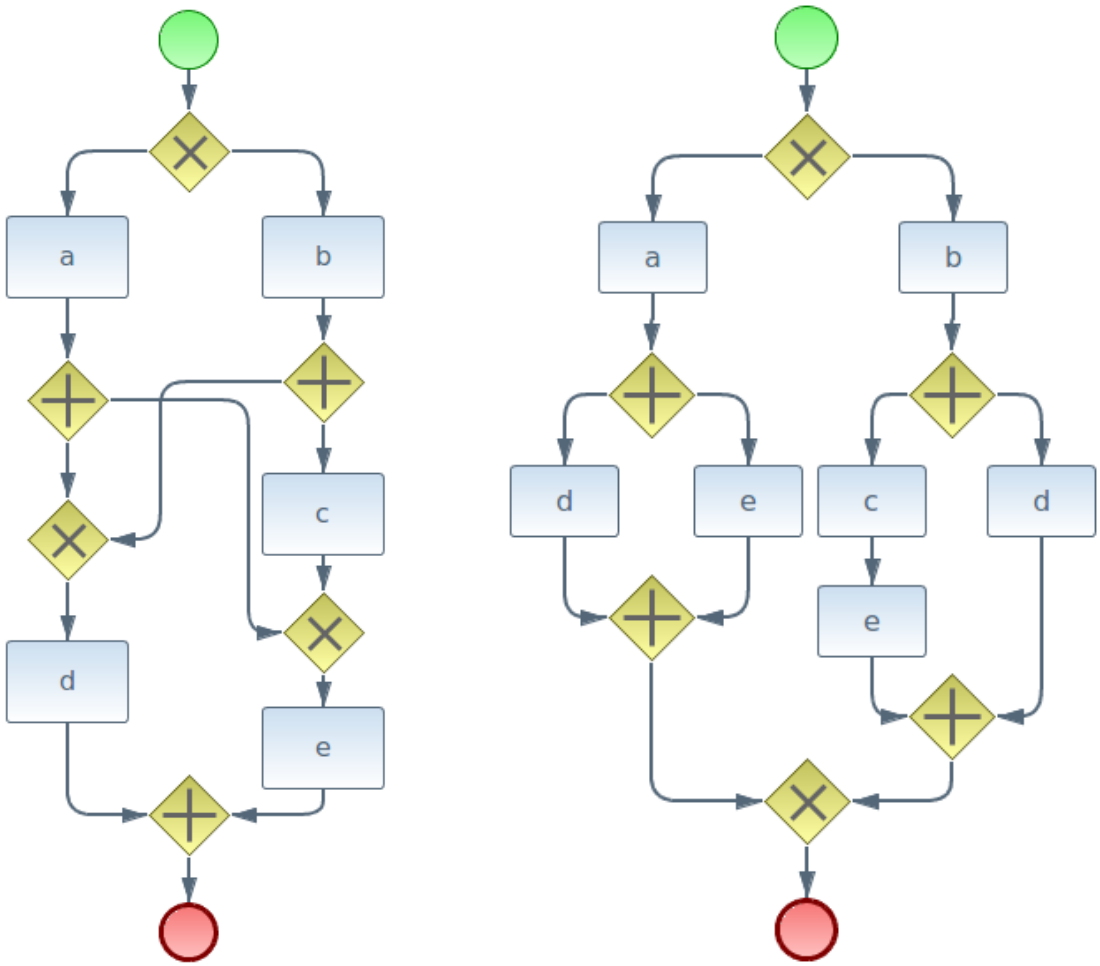
3.5 Lifting e Grounding

As etapas de *lifting* e *grounding* possuem a finalidade de converter a especificação de um processo de negócio em uma definição equivalente em GWM e vice-versa, respectivamente. As estratégias para transformar especificações de processos de negócio em GWM dependem inicialmente da estrutura da linguagem de especificação, a qual pode ser baseada em blocos ou grafos ou serem híbridas (KOPP, MARTIN, *et al.*, 2009).

Linguagens baseadas em blocos possuem obrigatoriamente todos os seus padrões de forma estruturada, ou seja, existe a definição tanto do início quanto do término de um padrão. Linguagens baseadas em grafos, como a BPMN, definem processos de negócio através de links de controle explícitos entre atividades, sem que haja a necessidade de seguir determinados padrões de estrutura. Entretanto, processos de negócio que seguem tais padrões, denominados processos de negócio estruturados, são mais adequados para transformações em outros modelos (POLYVYANYY, GARCÍA-BAÑUELOS, *et al.*, 2014).

Um processo acíclico[2] é estruturado se para todo nó com múltiplas arestas de saída (divisão) existe um nó correspondente com múltiplas arestas de entrada (junção) e vice-versa, de tal forma que os fragmentos entre a divisão e a junção formam um padrão que possui uma única entrada e uma única saída. Caso contrário o processo é não estruturado (POLYVYANYY, GARCÍA-BAÑUELOS, *et al.*, 2014). A Figura 3.28(a) ilustra um processo de negócio não estruturado especificado em BPMN, enquanto a Figura 3.28(b) ilustra sua forma estruturada equivalente.

[2] Processos acíclicos não possuem laços de repetição.

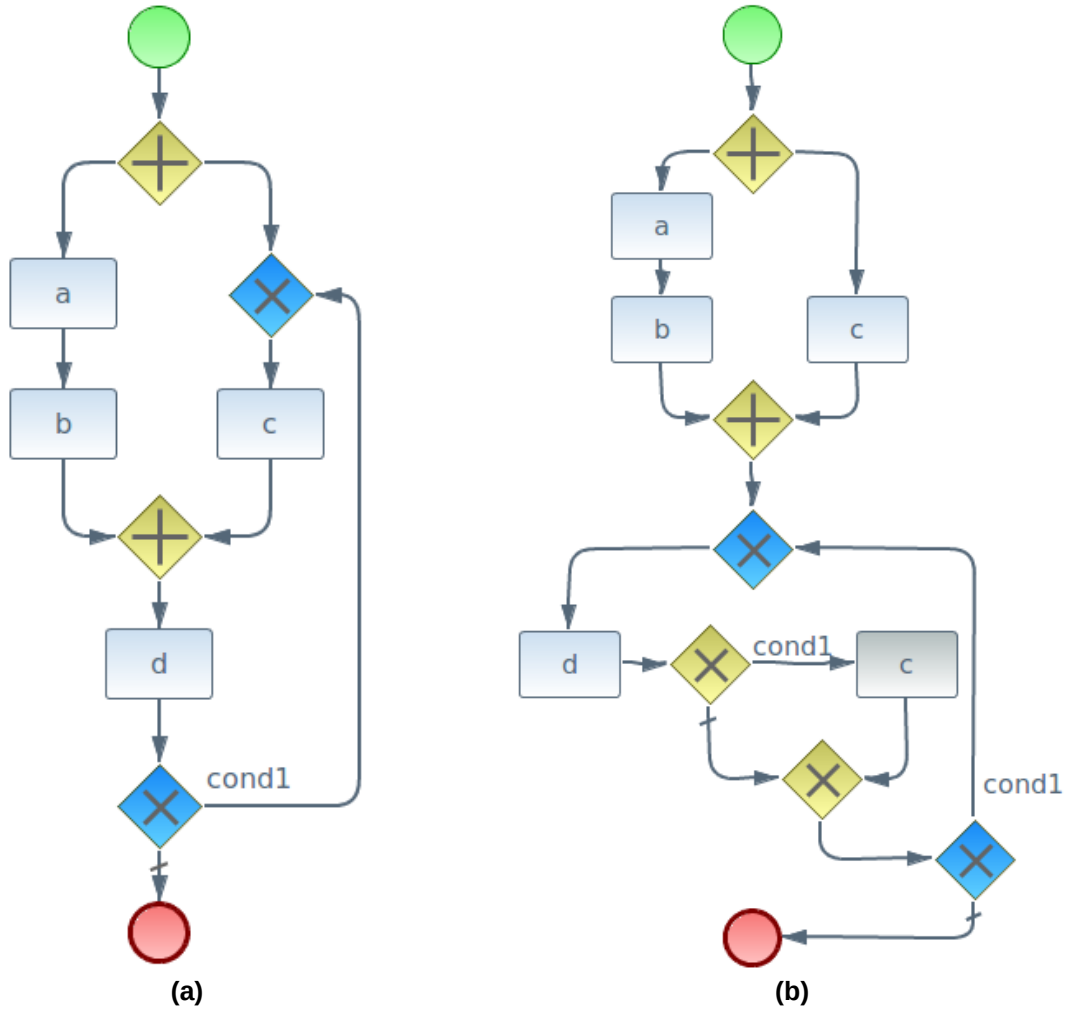


(a) **(b)**

Figura 3.28 – Exemplos de processo de negócio (a) estruturado e (b) não estruturado (POLYVYANYY, GARCÍA-BAÑUELOS, *et al.*, 2014).

Em (POLYVYANYY, GARCÍA-BAÑUELOS, *et al.*, 2014) são definidas provas formais de equivalência para processos de negócio acíclicos como os ilustrados na Figura 3.28, além de algoritmos de transformação entre tais processos. No entanto, processos de negócio cíclicos também devem ser suportados pelas transformações de *lifting* e *grounding*, o que inviabiliza a aplicação de tais algoritmos na abordagem proposta.

Laços de repetição podem ser arbitrários ou estruturados (RUSSELL, HOFSTEDE, *et al.*, 2006). Um laço estruturado possui um único ponto de entrada e um único ponto de saída e seus nós de início e término são diretamente aninhados, quando aninhados, ao mesmo padrão de divisão/junção. Caso contrário o laço é arbitrário. A Figura 3.29(a) ilustra um processo de negócio cíclico especificado em BPMN com um laço de repetição arbitrário, enquanto a Figura 3.29(b) ilustra sua forma estruturada equivalente, definida de forma empírica.



(a) **(b)**

Figura 3.29 – Exemplos de processo de negócio com laço de repetição estruturado (a) e não estruturado (b).

Processos como os ilustrados na Figura 3.29 não possuem uma prova de equivalência até onde se sabe (POLYVYANYY, GARCÍA-BAÑUELOS, *et al.*, 2014) e consecutivamente algoritmos que façam a conversão adequada entre tais processos também não. Dessa forma, todo processo de negócio, especificado em uma linguagem baseada em grafos, utilizado como entrada na etapa de *lifting* e emitido pela etapa de *grounding* pode ser cíclico ou acíclico, sendo que tanto os padrões de divisão/junção quanto os laços de repetição devem ser estruturados.

Para a abordagem proposta foram definidas transformações entre o GWM e as linguagens de especificação BPMN e WS-BPEL. A primeira tarefa para a definição dos algoritmos de *lifting* e *grounding* foi elaborar o mapeamento entre cada tipo de padrão das linguagens adotadas e os aspectos do GWM. Como tanto a linguagem BPMN quanto a WS-BPEL possuem uma padronização baseada na linguagem XML, as transformações de *lifting* e *grounding* recebem e emitem

processos especificados nessa linguagem. Em função da estrutura de árvore da XML, os algoritmos de transformação têm a finalidade de converter estruturas de árvores em grafos e vice-versa. A Figura 3.30 ilustra um exemplo de execução do algoritmo de *lifting* sobre o padrão de desvio condicional das linguagens BPMN e WS-BPEL e do algoritmo de *grounding* sobre o aspecto Ramo Condicional do GWM.

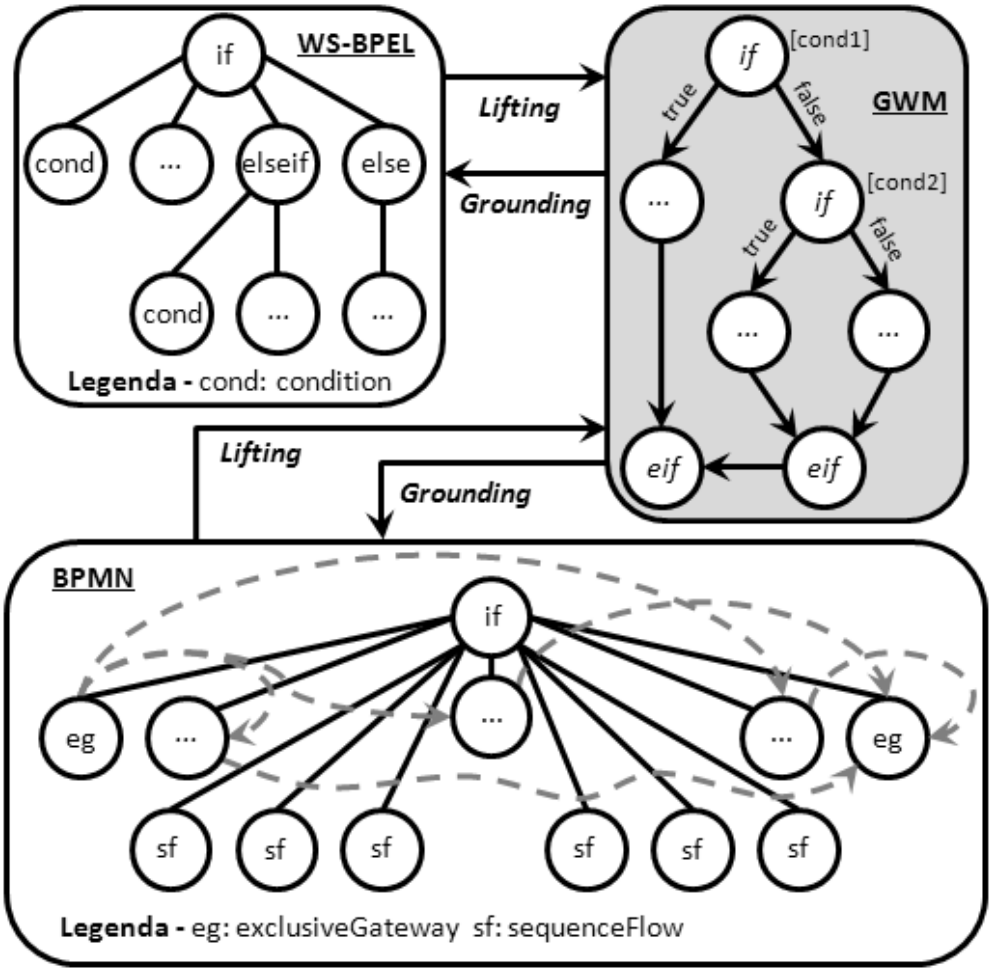


Figura 3.30 – Transformações de *lifting* e *grounding* para os padrões de desvio condicional das linguagens BPMN e WS-BPEL.

Em BPMN a sequência das atividades é definida pelos nós filhos do tipo *incoming* e *outgoing*, sendo que a ordem empregada por tais nós é representada na Figura 3.30 através de linhas cinza tracejadas. Em WS-BPEL a sequência das atividades é definida pela ordem dos nós.

Com a finalidade de generalizar as transformações, foram definidos algoritmos recursivos para o *lifting* e o *grounding*, sendo que tais algoritmos possuem uma parte geral e partes específicas para cada tipo de padrão e aspecto.

A parte geral do algoritmo de *lifting* recebe uma estrutura em árvore e identifica o tipo do nó raiz, chamando uma parte específica apropriada para o tipo identificado, a qual recebe a subárvore a partir do nó raiz. As partes específicas detêm todas as informações sobre a estrutura a partir do nó raiz e continuam em direção aos nós filhos, gerando construções equivalentes em grafo. Quando uma parte específica se depara com um tipo de nó para o qual não é preparada, a parte geral é invocada recursivamente recebendo a subárvore a partir do nó corrente.

A parte geral do algoritmo de *grounding* recebe uma estruturada de grafo GWM e identifica o tipo de aspecto a partir do primeiro nó. O aspecto é enviado como um todo para a parte específica que detém o conhecimento sobre sua estruturada, a qual continua através dos nós aninhados e gera uma estrutura em árvore seguindo os padrões da linguagem de especificação utilizada. As partes específicas chamam recursivamente a parte geral quando um aspecto não esperado pela parte específica é encontrado.

3.5.1 BPMN

A linguagem de especificação de processos de negócio BPMN possui um escopo abrangente e completo. A especificação da versão BPMN 2.0 (OBJECT MANAGEMENT GROUP, 2011) separa os elementos e atributos da linguagem em três subclasses:

- ***Descriptive***, concentrada em elementos e atributos relativos à visualização, os quais são empregados em uma modelagem com alto nível de abstração;
- ***Analytic***, a qual contém todos os elementos da subclasse ***Descriptive*** e elementos e atributos mais complexos; e
- ***Common Executable***, a qual possui todos os elementos e atributos necessários para modelagem de processos de negócio executáveis.

Os algoritmos de *lifting* e *grounding* foram definidos com foco nos elementos e atributos pertencentes à subclasse ***Common Executable***, já que contempla os recursos necessários para a definição de processos de negócio executáveis.

3.5.1.1 Mapeamento

Os padrões *task*, *manual task*, *user task*, *script task, business rule task* e *intermediate throw event*, ilustrados na Figura 3.31, são mapeados como nós de atividades.

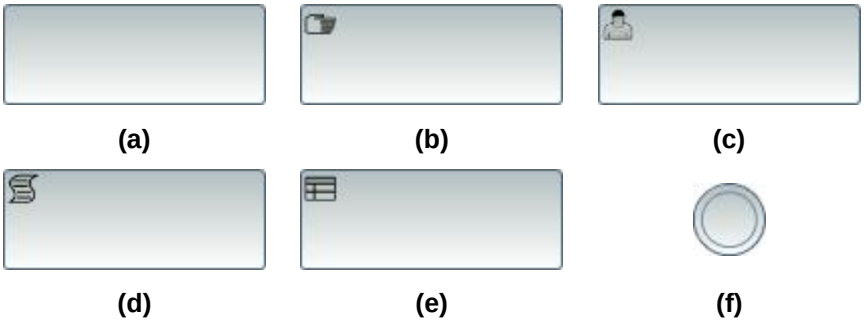


Figura 3.31 – Padrões *task* (a), *manual task* (b), *user task* (c), *script task* (d), *business rule task* (e) e *intermediate throw event* (f).

O padrão para execução sequencial de atividades é definido por arestas conectadas entre atividades, ilustrado na Figura 3.32, é mapeado como um aspecto Sequência.

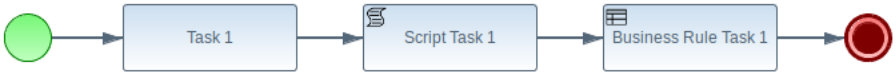


Figura 3.32 – Padrão para execução sequencial de atividades.

O padrão de execução condicional de atividades, ilustrado na Figura 3.33, é definido através de um par de padrões *exclusive gateway*, um para dividir o fluxo de controle e outro para juntá-lo. Tal padrão é mapeado como um aspecto Ramo Condicional.

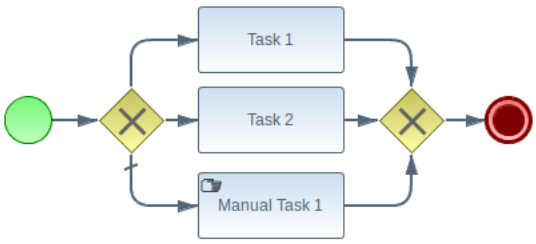


Figura 3.33 – Padrão para execução condicional de atividades.

O padrão de execução paralela de atividades, ilustrado na Figura 3.34, é definido por dois padrões do tipo *parallel gateway*, um para dividir o fluxo de controle e outro para juntá-lo. Tal padrão é mapeado como um aspecto Ramos Paralelos.

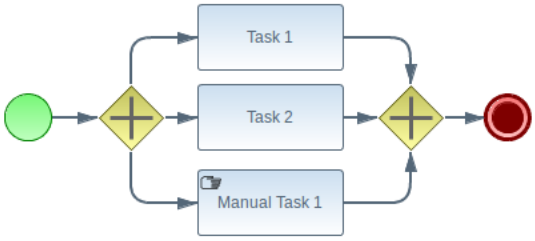


Figura 3.34 – Padrão para execução paralela de atividades.

O padrão de junção parcial, ilustrado na Figura 3.35, é definido por padrões do tipo *parallel gateway* ou *exclusive gateway* aninhados a um padrão de execução paralela. Tal padrão é mapeado como um aspecto Junção Parcial.

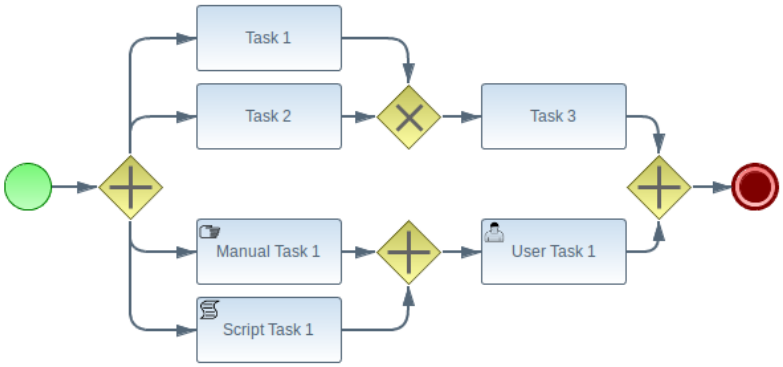


Figura 3.35 – Padrão de junção parcial de atividades paralelas.

O padrão de laço de repetição é definido por um par de padrões do tipo *exclusive gateway*. Tal padrão é mapeado como um aspecto Laço de Repetição. A Figura 3.36 ilustra um padrão de laço de repetição com a condição antes da primeira execução das atividades iterativas, enquanto a Figura 3.37 o ilustra com a condição depois da primeira execução das atividades iterativas.

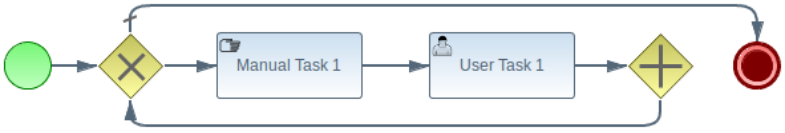


Figura 3.36 – Padrão de laço de repetição com condição antes.

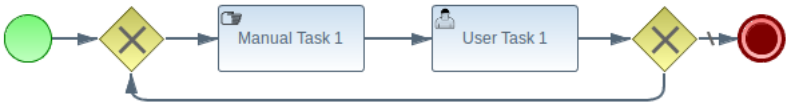


Figura 3.37 – Padrão de laço de repetição com condição depois.

Os padrões de atividades iterativas e atividades com múltiplas instâncias sequenciais, ilustrados pelas Figuras 3.34(a) e 3.34(b), respectivamente, também são mapeados como um aspecto Laço de Repetição.

(a) **(b)**

Figura 3.38 – Padrões de atividades iterativas (a) e atividades com múltiplas instâncias sequenciais (b).

O padrão múltiplas instâncias paralelas, ilustrado na Figura 3.39, é mapeado como um aspecto Múltiplas Instâncias.

Figura 3.39 – Padrão múltiplas instâncias.

O padrão mensagem externa, ilustrado pela Figura 3.40, é definido por padrões do tipo *event-based gateway*, *intermediate catch message* e *exclusive gateway*. O primeiro divide o fluxo de controle, o segundo define as operações que podem ser acessadas por parceiros externos e o terceiro une os fluxos de controle em um único. Tal padrão é mapeado como um aspecto Escolha Exclusiva Externa.

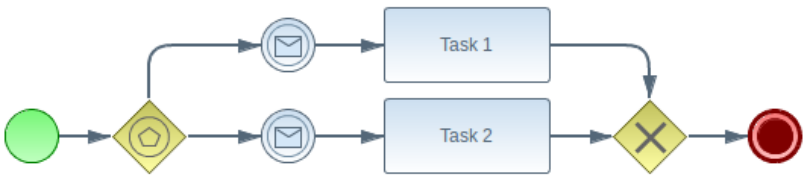


Figura 3.40 – Padrão mensagem externa.

O padrão mensagem externa pode ter aninhado a padrões do tipo *intermediate catch timer*, como ilustrado pela Figura 3.41. Padrões de tal tipo e suas atividades relacionadas são mapeados como um aspecto Tratamento de Tempo Limite.

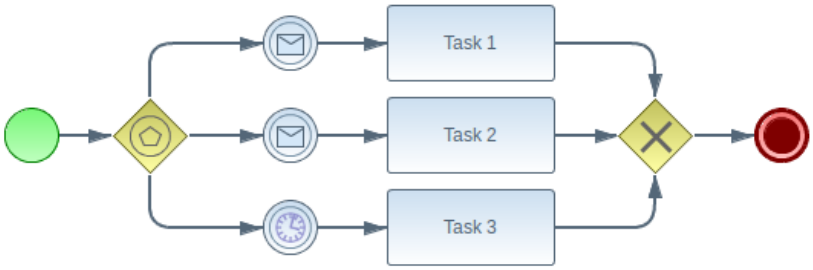


Figura 3.41 – Padrão de mensagem externa.

O padrão do tipo *event-based gateway* pode ser definido como paralelo. Caso isso ocorra, o padrão mensagem externa é mapeado como um aspecto Ramos Paralelos com nós do tipo *rec* aninhados, um para cada padrão *intermediate catch message*. Caso haja um padrão *intermediate catch timer* aninhado, esse é mapeado como um aspecto Tratamento de Tempo Limite relacionado a cada nó *rec*;

O padrão do tipo *end event terminate* é mapeado como um aspecto Finalização Explícita.

Padrões *transaction, subprocess* e *subprocess ad hoc* não possuem um aspecto correspondente, mas definem o escopo dos seus padrões aninhados. Padrões *start event message* e *receive task*, ilustrados na Figura 3.42, são mapeadas como um aspecto Receber Mensagem.

(a) **(b)**

Figura 3.42 – Padrões *start event message* (a) e *receive message task* (b).

Padrões *end event message* e *send task*, ilustrados na Figura 3.43, são mapeados como um aspecto Responder Mensagem.

(a) **(b)**

Figura 3.43 – Padrões *end event message* (a) e *receive task* (b).

O padrão *service task* pode ser mapeado de duas maneiras. A primeira se refere à comunicação sem uma resposta esperada com outro processo ou serviço. Esse tipo de comunicação contém um nó do tipo *req*, um nó do tipo *rec* e um nó de atividade. Entre o primeiro e o segundo nó há uma aresta de comunicação e entre os dois últimos uma aresta de controle. A comunicação com uma resposta esperada é descrita com os mesmos elementos da primeira situação com o acréscimo de dois nós, um do tipo *rep* e outro do tipo *get*. Entre o nó de atividade e o nó *rep* há uma aresta de controle, entre os nós *rep* e *get* uma aresta de comunicação e uma última aresta de controle entre os nós *req* e *get*.

O padrão de tempo limite também pode ser definido por um *boundary event time* atrelado a uma atividade ou subprocesso, como ilustrado na Figura 3.44. Tal padrão é mapeado como um aspecto Tratamento de Tempo Limite.

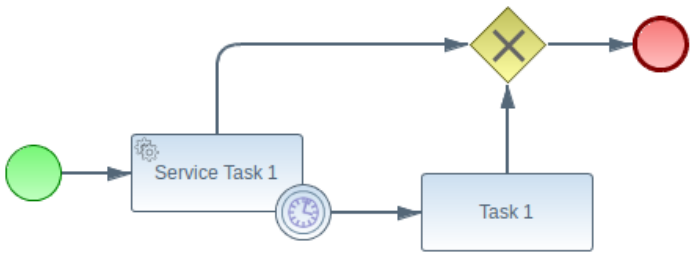


Figura 3.44 – Padrão de tempo limite.

O padrão de gerenciamento de erros, ilustrado na Figura 3.45, é definido através do padrão *boundary event error*, o qual deve ser atrelado a uma atividade ou subprocesso. Tal padrão é mapeado como um aspecto Tratamento de Exceções.

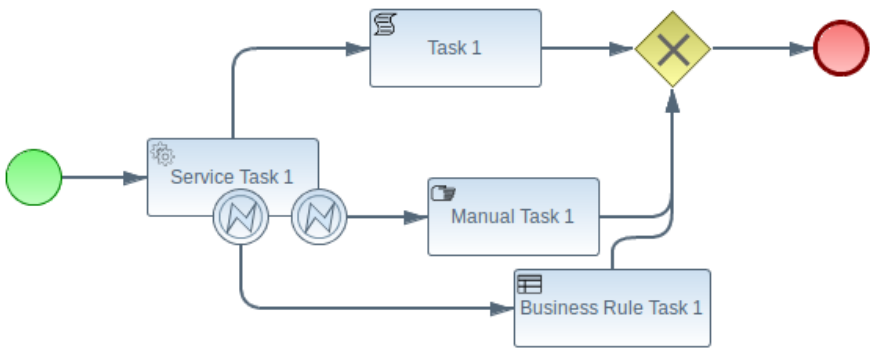


Figura 3.45 – Padrão de gerenciamento de erros.

O padrão de gerenciamento de finalização, ilustrado na Figura 3.46, também utiliza o padrão *boundary event error* para sua definição.

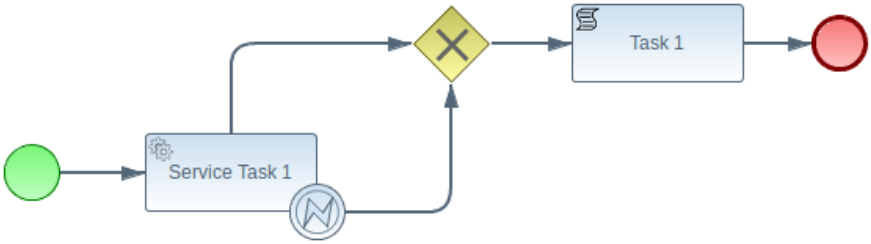


Figura 3.46 – Padrão de gerenciamento de finalização.

O fluxo de dados de processos de negócio especificados em BPMN é extraído a partir de padrões *data object*, *data input* e *data output*, sendo que os detalhes de cada item de dado é extraído da definição XML *itemDefinition*. A relação entre as atividades e os objetos de dados é mapeada como aresta de dados. A Figura 3.47 ilustra o referido padrão.

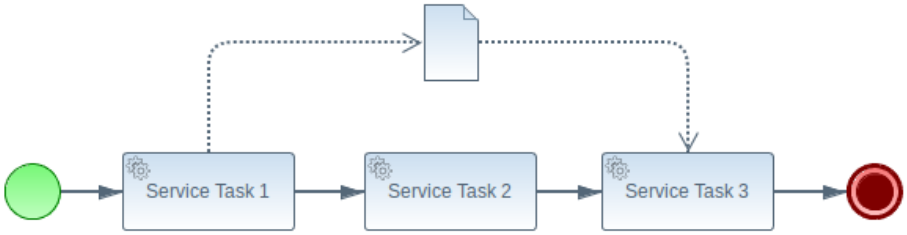


Figura 3.47 – Padrão de troca de dados em BPMN.

A coreografia em BPMN é definida através de arestas de comunicação entre os padrões *start event message*, *receive task, end event message, send message, intermediate throw event message* e *intermediate catch event message*. Tais arestas são mapeadas como arestas de comunicação no GWM. A Figura 3.48 ilustra os padrões de definição de coreografia.

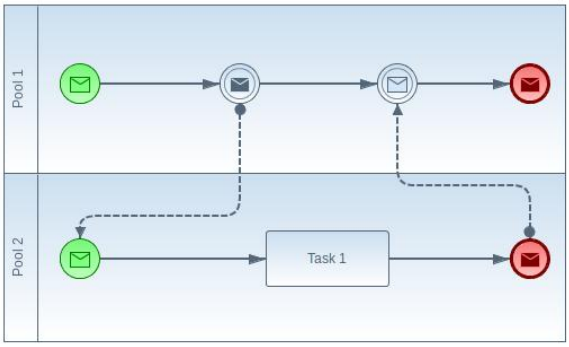


Figura 3.48 – Padrão de comunicação em BPMN.

3.5.1.2 Algoritmo

Esta seção exemplifica os algoritmos de *lifting* e *grounding* para a linguagem BPMN através do padrão de desvio condicional. O Algoritmo 3.7 mostra o pseudocódigo para o *lifting* do padrão de desvio condicional, enquanto o Algoritmo 3.8 mostra o pseudocódigo para o *grounding* do aspecto Ramo Condicional que gera um padrão de desvio condicional correspondente. A Figura 3.30 ilustra padrões e aspectos definidos através dessas transformações.

Em função da estrutura baseada em grafos da linguagem BPMN, foi necessário o emprego de uma variável global *c* para manter as informações de qual nó está sendo analisado, a qual é alterada ao passo que o algoritmo se move sobre os nós. A definição de qual padrão é esperado a partir da identificação do tipo de um conjunto de nós é feita pela parte geral do algoritmo de *lifting*, denominada *Parser*.

A parte específica *ConditionalPartnerParser* referente ao algoritmo de *lifting* verifica o tipo do nó atual e caso esse seja compatível como o início do padrão de desvio condicional a construção de um aspecto Ramo Condicional é iniciada. O primeiro ramo de saída é convertido para um ramo *true* do aspecto. O restante dos ramos são passados para a função *FalseBranchParser*, o qual constrói o ramo *false* ou cria um aspecto Ramo Condicional aninhados ao ramo *false* do primeiro. A função *BranchParser* tem a finalidade caminhar sobre cada atividade de um ramo e convertê-lo em um ramo GWM correspondente.

A parte específica *IfGenerator* referente ao algoritmo de *grounding* da linguagem BPMN cria um ramo aninhado a um par de nós do tipo *exclusive gateway* para cada ramo do aspecto Ramo Condicional. Caso existam aspectos do tipo Ramo Condicional aninhados aos ramos *false*, seus ramos corresponderão a mais ramos aninhados aos nós *exclusive gateway*.

Algoritmo 3.7 – Algoritmo de *lifting* para o padrão de desvio condicional da linguagem BPMN.

```
graph <- new Graph
c <- CurrentNode()

procedure ConditionalPartnerParser()
 cond ← {}
 if c of type ExclusiveGateway and c.type = Diverging
      and |c.incomingEdges| = 1 and |c.outgoingEdges| > 0 then
  cond ← new IfGraph()
  cond.trueBranch ←
   BranchParser(
    c.incomingEdges.get(0), |c.outgoingEdges|)
  cond.falseBranch ←
   FalseBranchParser(
    c.incomingEdges − {c.incomingEdges.get(0),
    c.defaultOutgoingEdge, |c.outgoingEdges|}
  graph.nodes = graph.nodes ∪ {cond}
 end if
end procedure

function FalseBranchParser(e, d, n)
  if |e| > 1 then
   cond ← new IfGraph
   cond.condition ← e.condition
   cond.trueBranch ← BranchParser(e.get(0), n)
   cond.falseBranch ← FalseParser(e − {e.get(0)}, d, n)
   return cond
  else if |e| = 1 and d ≠ {} then
   cond ← new IfGraph
   cond.condition ← e.condition
   cond.trueBranch ← BranchParser(e.get(0), n)
   cond.falseBranch ← BranchParser(d, n)
   return cond
  else if |e| = 1 and d = {} then
   cond ← new IfGraph
   cond.condition ← e.condition
   cond.trueBranch ← BranchParser(e.get(0), n)
   return cond
  else if d ≠ {} then
   return BranchParser(d, n)
  end if
end function

function BranchParser(e, n)
 c ← e.destinationNode
 b ← new Branch
 while not (c of type ExclusiveGateway and c.type = Diverging and
                      |c.incomingEdges| = n and |c.incomingEdges| = 1) do
  node ← Parser() ◄ this instruction identifies the type of, parses and moves the current node.
  if |b.nodes| > 0 then
   b.controlEdges ← b.controlNodes ∪ new ControlEdge(b.lastNode, node)
   b.nodes ← b.nodes ∪ {node}
  else
   b.startNode ← node
   b.nodes ← {node}
  end if
 end while
 return b
end function
```

Algoritmo 3.8 – Algoritmo de *grounding* do aspecto Ramos Condicionais para BPMN.

```
function IfGenerator(g)
 t ← IfTree()
 t.children ← t.children ∪ {Generator(g.true)}
 t.children ← t.children ∪ {FalseGenerator(g.false)}
end function

function FalseGenerator(f)
 r ← {}
 while f ≠ {} do
  if  |f.nodes| = 1 ∧ f type of ElseIfTree then
   t ← ElseIfTree()
   t.children ← CondGenerator(f.cond) ∪ Generator(f.true)
   r ← r ∪ t
  else
   r ← r ∪ ElseGenerator(f)
  end if
  f ← f.false
 end while
 return r
end function
```

3.5.2 WS-BPEL

Apesar da linguagem WS-BPEL ser híbrida, a maioria das suas construções são baseadas em blocos. Dessa forma, seus algoritmos de *lifting* e *grounding* possuem estratégias focadas nesse tipo de estrutura.

As construções dessa linguagem podem ser aplicadas de duas formas: abstrata ou executável: Um processo de negócio abstrato é um processo especificado parcialmente e deve ser explicitamente definido como *abstract*. Processos de negócio executáveis possuem todos os detalhes necessários para que mecanismos de processo consiguam gerar instâncias desses.

Como a intenção é implantar os subprocessos em determinados mecanismos e executá-los para verificar possíveis vantagens na decomposição, o foco das transformações de *lifting* e *grounding* é em processos de negócio executáveis.

3.5.2.1 Mapeamento

O elemento *sequence* é mapeado como um conjunto de aspectos interconectados por arestas de controle. Tais aspectos representam os padrões aninhados ao *sequence*. Os elementos *assign*, *wait*, *empty*, *throw*, *rethrow* e *extensionActivity* são mapeados como nós de atividade.

O elemento *if* é mapeado como um aspecto Ramo Condicional, sendo que elementos com mais de uma condição, ou seja, que contêm elementos *elseif*, são mapeados com aspectos Ramo Condicional aninhados junto ao ramo *false*.

O elemento *flow* é mapeado como um aspecto Ramos Paralelos. Os elementos *links*, *sources* e *targets* aninhados ao elemento *flow* podem ser mapeados como o aspecto Junção Parcial, caso a definição de tais elementos tenha a mesma finalidade de tal aspecto. Caso não sejam aninhados ao elemento *flow* ou não tenham a finalidade esperada, tais elementos são mapeados como atributos do elemento ao qual pertencem, para que não haja perda de informação durante a decomposição.

Os elementos *while* e *repeatUntil* são mapeados como um aspecto Laço de Repetição. O primeiro deles possui a avaliação da condição antes das atividades iterativas, enquanto o segundo possui a avaliação da condição após. O elemento *forEach* pode ser mapeado de duas formas. Caso o seu atributo *parallel* tenha o valor *no*, esse elemento é mapeado como um aspecto Laço de Repetição com a avaliação da condição antes das atividades iterativas. Caso o atributo tenha o valor *yes*, o elemento é mapeado como um aspecto Múltiplas Instâncias.

O elemento *pick* é mapeado como um aspecto Escolha Exclusiva Externa. Caso haja um elemento *onAlarm* aninhado, esse é mapeado como um aspecto Tratamento de Tempo Limite. O elemento *exit* é mapeado como um aspecto Finalização Explícita. O elemento *scope* define o escopo dos elementos aninhado a ele e é mapeado de formal identitica ao elemento *sequence*.

Os elementos *receive* e *reply* são mapeados por nós de comunicação dos tipos *rec* e *rep*, respectivamente. O elemento *invoke* é mapeado igualmente ao padrão *service task* da linguagem BPMN.

O elemento *eventHandlers* é mapeado como um aspecto Ramos Paralelos com nós do tipo *rec* aninhados, um para cada elemento *onEvent*. Caso haja um elemento *onAlarm* aninhado, esse é mapeado como um aspecto Tratamento de Tempo Limite relacionado a cada nó *rec*. O elemento *faultHandlers* é mapeado como um aspecto Tratamento de Exceções. O elemento *catchAll* corresponde ao ramo rotulado como *otherwise*. O elemento *terminationHandler* é mapeado como um aspecto Tratamento de Finalização.

Os padrões de dados da linguagem WS-BPEL partem do mapeamento dos elementos *variable* para os aspectos Variáveis Locais ou Variáveis Globais. A partir

desse mapeamento são identificadas quais atividades se relacionam com quais variáveis, e então esses relacionamentos são mapeados como arestas de dados.

A coreografia em WS-BPEL é feita através dos elementos *invoke*, *receive*, *reply e pick,* os quais definem e acessam interfaces de comunicação de e em parceiros externos.

3.5.2.2 Algoritmo

O Algoritmo 3.9 mostra o pseudocódigo para o *lifting* do padrão *if*, enquanto o Algoritmo 3.10 mostra o pseudocódigo para o *grounding* do aspecto Ramo Condicional A Figura 3.30 ilustra padrões e aspectos definidos através dessas transformações.

Algoritmo 3.9 – Algoritmo de lifting para o padrão *if* da linguagem WS-BPEL

```
function IfParser(t)
 cond ← {}
 if t type of IfTree then
  cond ← IfGraph()
  for all c ∈ t.children do
   if c type of ExceptionTree then
    ExceptionPart(c)
   else if c type of Condition then
    cond.cond ← CondParser(c)
   else if c type of ElseTree V c type of ElseIfTree then
    cond.false ← FalseParser(t.children)
    return cond
   else if c type of Tree then
     cond.true ← Parser(c)
   end if
   t.children ← t.children − {c}
  end for
 end if
 return cond
end function

function FalseParser(s)
 if s = {} then
  return s
 end if
 falseBranch ← Graph()
 if s.first type of ElseIfTree then
  cond ← IfBranch()
  cond.true ← ElseParse(s.first)
  cond.false ← FalseParse(s-{s.first})
  falseBranch.nodes ← {cond}
 else if s.first type of ElseTree then
  falseBranch.nodes ← {ElseParse(s.first)}
 else
  return FalseParse(s-{s.first})
 end if
 return falseBranch
end function
```

Algoritmo 3.10 – Algoritmo de *grounding* do aspecto Ramos Condicionais para WS-BPEL.

```
function IfGenerator(g)
 t ← IfTree()
 t.children ← t.children ∪ {CondGenerator(g.cond)}
 t.children ← t.children ∪ {Generator(g.true)}
 t.children ← t.children ∪ {FalseGenerator(g.false)}
end function

function FalseGenerator(f)
 r ← {}
 while f ≠ {} do
  if  |f.nodes| = 1 ∧ f type of ElseIfTree then
   t ← ElseIfTree()
   t.children ← CondGenerator(f.cond) ∪ Generator(f.true)
   r ← r ∪ t
  else
   r ← r ∪ ElseGenerator(f)
  end if
  f ← f.false
 end while
 return r
end function
```

A parte específica *IfParser* referente ao algoritmo de *lifting* caminha através dos nós aninhados na árvore *if*, verificando a condição de execução e construindo o ramo rotulado *true* do grafo *if*, correspondente ao aspecto Ramo Condicional, com as atividades relacionadas. A parte *FalseParser* trabalho sobre o ramo rotulado *false*. Caso a árvore tenha mais de uma condição, o ramo *false* conterá um grafo *if* para a segunda condição, e esse conterá um grafo *if* aninhado ao ramo *false* para a terceira condição e assim por diante.

A parte específica *IfGenerator* referente ao algoritmo de *grounding* caminha sobre o ramo *true* do grafo *if*, verificando e adicionando a árvore *if* a condição de execução com as atividades relacionadas. O ramo *false* é processado pela parte específica *FalseGenerator*, a qual verifica se há um grafo *if* aninhado. Nesse caso, uma árvore *elseif* com uma condição de execução e as atividades relacionadas é adicionada a árvore *if* e seu ramo *false* é direcionado para ser processado recursivamente pela parte *FalseGenerator*. Caso contrário, as atividades restantes são aninhadas a uma árvore *else*.

Capítulo 4

FORMALIZAÇÃO DO MODELO INTERMEDIÁRIO

Este capítulo apresenta a formalização do modelo intermediário. A Seção 4.1 introduz os conceitos da álgebra de processos π-Calculus; a Seção 4.2 descreve a formalização da estrutura do modelo intermediário; e a Seção 4.3 apresenta a formalização da semântica do modelo intermediário.

4.1 π-Calculus

π-Calculus (MILNER, PARROW e WALKER, 1992) é uma extensão do *Calculus of Communication Systems (CCS)* (MILNER, 1980) e é baseado em três conceitos principais: agentes, transições e ações.

Para a definição de agentes é assumido um conjunto infinito de nomes, sendo que nome é um termo genérico que se refere indistintamente a variáveis, constantes, valores ou links: *t*, *u*, *v*, *w*, *x*, *y*, e *z* são meta-variáveis sobre nomes, e *P*, *Q*, e *R* são meta-variáveis sobre agentes. A sintaxe de agentes pode ser resumida como as

$$P ::= \mathbf{0} \mid \bar{x}y.P \mid x(y).P \mid \tau.P \mid P + Q \mid P|Q \\ (x)P \mid [x = y]P \mid !P \mid A(y_1, \dots, y_n) \tag{1}$$

onde:

- **0** é um agente vazio, o qual não pode executar quaisquer ações;

- $\bar{x}y.P$ é um prefixo de saída, o qual representa um agente que emite um nome y através do nome $\bar{x}$, e se comporta como P. Se não há tal saída, esse agente é descrito como $\bar{x}.P$;
- $x(y).P$ é um prefixo de entrada, o qual representa um agente que recebe um nome arbitrário z através do nome x, e se comporta como $P\{z/y\}$, onde todas as ocorrências livres de y em P são substituída por z. Se não há tal entrada, esse agente é descrito como $x.P$;
- $\tau.P$ é um prefixo silencioso, o qual representa um agente que pode desenvolver-se para P sem interagir com seu ambiente;
- $P + Q$ é o somatório, o qual representa um agente que se comporta exclusivamente como P ou Q. $\sum_{i=1}^{n} P_i = P_1 + P_2 + \cdots + P_n$, onde para $n = 0$ tal padrão se comporta como um agente vazio ***0***;
- $P|Q$ é a composição, a qual representa um agente que se comporta concorrentemente como P e Q. $\prod_{i=1}^{n} P_i = P_1 | P_2 | \ldots | P_n$, onde para $n = 0$ tal padrão se comporta como um agente vazio ***0***;
- $(x)P$ é uma restrição, a qual representa um agente que se comporta como P, onde ações com o seu ambiente através do nome x são proibidas, enquanto entre os componentes de P são permitidas;
- $[x = y]P$ é uma condição, a qual representa uma agente que se comporta como P se x e y forem nomes idênticos, caso contrário se comporta como um agente vazio ***0***;
- $!P$ é uma replicação, a qual representa um agente que pode sempre criar cópias de P; e
- $A(y_1, \ldots, y_n)$ é um agente definido com uma equação de definição $A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P$, a qual representa um agente que se comporta como $P\{y_1/\mathrm{x}_1, \ldots, y_n/x_n\}$, onde todas as ocorrências livres de x_i em P são substituídas por y_i.

Uma transição em π-*Calculus* é definida na forma $P \xrightarrow{\alpha} Q$, significando que P pode desenvolver-se para Q executando uma ação α, a qual pode ser de um dos quatro tipos a seguir: ação silenciosa τ; ação de entrada $x(y)$; ação de saída livre $\bar{x}y$;

e ação de saída vinculada $\bar{x}(y)$. Seja o conjunto de nomes de um agente P definido como $n(P) = fn(P) \cup bn(P)$, onde $fn(P)$ é o conjunto de nomes livres de P e $bn(P)$ é o conjunto de nomes vinculados de P, as relações de transições sobre agente são as menores relações satisfazendo as principais regras de ação no Quadro 4.1.

Quadro 4.1 – Principais regras de ação.

Tau action $\dfrac{-}{\tau.P \xrightarrow{\tau} P}$	***Output action*** $\dfrac{-}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$
Input action $\dfrac{-}{x(z).P \xrightarrow{x(w)} P\{w\backslash z\}}, w \in fn((z)P)$	
Sum $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	***Match*** $\dfrac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$
Parallel $\dfrac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}, bn(\alpha) \cap fn(Q) = \emptyset$	
Communication $\dfrac{P \xrightarrow{\bar{x}y} P' Q \xrightarrow{x(z)} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'\{y\backslash z\}}$	***Close*** $\dfrac{P \xrightarrow{\bar{x}(w)} P' Q \xrightarrow{x(w)} Q'}{P\|Q \xrightarrow{\tau} (w)(P'\|Q')}$
Restriction $\dfrac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'}, y \in n(\alpha)$	

4.2 Formalização da Estrutura do Modelo Intermediário

A estrutura do GWM é formalmente definida por uma 5-tupla $(N, E, L, \Omega, \omega)$, onde N é um conjunto de nós, E é um conjunto de arestas, L é um conjunto de rótulos, Ω é uma tupla de funções, e ω é uma função parcial. Para a completa

definição de cada elemento da 5-tupla, assume-se que $T_n = \{Act\} \cup T_{ctrl} \cup T_{comm} \cup T_{excp}$ é um conjunto de tipos de nós, onde:

- Act é um tipo genérico de nó;
- $T_{ctrl} = \{if, eif, par, epar, and.or, xor, exor, loop, exit\}$ é um conjunto de tipos de nós de controle;
- $T_{comm} = \{rec, rep, req, get\}$ é um conjunto de tipos de nó de comunicação; e
- $T_{excp} = \{exp, eexp, ddl, eddl, fin, efin\}$ é um conjunto de tipos de nós de exceção.

Portanto, o conjunto de nós $N = N_{act} \cup N_{ctrl} \cup N_{comm} \cup N_{excp}$, onde:

- $N_{act} = \{(id, t): id \in L \wedge t \in \{Act\}\}$ é um conjunto de nós de atividades;
- $N_{ctrl} = \{(id, t): id \in L \wedge t \in T_{ctrl}\}$ é um conjunto de nós de controle;
- $N_{comm} = \{(id, t): id \in L \wedge t \in T_{comm}\}$ é um conjunto de nós de comunicação; e
- $N_{excp} = \{(id, t): id \in L \wedge t \in T_{excp}\}$ é um conjunto de nós de exceção.

Seja $T_e = \{ctrl, data, comm, excp\}$ um conjunto de tipos de arestas. Então, o conjunto de arestas $E = E_{ctrl} \cup E_{data} \cup E_{comm} \cup E_{excp}$, onde:

- $E_{ctrl} = \{(id, t, n_{src}, n_{dest}): id \in L \wedge t = crtl \in T_e \wedge n_{src}, n_{dest} \in N\}$ é um conjunto de arestas de controle;
- $E_{data} = \{(id, t, n_{src}, n_{dest}): id \in L \wedge t = data \in T_e \wedge n_{src}, n_{dest} \in N - N_{excp}\}$ é um conjunto de arestas de dados;
- $E_{comm} = \{(id, t, n_{src}, n_{dest}): id \in L \wedge t = comm \in T_e \wedge n_{src}, n_{dest} \in N_{comm}\}$ é um conjunto de arestas de comunicação; e
- $E_{excp} = \{(id, t, n_{src}, n_{dest}): id \in L \wedge t = excp \in T_e \wedge n_{src} \in N - N_{excp} \wedge n_{dest} \in N_{excp}\}$ é um conjunto de arestas de exceção.

A tupla de funções $\Omega = (\delta, \zeta, \lambda, \sigma, \phi, \chi, \iota, \kappa, o, \psi)$, onde:

- $\delta: E_{data} \rightarrow (2^L)^+$ é uma função que atribui um conjunto de rótulos, representando item de dados, para uma aresta de dados, onde $(2^L)^+$ é o conjunto potência de L sem o elemento $\emptyset$;

- $\zeta: N_{if} \rightarrow L$ é uma função que atribui um rótulo, representando uma sentença lógica, a um nó *if*, onde $N_{if} \subseteq N_{ctrl}$ é um conjunto de nós *if*;
- $\lambda: E_{if} \rightarrow L$ é uma função que atribui um rótulo, representando um valor Booleano, a uma aresta de controle de saída de um nó *if,* onde E_{if} é o conjunto de arestas de saída de nós *if*;
- $\sigma: N_{loop} \rightarrow L$ é uma função que atribui um rótulo, representando uma sentença lógica, para um nó *loop*, onde $N_{loop} \subseteq N_{ctrl}$ é o conjunto de nós *loop*;
- $\phi: E_{loop} \rightarrow L$ é uma função que atribui um rótulo, representando um valor Booleano, para um aresta de controle de saída de um nó *loop*, onde E_{loop} é um conjunto de arestas de controle de saída de nós *loop;*
- $\chi: E_{cxor} \rightarrow L$ é um função que atribui um rótulo, representando um nome de operação, para uma aresta de controle de saída de um nó *xor*, onde E_{cxor} é o conjunto de arestas de controle de saída de nós *xor*;
- $\iota: E_{comx} \rightarrow L$ é uma função que atribui um rótulo, representando um nome de operação, para uma aresta de entrada de um nó *xor*, onde E_{comx} é um conjunto de arestas de entrada de nós *xor*;
- $\kappa: E_{cexp} \rightarrow L$ é uma função atribuindo o rótulo, representando um tipo de exceção, para uma aresta de controle de saída de um nó *exp*, onde E_{cexp} é um conjunto de arestas de controle de saída de nós *exp*;
- $o: E_{eexp} \rightarrow L$ é uma função que atribui um rótulo, representando um tipo de exceção, para uma aresta de exceção de entrada de um nó *exp*, onde E_{eexp} é um conjunto de arestas de exceção de entrada de nós *exp*; e
- $\psi: E_{ddl} \rightarrow L$ é uma função que atribui um rótulo, representando um tempo limite, a uma aresta de exceção de entrada de um nó *ddl*, onde E_{ddl} é o conjunto de arestas de exceção de entrada de nós *ddl*.

A função parcial $\omega: N_{act} \rightarrow M$ atribui uma tupla de múltiplas instâncias paralelas *M = (start, end, sync)* a um nó de atividade, onde:

- *start, end* ∈ *L* são rótulos representando números inteiros, sendo o número de instâncias paralelas definido por *end - start*; e

- *sync* é um rótulo representando um valor Booleano definindo se as instâncias são sincronizadas ou não antes que o nó subsequente seja disparada.

4.3 Formalização da Semântica do Modelo Intermediário

Com a finalidade de formalizar a semântica do GWM é assumido que cada nó é um agente *π-Calculus* independente (PUHLMANN e WESKE, 2005), e cada aresta é um link independente. A semântica do nó de atividade é definida como

$$ACT(x, y) \stackrel{\text{def}}{=} x.\tau.\bar{y} \tag{11}$$

onde um nó de atividade é disparado através do link x, executa uma atividade interna τ e dispara um nó subsequente através do link y quando requerido.

A seguir é apresentada a formalização da semântica do GWM por meio de cada aspecto simples introduzido na Seção 3.2.3.

4.3.1 Fluxo de Controle

A semântica do aspecto Sequência é definida como

$$SEQ(\tilde{x}) \stackrel{\text{def}}{=} \prod_{i=1}^{n} x_i.\tau_i.\overline{x_{i+1}} \tag{12}$$

onde $\tilde{x}$ representa um conjunto de n links, o nó de atividade i é disparado via link x_i, executa uma ação interna τ_i e dispara um nó de atividade $i + 1$ via link x_{i+1}.

A semântica do aspecto Ramo Condicional é definida como

$$
\begin{aligned}
&CBC(x,y,w) \stackrel{\text{def}}{=} IF(x,y) \mid EIF(y,w)\\
&IF(x,y) \stackrel{\text{def}}{=} x.\tau_{if}.(P+Q).\overline{y}\\
&EIF(y,w) \stackrel{\text{def}}{=} y.\tau_{eif}.\overline{w}
\end{aligned}
\tag{13}
$$

onde o nó *if* é disparado via link x, executa uma ação interna τ_{if}, divide um ramo em dois, os quais são representados pelos agentes P e Q e dispara um deles. O nó *eif* junta esses ramos via link y, executa uma ação interna τ_{eif} e dispara um nó subsequente via link w.

A semântica do aspecto Ramos Paralelos é definida como

$$
\begin{aligned}
&PBC(x,\tilde{y},w) \stackrel{\text{def}}{=} PAR \mid EPAR\\
&PAR(x,\tilde{y}) \stackrel{\text{def}}{=} x.\tau_{par}.\left(\prod_{i=1}^{n} B_i\,.\,\overline{y_i}\right)\\
&EPAR(\tilde{y},w) \stackrel{\text{def}}{=} \left(\prod_{i=1}^{n} y_i\right).\tau_{epar}.\overline{w}
\end{aligned}
\tag{14}
$$

onde o nó *par* é disparado via link x, executa uma ação interna τ_{par}, divide um ramo em n ramos, os quais são representados pelos agentes $B_1, \ldots, B_n$. O nó *epar* junta os ramos paralelos via links $y_1, \ldots, y_n$, executa uma ação interna τ_{epar} e ativa um nó subsequente via link w.

A semântica do aspecto Junção Parcial é definida por meio da semântica dos nós *and* e *or*. A semântica do nó *and* é definida como

$$
AND(\tilde{x},y) \stackrel{\text{def}}{=} \left(\prod_{i=1}^{n} x_i\right).\tau_{and}.\overline{y} \tag{15}
$$

onde esse nó junta n ramos paralelos via links $x_1, \ldots, x_n$, executa uma ação interna τ_{and} e dispara um nó subsequente via link y.

A semântica do nó *or* é definida como

$$OR(\tilde{x}, y) \stackrel{\text{def}}{=} \left(\sum_{i \in I} x_i\right).\tau_{or}.\bar{y} \tag{16}$$

onde esse nó junta n ramos paralelos via a link x_i, executa uma ação interna τ_{or} e dispara um nó subsequente via link y.

A semântica do aspecto Laço de Repetição é definida como

$$\begin{aligned} &LC(x, y, z) \stackrel{\text{def}}{=} LOOP(x, y, z) \mid IB(y, x) \\ &LOOP(x, y, z) \stackrel{\text{def}}{=} !\left(x.\tau_{loop}.(\bar{y} + \bar{z})\right) \\ &IB(y, x) \stackrel{\text{def}}{=} !\,(y.B.\bar{x}) \end{aligned} \tag{17}$$

onde o nó *loop* é disparado via link x, executa uma ação interna τ_{loop} e dispara o ramo iterativo B via link y ou um nó subsequente via link z. No primeiro caso, B é executado e o nó *loop* é disparado novamente via link x. Essa definição é independente da localização do nó *loop* referente ao ramo iterativo, uma vez que permite que o primeiro disparo seja do nó *loop* ou do ramo iterativo.

A semântica do aspecto Escolha Exclusiva Externa é definida como

$$\begin{aligned} &EEC(u, \tilde{v}, w, x, y, z) \stackrel{\text{def}}{=} XOR \mid EXOR \\ &XOR(u, \tilde{v}, w, x, y) \stackrel{\text{def}}{=} x.w(u).\tau_{xor}.\bar{u} \mid \left(\sum_{i=1}^{n} v_i.B_i\right).\bar{y}, \;\; \forall u \in \tilde{v} \\ &EXOR(y, z) \stackrel{\text{def}}{=} y.\tau_{exor}.\overline{w} \end{aligned} \tag{18}$$

onde o nó *xor* é disparado via link x, recebe um link u a partir de um parceiro externo via link w, executa uma ação interna τ_{xor}, divide um ramo em n ramos, os quais são representados pelos agentes $B_1, \ldots, B_n$, e dispara um ramo B_i via link u. O nó *exor* junta esses ramo via link y, executa uma ação interna τ_{exor} e dispara um nó subsequente via link w.

A semântica do aspecto Finalização Implícita define que uma instância de workflow é finalizada quando não há mais nós em execução ou esperando para ser disparado.

Com a finalidade de força a finalização de um workflow composto por n nós, todo nó A é assumido ser da forma $A + \mathcal{E}$ com $\mathcal{E}$ sendo um agente de cancelamento $cancel_A.\tau_{cancel}$. Portanto, a semântica do aspecto Finalização Explícita é definida como

$$EXIT(x, \tilde{c}) \stackrel{\text{def}}{=} x.\tau_{cancel}.\prod_{i=1}^{n} \overline{cancel_i}, \quad \forall cancel_i \in \tilde{c} \tag{19}$$

onde o nó *exit* é disparado via link x, executa uma ação interna τ_{cancel} e dispara todos os agentes de cancelamento de todos os nó via links $cancel_1, \ldots, cancel_n$.

A semântica do aspecto Múltiplas Instâncias assíncrono é definida como

$$AMIC(x, z) \stackrel{\text{def}}{=} x.\left(\prod_{s}^{e} B \mid \bar{z}\right) \tag{20}$$

onde B representa um nó que é disparado via link x, então disparada um nó subsequente via link z, e possui e-s+1 instâncias paralelas sem sincronização.

A semântica do aspecto Múltiplas Instâncias síncrono é definida como

$$SMIC(x, \tilde{y}) \stackrel{\text{def}}{=} x.\left(\prod_{i=s}^{e} B.\overline{y_i}\right) \mid \left(\prod_{i=s}^{e} y_i\right).\bar{z} \tag{21}$$

onde B representa um nó disparado via link x, então dispara um nó subsequente via link z, e possui e-s+1 instâncias paralelas sem sincronização via links $y_s, \ldots, y_e$.

4.3.2 Fluxo de Dados

O aspecto Variável Local permite definir variáveis as quais podem somente ser acessadas por um conjunto restrito de nó. Então, $d \in bn(P)$ significa que uma variável local d pertence ao conjunto de variáveis do agente P.

O aspecto Variável Global permite definir variáveis que podem ser acessadas por qualquer nó no workflow. Então, $(\forall P)\big(d \notin bn(P)\big)$ significa que uma variável

global d não pode pertencer a um conjunto de variáveis atreladas de qualquer agente P.

A semântica do aspecto Troca de Dados é definida como

$$x.\tau_A.\bar{k}(d).\bar{y} \mid k(d).y.\tau_B.\bar{z} \tag{22}$$

onde o agente $x.P_A.\bar{y}$ envia um dado d a um agente $y.P_B.\bar{z}$ via link k.

4.3.3 Communication

A semântica do aspecto Receber Mensagem é definida como

$$REC(x,y) \stackrel{\text{def}}{=} x.z(u).\tau_{rec}.\bar{y} \tag{23}$$

onde o nó *rec* é disparado via link x, recebe um dado u via link z, executa uma ação interna τ_{rec} e dispara um nó subsequente via link y.

A semântica do aspecto Responder Mensagem é definida como

$$REP(x,w,z) \stackrel{\text{def}}{=} x.\tau_{rep}.\bar{w}(u).\bar{z} \tag{24}$$

onde o nó *rep* é disparado via link x, executa uma ação interna τ_{rep}, envia um dado u via link w é ativa um nó subsequente via link z.

A semântica do aspecto Requisitar Serviço é definida como

$$\begin{aligned} RSC_1(v,w,x) &\stackrel{\text{def}}{=} REQ(v,w,x) \\ REQ(v,w,x) &\stackrel{\text{def}}{=} v.\tau_{req}.\bar{w}d.\bar{x} \end{aligned} \tag{25}$$

para enviar uma mensagem para um parceiro externo sem uma resposta de retorno. O nó *req* é disparado via link v, executa uma ação interna τ_{req}, envia um dado d para um parceiro externo via link w e ativa um nó subsequente via link x.

A semântica do aspecto Requisitar Serviço é definida como

$$RSC_2(v,w,x,y,z) \stackrel{\text{def}}{=} REQ(v,w,x) \mid GET(x,y,z)$$
$$GET(x,y,z) \stackrel{\text{def}}{=} x.y(u).\tau_{get}.\bar{z} \tag{26}$$

para enviar uma mensagem para um parceiro externo sem uma resposta de retorno. O nó *get* é disparado via link x, espera por uma resposta contendo um dado u via link y, executa uma ação interna τ_{get} e ativa um nó subsequente via link z.

4.3.4 Tratamento de Exceções

A semântica do aspecto Tratamento de Exceções é definida como

$$EHC(u,v,\tilde{w},x,y) \stackrel{\text{def}}{=} EXP(u,v,w,\tilde{x}) \mid EEXP(v,y)$$
$$EXP(u,v,\tilde{w},x) \stackrel{\text{def}}{=} u(v).\tau_{exp}.\left(\bar{v} \mid \prod_{i\in I} w_i.B_i\right).\bar{x}, \quad \forall v \in \tilde{w} \tag{27}$$
$$EEXP(x,y) \stackrel{\text{def}}{=} x.\tau_{eexp}.\bar{y}$$

onde o nó *exp* é disparado via link u, recebe uma tipo de exceção v via o mesmo link, executa uma ação interna τ_{exp}, divide um ramo em múltiplos ramos B_i, casa um disparado via link w_i, e dispara o ramo relacionado à exceção recebida v. O nó *eexp* junta via link x os ramos divididos pelo nó *exp*, executa uma ação interna τ_{eexp}, e dispara um nó subsequente via link y.

A semântica do aspecto Tratamento de Tempo Limite é definida como

$$DHC(x,y) \stackrel{\text{def}}{=} x.\left(P.\overline{w}(t_d).\bar{u} \mid \overline{w}(t)\right) \mid$$
$$!\,w(t_0).\left([t_0 = t_d]\bar{d} + \overline{w}(t_1)\right) \mid DDL(d,e,f) \mid EDDL(f,u) \tag{28}$$
$$DDL(d,e,f) \stackrel{\text{def}}{=} d.\tau_{ddl}.\bar{e} \mid e.Q.\bar{f}$$
$$EDDL(f,u) \stackrel{\text{def}}{=} f.\tau_{eddl}.\bar{u}$$

onde t_d é um tempo limite agregado ao nó representado pelo agente P, t_0 e t_1 são variáveis contendo o tempo corrente. Se $\mathrm{t_0} = \mathrm{t_d}$, o nó *ddl* é ativado via link d, executa uma ação interna τ_{ddl} e dispara um ramo de tratamento de tempo limite via link e. O nó *eddl* junta tal ramo via link f, executa uma ação interna τ_{eddl}, e dispara um nó subsequente via link u.

A semântica do aspecto Tratamento de Finalização é definida como

$$\begin{aligned} &FHC(w,x,y) \stackrel{\text{def}}{=} FIN(x,w)|EFIN(x,y) \\ &FIN(y) \stackrel{\text{def}}{=} \tau.\overline{w} \mid w.P.\bar{x} \\ &EFIN(x,y) \stackrel{\text{def}}{=} x.\tau.\bar{y} \end{aligned} \tag{29}$$

onde o nó *fin* pode ser disparado via link v, se um conjunto de atividades foi executado com sucesso, ou via link w, se uma exceção foi disparada. Então, o nó *fin* executa uma ação interna τ_{fin} e dispara um ramo representado pelo agente P via link x. O nó *efin* junta o ramo P via link y, executa uma ação interna τ_{efin}, e dispara um nó subsequente via link z.

Capítulo 5

IMPLEMENTAÇÃO

Este capítulo apresenta os detalhes da implementação da abordagem proposta. A Seção 5.1 apresenta as técnicas, ferramentas e componentes de software utilizados na implementação; a Seção 5.2 descreve o meta-modelo orientado a objetos do GWM; a Seção 5.3 apresenta a arquitetura da etapa de seleção de localização; a Seção 5.4 a arquitetura da etapa de decomposição; e a Seção 5.5 apresenta a arquitetura das etapas de lifting e grounding.

5.1 Técnicas, Ferramentas e Componentes de Software

Com base na abordagem definida foi desenvolvido um protótipo através da técnica de desenvolvido denominada *Test Driven Development* (*TDD*) (BECK, 2002) utilizando a tecnologia Java 7 (ORACLE, 2014) junto ao framework de testes jUnit (BECK e GAMMA, 2014). Seu desenvolvimento foi gerenciado através do sistema de controle de versões Subversion (*SVN*) (PILATO, COLLINS-SUSSMAN e FITZPATRICK, 2008), sendo possível encontrar o código-fonte do protótipo no endereço https://code.google.com/p/bpd4cd/.

O protótipo é composto por seis componentes: GWM, *Languages*, *Transformation*, *Selection*, *Decomposition* e *Utils*. O componente GWM contém o meta-modelo orientado a objetos das definições apresentadas na Seção 3.2. O componente *Languages* possui as definições dos padrões das linguagens BPMN e WS-BPEL. O componente *Transformation* possui as classes com as implementações dos algoritmos de transformação, *lifting* e *grounding*, das linguagens BPMN e WS-BPEL. O componente *Selection* detém as classes responsáveis por calcular e definir

a localização de cada atividade e os dados associados de um processo monolítico. O componente *Decomposition* possui as classes que implementam as etapas e regras de decomposição. O componente *Utils* contém classes que auxiliam em várias etapas da abordagem, como por exemplo, na transformação e na seleção de localização. A Figura 5.1 ilustra cada um dos componentes e a relação entre cada um deles através do diagrama de componentes (OBJECT MANAGEMENT GROUP, 2014).

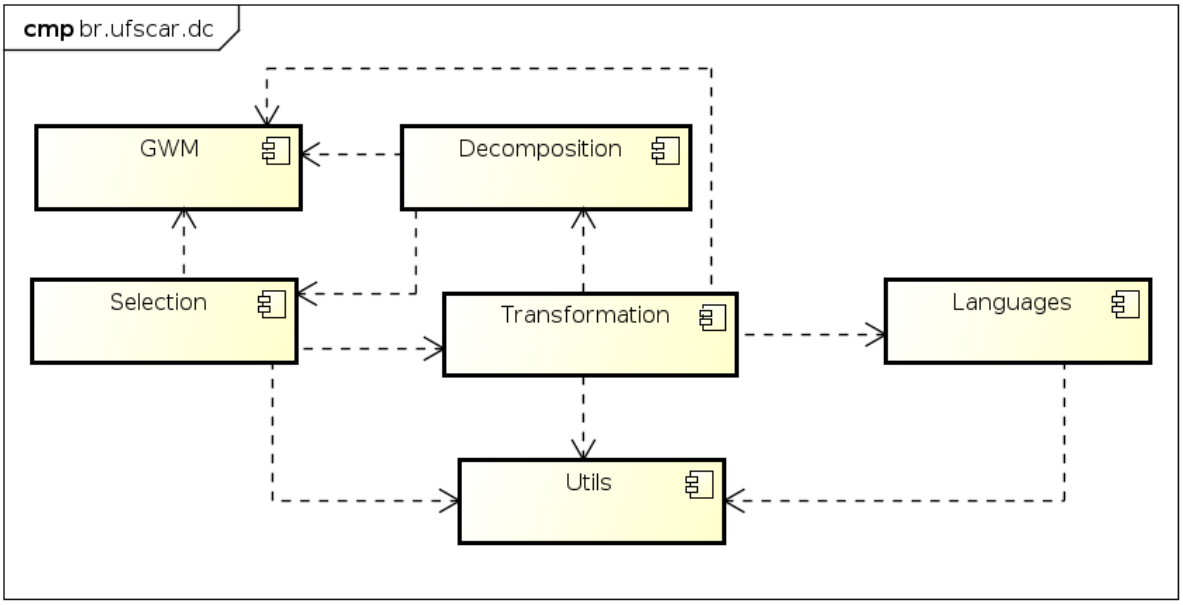


Figura 5.1 – Diagrama de componentes do protótipo da abordagem.

5.2 Modelo Intermediário

O componente principal da abordagem proposta é o meta-modelo orientada a objetos do GWM, o qual é utilizado por todas as etapas da abordagem. A Figura 5.2 ilustra o diagrama de classes referente ao meta-modelo do GWM, sendo que os papais de cada classe são:

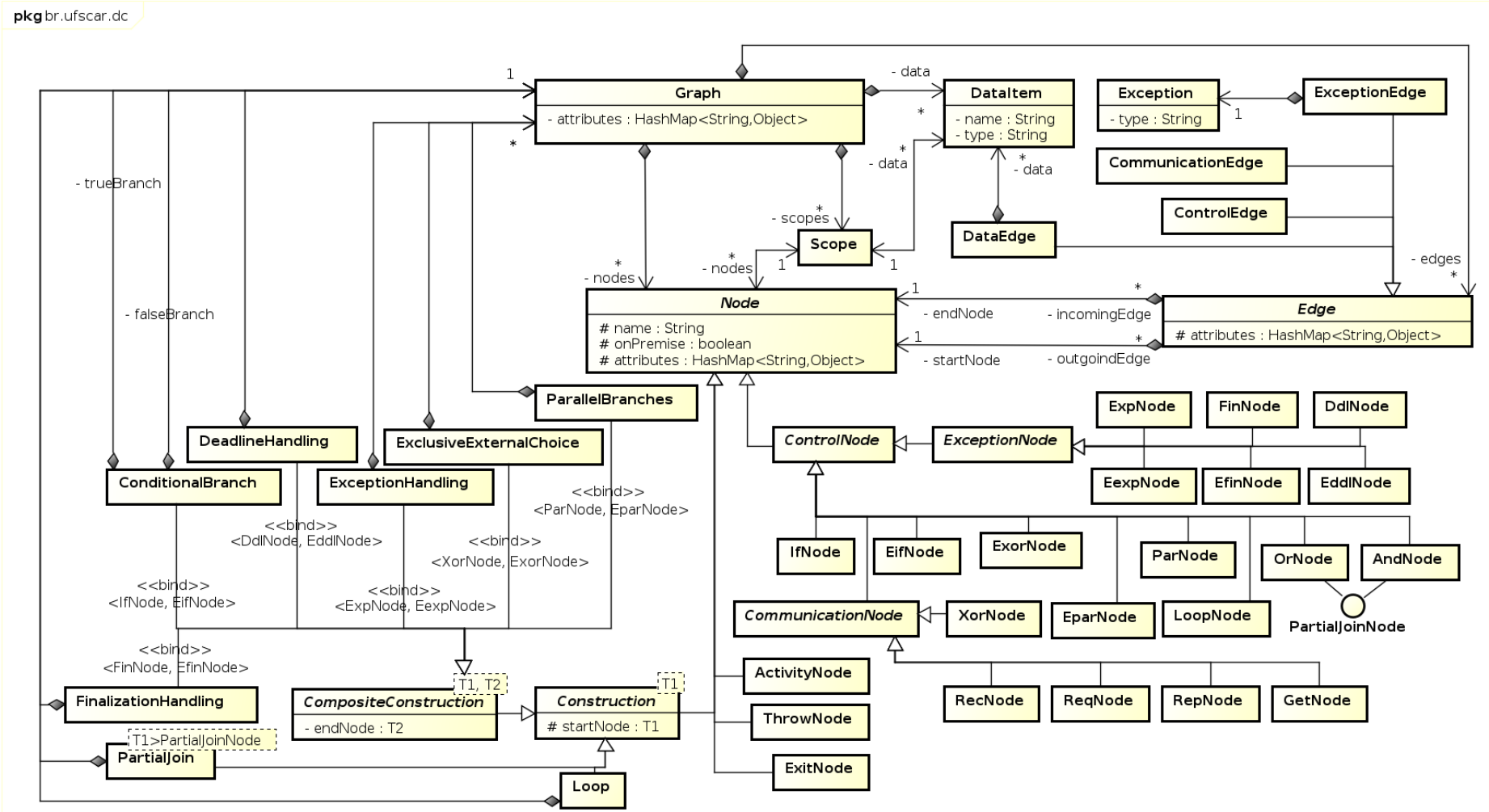


Figura 5.2 –Diagrama de classes do meta-modelo do GWM.

- **Graph** é a classe base para a definição de processos de negócio, a qual é composta por nós representando atividades, por item de dados e por arestas de controle, dados, comunicação e exceção. Além disso, essa classe captura atributos dos processos de negócio, como informações referentes às portas de acesso ao processo, nome do processo, endereço do servidor de alocação, entre outros dados. Além disso, um grafo representa ramos aninhados a aspectos do GWM;
- **Node** é uma classe abstrata que define a estrutura básica a ser seguida pelas classes representam determinados tipos de atividades. Dentre essas características são capturas a localização de alocação, na nuvem ou na premissa, o nome e atributos das atividades;
- **Edge** é uma classe abstrata cuja finalidade é definir a estrutura básica a ser seguida pelas classes que representam arestas de tipos específicos. Além disso, tal classe define a relação entre os nós que compõem um processo. Em particular, a classe **DataEdge** define quais itens de dados são transferidos entre dois nós, a classe **ExceptionEdge** possui a capacidade de capturar a exceção disparada por um nó em particular, classe **CommunicationEdge** conecta dois nós de comunicação para definir a coreografia entre processos distintos ou o acesso a atividades alocadas em parceiros externos e a classe **ControlEdge** define o fluxo de controle entre os nós do processo;
- **Scope** é a classe que define o escopo de um conjunto de nós e itens de dados;
- **DataItem** é a classe que define um item de dados utilizado pelas atividades do processo;
- **Exception** é a classe que define uma variável para capturar exceções;
- **ActivityNode** é a classe que define atividades dentro de um processo;
- **ThrowNode** é a classe que define atividades que disparam um determinado tipo de exceção explicitamente;
- **ExitNode** é a classe que define atividades que finalizam a instância do processo de negócio explicitamente;

- **ControlNode** é a classe que generaliza a estrutura de classes que representam nós de controle, as quais são **IfNode**, **EifNode**, **ExorNode**, **ParNode**, **EparNode**, **PartialJoinNode**, **OrNode**, **AndNode**, **LoopNode***;*
- **ExceptionNode** é a classe que generaliza a estrutura de classes que representam nós utilizados para tratamento de exceções, sendo a generalização das classes **ExpNode**, **EexpNode**, **FinNode**, **EfinNode**, **DdlNode**, **EddlNode**;
- **CommunicationNode** é uma classe abstrata que define a estrutura básica que nós de comunicação devem seguir, sendo a generalização das classes **RecNode**, **RepNode**, **ReqNode**, **GetNode** e **XorNode**;
- **Construction** é a classe abstrata que define a estrutura básica dos aspectos do GWM, a qual representa a generalização das classes **Loop**, **CompositeConstruction** e **PartialJoin**;
- **Loop** é a classe que define o aspecto Laço de Repetição do GWM, a qual contém um grafo a ser executado iterativamente e um atributo do tipo **LoopNode** que possui a condição de iteração, a qual pode ser avaliada antes ou depois da primeira execução do grafo iterativo; e
- **CompositeConstruct** é a classe abstrata que define a estrutura básica das classes que capturam aspectos do GWM que possuem um nó de início e um nó de término, os quais são definidos pelas classes **DeadlineHandling**, **ConditionalBranch**, **ExceptionHandling**, **ExclusiveExternalChoice** e **ParallelBranches**.

5.3 Seleção de Localização

As implementações das equações e variáveis da etapa de seleção de localização são concentradas na classe denominada **Selection**. A Figura 5.3 ilustra o diagrama de classes referente aos componentes da referida etapa.

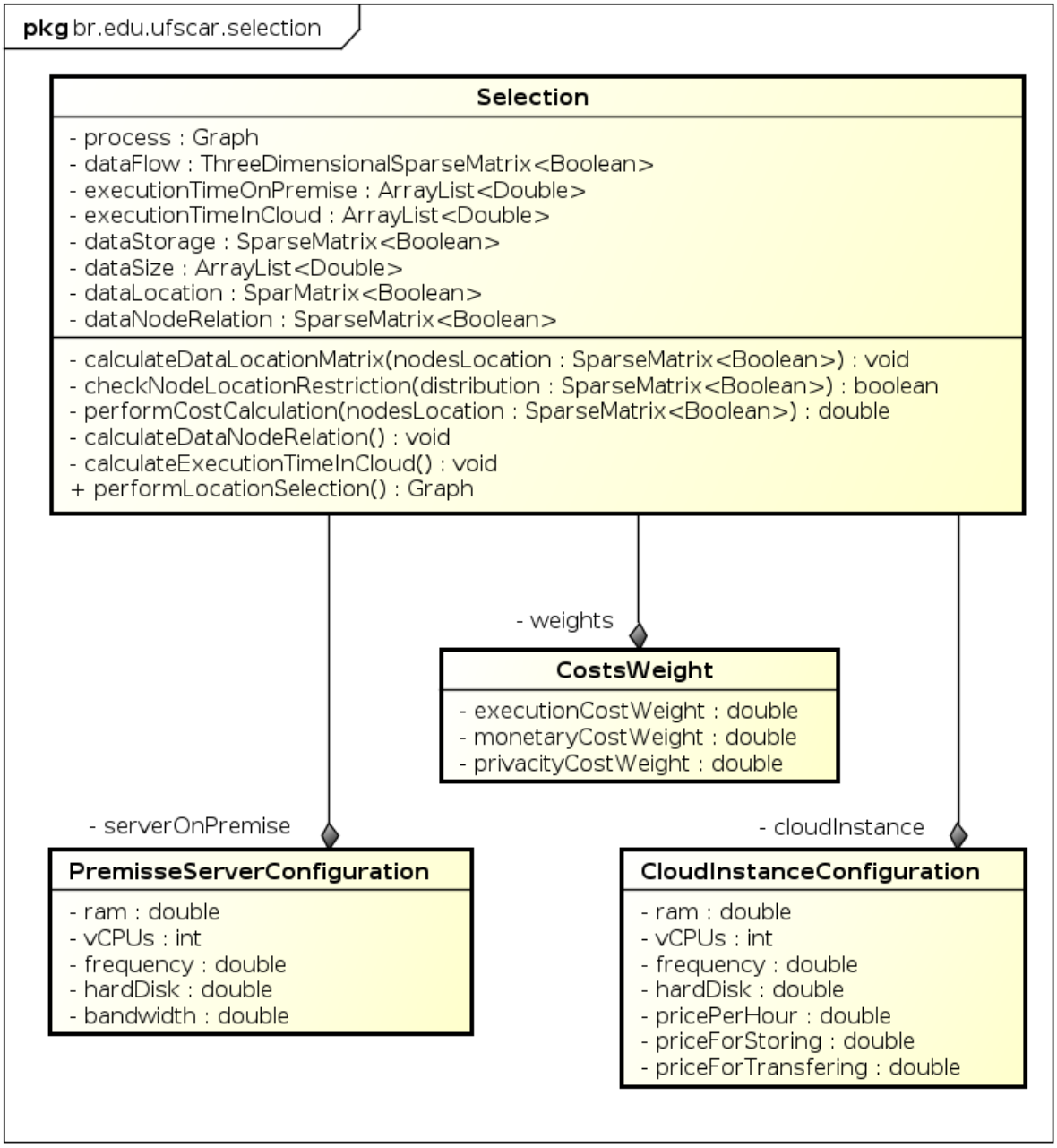


Figura 5.3 – Diagrama de classes da etapa de seleção de localização.

Em particular, a classe **CostsWeight** representa os pesos de cada custo envolvido na etapa de seleção, a classe **PremiseServerConfiguration** representa a configuração do servidor da premissa e a classe **CloudInstanceConfiguration** representa a configuração da instância de nuvem. As três classes descritas compõem três dos onze atributos da classe **Selection**, os quais são referentes às variáveis utilizadas na etapa de seleção de localização. A classe **Selection** é composta por seis métodos principais:

- *calculateDataLocationMatrix* tem a finalidade de preencher os valores da matriz esparsa ***R***, a qual é representada pelo atributo *dataLocation*, a partir do atributo *process*;
- *checkNodeLocationRestriction* tem a finalidade de checar se uma determinada distribuição infringe ou não as restrições de localização de algumas atividades, a qual é descartada caso infrinja alguma restrição;
- *performCostCalculation* é a implementação da Equação (1) e consecutivamente das Equações (4) a (10). Tal método tem como entrada a distribuição que terá o custo calculado;
- *calculateDataNodeRelation* é a implementação direta da Equação (2), o qual preenche os valores da matriz esparsa ***Q*** representada pelo atributo *dataNodeRelation*;
- *calculateExecutionTimeInCloud* é a implementação direta da Equação (3), sendo executado sobre os atributos *cloudInstance*, *serverOnPremise* e *executionTimeOnPremise*. A finalidade é desse método é preencher os valores do atributo *executionTimeInCloud*; e
- *performLocationSelection* tem a finalidade de gerar todas as distribuições possíveis do GWM de entrada e calcular iterativamente os custos para cada uma delas.

5.4 Decomposição

Os algoritmos da etapa de seleção, e consecutivamente as regras de decomposição, foram implementados na classe **Decomposition**. Para cada etapa da decomposição existe um conjunto de classes correspondente a sua entrada e a sua saída. A classe **Fragment** representa a saída da etapa de fragmentação, a entrada da etapa de agrupamento e umas das entradas da etapa de coreografia. A classe **Cluster** representa a saída da etapa de agrupamento e a segunda entrada da etapa de coreografia. A classe **Coreography** representa a saída da etapa de

coreografia e a entrada e saída da etapa de fluxo de dados. A Figura 5.4 ilustra o diagrama de classes da implementação da etapa de decomposição.

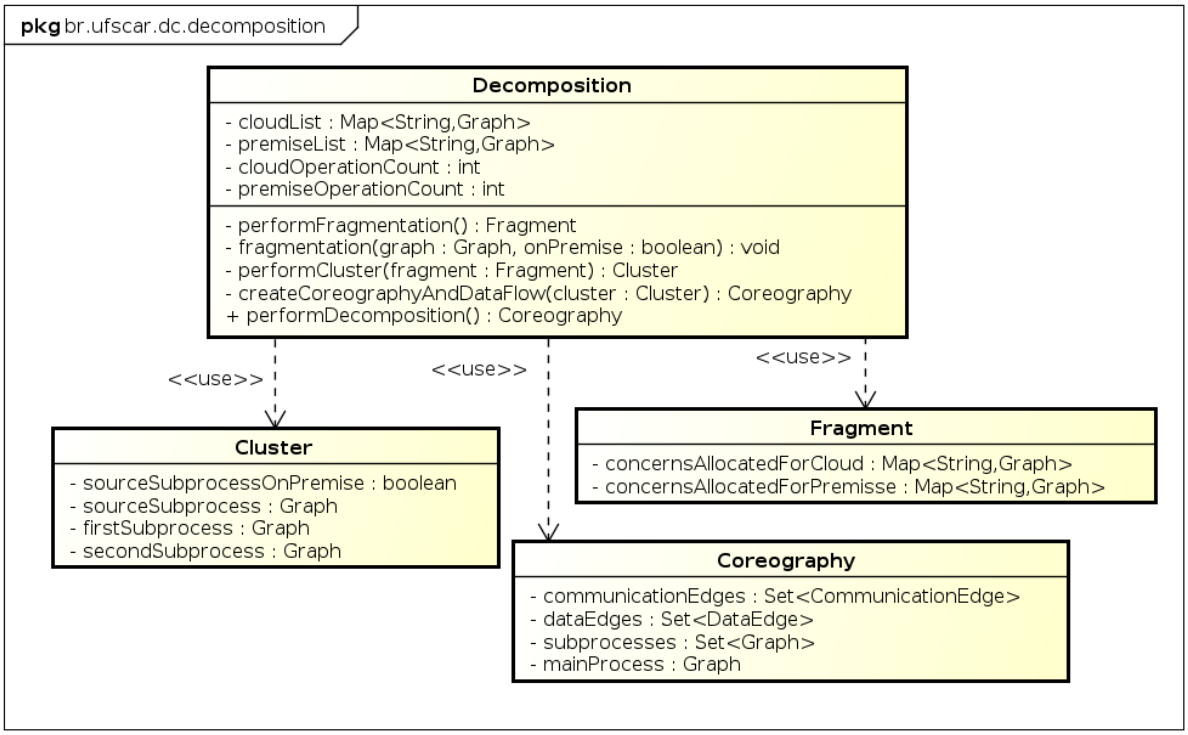


Figura 5.4 – Diagrama de classes da etapa de decomposição.

Os métodos da classe **Decomposition** possuem as seguintes finalidades:

- *performFragmentation* corresponde à implementação do Algoritmo 3.1;
- *fragmentation* corresponde à implementação dos Algoritmos Algoritmo 3.2 e Algoritmo 3.3;
- *performCluster* corresponde à implementação do Algoritmo 3.4;
- *createCoregraphyAndDataFlow* corresponde à implementação dos Algoritmos Algoritmo 3.5 e Algoritmo 3.6; e
- *performDecomposition* recebe o processo monolítico emitido pela etapa de seleção de localização e gerencia o fluxo entre os métodos correspondentes a cada etapa da decomposição.

5.5 Lifting e Grounding

A implementação das transformações de *lifting* e *grounding* teve como principal desafio a definição de uma arquitetura genérica o suficiente para que a definição das transformações para outras linguagens de especificação de processos de negócio fosse possível com um baixo esforço. A Figura 5.5 ilustra o diagrama de classes referente à generalização das transformações de *lifting* e *grounding*.

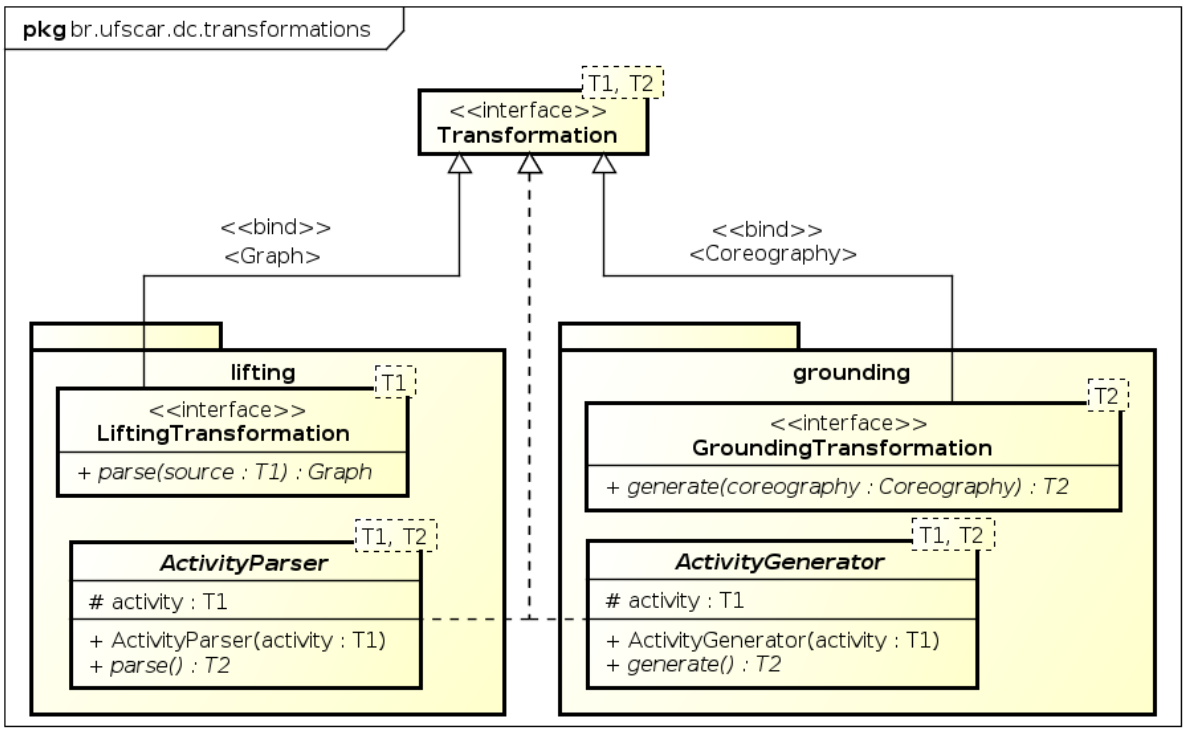


Figura 5.5 – Diagrama de classes da implementação do *lifting* e *grounding*.

A interface **Transformation** define dois tipos genéricos T1 e T2, os quais são respectivamente correspondentes ao tipo da entrada e saída esperado pelos métodos de transformações das classes que implementam tal interface.

A interface **LiftingTransformation** define que toda transformação de *lifting* parte de um objeto do tipo genérico T1 e emite um objeto do tipo **Graph**, sendo que o método *parse* é o responsável por tal tarefa. A classe abstrata ***ActivityParser*** define o atributo *activity* do tipo genérico T1, o qual representa um padrão específico de uma linguagem de definição de processos de negócio, e o método *parse* que

efetivamente transforma um padrão de entrada em um aspecto ou nó GWM correspondente representado por um objeto do tipo genérico T2.

A interface **GroundingTransformation** define que toda transformação de *grounding* parte de um objeto do tipo **Coreography** e emite um objeto do tipo genérico T2, sendo que o método *generate* é o responsável por tal tarefa. A classe abstrata ***ActivityGenerator*** define o atributo *activity* do tipo genérico T1, o qual é correspondente a um aspecto ou nó GWM, e o método *generate* que efetivamente deverá transformar um aspecto ou nó de entrada em um padrão correspondente na linguagem de definição de processos de negócio.

A partir da classe abstrata **AcitivityParser**, são definidas as classes concretas. Tais classes detêm os métodos que implementam os algoritmos de *lifting* relativos a cada padrão da linguagem em foco, e a partir da classe abstrata **ActivityGenerator** são definidas as classes concretas que detém os métodos que implementam os algoritmos de *grounding* relativos a cada aspecto e nó do GWM.

Como essas classes são responsáveis pelas partes específicas dos algoritmos de *lifting* e *grounding* devem ser criadas duas classes adicionais, uma para possuir o método referente à parte geral do algoritmo de *lifting* e outra para possuir o método referente à parte geral do algoritmo de *grounding*. A primeira delas deve implementar a interface **LiftingTransformation** e a última a interface **GroundingTransformation**.

Capítulo 6

ESTUDO DE CASO

Neste capítulo as etapas da abordagem proposta nesta pesquisa são executadas em um processo de negócio do domínio da Saúde. A Seção 6.1 descreve o processo de negócio utilizado no estudo de caso; a Seção 6.2 apresenta a etapa de lifting sobre o processo monolítico; a Seção 6.3 descreve os detalhes da etapa de seleção de localização; a Seção 6.4 mostra a aplicação da etapa de decomposição; a Seção 6.5 apresenta a etapa de grounding; a Seção 6.6 descreve a análise de desempenho comparando os processos monolítico e decomposto; e a Seção 6.7 apresenta um modelo para o cálculo do custo por hora da nuvem e a relação entre esse custo e o ganho de desempenho obtido.

6.1 Processo de Negócio no Domínio da Saúde

O estudo de caso apresentado neste capítulo é baseado em *Picture Archiving and Communication Systems* (*PACS*) (OOSTERWIJK, 2004). Esse tipo de sistema, proveniente do domínio da Saúde, possui a finalidade de captar, armazenar, transmitir e apresentar imagens digitais (e.g., tomografia mamária, radiografia) e informações relacionadas (e.g., dados demográficos, relatórios de diagnostico, histórico clínico). A Figura 6.1 ilustra o processo de negócio monolítico do PACS descrito na linguagem BPMN, onde os nós de divisão e junção correspondentes são ilustrados com cores iguais. Tal processo efetua o armazenamento de diagnósticos e tomografias mamárias, as quais são analisadas para que a existência de possíveis nódulos seja verificada, e emite como resposta um vetor contendo as imagens com nódulos em potencial.

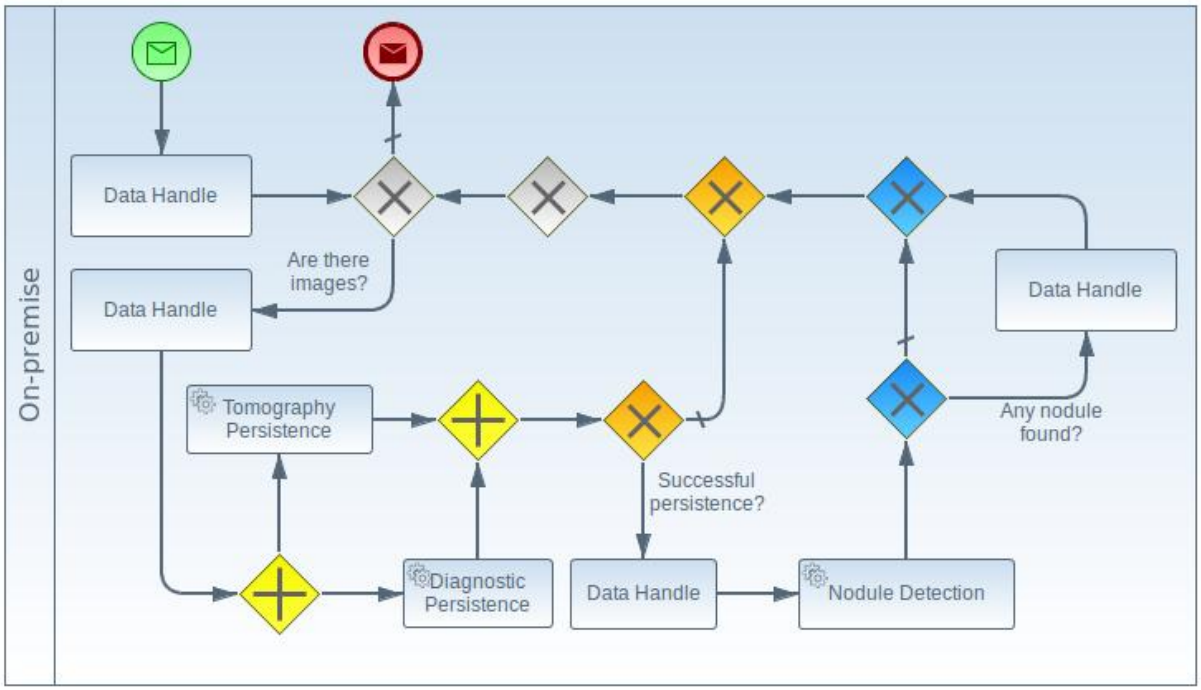


Figura 6.1 – PACS monolítico descrito em BPMN.

O processo de negócio do PACS é iniciado através de uma mensagem emitida por um parceiro externo com um conjunto de imagens e seus respectivos diagnósticos e identificadores, sendo que um laço de repetição percorre cada elemento do conjunto de entrada. Concorrentemente é executado o armazenamento da imagem e seu respectivo diagnóstico através dos nós rotulados como *Tomography Persistence* e *Diagnostic Persistence*. Em caso de sucesso na execução do armazenamento, a detecção de nódulos em potencial é efetuada através do nó rotulado como *Nodule Detection*, e caso algum nódulo seja identificado, a imagem é adicionada ao vetor a ser emitido como resposta. Após o termino do laço de repetição, o processo envia uma mensagem ao parceiro externo requisitante contendo o vetor composto pelas imagens com possíveis nódulos.

6.2 Lifting

A primeira fase da etapa de *lifting* é exportar o modelo do processo de negócio para uma representação XML correspondente. O Quadro 6.1 descreve

trecho do código XML correspondente ao modelo do processo de negócio monolítico do PACS ilustrado na Figura 6.1.

Quadro 6.1 – Treco do código XML corresponde ao PACS monolítico.

```
<process id="monolithic-onpremise" ...>
 <startEvent id="StartEvent_1" ...>
  ...
  <messageEventDefinition id="MessageEventDefinition_1"/>
 </startEvent>
 <task id="Task_1" name="Data Handle">
  <incoming>SequenceFlow_1</incoming>
  <outgoing>SequenceFlow_2</outgoing>
  ...
 </task>
 <exclusiveGateway id="ExclusiveGateway_1"
  gatewayDirection="Diverging" default="SequenceFlow_3">
  ...
 </exclusiveGateway>
 <parallelGateway id="ParallelGateway_2" gatewayDirection="Diverging">
  ...
 </parallelGateway>
 <serviceTask id="ServiceTask_2"
  name="Tomography Persistence">
  ...
 </serviceTask>
 <serviceTask id="ServiceTask_4" name="Diagnostic Persistence">
  ...
 </serviceTask>
 <parallelGateway id="ParallelGateway_4" gatewayDirection="Converging">
  ...
 </parallelGateway>
 ...
 <endEvent id="EndEvent_1">
  ...
  <messageEventDefinition .../>
 </endEvent>
 <sequenceFlow id="SequenceFlow_1" sourceRef="StartEvent_1"
  targetRef="Task_1"/>
 <sequenceFlow id="SequenceFlow_2" sourceRef="Task_1"
  targetRef="ExclusiveGateway_1"/>
 <sequenceFlow id="SequenceFlow_3"
  sourceRef="ExclusiveGateway_1" targetRef="EndEvent_1"/>
 <sequenceFlow id="SequenceFlow_31" name="Are there images?"
  sourceRef="ExclusiveGateway_1" targetRef="Task_9">
  ...
 </sequenceFlow>
 ...
</process>
```

Após a etapa de *Lifting* sobre o código XML do PACS foi obtido o GWM ilustrado na Figura 6.2, onde o fluxo de dados está omitido.

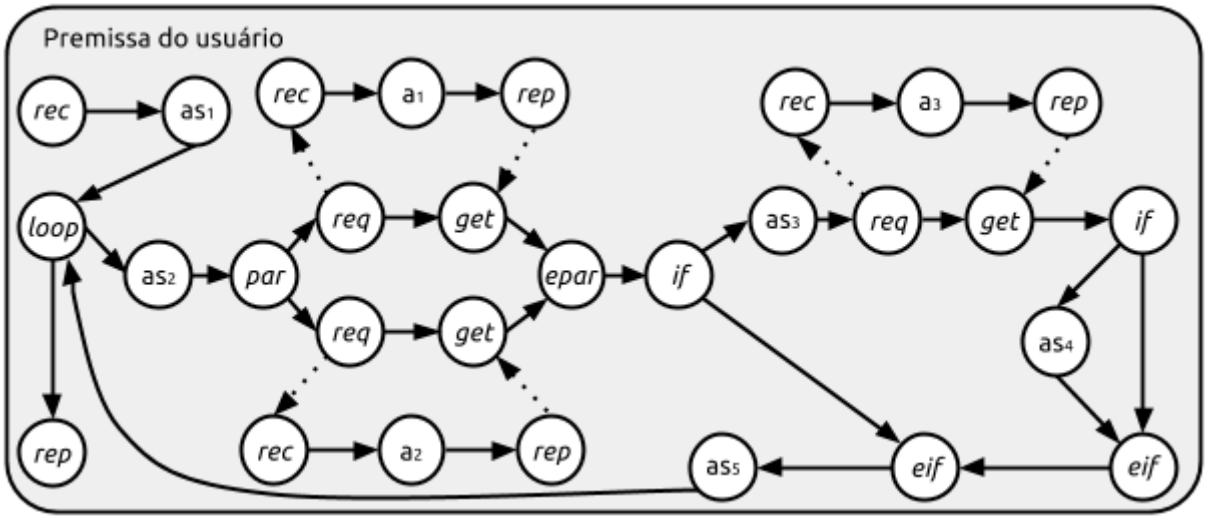


Figura 6.2 – PACS monolítico descrito em GWM.

A Figura 6.3 ilustra o fluxo de controle junto ao fluxo de dados, destacando esse último. Os nomes das variáveis foram abreviados com a finalidade de tornar a representação mais legível.

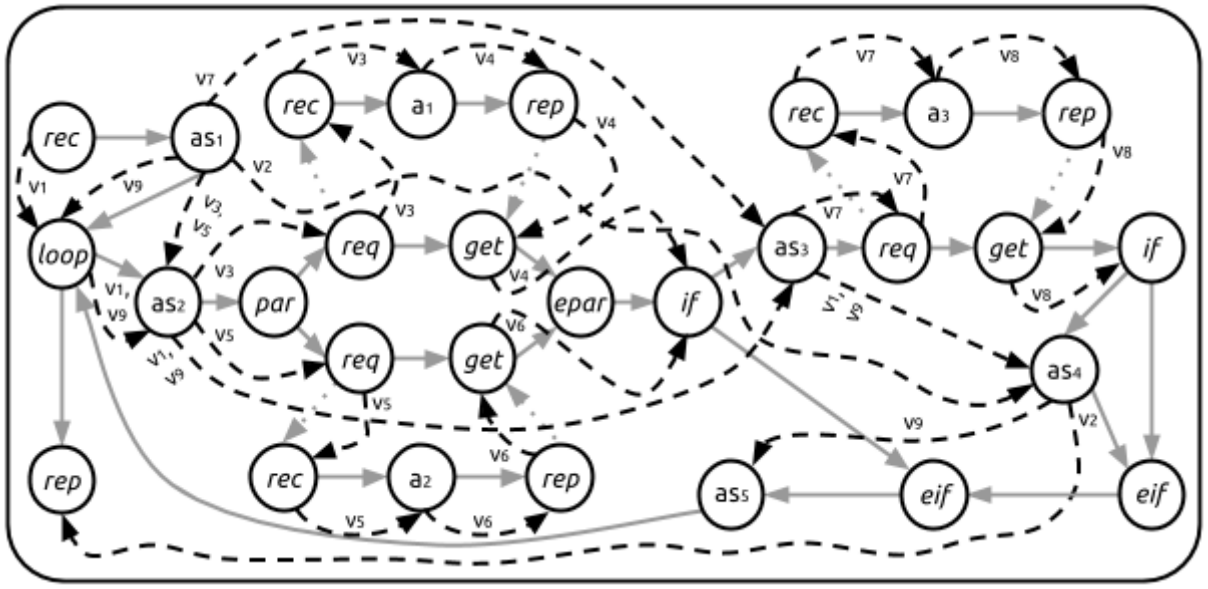


Figura 6.3 – PACS monolítico com fluxo de dados destacado.

6.3 Seleção de Localização

A partir do GWM obtido na etapa de *lifting* é executada a etapa de seleção de localização. Inicialmente são capturadas as atividades para compor o conjunto ***A*** e os itens de dados para compor o conjunto ***D***. A partir do fluxo de dados é extraída a matriz esparsa ***R***, a qual é representada pelo gráfico ilustrado na Figura 6.4. Nessa figura blocos laranja representam a existência de relação (um) entre a atividade e o item de dado e blocos cinza a ausência de tal relação (zero).

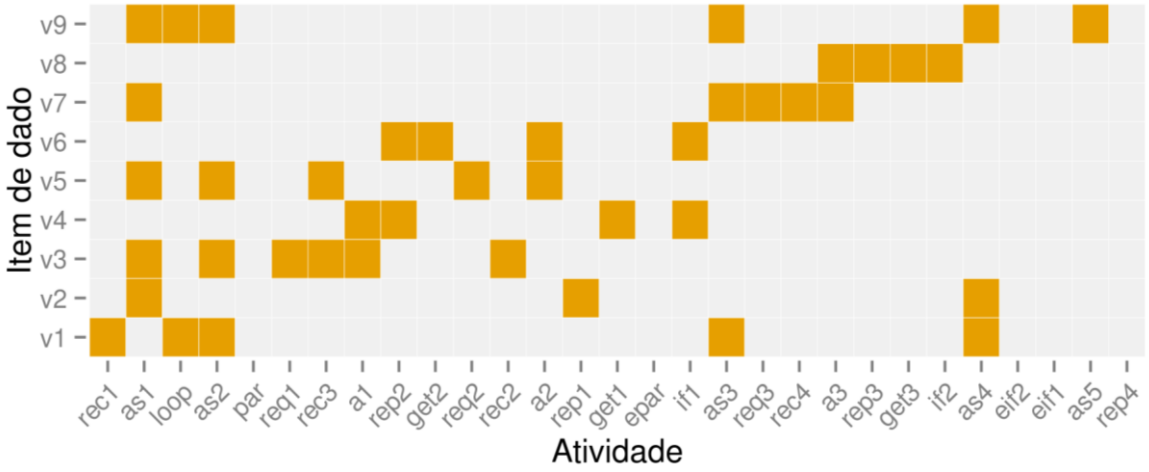


Figura 6.4 – Representação da matriz esparsa *R* referente ao PACS monolítico.

A partir do fluxo de dados também é extraída a matriz esparsa 3-dimensional ***V*** requerida pela etapa de seleção de localização. A matriz ***V*** correspondente ao fluxo de dados do GWM ilustrado pela Figura 6.3 é representada pelo gráfico da Figura 6.5, onde pontos em vermelho correspondem ao valor um e pontos cinza ao valor zero.

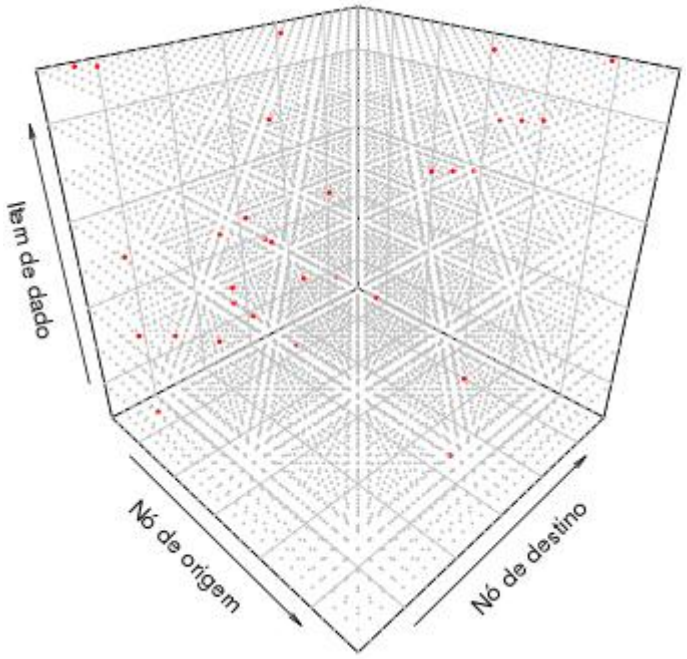


Figura 6.5 – Representação da matriz esparsa *V* referente ao PACS monolítico.

Duas informações sobre cada item de dado utilizado no processo de negócio, as quais são informadas por uma entidade externa, também são necessárias para a seleção de localização. A primeira delas é inerente à sensibilidade dos itens de dados, ou seja, se esses podem ou não ser alocados na nuvem, sendo que tal informação compõe o vetor ***C***. A segunda informação se refere ao tamanho do item de dado, a qual compõe o conjunto imagem da função *size*. A Tabela 6.1 apresenta o nome real dos itens de dados utilizados no processo de negócio do PACS e sua respectiva abreviação, além das informações sobre sua sensibilidade e seu tamanho em megabytes.

Tabela 6.1 – Variáveis do PACS.

Item de dado	Abreviação	Sensível	Tamanho (~)
request	v1	Sim (1)	22,40 MiB
response	v2	Não (0)	11,20 MiB
imagePersistenceRequest	v3	Não (0)	11,20 MiB
imagePersistenceResponse	v4	Não (0)	1,00 Byte
diagnosticPersistenceRequest	v5	Sim (1)	32,00 Bytes
diagnosticPersistenceResponse	v6	Não (0)	1,00 Byte
analysisRequest	v7	Não (0)	11,20 MiB
analysisResponse	v8	Não (0)	1,00 Byte
i	v9	Não (0)	1,00 Byte

Duas informações sobre as atividades do processo de negócio, as quais também são fornecidas por uma entidade externa, são necessárias para a etapa de seleção de localização. A primeira delas se refere ao tempo de execução da atividade enquanto alocada na premissa, a qual compõe o conjunto imagem da função $exec_p$. A segunda se refere à questão de armazenamento de dados, ou seja, se a atividade efetua o armazenamento de algum item de dado que se relaciona ou não, compondo a matriz esparsa ***P***.

A Figura 6.6 apresenta o tempo de execução médio (aproximado) em segundos das atividades que não possuem restrições de localização durante o cálculo do custo total, sendo que as atividades com tais restrições são consideradas com tempo zero de execução.

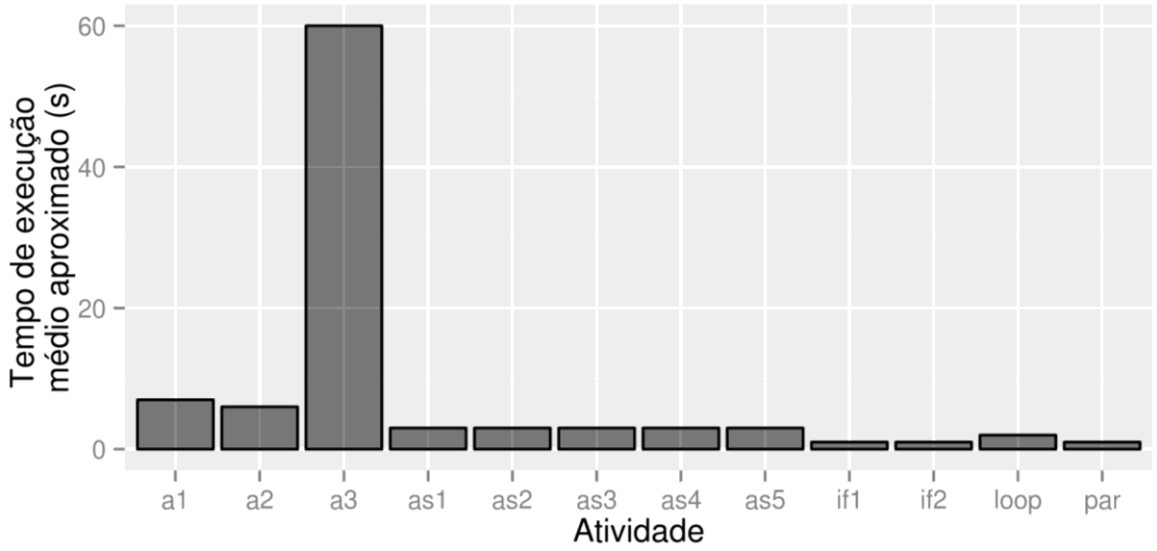


Figura 6.6 – Tempo de execução de cada atividade do PACS monolítico.

A Figura 6.7 ilustra o gráfico que representa a matriz esparsa ***P*** correspondente ao processo de negócio do PACS, onde blocos laranja definem que a atividade armazena o item de dado associado (um) e blocos cinza que a atividade não armazena tal item de dado (zero).

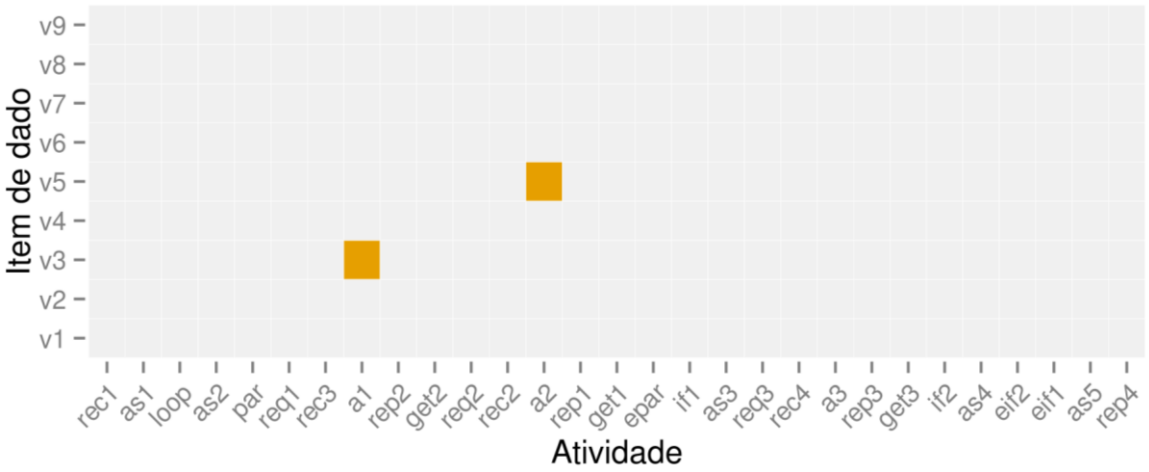


Figura 6.7 – Representação da matriz esparsa *P* referente ao PACS monolítico.

As informações sobre o servidor da premissa, as quais devem ser fornecidas por uma entidade externa, compõe a tupla $c_{premise}$. Para o estudo de caso apresentado foi utilizado um servidor na premissa contendo 2 GB de RAM, 50 GB de HD, 1 CPU virtual (vCPU) com 0,8 GHz de frequência, e uma banda entre a premissa e a nuvem de 125 Mbps.

A configuração do servidor da nuvem (instância da nuvem) compõe a tupla C_{cloud}. Para o estudo de caso foram empregadas três instâncias no provedor de nuvem Amazon EC2 (AMAZON WEB SERVICES, 2014), as quais possuem suas configurações descritas na Tabela 6.2.

Tabela 6.2 – Configurações das instâncias da nuvem.

Instância	Número de CPUs	Frequência (GHz)	RAM (GB)	HD (GB)
c1.xlarge	8	2.75	7.00	4 x 420
m2.4xlarge	8	3.58	68.40	2 x 840
hs1.8xlarge	16	2.41	117.00	45 x 2048

O preço por hora de execução, o preço por GB transferido para e da nuvem e o preço/hora por TB armazenado na nuvem referentes a cada instância utilizada no estudo de caso são descritos na Tabela 6.3.

Tabela 6.3 – Preços relacionadas às instâncias de nuvem.

Instância	Preço por hora em execução	Preço por GB transferido	Preço/hora por TB armazenado (US$)
c1.xlarge	US$ 0,06	US$ 0,01	US$ 0,27
m2.4xlarge	US$ 0,10	US$ 0,01	US$ 0,27
hs1.8xlarge	US$ 4,60	US$ 0,01	US$ 0,27

A Figura 6.8 destaca as atividades do processo de negócio que não possuem restrições de localização para o cálculo do custo total, como descrito na Seção 3.3.

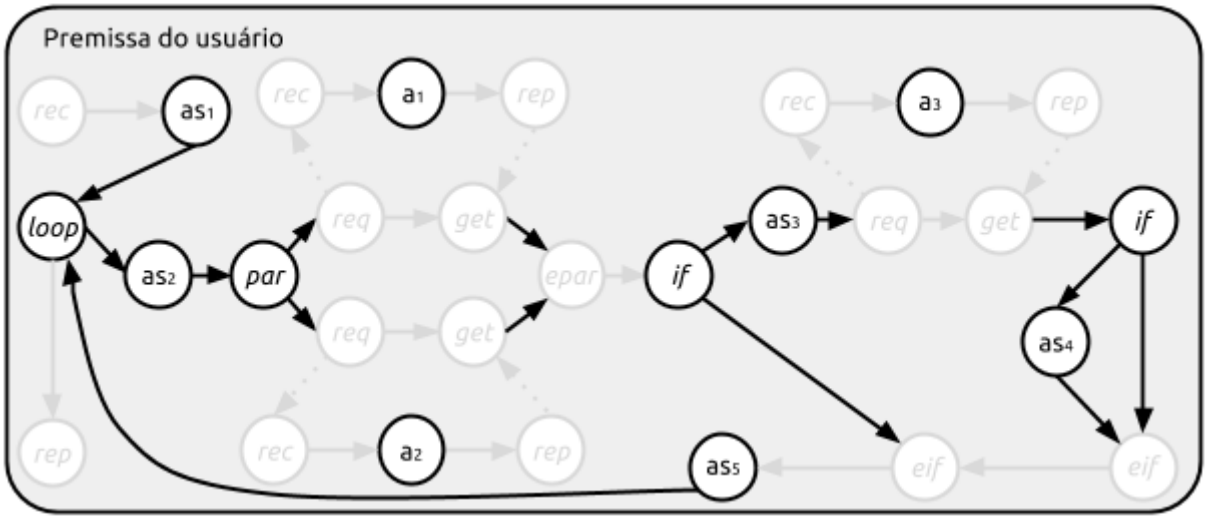
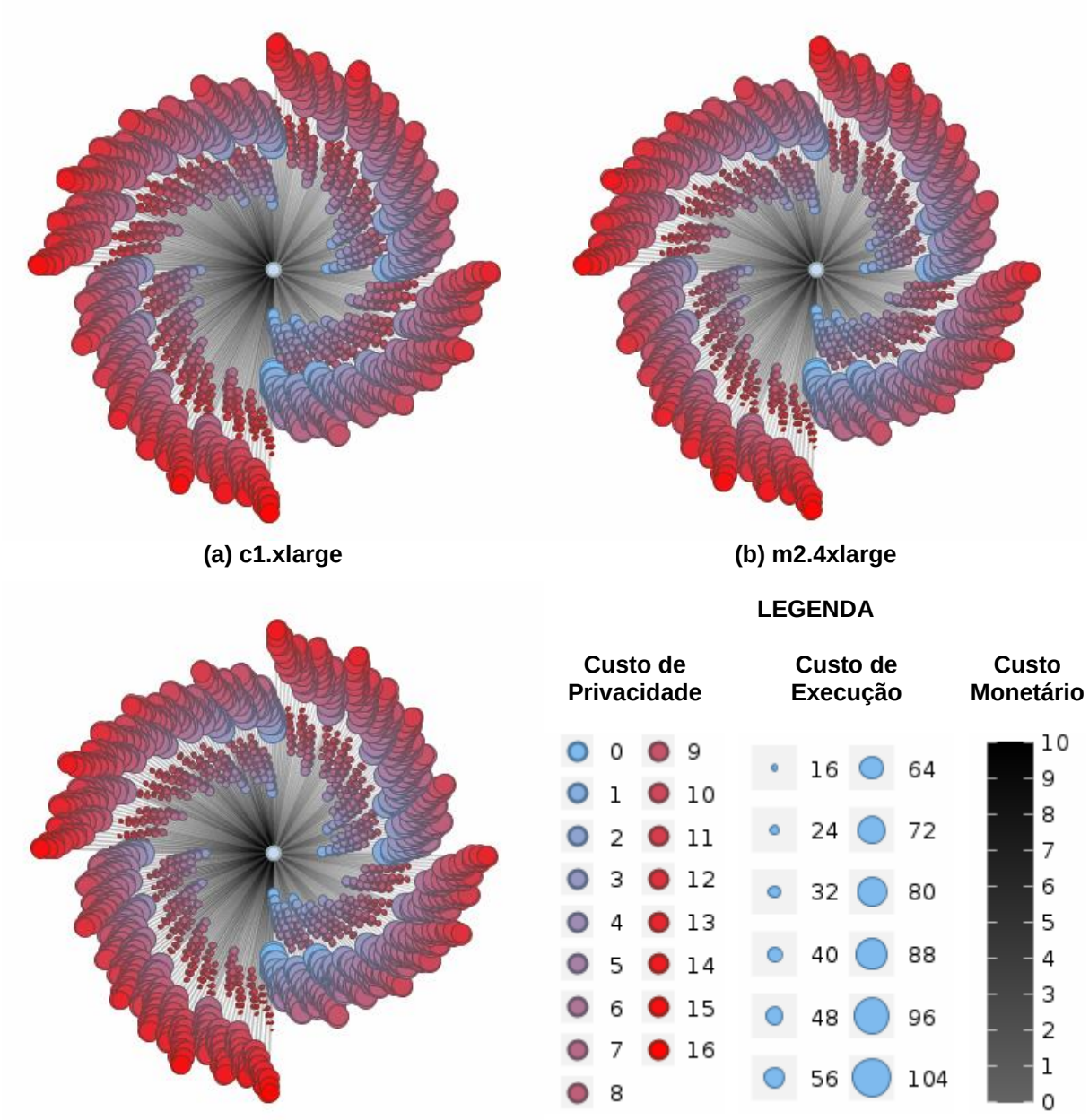


Figura 6.8 – PACS monolítico com restrições de localização.

Com base nas informações descritas anteriormente, o framework de seleção de localização determinou o custo total para cada distribuição de atividades e dados associados. A Figura 6.9 ilustra três grafos de dispersão, sendo que cada um deles representa os custos de execução, monetário, de privacidade e total, empregando cada uma das três instâncias de nuvem descritas na Tabela 6.2. Tais grafos devem ser interpretados da seguinte forma:

- o nó central representa o melhor custo total teórico possível, ou seja, zero;
- a coloração das arestas entre o nó central e os nós restantes, denominados de nós de custo, representa o custo de privacidade;
- o tamanho dos nós dos custos corresponde ao custo de execução;
- a coloração dos nós dos custos é referente ao custo de privacidade; e
- a distância entre o nó central e os nós dos custos corresponde ao custo total.



(a) c1.xlarge **(b) m2.4xlarge**

(c) hs1.8xlarge

Figura 6.9 – Grafos dos custos empregando diferentes instâncias da nuvem.

O nó de custo mais próximo ao nó central representa o menor custo total, sendo que para as três diferentes instâncias na nuvem, o menor custo total é referente à mesma distribuição, ou seja, o mesmo vetor ***s***.

A Figura 6.10 ilustra o GWM de saída da etapa de seleção de localização com nós marcados com a distribuição associada ao menor custo total, onde nós com fundo destacado foram selecionados para alocação na nuvem.

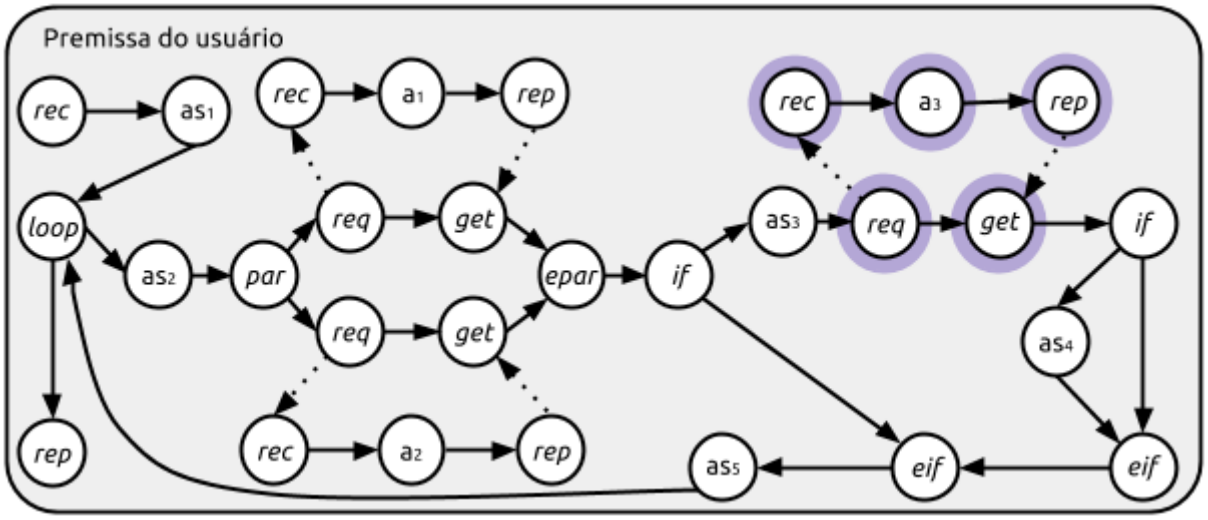


Figura 6.10 – GWM com atividades marcadas para alocação na nuvem ou na premissa seguindo a distribuição referente ao menor custo total.

6.4 Decomposição

A etapa de decomposição é executada a partir do GWM com distribuição marcada pela etapa de seleção de localização. A decomposição é iniciada pela fase de fragmentação. A Figura 6.11 ilustra o GWM e a lista resultante ao término da primeira fase da decomposição aplicada ao GWM ilustrado na Figura 6.10.

A partir da lista emitida pela fase de fragmentação é executada a fase de agrupamento, a qual cria o novo subprocesso com base nos aspectos que compõem a lista de entrada. A Figura 6.12 ilustra o subprocesso resultante da fase de agrupamento a partir da lista ilustrada na Figura 6.11.

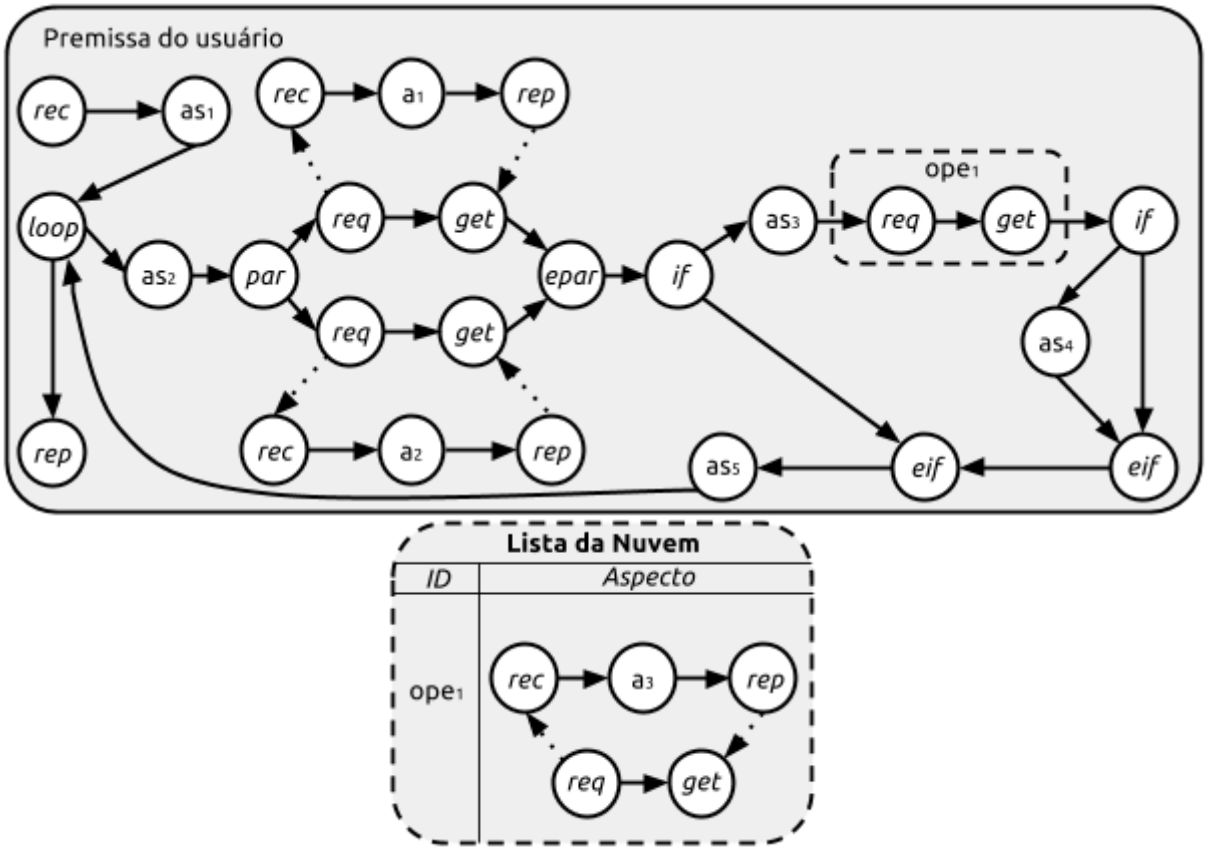


Figura 6.11 – GWM e lista emitidos na fase de fragmentação.

A última fase da decomposição, a criação da coreografia e fluxo de dados, é executada a partir do GWM emitido pela fase de fragmentação e dos GWMs emitidos pela fase de agrupamento. A Figura 6.13(a) destaca a coreografia criada entre o GWM da Figura 6.11 e o GWM da Figura 6.12. O novo fluxo de dados criado entre os mesmos GWMs é destacado na Figura 6.13(b).

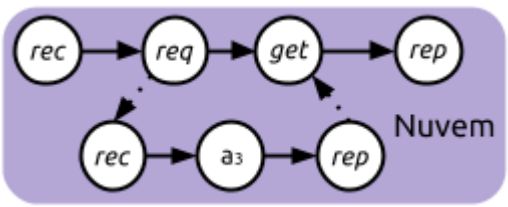


Figura 6.12 – GWM resultante da fase de agrupamento.

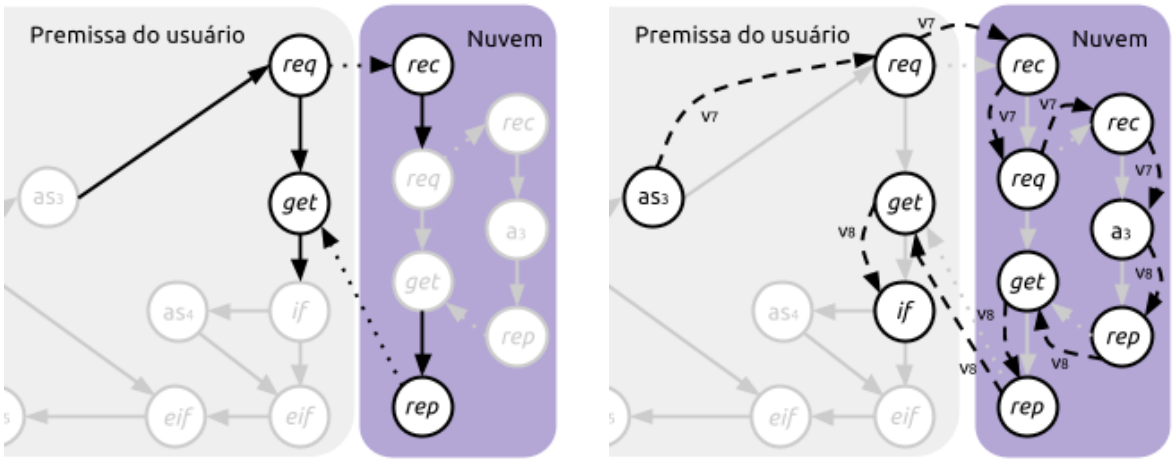


Figura 6.13 – GWMs com destaque da comunicação (a) e do fluxo de dados (b).

A Figura 6.14 ilustra o processo de negócio em GWM emitido pela etapa de decomposição, sendo que o fluxo de dados e os rótulos das arestas estão omitidos.

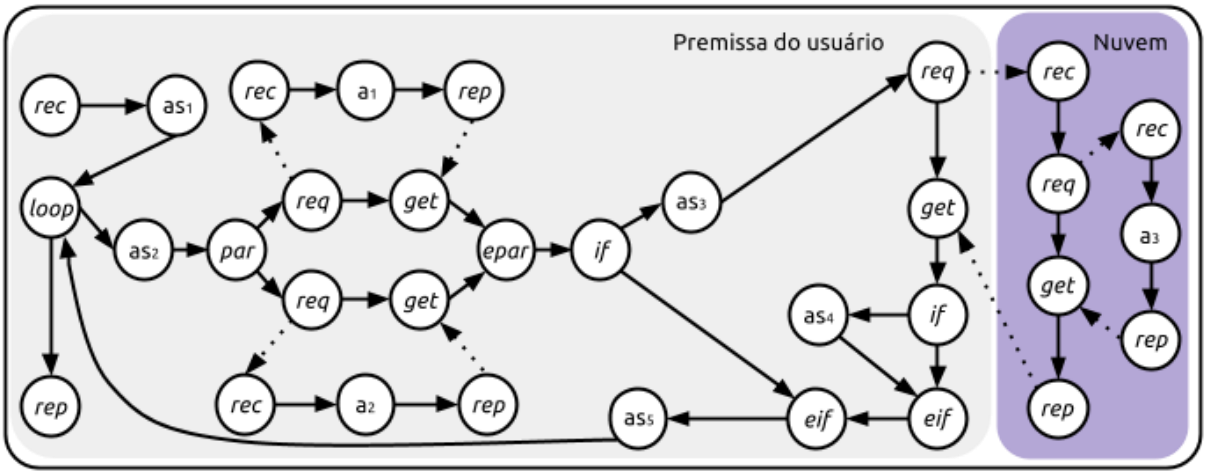


Figura 6.14 – PACS decomposto em GWM.

6.5 Grounding

A partir do GWM emitido pela etapa de decomposição é executada a etapa de *grounding*. Tal etapa gera o código XML correspondente a cada subprocesso gerado. O Quadro 6.2 descreve trecho do código XML referente ao subprocesso da

premissa, destacando a principal mudança em relação ao processo monolítico, e o Quadro 6.3 descreve trecho do código XML referente ao subprocesso da nuvem.

Quadro 6.2 – Trecho do código XML correspondente ao subprocesso da premissa gerado pela etapa de decomposição.

```
<process id="decomposed-onpremise" ...>
 ...
 <intermediateThrowEvent id="IntermediateThrowEvent_2" ...>
  <incoming>SequenceFlow_16</incoming>
  <outgoing>SequenceFlow_17</outgoing>
  <messageEventDefinition .../>
  ...
 </intermediateThrowEvent>
 <intermediateCatchEvent id="IntermediateCatchEvent_2" ...>
  <incoming>SequenceFlow_17</incoming>
  <outgoing>SequenceFlow_21</outgoing>
  <messageEventDefinition .../>
  ...
 </intermediateCatchEvent>
 ...
</process>
```

Quadro 6.3 – Trecho do código XML correspondente ao subprocesso da nuvem gerado pela etapa de decomposição.

```
<process id="decomposed-cloud" ...>
 <startEvent id="StartEvent_1">
  <outgoing>SequenceFlow_5</outgoing>
  <messageEventDefinition .../>
  ...
 </startEvent>
 <serviceTask id="ServiceTask_6" name="Nodule Detection" ...>
  <incoming>SequenceFlow_5</incoming>
  <outgoing>SequenceFlow_10</outgoing>
  ...
 </serviceTask>
 <endEvent id="EndEvent_1">
  <incoming>SequenceFlow_10</incoming>
  <messageEventDefinition .../>
  ...
 </endEvent>
 <sequenceFlow id="SequenceFlow_10"
  sourceRef="ServiceTask_6" targetRef="EndEvent_1"/>
 <sequenceFlow id="SequenceFlow_5"
  sourceRef="StartEvent_1" targetRef="ServiceTask_6"/>
</process>
```

A Figura 6.15 ilustra o modelo BPMN correspondente aos códigos XMLs descritos anteriormente.

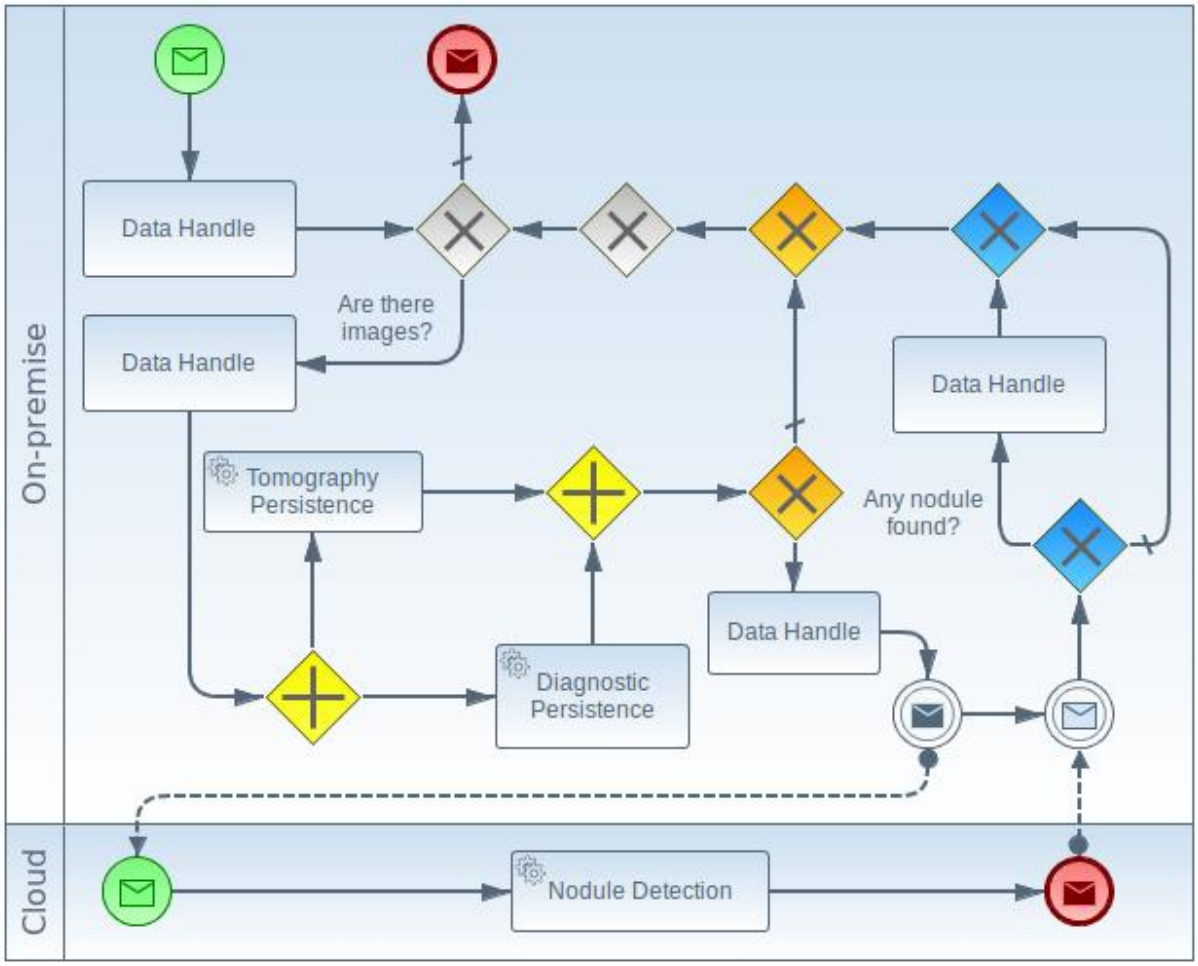


Figura 6.15 – Processo de negócio decomposto do PACS descrito em BPMN.

6.6 Análise de Desempenho

A fim de comparar o desempenho entre os processos monolítico e decomposto, o processo de negócio do PACS foi também especificado em WS-BPEL, o qual também foi submetido à abordagem de decomposição através do protótipo desenvolvido. A plataforma utilizada para os testes de desempenho foi composta pelo sistema operacional Debian 6, o servidor de aplicação Apache Tomcat 6, a tecnologia Java 6, o mecanismo de processos BPEL Apache ODE e o framework para disponibilizar Web Services Apache AXIS 2. O processo monolítico e o subprocesso da premissa foram executados no mesmo servidor da premissa considerado na etapa de seleção de localização descrito na Seção 6.3, enquanto o subprocesso da nuvem foi executado nas instâncias descritas na Tabela 6.2. A

Figura 6.16 ilustra a infraestrutura e a plataforma em que o processo decomposto foi executado, sendo que a parte que se refere à premissa também foi empregada para execução do processo monolítico.

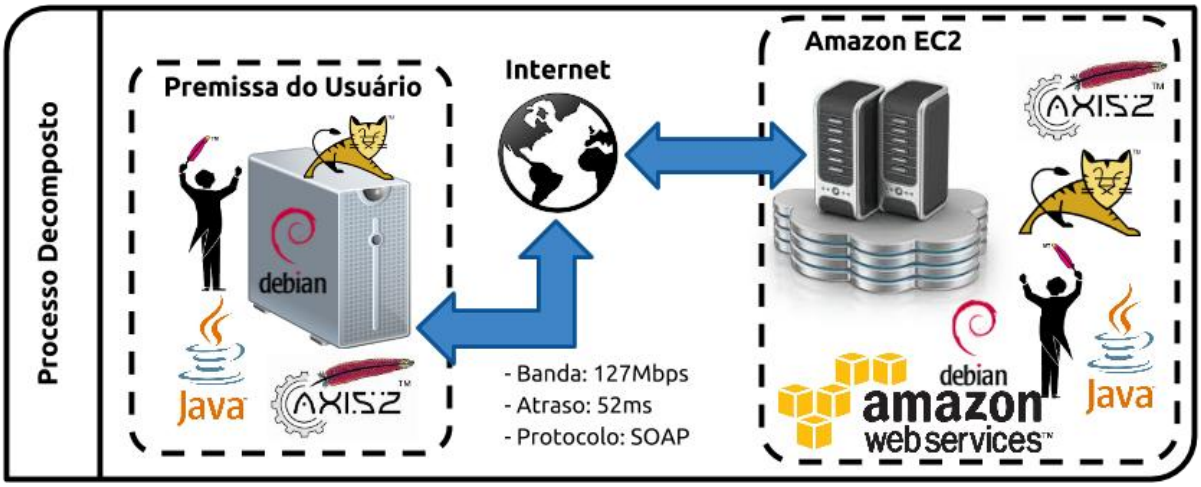


Figura 6.16 – Ambiente utilizado na análise de desempenho.

As execuções dos processos empregaram uma carga de trabalho composta por duas tuplas na forma <*id, diagnostic, image*>, onde *id* é um identificador de 4 bytes, *diagnostic* é um texto de 40 bytes e *image* é uma tomografia mamária de 11,1 MB. Foram coletadas cem amostras dos tempos de resposta dos processos monolítico e decomposto para cada instância *i* da nuvem. O percentual P_i de desempenho ganho do processo decomposto em relação ao monolítico para a $i^{ésima}$ configuração pode ser definido como (JAIN, 1991):

$$P_i = 1 - \frac{\bar{T}_{decomposto_i}}{\bar{T}_{monolítico}} \tag{30}$$

onde:

- $\bar{T}_{decomposto_i}$ é o tempo de resposta médio do processo decomposto na configuração *i*; e
- $\bar{T}_{monolítico}$ é o tempo de resposta médio do processo monolítico.

A Figura 6.17 ilustra o percentual de ganho de desempenho do processo decomposto em relação ao monolítico para cada uma das instâncias da nuvem, sendo que o percentual mínimo é superior a 14%.

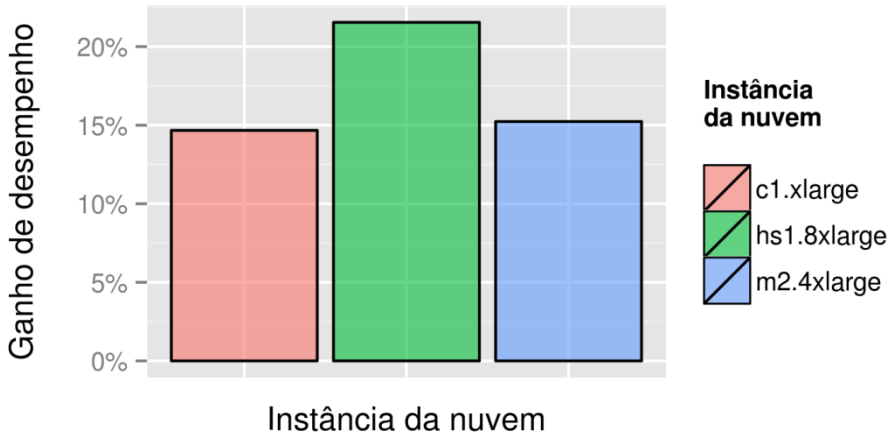


Figura 6.17 – Ganho de desempenho do processo decomposto.

Para verificar a hipótese de que as médias dos tempos de resposta do processo decomposto foram significativamente menores que a do processo monolítico, foi empregada a estatística do teste *t* (R CORE TEAM, 2013) a um nível de significância de 5%. Os testes resultaram em valores da probabilidade *p-value* na ordem de $1{,}3 \times 10^{-19}$, confirmando essa hipótese.

Observa-se na Figura 6.17 que o maior ganho de desempenho foi obtido com instância *hs1.8large*, a qual proporciona uma redução de aproximadamente 22% no tempo de resposta do processo de negócio.

6.7 Custos Relativos à Nuvem

Com a finalidade de analisar os custos adicionais agregados aos ganhos de desempenho, foi definido um modelo de regressão linear (KLOKE e MCKEAN, 2012) com base em quarenta e cinco observações de preços de três grandes provedores de IaaS. Tal modelo tem como base os dados da configuração da instância da nuvem utilizada, quantidade de RAM em MiB, quantidade de disco em GiB, o número de núcleos virtuais e a frequência em GHz de cada um desses núcleos, e

retorna o preço por hora em US$ da execução da instância. O valor estimado $\hat{y}$ do preço em US$/hora do recurso alocado na nuvem é definido como:

$$\hat{y} = 2{,}4882 \times 10^{-16} + 0{,}1350 \times a_1 + 0{,}0724 \times a_2 + 0{,}0835 \times a_3 + 0{,}0001 \times a_4 \qquad (31)$$

onde a_1, a_2, a_3 e a_4 são referentes à quantidade de RAM, ao número de vCPUs, à frequência de cada vCPU e à quantidade de HD, respectivamente.

Esse modelo possui o coeficiente de determinação R^2 de 89,62% e erro aleatório médio de US$ 0,0827, o qual foi determinado com a técnica de validação cruzada *leave-one-out* (KOHAVI, 1995). A Figura 6.18 ilustra a aderência dos valores estimados pela Equação (31) aos valores observados.

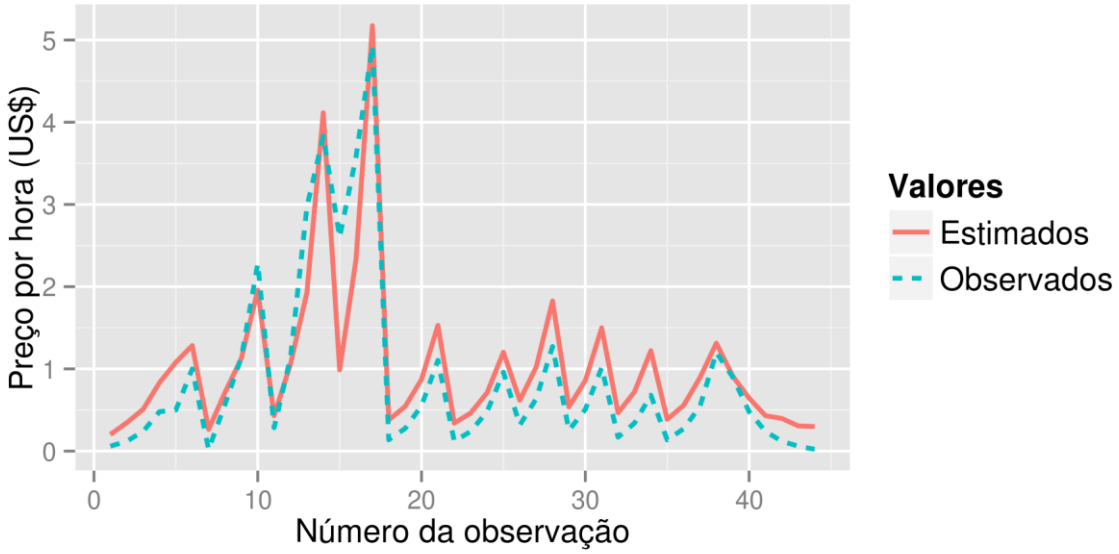


Figura 6.18 – Aderência dos valores estimados aos valores observados.

A Figura 6.19 ilustra a relação entre o custo por hora estimado pela Equação (31) de cada instância da nuvem empregada na análise de desempenho entre os processos monolítico e decomposto.

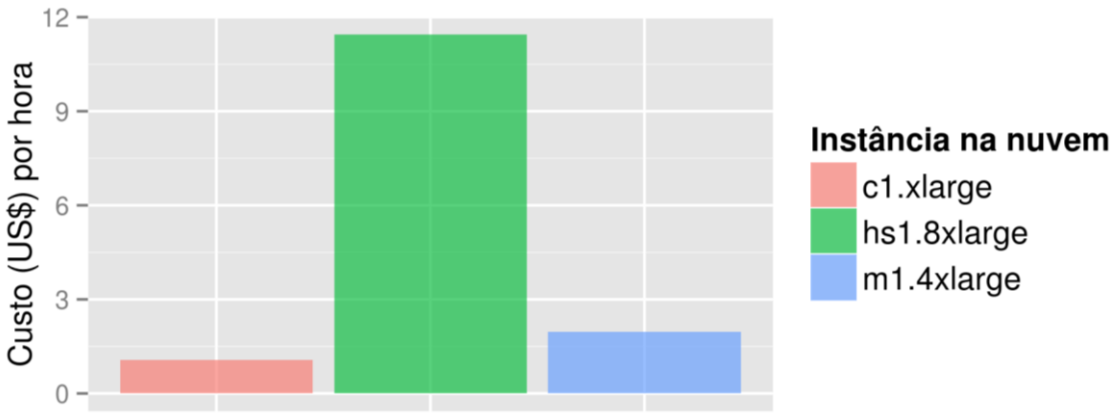


Figura 6.19 – Relação entre o ganho de desempenho e custo/hora da nuvem.

Capítulo 7

TRABALHOS CORRELATOS

O problema de decomposição de processos de negócio, também conhecido com descentralização de serviços web compostos, foi tratado primeiramente em (NANDA, CHANDRA e SARKAR, 2004). Nesse trabalho, orquestrações são criadas para cada serviço usado por um processo de negócio, resultando em uma comunicação direta entre esses ao invés de terem um único coordenador. A finalidade dessa abordagem é decompor um processo de negócio de tal forma que haja a maior quantidade de paralelização possível com o intuito de aumentar a vazão do sistema. O processo escrito em WS-BPEL é convertido para um grafo de fluxo de controle, que gera um PDG, a partir do qual são realizadas as transformações, e os novos grafos gerados são reconvertidos para WS-BPEL. Como nessa abordagem cada serviço no processo corresponde a um nó fixo para o qual uma partição é gerada, o PDG não é adequado para a abordagem desta pesquisa, pois esta visa à criação de processos nos quais múltiplos serviços possam ser usados.

Em (BARESI, MAURINO e S., 2006) é proposta a descentralização de processos WS-BPEL através de transformações baseadas em grafos. As regras dessas transformações são definidas, assim como o tipo de grafo com o qual essas são realizadas, e esses poderiam ser reutilizados em outras abordagens, desde que a linguagem de especificação de processos seja WS-BPEL. A abordagem desta pesquisa é mais genérica, lidando com linguagens de diferentes tipos.

O foco de (KOPP, KHALAF e LEYMANN, 2008) é na descentralização de orquestrações de processos WS-BPEL usando *Dead Path Elimination* (*DPE*) para garantir a conclusão da execução de processos descentralizados, mas DPE também torna a abordagem muito dependente da linguagem empregada na especificação do

processo de negócio. A Representação Intermediária (modelo intermediário) aqui empregada é independente dessa linguagem e, consequentemente também as estratégias e algoritmos de decomposição.

O trabalho apresentado em (AI, TANG e FIDGE, 2011) fornece uma solução desenvolvida com algoritmo genético baseado em penalidade para o mesmo problema apresentado em (NANDA, CHANDRA e SARKAR, 2004). Dessa forma, os resultados apresentados no primeiro são igualmente inviáveis aos resultados apresentados no último.

Em (CHENG e ZENG, 2012) é definida uma abordagem para decomposição de processos de negócio orientada a agentes móveis. A abordagem utiliza Grafos Acíclicos Direcionados (*DAGs*) para representar o processo de negócio. Essa representação consiste somente em estruturas sequenciais, paralelas e condicionais, considerando laços de repetição como atividades atómicas. A abordagem busca encontrar a partição que fornece os melhores resultados para dois itens de QoS denominados Custo de Execução Mínimo dentro de um Prazo (*MCD*) e Tempo de Execução Mínimo dentro de um Custo (*MTB*). Os cálculos do MCD e MTB são modelados através de Processos de Decisão de Markov (*MDP*). No entanto, dados os critérios e a arquitetura baseada em agentes móveis e também o alto nível de abstração das DAGs, os resultados apresentados não podem ser reaproveitados na definição da abordagem desta pesquisa.

Em (DUIPMANS, PIRES e SANTOS, 2013) é apresentado um *framework* para a decomposição de processos de negócio escritos na linguagem *AMBER* (EERTINK, JANSSEN, *et al.*, 1999), cujo código XML equivalente às representações do processo de negócio é exportado da ferramenta proprietária BiZZdesigner (BIZZDESIGN, 2014). Tal decomposição é executada com base em uma lista de distribuição de atividades e dados definida manualmente, onde restrições podem ser determinadas para assegurar que dados sensíveis permaneçam na premissa. Esse *framework* emprega uma abordagem que possui três etapas denominadas *lifting*, decomposição e *grounding*.

Em (DUIPMANS, PIRES e SANTOS, 2013) também é definido um modelo intermediário baseado em grafos, cujo objetivo é prover um modelo abstrato para captura de padrões de processos de negócio. Dessa forma, a etapa de decomposição ocorre sobre o modelo intermediário, o que torna necessário o mapeamento e a transformação da linguagem de processo de negócio empregada

no modelo intermediário, tarefa que caracteriza a etapa denominada *lifting*, e o processo inverso caracteriza o *grounding*. Essa trabalho também efetuou uma análise para definir as regras de decomposição suportadas pelo *framework*, concebidos algoritmos para a sua implementação e concebido um algoritmo para verificar se há restrições relativas aos dados violadas pela decomposição.

Entretanto, os resultados apresentados em (DUIPMANS, PIRES e SANTOS, 2013) possuem limitações que foram investigadas a partir dos resultados desta pesquisa:

- A linguagem AMBER, atrelada a ferramenta BiZZdesigner, só possui a capacidade de ser simulada, limitando as análises dos custos computacionais e financeiros e dos possíveis ganhos de desempenho, vazão e escalabilidade dados através da distribuição de processos de negócio para a execução em nuvens computacionais;
- A linguagem de especificação de processo de negócio empregada não possui um padrão aberto e não contém uma ferramenta livre para editar processos de negócio e simular seus comportamentos até onde se sabe;
- O modelo intermediário se baseia em um conjunto restrito de padrões de fluxo de controle de workflow, além disso, não foram considerados padrões de gerenciamento de exceção, dados e comunicação para a composição do modelo intermediário, o que inviabiliza sua aplicação para processos de negócio mais complexos;
- A formalização do modelo intermediário foi definida somente para a estrutura do modelo, desconsiderando a formalização de sua semântica, o que inviabiliza a possiblidade de provas formas de equivalência entre os processos monolítico e decomposto; e
- Os resultados não analisam os custos e ganhos de se decompor um processo de negócio para alocação de parte dele na nuvem, o que torna difícil ter uma percepção da real vantagem e dos custos inerentes à aplicação da abordagem.

O trabalho apresentado em (FDHILA, YILDIZ e GODART, 2009) aponta que a maioria das pesquisas em descentralização de orquestrações foca em determinadas linguagens de especificação de processos de negócio. Não focar tanto nessas

linguagens foi um dos principais desafios da pesquisa desta dissertação de mestrado, sendo que outro desafio foi não se preocupar somente com problemas de desempenho, mas também com medidas de segurança e custos relativos aos provedores de nuvem.

Capítulo 8

CONSIDERAÇÕES FINAIS

Este capítulo discorre as considerações finais da pesquisa desta dissertação de mestrado. A Seção 8.1 pontua as contribuições e publicações resultantes desta dissertação; a Seção 8.2 apresenta as respostas das questões de pesquisa definidas no início desta dissertação; e a Seção 8.3 apresenta os trabalhos futuros.

8.1 Contribuições

A pesquisa desta dissertação de mestrado resultou em uma abordagem para decomposição de processos de negócio para executar parte desses na premissa e parte na nuvem, considerando restrições de dados, custos inerentes aos provedores de nuvem e desempenho. Os processos de negócio monolíticos são capturados através de um modelo intermediário, denominado GWM, o qual é baseado em grafos direcionados e padrões de controle, dado, exceção e comunicação. A seleção da localização de atividades e dados associados de um processo é definida por meio de um framework de seleção baseado em um modelo de otimização inteiro. Em suma, as contribuições desta dissertação de mestrado são:

- A definição de um modelo intermediário baseado em grafos direcionados independente de linguagens de especificação de processos de negócio e com aspectos inerentes a fluxo de controle, fluxo de dados, comunicação e tratamento de exceções. Tal modelo permite que a abordagem proposta seja abstrata o suficiente para cobrir os padrões das principais linguagens de processos de negócio;

- A definição formal da estrutura e da semântica do modelo intermediário;
- O mapeamento e os algoritmos de transformação entre as linguagens de especificação de processos de negócio BPMN e WS-BPEL e o modelo intermediário;
- O framework de seleção de localização, na nuvem ou na premissa, de atividades e dados associados, considerando restrições de dados, desempenho e custos inerentes a provedores de nuvem;
- O conjunto de regras e algoritmos de decomposição para decompor processos de negócio de forma a manter o comportamento equivalente ao original; e
- O modelo de regressão para estimar o preço por hora de instâncias na nuvem de maneira independente de provedores específicos.

Além disso, esta dissertação de mestrado resultou nas seguintes publicações em eventos nacionais e internacionais:

- Wanderley Lopes de Souza, Luís Ferreira Pires, Evert F. Duipmans, Antonio Francisco do Prado, Lucas Venezian Povoa. Processo como um Serviço na Computação em Nuvem para o Desenvolvimento de Aplicações na Saúde. In Workshop de Informática Biomédica, Semana de Informática Biomédica, v. 2, p. 5-11, Ribeirão Preto. 2012. ISSN 2237-3594.
- Lucas Venezian Povoa, Wanderley Lopes de Souza, Luís Ferreira Pires, Evert F. Duipmans, Antonio Francisco do Prado. An Approach to Business Processes Decomposition for Cloud Deployment. In 2013 27th Brazilian Symposium on Software Engineering (SBES), p. 109-118, outubro 2013. DOI 10.1109/SBES.2013.1.
- Lucas Venezian Povoa, Wanderley Lopes de Souza, Luís Ferreira Pires, Antonio Francisco do Prado. An Approach to Decomposition of Business Processes for Execution in the Cloud. In 2013 11th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA). 2014.

8.2 Respostas às Questões de Pesquisa

As questões de pesquisa que foram definidas no início desta pesquisa foram respondidas:

QP1: Como modelar processos de negócio independentemente da linguagem utilizada para a especificação dos mesmos?

Na Seção 3.2 foi apresentado um modelo intermediário baseado em grafos com a finalidade de capturar padrões de linguagens de especificação de processos de negócio. Tal modelo, intitulado GWM, foi baseado em padrões de workflow (fluxo de controle, fluxo de dados e tratamento de exceções) e padrões de comunicação. Uma vez que tais padrões são definidos independentes de linguagens de especificação, o modelo também o é, o que tornou o GWM abstrato o suficiente para capturar padrões de linguagens de especificação de processos de negócio de diferentes tipos, ou seja, baseadas em blocos ou grafos. Portanto, os resultados desta pesquisa apresentam evidências que uma forma viável de modelar processos de negócio independentemente da linguagem utilizada para sua especificação é empregar modelos com fundamentação em padrões de workflow e comunicação.

QP2: Como decompor um processo de negócio para executar parte na nuvem e parte na premissa considerando restrições de segurança, desempenho e custo?

O problema de decompor processos de negócio nesta pesquisa foi definido a partir da definição do GWM. Portanto, sobre tal modelo foram definidas regras de decomposição que quando aplicadas geram processos de negócio decompostos equivalentes em observação a sua forma monolítica. A formalização do GWM e as provas de equivalência para cada regra de decomposição são apresentadas no Capítulo 4. Também a partir do GWM, foi construído um framework para definição da localização, na nuvem ou na premissa, das atividades e dados associados de um processo de negócio. Esse framework considera restrições de dados, custos inerentes à nuvem (hora de execução, armazenamento e transferência de dados) e também questões de desempenho. A partir dos resultados desta pesquisa é

aceitável que uma das maneiras possíveis de se decompor processos de negócio considerando restrições de segurança, desempenho e custo é unir um modelo intermediário com alguma heurística ou modelo de otimização que considera tais aspectos.

8.3 Trabalhos Futuros

A abordagem proposta considera como local de execução das partes de um processo de negócio decomposto um servidor na nuvem e um servidor na premissa. Entretanto, pode ser conveniente empregar múltiplos servidores na premissa e na nuvem, os quais podem estar alocados em diferentes provedores. Nesse sentido, é plausível concentrar esforços na extensão das regras e algoritmos de decomposição com a finalidade de considerar múltiplos servidores na premissa e na nuvem em diferentes provedores. Como consequência, existe também a necessidade de redefinir o framework de seleção de localização de atividades e dados associados, para que custos inerentes a diferentes provedores e também velocidades de conexão distintas sejam considerados.

A abordagem assume que todo serviço pode ser movido da premissa para a nuvem ou vice-versa sem qualquer tipo de restrição, a não ser de dados. Serviços externos podem pertencer a empresas ou instituições que não permitem tal mobilidade, disponibilizando acesso somente às interfaces de comunicação de tais serviços. Tais fatos justificam o estudo de heurísticas que considerem a velocidade de conexão entre as localidades disponíveis para a execução das partes do processo de negócio e os serviços não móveis.

Apesar da formalização da estrutura e da semântica do GWM, não foram desenvolvidas provas formais de equivalência entre os processos monolítico e decomposto definido através das regras de decomposição. Definir as provas de equivalência através dos teoremas de expansão e de congruência utilizando os axiomas da álgebra π-Calculus junto à relação de *weak bisimulation* (PARRONW, 2001) pode ser uma alternativa válida para definir as provas necessárias.

Finalmente, a acoplamento da abordagem a um BPMS é fundamental para torná-la o mais independente possível de intervenções do usuário, o que seria o primeiro passo para a decomposição de processos de negócio sob demanda.

REFERÊNCIAS

AI, L.; TANG, M.; FIDGE, C. Partitioning composite web services for decentralized. **Future Generation Computer Systems**, Amsterdam, The Netherlands, The Netherlands, v. 27, n. 2, p. 157-172, fevereiro 2011. ISSN 0167-739X. DOI 10.1016/j.future.2010.08.003.

AMAZON WEB SERVICES. Amazon Elastic Compute Cloud: user guide. **AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting**, 2014. Disponivel em: <http://goo.gl/Tg1XsA>. Acesso em: 27 agosto 2014.

ARMBRUST, M. et al. **Above the clouds: a Berkeley view of cloud computing**. University of California at Berkeley. Berkeley. 2009. (UCB/EECS-2009-28). Disponível em: http://goo.gl/bCruJb. Acessado em: 19 de junho 2014.

BALLANI, H. et al. Towards predictable datacenter networks. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v. 41, n. 4, p. 242-253, Agosto 2011. ISSN 0146-4833. DOI 10.1145/2043164.2018465.

BANG-JENSEN, J.; GUTIN, G. Z. **Digraphs:** theory, algorithms, and applications. 2ª. ed. Nova York, NY, EUA: Springer-Verlag, v. XXII, 2009. 798 p. ISBN 978-1-84800-998-1.

BANKS, D. L. The Health Insurance Portability and Accountability Act: Does It Live Up to the Promise? **Journal of Medical Systems**, v. 30, n. 1, p. 45-50, Fevereiro 2006. ISSN 1573-689X. DOI 10.1007/s10916-006-7403-2.

BARESI, L.; MAURINO, A.; S., M. Towards Distributed BPEL Orchestrations. **Electronic Communication of The European Association of Software Science and Technology** , v. 3, n. 1, p. 1-14, 2006. ISSN 1863-2122.

BECK, K. **Test Driven Development:** by example. Boston, MA, EUA: Addison-Wesley Logman Publishing Co., Inc, 2002. ISBN 0321146530.

BECK, K.; GAMMA, E. JUnit Cookbook, 2014. Disponivel em: <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>. Acesso em: 8 setembro 2014.

BENSON, T. et al. MicroTE: fine grained traffic engineering for data centers. **Proceedings of the Seventh Conference on emerging Networking Experiments and Technologies**, New York, NY, USA, 2011. 8:1--8:12. DOI 10.1145/2079296.2079304.

BIZZDESIGN. BiZZdesigner 11.5.1. **BiZZdesign:** building strong organizations, 2014. Disponivel em: <http://www.bizzdesign.com/downloads/register/41>. Acesso em: 9 setembro 2014.

BRASSCOM. **Índice BRASSCOM de convergência digital - 6ª Edição 2012**. Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (BRASSCOM). Olímpia, Brasil, p. 103. 2012. Disponível em: http://goo.gl/Rha6E. Acesso em: 20 de junho de 2014.

BUYYA, R. et al. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, v. 25, n. 6, p. 599–616, Junho 2009. ISSN 0167-739X. DOI 10.1016/j.future.2008.12.001.

CHENG, J.; ZENG, G. An agent-oriented approach to process partition and planning. **Engineering Applications of Artificial Intelligence**, Philadelphia, PA, EUA, v. 25, n. 4, p. 837 - 845, julho 2012. ISSN 0952-1976. DOI 10.1016/j.engappai.2011.09.027.

CURTISS, E. T.; EUSTIS, S. **Business Process Management (BPM) Market Shares, Strategies, and Forecasts, Worldwide, 2012 to 2018**. WinterGreen Research. [S.l.]. 2012.

DEELMAN, E. Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments. **International Journal of High Performance Computing Applications**, Thousand Oaks, CA, USA, v. 34, n. 3, p. 284-298, Agosto 2010. ISSN 1094-3420. DOI 10.1177/1094342009356432.

DUIPAMANS, E. F. **Business Process Management in the Cloud with Data and Activity Distribution**. Enschede, Holanda: University of Twente, 2012.

DUIPMANS, E. F.; PIRES, L. F.; SANTOS, L. O. B. D. S. A transformation-based approach to business process management in the cloud. **Journal of Grid Computing**, Dordrecht, the Netherlands, 2013. 1–29. DOI 10.1007/s10723-013-9278-z.

DUIPMANS, E. F.; PIRES, L. F.; SANTOS, L. O. B. S. Towards a BPM cloud architecture with data and activity distribution. **2012 IEEE 16th International Enterprise Distributed Object Computing Conference Workshops (EDOCW)**, Beijing, China, Setembro 2012. 165-171. DOI 10.1109/EDOCW.2012.30.

DUTTA, S.; BILBAO-OSORIO, B. **The Global Information Technology Report 2012: living in a hyperconnected world.** World Economic Forum and INSEAD. Genebra, Suiça, p. 441. 2012. ISBN 92-95044-33-9.

EERTINK, H. et al. A Business Process Design Language. In: WING, J. M.; WOODCOCK, J.; DAVIES, J. **Proceedings of the World Congress on Formal Methods in the Development of Computing Systems**. Toulouse: Springer-Verlag Berlin Heidelberg, 1999. p. 76-95. ISBN 03029743. Lecture notes in computer science, vol. 1708.

FDHILA, W.; YILDIZ, U.; GODART, C. A Flexible Approach for Automatic Process Decentralization Using Dependency Tables. **Web Services, 2009. ICWS 2009. IEEE International Conference on**, julho 2009. 847-855. DOI 10.1109/ICWS.2009.41.

FERRANTE, J.; OTTENSTEIN, K. J.; WARREN, J. D. The Program Dependence Graph and Its Use in Optimization. **ACM Transactions on Programming Languages and Systems**, New York, NY, EUA, v. 9, n. 3, p. 319-349, Julho 1987. ISSN 0164-0925. DOI 10.1145/24039.24041.

FURHT, B. Cloud computing fundamentals. In: FURHT, B.; ESCALANTE, A. **Handbook of Cloud Computing**. Nova York, NY, EUA: Springer, 2010. Cap. 1, p. 3-19. ISBN 9781441965240. DOI 10.1007/978-1-4419-6524-0.

HAN, Y. B. et al. A cloud-based BPM architecture with user-end distribution of non-compute-intensive activities and sensitive data. **Journal of Computer Science and Technology**, v. 25, n. 6, p. 1157–1167, Novembro 2010.

IBM SOFTWARE. **Comparing IBM WebSphere and Oracle WebLogic:** benefits of an IBM WebSphere application infrastructure, 2011. Disponivel em: <http://goo.gl/dri4G>. Acesso em: 19 junho 2014.

JAIN, R. **The art of computer systems performance analysis:** techniques for experimental design, measurement, simulation, and modeling. [S.l.]: Wiley, 1991. 1-685 p. ISBN ISBN: 978-0-471-50336-1.

JBOSS COMMUNITY. **jBPM**, 2014. Disponivel em: <http://goo.gl/0dOZf>. Acesso em: 19 junho 2014.

JIANG, J. et al. Analysis on Development Tendency of Business Process Management. In: LIU, C.; CHANG, J.; YANG, A. **Information Computing and Applications**. Berlin, Alemanha: Springer Berlin Heidelberg, v. 244, 2011. p. 629-636.

JIN, H. et al. Cloud types and services. In: FURHT, B.; ESCALANTE, A. **Handbook of Cloud Computing**. 1. ed. [S.l.]: Springer, v. XIX, 2010. Cap. 14, p. 335–355.

KLOKE, J. D.; MCKEAN, J. W. Rfit: Rank-based Estimation for Linear Models. **The R Journal**, v. 4, n. 2, p. 57-64, 2012.

KO, R. K. A computer scientist's introductory guide to business process management. **Crossroads**, New York, NY, USA, v. 15, n. 4, p. 4:11-4:18, Junho 2009. ISSN 1528-4972. DOI 10.1145/1558897.1558901.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: _____ **Proceedings of the 14th international joint conference on Artificial intelligence**. San Francisco: Morgan Kaufmann Publishers Inc., v. 2, 1995. p. 1137-1143. ISBN 1-55860-363-8.

KOPP, O. et al. The difference between graph-based and block-structured business process modelling languages. **Enterprise Modelling and Information Systems**, v. 4, n. 1, p. 3-13, 2009.

KOPP, O.; KHALAF, R.; LEYMANN, F. Deriving explicit data links in WS-BPEL processes. In: _____ **Proceedings of 2008 IEEE International Conference on Services Computing**. Washington, DC, USA: IEEE Computer Society, v. 2, 2008. p. 367-376. ISBN 9780769532837. DOI 10.1109/SCC.2008.122.

LINTHICUM, D. S. **Cloud Computing and SOA Convergence in Your Enterprise:** a step-by-step guide. [S.l.]: Addison-Wesley, 2010. ISBN 0136009220.

LIU, W. Research on Cloud Computing Security Problem and Strategy. **Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)**, 2012. 1216-1219. ISBN 978-1-4577-1412-2.

MARSTON, S. et al. Cloud Computing: the business perspective. **Decision Support System**, EUA, v. 51, n. 1, p. 179 – 189, 2011. ISSN 0167-9236. DOI 10.1016/j.dss.2010.12.006.

MAZANEK, S.; HANUS, M. Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language. **Journal of Visual Languages & Computing**, v. 22, n. 1, p. 66-89, 2011. ISSN 1045-926X. DOI 10.1016/j.jvlc.2010.11.005.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. National Institute of Standards and Technology. Estados Unidos da América, p. 1-3. 2011. (Special Publication 800-145). Disponível em: http://goo.gl/4LwvP. Acessado em: 12 de junho de 2014.

MILNER, R. **A Calculus of Communication Systems**. Nova York, NY, EUA: Springer-Verlag, 1980.

MILNER, R.; PARROW, J.; WALKER, D. A calculus of mobile processes, part i/ii. **Information and Computation**, v. 100, n. 1, p. 1-77, Setembro 1992.

MINERAUD, J. et al. Fs-PGBR: A Scalable and Delay Sensitive Cloud Routing Protocol. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v. 42, n. 4, p. 301-302, agosto 2012. ISSN 0146-4833. DOI 10.1145/2377677.2377741.

MOSHREF, M. et al. vCRIB: Virtualized Rule Management in the Cloud. **Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing**, Berkeley, CA, 2012. 1-6.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, v. 77, n. 4, p. 541-580, Abril 1989. ISSN 0018-9219. DOI 10.1109/5.24143.

NANDA, M. G.; CHANDRA, S.; SARKAR, V. Decentralizing Execution of Composite Web Services. **Proceedings of the Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications**, Nova York, EUA, 19, 2004. 170--187. DOI 10.1145/1028976.1028991.

OASIS STANDARD. **Web services business process execution language version 2.0**. Organization for the Advancement of Structured Information Standards. Burlington, MA, EUA, p. 264. 2007. (wsbpel-v2.0-OS).

OBJECT MANAGEMENT GROUP. **Business Process Model and Notation (BPMN): version 2.0**. Object Management Group. Needham, MA, EUA, p. 508. 2011. (Technical Report formal/2011-01-03).

OBJECT MANAGEMENT GROUP. Unified Modeling Language (UML), 2014. Disponivel em: <http://www.uml.org/>. Acesso em: 03 setembro 2014.

OOSTERWIJK, H. **PACS fundamentals**. [S.l.]: OTech, 2004. ISBN 0971886733.

ORACLE. java.com: Java + You, 2014. Disponivel em: <http://java.com>. Acesso em: 8 setembro 2014.

PAPAZOGLOU, M. P.; HEUVEL, W. Service Oriented Architectures: Approaches, Technologies and Research Issues. **The International Journal on Very Large Data Bases**, Secaucus, NJ, USA, v. 16, n. 3, p. 389-415, Julho 2007. ISSN 1066-8888. DOI 10.1007/s00778-007-0044-3.

PARRONW, J. An Introduction to the π-Calculus. In: BERGSTRA, J. A.; PONSE, A.; SMOLKA, S. A. **Handbook of Process Algebra**. Nova York, NY, EUA: Spring Science, 2001. p. 479-543. ISBN 0444828303.

PEREZ, R. S. **Business protocol adaptors for flexible chain formation and enactment**. Eindhoven University of Technology. Eindhoven, The Netherlands. 2012. (Tese de Doutorado). DOI 10.4121/uuid:c0d66060-4d6c-431d-9a10-567fe63b1bc8.

PILATO, C. M.; COLLINS-SUSSMAN, B.; FITZPATRICK, B. W. **Version Control with Subversion**. 1ª. ed. Sebastopol: O'Reilly Media, 2008. ISBN 9781449379353.

POLYVYANYY, A. et al. Maximal structuring of acyclic process models. **Computer Journal**, Cary, EUA, v. 57, n. 1, p. 12-35, julho 2014. ISSN 1460-2067. DOI 10.1093/comjnl/bxs126.

PUHLMANN, F.; WESKE, M. Using the π-calculus for formalizing workflow patterns. **Proceedings of the 3rd international conference on Business Process Management (BPM'05)**, Berlin, Heidelberg, 2005. 153-168. DOI 10.1007/11538394_11.

R CORE TEAM. **R: A Language and Environment for Statistical Computing**. R Foundation for Statistical Computing. Vienna, Austria. 2013. (3-900051-07-0).

RUH, W. A.; MAGINNIS, F. X.; BROWN, W. J. Basic Building Blocks. In: _____ **Enterprise application integration:** a Wiley tech brief. Nova York, NY, EUA: John Wiley & Sons, 2001. p. 39–60. ISBN 0471376418.

RUSSELL, N. et al. Workflow data patterns: identification, representation and tool support. **Proceedings of the 24th International Conference on Conceptual Modeling**, Lecture Nodes in Computer Science, vol. 3716, pp. 353–368, 2005. ISBN 3-540-29389-2.

RUSSELL, N. et al. **Workflow Control-Flow Patterns: A Revised View**. BPMcenter.org. BPM Center Report BPM-06-22. 2006.

RUSSELL, N.; VAN DER AALST, W.; TER HOFSTEDE, A. Workflow exception patterns. In: DUBOIS, E.; POHL, K. **Advanced Information Systems Engineering**.

Heidelberg: Springer Berlin Heidelberg, 2006. p. 288–302. LNCS vol. 4001. DOI 10.1007/11767138_20.

SPECTOR, A.; NORVIG, P.; PETROV, S. Google's Hybrid Approach to Research. **Communications of the ACM**, Nova York, EUA, v. 55, n. 7, p. 34-37, julho 2012. ISSN 0001-0782. DOI 10.1145/2209249.2209262.

VAN DER AALST, W. M. P.; HOFSTEDE, A. H. M. T.; WESKE, M. Business Process Management: A Survey. **Proceedings of the 2003 International Conference on Business Process Management**, Eindhoven, The Netherlands, 2003.

VAN DER AALST, W.; TER HOFSTEDE, A. YAWL: yet another workflow language. **Information Systems**, Ocford, UK, Reino Unido, v. 30, n. 4, p. 245-275, Junho 2005. ISSN 0306-4379. DOI 10.1016/j.is.2004.02.002.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J. A break in the clouds: towards a cloud defition. **SIGCOMM Comput. Commun. Re.**, New York, NY, USA, v. 39, n. 1, p. 50-55, dezembro 2009. ISSN 0146-4833. DOI 10.1145/1496091.1496100.

W3C. **Web Services Choreography Description Language Version 1.0**. World Wide Web Consortium. Cambridge, MA, EUA. 2005. (Candidate Recommendation CR-ws-cdl-10-20051109).

WESKE, M. **Business Process Management:** Concepts, Languages, Architectures. Secaucus: Springer-Verlag, 2007. ISBN 3540735216.

YU, S. et al. Achieving secure, scalable, and fine-grained data access control in cloud computing. **Proceedings of the 29th Conference on Information Communications (INFOCOM'10)**, San Diego, Califronia, Estados Unidos da América, 2010. 534-542.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, Heidelberg, Alemanhã, v. 1, n. 1, p. 7–18, 2010. ISSN 1867-4828. DOI 10.1007/s13174-010-0007-6.

ZISSIS, D.; LEKKAS, D. Addressing cloud computing security issues. **Future Generation Computer Systems**, Philadelphia, EUA, v. 28, n. 3, p. 583 – 592, março 2012. ISSN 0167-739X. DOI 10.1016/j.future.2010.12.006.