Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Melhoria de Modelos de Processo de Negócio com Mineração de Processos e Simulação Baseada em Agentes

Fernando Szimanski

Tese apresentada como requisito parcial
para conclusão do Doutorado em Informática

Orientadora
Prof.ª Dr.ª Célia Ghedini Ralha

Coorientador
Prof. Dr. Diogo R. Ferreira

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Doutorado em Informática

Coordenador: Prof. Dr. Ricardo Pezzuol Jacobi

Banca examinadora composta por:

Prof.ª Dr.ª Célia Ghedini Ralha (Orientadora) — CIC/UnB
Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB
Prof.ª Dr.ª Genaína Nunes Rodrigues — CIC/UnB
Prof.ª Dr.ª Flávia Maria Santoro — UNIRIO
Prof. Dr. João Paulo Andrade Almeida — UFES

CIP — Catalogação Internacional na Publicação

Szimanski, Fernando.

Melhoria de Modelos de Processo de Negócio com Mineração de Processos e Simulação Baseada em Agentes / Fernando Szimanski. Brasília : UnB, 2013.
206 p. : il. ; 29,5 cm.

Tese (Doutorado) — Universidade de Brasília, Brasília, 2013.

1. Agent-Object Relationship framework,
2. Expectation-Maximization, 3. Modelo Hierárquico de Markov

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Melhoria de Modelos de Processo de Negócio com Mineração de Processos e Simulação Baseada em Agentes

Fernando Szimanski

Tese apresentada como requisito parcial
para conclusão do Doutorado em Informática

Prof.ª Dr.ª Célia Ghedini Ralha (Orientadora)
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof.ª Dr.ª Genaína Nunes Rodrigues
CIC/UnB

Prof.ª Dr.ª Flávia Maria Santoro
UNIRIO

Prof. Dr. João Paulo Andrade Almeida
UFES

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Doutorado em Informática

Brasília, 18 de dezembro de 2013

Dedicatória

Dedico este trabalho a minha esposa Danusa Pegoraro Szimanski. Agradeço a companhia, paciência e apoio incondicional. Foram muitos acontecimentos durante este período, tivemos ótimas histórias e você sempre esteve ao meu lado. Eu te amo!

Agradecimentos

A professora Célia Ghedini Ralha por me introduzir nesta área de pesquisa. Sou muito grato pela excelente orientação, dedicação, colaboração e paciência em todos os momentos durante esses anos de estudo. Foi um período de grande aprendizado, muitas coisas boas aconteceram neste período.

Ao professor Diogo R. Ferreira pelo compartilhamento do seu conhecimento na área, total apoio no desenvolvimento e acompanhamento dos trabalhos que foram realizados. Esta parceria exerceu um papel fundamental nesta jornada.

Aos meus familiares, principamente aos meus pais, Alcindo e Neusa Szimanski por sempre me encorajarem durante todo o período acadêmico. Agradeço a Deus pela família que tenho e pela educação que recebi dos meus pais, espero um dia representar para meus filhos o que eles representam para mim.

A todos os colegas de estudo, pela companhia, colaboração e solidariedade em todos os momentos desta empreitada.

Resumo

Técnicas de mineração de processos permitem a descoberta de modelos de processos em logs de eventos. Apesar dos importantes progressos nesta área, os modelos de processo minerados se apresentam frequentemente de forma complexa dificultando a compreensão dos processos de negócio. Essa complexidade envolve de forma implícita as relações entre as atividades de alto nível de abstração, utilizadas pelos analistas ao descrever seus processos de negócio, e as atividades de baixo nível representadas nos logs de eventos. De fato, nas técnicas atuais de mineração de processos existe uma lacuna entre os eventos de baixo nível e a abstração de alto nível das atividades empresariais. Esta pesquisa aborda o problema através do desenvolvimento de uma técnica de mineração de processos denominada modelo hierárquico de Markov. Este modelo é capaz de produzir a hierarquização de processos relacionando os eventos de alto e baixo nível de abstração utilizando um procedimento de descoberta baseado na técnica denominada *Expectation-Maximization*. Para a geração dos logs de eventos mais próximos a realidade organizacional, relacionando os diversos agentes aos múltiplos cenários dos processos de negócio, foi utilizado uma plataforma de simulação baseada em agentes denominada *Agent-Object Relationship* (AOR) *framework*. Com a finalidade de avaliar experimentalmente a abordagem proposta foram descritos três modelos de processo de negócio utilizando *Business Process Modeling Notation* associados à simulação baseada em agentes para gerar os logs de eventos de baixo nível na plataforma AOR e também implementado um estudo de caso utilizando um log de eventos real. Os resultados foram comparados com técnicas de mineração de processos existentes no *framework* ProM. Também foram estudadas e selecionadas métricas para avaliar quantitativamente aspectos de complexidade dos modelos de processos hierárquicos gerados. O resultado da avaliação indicou a viabilidade do modelo hierárquico proposto para preencher a lacuna entre os dois níveis de abstração nos processos de negócio existentes nas organizações.

Palavras-chave: Agent-Object Relationship framework, Expectation-Maximization, Modelo Hierárquico de Markov

Abstract

Process mining techniques allow the discovery of process models in event logs. Despite the significant progress in this field, the mined process models are often presented in a complex form, difficulting the understand of business processes. This complexity involves implicitly the relationship between the high-level abstraction used by analysts to describe their business processes, and the low-level activities, represented in the event logs. In fact, there is a gap between the low-level events and the high-level abstraction of business activities. This research addresses this problem by developing a process mining technique so called hierarchical Markov model. This model is able to produce a hierarchical process model relating the low- and high-level events through the use of a discovering procedure based on the Expectation-Maximization technique. For the event logs generation closest to the organizational reality, where agents are involved in different business scenarios, we used the Agent-Object Relationship (AOR) simulation framework. In order to evaluate the proposed approach three different business process models were described using the Business Process Modeling Notation, together with the agent-based simulation platform (AOR) to generate the event logs, and also implemented a case study using a real-life event log. The results were compared with existing process mining techniques in the ProM framework. Also, metrics were studied and selected to quantitatively evaluate aspects of complexity in the generated hierarchical process models. The evaluation results indicated the feasibility of the hierarchical proposed model to fill the gap between the two business process abstraction levels existents in organizations.

Keywords: Agent-Object Relationship framework, Expectation-Maximization, Hierarchical Markov Model

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ABS Agent-based Simulation

AD UML 2.0 Activity Diagram

AOR Agent-Object-Relationship

AORML AOR Modeling language

AORSL AOR Simulation Language

BPDM Business Process Definition Metamodel

BPI Business Process Improvement

BPM Business Process Management

BPMI Business Process Management Initiative

BPMN Business Process Modeling Language

BPR Business Process Reengineering

CRM Customer Relationship Management

EM Expectation-Maximization

EPC Event-driven Process Chain

ERP Enterprise Resource Planning

FSM Finite State Machine

GQM Goal Question Metric

HMM Hidden Markov Model

MXML Mining XML

PAIS Process-Aware Information Systems

SI Sistemas de Informação

SMA Sistema Multiagente

TI Tecnologia da Informação

UML Unified Modeling Language

WFM Workflow Management

XES eXtensible Event Stream

XML eXtensible Markup Language

YAWL Yet Another Workflow Language

Capítulo 1

Introdução

Este capítulo apresenta a contextualização, justificativa, a definição do problema, os principais objetivos da pesquisa, as contribuições, a metodologia utilizada no desenvolvimento deste trabalho, bem como a estrutura e conteúdo dos demais capítulos.

1.1 Contextualização

O entendimento dos processos de negócio é estratégico nos ambientes empresariais, uma vez que facilita o alcance eficiente dos objetivos organizacionais estabelecidos (Georgakopoulos et al. 1995, Laguna and Marklund 2004, Weske 2007). Além de proporcionar resposta mais rápida aos clientes, melhorar a qualidade do trabalho e promover a utilização eficaz dos recursos, os processos de negócio fomentam o desenvolvimento de Sistemas de Informação (SI) empresariais. Desta forma, mais valor é agregado através da definição dos papéis e das responsabilidades dos *stakeholders*, bem como melhores definições dos "objetos" que serão tratados pelo negócio como base para a criação dos SI.

Dentro de um ambiente empresarial, traduzir requisitos de especificação de sistema é uma resposta crucial para qualquer projeto de engenharia de software. O maior desafio nesta área são as diferentes perspectivas dos analistas de negócio e os analistas de sistemas ao interpretar os fenômenos do mundo real em um ambiente empresarial limitado a específicos cenários.

A modelagem de processos de negócio foi identificada como uma importante ferramenta para estabelecer uma ponte entre os negócios e o desenvolvimento de software; a qual entre outras vantagens facilita o projeto estruturado (Yourdon 1989), viabiliza o alinhamento entre negócios de Tecnologia da Informação (TI) (Luftman et al. 1999) e contribui para o processo de conhecimento na engenharia de sistemas (Kindler 2009).

Nesta direção, a *Business Process Management Initiative*(BPMI) desenvolveu o padrão para modelagem de processos de negócio *Business Process Modeling Notation* (BPMN) (OMG 2011). O principal objetivo do BPMN é fornecer uma notação que seja facilmente compreensível por todos os usuários de negócios de tal forma que os processos de negócios possam ser representados através de uma notação padrão compreensível tanto por analistas de negócios que criam os rascunhos iniciais dos processos, como por desenvolvedores de sistemas responsáveis por implementar a tecnologia que irá executar os processos e, finalmente, por pessoas de negócios que irão gerenciar e monitorar estes processos. Uma introdução completa sobre processos de negócios é fornecida por Weske (2007).

BPMN é capaz de apoiar o projeto de um processo de negócio, no entanto não é capaz de garantir a eficiência da execução. Tal como indicado por Wong and Gibbons (2008) a notação BPMN não tem uma semântica formal de comportamento, o que é importante na especificação e verificação do comportamento. Há várias abordagens que tentam resolver este problema. As abordagens tradicionais são baseadas em Redes de Petri (van der Aalst 1998) (e.g. *Event-driven Process Chains* (EPC's) (Scheer 2000) e *Workflow Nets* (Verbeek et al. 1999)). Neste sentido, consideramos que a utilização de técnicas de simulação baseada em agentes possibilita a execução de modelos de processos através de uma semântica formal de comportamento, que pode ser uma abordagem interessante para proporcionar melhoria no modelo da aplicação a partir de modelos de processos mais estáveis (i.e. "equilibrados" em termos de complexidade).

1.2 Justificativa

Se tentarmos entender um processo de negócio observando as pessoas trabalhando em uma organização, a natureza aparentemente caótica dos eventos pode ser muito confusa. No entanto, se uma descrição de alto nível do processo de negócio estiver disponível (e.g. particionando o processo em duas ou três etapas principais), a sequência dos eventos torna-se mais compreensível.

Neste contexto, os processos de negócio são normalmente modelados em um alto nível de abstração, no entanto, em tempo de execução, processos de negócio são executados por recursos (sejam eles humanos, sistemas ou máquinas) que interagem uns com os outros a fim de executar cada atividade no processo. Neste trabalho, vamos nos referir a estes recursos como *agentes*, no sentido de agentes inteligentes (Wooldridge and Jennings 1995), que são capazes de colaborar uns com os outros através da troca de mensagens (Lashkari et al. 1998). A Figura 1.1 ilustra um exemplo da interação de agentes representada como um diagrama de sequência UML, onde a atividade A é realizada através da troca de mensagens entre os agentes X, Y e Z. Em particular, o agente X envia a mensagem M1 para o agente Y, o agente Y envia a mensagem M2 para o agente Z e o agente Z responde a mensagem M3 para o agente Y.

Assumimos que estes agentes operam sobre uma infraestrutura de sistemas ou em um ambiente onde é possível registrar suas interações na forma de eventos de comunicação. Tipicamente, cada evento se refere a uma operação que pode ser executada por algum agente durante a execução de uma instância do processo e estes eventos de baixo nível são registrados em um log de eventos. Enquanto o processo de negócio é descrito em termos de atividades de alto nível que são familiares aos usuários de negócios, a colaboração dos agentes em tempo de execução é registrada em um log de eventos como uma sequência de eventos de baixo nível.

A partir deste log de eventos, é possível analizar o comportamento de agentes em tempo de execução através de técnicas de mineração de processos (van der Aalst 2011). Estas técnicas permitem um estudo a partir de diversas perspectivas, incluindo a sequência das operações, bem como as interações entre os agentes durante a execução do processo. Uma vez que o mesmo processo pode ser instanciado e executado múltiplas vezes, a sequência dos eventos registrados para uma instância específica do processo é referenciado como um *trace*, enquanto que o conjunto de eventos registrados para todas as instâncias do processo é referenciado como um log de eventos completo.

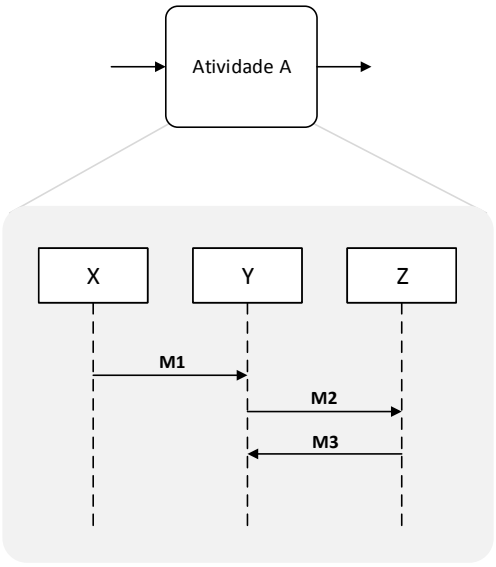


Figura 1.1: Atividade realizada como um conjunto de trocas de mensagens entre os agentes

Tipicamente, nas técnicas atuais de mineração cada evento em um log de eventos contém os seguintes atributos: a *instância* (*caseid*) do processo; a *atividade* que foi executada na instância do processo; o *agente* que executou a atividade; e a *data e hora* (*timestamp*) que o evento ocorreu. No entanto, no cenário da Figura 1.1, onde cada atividade é executada como um conjunto de troca de mensagens entre agentes, os eventos que acabam sendo registrados no log de eventos são as trocas de mensagens entre os agentes. Portanto, no contexto deste trabalho, cada evento tem os seguintes atributos:

- instância – identifica a instância do processo;
- remetente – identifica quem enviou a mensagem;
- mensagem – indica o tipo de mensagem a ser enviada;
- destinatário – identifica quem recebeu a mensagem;
- data e hora – registra a data e a hora em que a mensagem foi enviada.

Um exemplo de três eventos que são gerados com um resultado de execução da atividade A na Figura 1.1 é apresentado na Tabela 1.1. É importante notar que neste log de eventos aparentemente não existe relacionamento entre os eventos registrados e sua atividade correspondente no processo de negócio. Em outras palavras, não existe nada para indicar que estes eventos pertencem a atividade A. Isto cria uma dificuldade, porque apesar de que log de eventos tal como descrito na Tabela 1.1 possam ser analisados para estudar o comportamento dos agentes, na prática muitas vezes há uma lacuna entre os eventos de baixo nível registrados em um log de eventos e as atividades de alto nível que são utilizadas para descrever o processo de negócio.

Tipicamente, os logs de eventos podem ser analisados utilizando técnicas de mineração de processos para extrair diferentes tipos de modelos comportamentais (van der Aalst 2011). Em particular, existem técnicas para minerar processos sob diversas perspectivas,

instância	*remetente*	*mensagem*	*destinatário*	*data e hora*
...	...	...	...	...
1	X	M1	Y	2013-04-17 14:53
1	Y	M2	Z	2013-04-17 18:12
1	Z	M3	Y	2013-04-18 10:25
...	...	...	...	...
2	X	M1	Y	2013-04-18 17:10
2	Y	M2	Z	2013-04-19 09:37
2	Z	M3	Y	2013-04-19 13:21
...	...	...	...	...

Tabela 1.1: Log de eventos contendo os eventos gerados durante a execução do processo

como "controle de fluxo", "organizacional" e "performance". Geralmente estas técnicas assumem que cada evento do log de eventos corresponde a execução de uma certa atividade no processo de negócio. No entanto, no log de eventos da Tabela 1.1 cada evento se refere a uma troca de mensagem que ocorre entre os agentes, e o relacionamento entre troca de mensagens e as atividades no processo de negócio não é claro.

1.3 Definição do problema

Em geral, tanto a descrição de alto nível (nível macro) do processo de negócio e a sequência de baixo nível (nível micro) dos eventos podem ser obtidas: a primeira é fornecida por analistas de negócios ou na documentação do processo e a segunda pode ser encontrada em logs de eventos. No entanto, a forma mais provável em que os eventos de nível micro podem ser mapeados para as atividades de nível macro é desconhecida e é precisamente isso que queremos descobrir. Dado um log de eventos de nível micro e um modelo de processo de negócio de nível macro, o nosso objetivo é descobrir: (1) um modelo de nível micro para o comportamento dos agentes; e (2) como este modelo de nível micro se encaixa na descrição de nível macro do processo de negócio. Para esta finalidade, desenvolvemos um modelo hierárquico de Markov que é capaz de capturar o macro comportamento do processo de negócio, o micro comportamento dos agentes ao trabalhar em cada atividade e a relação entre os dois.

As abordagens de mineração tradicionais não permitem este relacionamento entre o alto-nível de abstração de modelos de processo com o baixo nível de eventos registrados no log de eventos, que é essencial em uma organização, conforme descrito nas Seções 2.3 e 2.5. Tal relacionamento é útil para aumentar o conhecimento do gestor no modelo de processo e assim verificar oportunidades de melhoria. Salienta-se que um excelente modelo de alto-nível (macro-modelo) não é uma restrição do trabalho, pois independente do macro-modelo fornecido como entrada, será apresentado o resultado mais provável para tal configuração, que pode ser um resultado mais/menos complexo em termos da métricas apresentadas na Seção 3.4.

Adicionalmente, visando encontrar a melhor relação entre o macro-modelo e os micro-modelos, torna-se necessária a avaliação desta relação de hierarquia em termos de complexidade, pois como poderemos dizer que o modelo hierárquico é "equilibrado" no sentido

de que todas as cadeias de Markov em seu modelo são de complexidade similar. Mas isso é o que acontece normalmente? Como podemos determinar se um determinado modelo hierárquico é equilibrado, e como podemos comparar dois modelos hierárquicos e dizer que um é mais complexo e equilibrado do que o outro?

A complexidade de um modelo de processo está relacionada ao equilíbrio com a simplicidade, pois se um modelo é muito simples, aspectos importantes que descrevem o processo podem estar faltando. Por outro lado, se um modelo é muito complexo, não será entendido por qualquer pessoa e assim pode reduzir a compreensibilidade. Esta também é outra questão que é abordada neste trabalho, onde através de uma abordagem iterativa de melhoria de processos propomos e ilustramos o uso de um conjunto de métricas para quantificar diversos fatores que impactam diretamente na complexidade dos modelos hierárquicos, como e.g. a densidade, o controle de fluxo, o tamanho e a modularidade.

O interesse em medir a complexidade destes modelos pode ser justificado pelo fato de que a aplicação de técnicas de mineração de processos para log de eventos do mundo real registra muitas vezes modelos espaguete (*spaghetti models*) (Günther and van der Aalst 2007). Estes são modelos que são muito complexos para o analista entender. Tais modelos surgem porque as técnicas de mineração de processos são aplicadas aos eventos de baixo nível no log de eventos e, quando se estuda o comportamento do processo com base nesses eventos de baixo nível, o processo pode se apresentar com uma complexidade elevada.

Portanto, nesta tese de doutorado foi construída uma abordagem de melhoria de modelos de processo de negócio com base nas seguintes premissas:

- A técnica de mineração de processos proposta não se refere às atividades de negócio que estão fora do escopo de sistemas transacionais;
- Em relação ao log de eventos, deve-se haver disponibilidade de informação sobre as "instâncias", pois caso a identificação única da instância esteja ausente, as atividades de mineração de processo podem ser prejudicadas;
- Em relação aos dados de entrada, deve-se haver disponibilidade de informações sobre o macro-modelo do processo, que é a descrição de alto nível do processo de negócio em questão;
- O trabalho pode ser aplicado no contexto de melhoria de modelos de processo de negócio em ambientes empresariais onde seja necessário preencher a lacuna entre diferentes níveis de abstração de modelos de processo de negócio;
- O modelo de processo minerado está em conformidade com o modelo de processo de negócio definido na organização.

1.4 Objetivos

A presente pesquisa tem como objetivo principal prospectar e definir um método híbrido de melhoria da compreensão de modelos de processo de negócio que integre o alto nível de abstração de processos negociais com o baixo nível de eventos registrados por SI. Faz parte deste objetivo a validação do modelo proposto com estudos exploratórios e um estudo de caso real comuns aos ambientes empresariais.

Adicionalmente, alguns objetivos secundários são propostos, a saber:

1. Investigar ambientes de simulação baseada em agentes para gerar logs de eventos a serem utilizados na mineração de processos. Através da simulação é possível gerar situações, evitando o desperdício de recursos na organizações;

2. Analisar diferentes algoritmos de mineração de processos para identificar as oportunidades e requisitos para a definição da abordagem de pesquisa;

3. Desenvolver uma técnica de mineração de processos com o foco no relacionamento entre a alto nível de abstração de processos de negócio e o baixo nível de eventos registrados por SI empresariais;

4. Investigar métodos e métricas para avaliação da complexidade de modelos de processo de negócio, apoiando a melhoria destes modelos;

5. Implementar as métricas de complexidade em estudos exploratórios comuns aos ambientes empresariais e também em um estudo de caso utilizando um log de eventos real. Através destes cenários de aplicação é possível demonstrar oportunidades de melhoria do modelo do processo de negócio, servindo de orientação a gestores de TI/analistas de negócio na melhoria destes modelos.

1.5 Contribuições

Ao alcançar os objetivos propostos, este trabalho avança o estado da arte nos seguintes pontos:

- Modelagem e implementação de exemplos de modelo de processo de negócio para simulação em uma plataforma baseada em agentes – especificamente, o *framework Agent-Object Relationship* (AOR) (Wagner 2004) – para gerar comportamento não-determinístico de baixo nível na execução do processo.
- Embora algumas abordagens de mineração de processos apresentem formas de abstração em modelos de processo de negócio, neste trabalho foi desenvolvida uma abordagem inovadora de mineração de processos que reduz a lacuna entre os eventos de baixo nível registrados em um log de eventos e as atividades de alto nível que são utilizadas para descrever o processo de negócio. Portanto, a abordagem descrita nesta pesquisa – particularmente, os algoritmos descritos na Seção 3.3 – podem ser utilizados em cenários reais para descobrir o comportamento em tempo de execução de processos de negócio em que um modelo de alto nível é conhecido.
- Formalização do método hierárquico de mineração de processos descrito na Seção 3.3.
- Um estudo abrangente sobre métricas aplicadas na avaliação de complexidade em modelos de processo de negócio e áreas correlatas, que serviram de base para a definição de métricas para avaliação da abordagem proposta neste trabalho, o modelo hierárquico de Markov e seus componentes.

- Uma abordagem iterativa para a melhoria de processos que permite analistas de negócio refinar os modelos de processo de negócio que são descobertos durante as iterações. Tal abordagem visa melhorar a compreensibidade dos modelos de processo de negócio dentro das organizações, o que pode levar tanto ao desenvolvimento de sistemas eficazes, eficientes e de agregação de valor ao negócio, como a redução de custos, identificação de gargalos, utilização eficaz dos recursos, entre outros fatores. Embora no Capítulo 4 voltamos a atenção para três cenários práticos de aplicação e um estudo de caso com um log de eventos real, a mesma abordagem pode ser utilizada para melhoria de modelos de processo de negócio em uma ampla gama de aplicações.

Apesar de eficientes as contribuições, a abordagem apresenta algumas limitações, como:

- A abordagem de melhoria proposta salienta as oportunidades de melhoria no modelo de processo de negócio, demonstrando a relação entre o relacionamento entre alto e baixo nível de abstração dos modelos. No entanto, a abordagem não executa esta melhoria de forma automatizada, pois somente auxilia gestores na identificação de oportunidades de melhoria no modelo de processo de negócio.
- Deve-se ter conhecimento do modelo de processo de negócio para modelagem e execução da simulação com agentes.
- A técnica de mineração de processos que foi desenvolvida nesta tese não visa realizar a verificação de conformidade de modelos, i.e. determinar até que ponto o processo está sendo executado como descrito no modelo existente. Sendo que o propósito deste trabalho é realizar a extensão de modelos, que consiste na melhoria de um modelo de processo existente com base no comportamento registrado no log de eventos;
- Em relação ao paralelismo de eventos inerentes a modelos de processo de negócio, a técnica de mineração de processos que foi desenvolvida não faz o tratamento deste tipo de ocorrência.
- Os cenários de simulação baseada em agentes que foram modelados nesta tese utilizam somente agentes reativos simples com troca de mensagens entre eles.

A relação detalhada de todas as publicações científicas desta pesquisa pode ser encontrada no Apêndice A.

1.6 Metodologia

Os objetivos da pesquisa foram obtidos seguindo uma metodologia que compreende o estudo, o desenvolvimento prático e teórico da abordagem proposta e exemplos de modelos de processos de negócio. A metodologia inclui as seguintes atividades:

- Explorar a partir de um ponto de vista teórico a aplicabilidade da simulação, mineração e medição de processos de negócio no contexto do nosso problema (objetivos 1 e 2);

- Desenvolver a abordagem proposta (objetivos 3 e 4), uma vez que isto é desenvolvido a partir da compreensão teórica dos estudos anteriores e de outras teorias relacionadas;
- Implementar os cenários de aplicação para validação da abordagem proposta (parte do objetivo principal).
- Avaliar os cenários de aplicação através de métricas de complexidade de processos de negócio auxiliando gestores a identificar oportunidades de melhoria nos modelos (objetivo 5).

O estudo de caso está entre as estratégias de pesquisa mais utilizadas na área de SI (Irani et al. 1999). Ele permite captar a realidade em grandes detalhes e pode ser utilizado para construir, testar e ampliar as teorias dentro de ambientes organizacionais (Galliers 1992). Conforme descrito por Yin (2003), o estudo de caso é uma abordagem viável em situações onde: (1) há uma necessidade de se realizar a pesquisa em seu ambiente natural; (2) os problemas residem dentro de um ambiente em rápida mutação; (3) existe uma ênfase nas questões *porquê* e *como*; e (4) existe uma carência de estudos anteriores e compreensão teórica elaborada no que diz respeito ao problema de pesquisa. Particularmente em nosso caso, os cenários de aplicação e o estudo de caso são utilizados para: (1) explorar, a partir de um ponto de vista prático, como a abordagem integrada de mineração de processos e simulação baseada em agentes pode ser utilizada na melhoria de processos; e (2) testar o desenvolvimento teórico da abordagem em ambientes de negócio.

1.7 Estrutura do documento

Conforme ilustrado na Figura 1.2, este documento é composto de teoria e prática compreendendo os seguintes capítulos.

O **Capítulo 2** desenvolve a base teórica da pesquisa com base no estudo da literatura de publicações disponíveis nas categorias teóricas envolvidas nesta pesquisa. A Seção 2.1 introduz os conceitos gerais sobre processos de negócio explorando a melhoria de modelos de processo através da reengenharia de redesenho de modelos, a modelagem e execução de processos. A Seção 2.2 apresenta os aspectos relacionados com a simulação baseada em agentes, como agentes inteligentes, os sistemas multiagentes e, em particular, o uso neste trabalho do *framework* AOR. A Seção 2.3 apresenta a mineração de processos e introduz os conceitos, técnicas e ferramentas existentes. A Seção 2.4 apresenta um estudo sobre as métricas tanto na área de processos de negócio como em áreas relacionadas que podem servir como base para avaliar a complexidade de modelos hierárquicos de processo de negócio. O capítulo termina com uma análise e reflexão sobre os resultados teóricos, bem como dos trabalhos correlatos que resultam na identificação de oportunidades e requisitos para a definição da proposta de pesquisa.

O **Capítulo 3** descreve a abordagem proposta neste trabalho: A Seção 3.1 introduz a abordagem iterativa de melhoria de modelos de processos de negócio através da integração de simulação baseada em agentes e mineração de processos. Na Seção 3.2 é discutido sobre a obtenção dos dados de entrada para a abordagem proposta. A Seção 3.2.1 descreve conceitualmente a modelagem e simulação baseada em agentes no sistema AOR para obter o comportamento de baixo nível de modelos de processo de negócio. A Seção 3.3

descreve o modelo hierárquico de Markov, que é uma técnica de mineração de processos que foi desenvolvida neste trabalho com o foco na descoberta de um modelo hierárquico a partir do comportamento de agentes em tempo de execução e uma descrição de alto nível do processo de negócio. Particularmente, esta seção relata a formalização da abordagem através de definições, a semântica de execução e os algoritmos que foram desenvolvidos para a mineração de processos através de um procedimento *expectation-maximization*. Também foi realizada uma verificação de integridade da abordagem de mineração proposta em padrões básicos de *workflow* para assegurar que é possível descobrir o comportamento dos agentes em processos de nível macro que envolvem mais do que apenas uma sequência linear de passos. Na Seção 3.4, são definidas as métricas utilizadas para avaliação de complexidade de modelos de processo de negócio.

O **Capítulo 4** inclui os estudos exploratórios que foram conduzidos em quatro cenários diferentes: (1) processo de compra; (2) processo de indenização de seguro; (3) processo de empréstimo bancário; e (4) por fim foi realizado um estudo de caso real no processo de gestão de incidentes da Volvo TI Bélgica. Em todos os cenários demonstramos o processo de aplicação da abordagem iterativa de melhoria de processos, com a geração do log de eventos através da simulação baseada em agentes (exceto o estudo de caso), a mineração de modelos hierárquicos através da abordagem de mineração proposta neste trabalho, a aplicação de métricas para avaliar a complexidade dos modelos minerados e análise dos modelos com o foco na melhoria de processos. Ao final os resultados foram comparados com outras abordagens existentes e analisados visando orientar as oportunidades de melhoria do modelo de processo de negócio através da redução de sua complexidade.

O **Capítulo 5** conclui o estudo com uma reflexão sobre os resultados da pesquisa, bem como a identificação das principais contribuições, direcionamentos futuros e as produções científicas no período desta pesquisa.

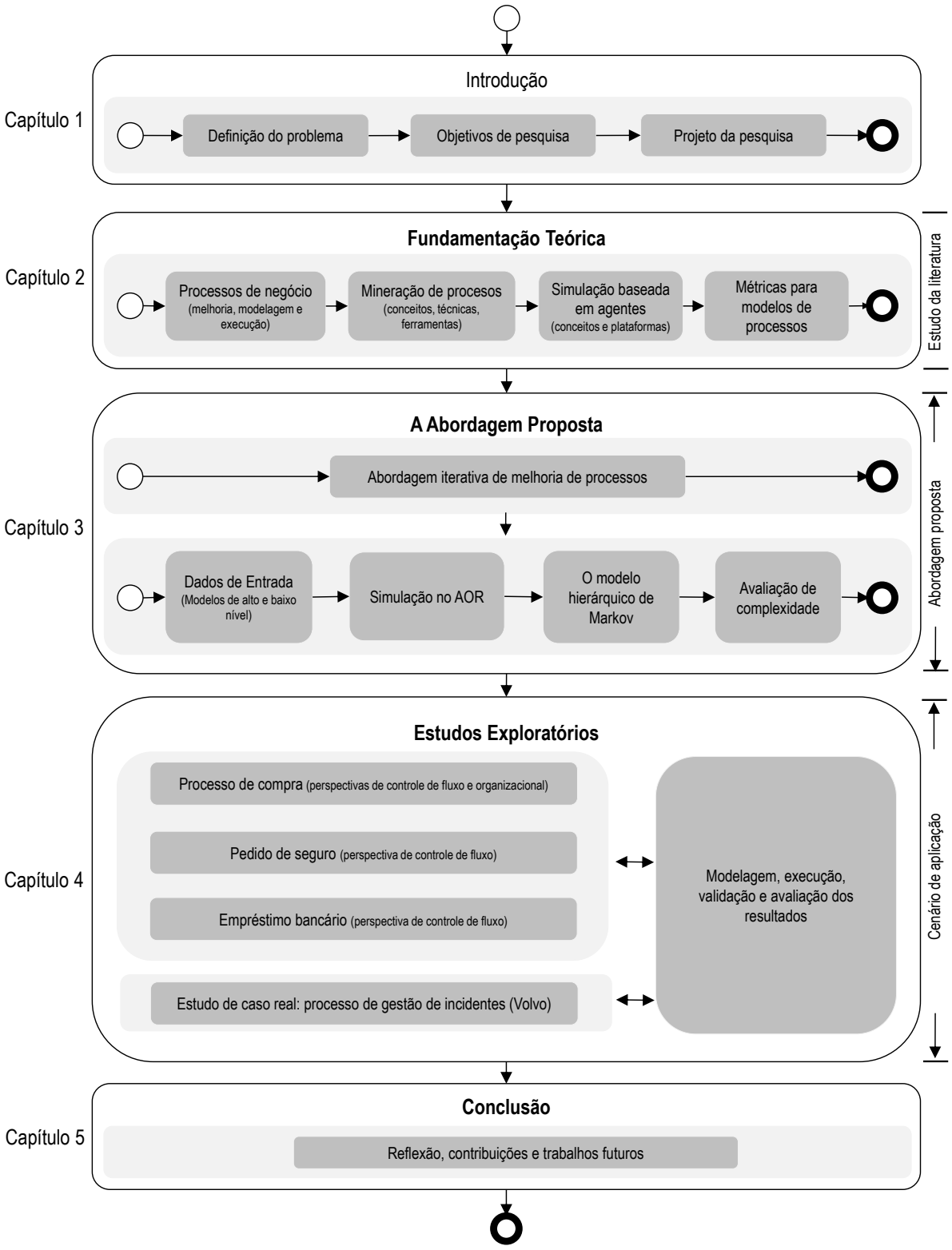


Figura 1.2: Visão geral do projeto de pesquisa

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta uma breve conceitualização dos principais temas relacionados à pesquisa, tais como: processos de negócio, simulação baseada em agentes, mineração de processos e métricas para modelos de processo. Ao final, alguns trabalhos relacionados ao problema de pesquisa também são descritos.

2.1 Processos de Negócio

Um processo de negócio é definido por Hammer and Champy (1993) como um conjunto de atividades com um ou mais tipos de entrada e cria uma saída com valor para o cliente. No entanto, Aguilar-Savén (2004) enfatiza que um processo de negócio está relacionado com a empresa, uma vez que define a maneira pela qual os objetivos da empresa sejam alcançados. Em outras palavras, um processo de negócio é a combinação de um conjunto de atividades dentro de uma organização com uma estrutura descrevendo sua ordem lógica e suas dependências, cujo objetivo é produzir um resultado desejado. Atingir a maturidade nos processos de negócio, que levam à redução de custos ou aumento de receitas, é crítico para as organizações, dada a crise financeira global (Mcafee and Brynjolfsson 2008). Na prática, as organizações acreditam que sempre há capacidade de se produzir mais resultados com menos custos, recursos, entre outros indicadores (Malhotra 2005).

Os processos de negócios são incorporados dentro das organizações na medida em que potencialmente agregam valor e, como tal, estão atraindo a atenção (exemplos dados em Hammer and Champy (1993), Davenport (1993), Martinez et al. (2001), Aguilar-Savén (2004)). Consequentemente, a modelagem e compreensão completa dos processos de negócios dentro das organizações podem levar a sistemas eficazes, eficientes e de agregação de valor. De acordo com Aguilar-Savén and Olhager (2002), os processos de negócio são elementos chave para alinhar o negócio e a TI nas organizações. Além disso, a modelagem conceitual dos processos de negócio é implantada em larga escala para facilitar o desenvolvimento de software que suporta os processos de negócios e para permitir a análise, reengenharia ou melhoria deles (Aguilar-Savén 2004).

Nos dias de hoje, as organizações necessitam de flexibilidade operacional e rápidas respostas para lidar com os desafios decorrentes de ambientes turbulentos de negócio, aumento de demanda de clientes, pressões de mercado, bem como os avanços tecnológicos. Uma abordagem muito popular de manipulação desta pressão que depende de um

entendimento completo do processo de negócio é o gerenciamento de processo de negócio (*Business Process Management* – BPM). BPM inclui métodos, técnicas e ferramentas para apoiar o projeto, melhoria, gerenciamento e análise de processos de negócio operacionais envolvendo humanos, organizações, aplicações, documentos e outras fontes de informação (van der Aalst et al. 2003b). BPM pode ser considerado como uma extensão de sistemas e abordagens clássicas de *Workflow Management* (WFM).

Nesta direção, a melhoria de processos de negócio (*Business Process Improvement* – BPI) introduzida por Harrington (1991), surge com uma abordagem focada no cliente para melhorar um processo de negócio visando alcançar as metas de redução de custos e aumentar a satisfação. Um conceito que tem sido associado com a BPI é a reengenharia de processos de negócio (*Business Process Reengineering* – BPR), que de acordo com Hammer and Champy (1993), é uma iniciativa de melhoria de processo de negócio definida como "*a reanálise fundamental e o redesenho radical dos processos de negócio a fim de alcançar melhorias drásticas em áreas críticas de desempenho, como: custos, qualidade, serviço e velocidade*". BPR se concentra em processos e não em tarefas, postos de trabalho ou pessoas. BPR tem como foco o redesenho dos processos estratégicos e de valor agregado que transcendem as fronteiras organizacionais.

Uma metodologia de BPR genérica foi proposta por Thong et al. (2003), a partir da consolidação de várias metodologias, ferramentas, técnicas e fatores críticos de sucesso publicados na literatura, com a finalidade de servir aos gerentes interessados em utilizar BPR nas organizações. Esta metodologia facilita o entendimento das atividades essenciais para BPR contemplando as seguintes fases:

1. Preparar para reengenharia – consiste em verificar a real necessidade do processo de reengenharia, i.e. a justificativa desta necessidade marca o início da atividade de preparação;

2. Mapear e analisar o processo (As-Is) – consiste em entender os processos existentes com o objetivo principal de identificar as desconexões (i.e. qualquer fato que impede o processo de obter os resultados desejados e, em particular, a transferência de informações entre as organizações ou pessoas) e processos de valor agregado;

3. Projetar o processo (To-Be) – após a identificação das melhorias em potencial para os processos existentes, esta fase se concentra em produzir uma ou mais alternativas para a situação corrente, que satisfaz os objetivos estratégicos da empresa, através dos métodos de modelagem disponíveis levando em consideração as princípios de projeto de processo;

4. Implementar a reengenharia de processos – é a fase onde os esforços de reengenharia apresentam maior resistência. O foco é elaborar o plano de transição do processo As-Is para o processo redesenhado (To-Be), onde utilizando técnicas de prototipagem e simulação, o plano de transição é validado e as versões do projeto piloto são projetadas e demonstradas. Os programas de treinamento para os trabalhadores são iniciados e o plano é executado em grande escala;

5. Melhorar continuamente o processo – uma parte vital para o sucesso de todos os esforços de reengenharia reside na melhoria continua do processo de reengenharia. Para este objetivo, o monitoramento do progressos das ações e dos resultados devem

ser realizados através de um sistema de acompanhamento de desempenho e aplicação de habilidades para resolver problemas.

2.1.1 Modelagem de processos de negócio

A modelagem de processos de negócio permite um entendimento comum entre analistas de negócio, analistas de sistemas e pessoas de negócios que irão gerenciar e monitorar estes processos. Um modelo de processo pode fornecer uma compreensão abrangente de um processo e uma empresa pode ser analisada e integrada através de seus processos de negócio. Para este fim, o termo modelagem de processos de negócio é usado para caracterizar a identificação e especificação dos processos de negócio. Esta fase inclui a modelagem das atividades e suas relações causais e temporais, bem como regras de negócio específicas que as execuções de processos têm de cumprir.

A modelagem de processos tem uma variedade de produtos que estão comercialmente disponíveis para apoiar esta fase, baseado em diferentes linguagens de processos. Dada esta situação, não é de se estranhar que a seleção de um determinado produto é um passo importante em muitos projetos de BPM, e consequentemente, os critérios adequados de seleção têm sido estudados extensivamente. Além de aspectos organizacionais, econômicos e aspectos relacionados com a infraestrutura geral de TI da empresa, o poder expressivo da linguagem de processo, bem como as interfaces para sistemas software relacionados são critérios importantes, i.e. a mais proeminente interface para sistemas de melhoria de processos e o software responsável pela modelagem e estruturas organizacionais da empresa. Não só o poder expressivo, mas também uma semântica bem definida da linguagem do processo merece um papel central durante a seleção do produto.

Entre as linguagens de modelagem de processos que tem um potencial futuro ou estão bem estabelecidas na indústria ou em pesquisas podemos destacar: Diagrama de Atividades *UML 2.0* (Korherr and List 2006); *Business Process Definition Metamodel* (BPDM) (OMG 2008); *Business Process Modelling Notation* (BPMN) *(OMG 2011)*; *Event Driven Process Chain* (EPC) (Scheer 2000); *Petri Net*; (van der Aalst 1998); *Yet Another Workflow Language* (YAWL) (van der Aalst and ter Hostede 2004). List and KorList (2006) apresentam um estudo comparativo entre estas linguagens através de diversas perspectivas, apontando que as linguagens de modelagem de processos de negócio devem fornecer formas de execução do processo, e, por sua vez, oferecer elementos de notação mais explícitos sobre todas as perspectivas. Mas um grande número de diferentes elementos são confusos para fins de descrição do processo, portanto, um conjunto básico de elementos, como é oferecido pelo BPMN também é necessário.

Atualmente, existem diversas linguagens conceituais disponíveis para modelagem de processos de negócio. Para descrever adequadamente um processo de negócio, muitas formas de informação devem ser integradas em um modelo de processo. As informações que as pessoas querem extrair a partir de modelos de processo podem ser: o que vai ser feito, quem vai fazê-lo, quando e onde é que vai ser feito, como e porque é que vai ser feito e o que é dependente de ser feito (Curtis et al. 1992). Estas linguagens se diferem na forma em que suas construções destacam a informação que respondem a estas diferentes perguntas. As diferenças resultam dos diferentes domínios de origem (e.g. processo, engenharia de software, etc.), bem como a partir das áreas de aplicação.

2.1.2 Execução e análise de processos de negócio

A execução e análise de processos de negócios tem como objetivo investigarem as propriedades dos processos de negócio que não é óbvia nem trivial. Para este fim, a análise é um termo utilizado com um significado bastante amplo, incluindo, por exemplo, simulação e diagnóstico, verificação e desempenho. A simulação de processos facilita o diagnóstico do processo no sentido de que através da simulação de casos reais, os especialistas de domínio podem reconhecer a modelagem correta ou propor alterações ao modelo de processo original. Se os modelos de processos de negócios são expressos em linguagens de processo com uma clara semântica, suas propriedades estruturais podem ser analisadas. Se, por exemplo, certas partes dos processos nunca possam ser atingidas, um erro óbvio de modelagem ocorreu que deve ser corrigido.

Enquanto as propriedades estruturais básicas de modelos de processos têm sido estudadas há algum tempo, é notável que alguns produtos de software realmente podem dar o suporte. No entanto, a análise estrutural de modelos de processos exige uma clara semântica formal da linguagem do processo, que pode não estar presente. Em alguns produtos, uma abordagem pragmática para a modelagem de processos é preferível a uma formal, especialmente se o objetivo principal da modelagem de processos é a discussão com especialistas de domínio, em vez de análise do processo ou melhoria. No entanto, mencionamos que a semântica formal de linguagens de processo, intuitividade e facilidade de uso não são objetivos contraditórios e abordagens recentes parecem apoiar esta observação.

Durante a última década, conceitos de processos e WFM estão sendo aplicados em diferentes SI empresariais (van der Aalst 2000), por exemplo como encontramos nos sistemas transacionais (e.g. *Enterprise Resource Planning* – ERP, SAP, *PeopleSoft*, *Oracle*, *Customer Relationship Management* – CRM, sistemas de WFM, entre outros). Através de definições de processos, i.e. modelos que descrevem o ciclo de vida de um caso típico (i.e. instância do processo) isolado, podemos configurar estes sistemas para apoiar a execução de processos. Desta forma, torna-se possível coletar as informações das atividades que foram executadas no processo através de um log de eventos. Estes logs podem ser utilizados para construir uma especificação do processo que são os modelos adequados do comportamento registrado. Neste caso, a mineração de processos pode ser utilizada como um método para refinar a descrição de um processo estruturado a partir deste conjunto de execuções reais, que é o comportamento registrado no log eventos, conforme descrito na Tabela 2.1.

A Tabela 2.1 contém algumas informações que podem estar presentes no log de eventos. Em diversos SI, o log de eventos pode conter diversas outras informações, conforme descrito no log da Tabela 2.3. Podemos observar que este log da Tabela 2.1 contém informações sobre cinco instâncias do processo. O log mostra que para quatro casos (1, 2, 3 e 4), as atividades A, B, C e D foram executadas. Para o quinto caso, somente três atividades foram executadas: atividades A, E e D. Cada caso inicia com a execução de A e termina com a execução de D. Se a atividade B é executada, então a atividade C também é executada. No entanto, para alguns casos, a atividade C é executada antes da atividade B. Baseada na informação apresentada na Tabela 2.1 e fazendo algumas suposições de completude (i.e. assumindo que os casos são representativos e um grande subconjunto suficiente de possíveis comportamentos é observado), podemos deduzir por exemplo o modelo de processo ilustrado na Figura 2.1.

instância	atividade
instância 1	atividade A
instância 2	atividade A
instância 3	atividade A
instância 3	atividade B
instância 1	atividade B
instância 1	atividade C
instância 2	atividade C
instância 4	atividade A
instância 2	atividade B
instância 2	atividade D
instância 5	atividade A
instância 4	atividade C
instância 1	atividade D
instância 3	atividade C
instância 3	atividade D
instância 4	atividade B
instância 5	atividade E
instância 5	atividade D
instância 4	atividade D

Tabela 2.1: Exemplo de um log de eventos resultante da execução de um processo

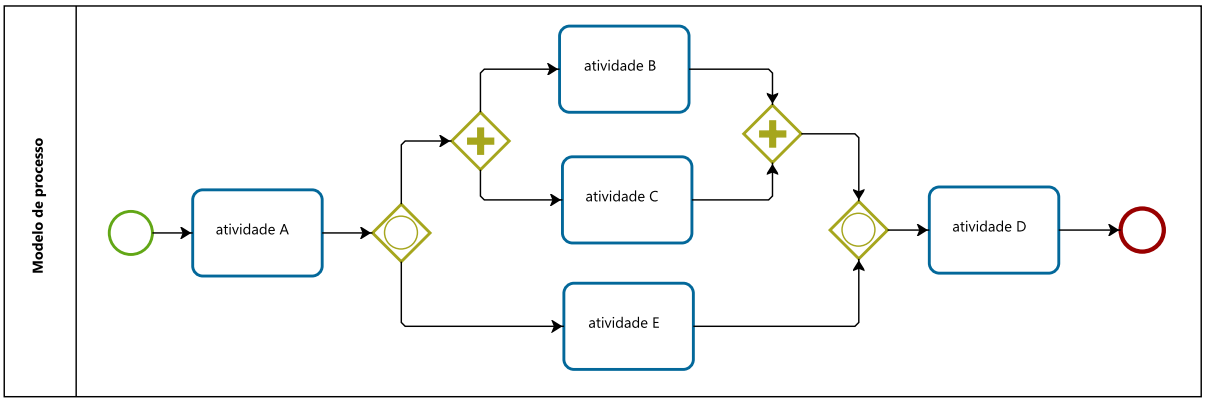


Figura 2.1: Modelo de processo correspondente ao log de eventos da Tabela 2.1

O modelo de processo ilustrado na Figura 2.1 está representado em BPMN. Ele inicia com a atividade A e termina na atividade D. Estas atividades são representadas por transações. Depois de executar A, existe uma escolha entre executar as atividades B e C em paralelo ou somente executar a atividade E. Portanto, para este exemplo simples, é fácil construir um modelo de processo que é capaz de representar o comportamento registrado no log de eventos.

2.2 Simulação baseada em agentes

Um agente é uma entidade de software que está situado em algum ambiente e que é capaz de executar ações de maneira autônoma neste ambiente com o propósito de atingir seus objetivos (Wooldridge 2009). Na literatura encontramos variações para a definição de agentes. Segundo Russell and Norvig (2003) um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Para Macal and North (2005) um agente deve ter as seguintes características:

- ser identificável – um indivíduo discreto com um conjunto de características e regras que governam seu comportamento e capacidade de tomada de decisões;
- estar situado – habitar em um ambiente com o qual interage e também no qual interage com outros agentes;
- ser orientado por objetivos;
- ser autônomo;
- ser flexível e possuir habilidade para aprender e adaptar seu comportamento através do tempo baseado em experiências.

Franklin and Graesser (1997) discutem acerca de diversas definições de agentes e enumeram alguns comportamentos apresentados por estes, identificados a partir destas definições:

- são reativos;
- são autônomos;
- são orientados por objetivo(s)/pró-ativos;
- são temporalmente contínuos;
- são sociáveis/comunicativos;
- possuem capacidade de aprender/se adaptar;
- são móveis;
- são flexíveis;
- possuem personalidade.

O conceito de agente inteligente está relacionado a racionalidade. Segundo Russell and Norvig (2003), a racionalidade é associada a quatro fatores: (1) a medida de desempenho que define critérios de sucesso; (2) o conhecimento prévio do agente sobre o ambiente; (3) as ações que o agente é capaz de executar; (4) a sequência de percepções do agente até o momento. Estes levam à definição de agente racional: Para cada sequência de percepções possível, um agente racional deve selecionar uma ação que ele espera que venha maximizar sua medida de desempenho, dada a evidência fornecida pela sequência de percepções e qualquer conhecimento prévio interno do agente.

Agentes são entidades de software que podem operar com robustez, em ambientes que se modificam rapidamente e que necessitam de respostas precisas e rápidas a eventos que podem ser inesperados. Eles conseguem reagir rapidamente e possuem características que os permitem atuar em situações não programadas, além de possuir capacidade para interagir com outros, sejam humanos ou agentes. O comportamento flexível e racional é alcançado através de processos-chave, resolução de problemas, planejamento, tomada de decisão e aprendizado (Weiss 1999).

A simulação baseada em agentes (*Agent-based Simulation* – ABS) (Bonabeau 2002, Axelrod 2006, Davidsson et al. 2007) é um meio eficaz para estudar o comportamento de sistemas que envolvem as ações e interações de agentes autônomos. A ABS tem sido aplicada em problemas de simulação para diversas áreas: Ciências Biológicas; Engenharias; Ciências Econômicas; e Ciências Sociais. Por exemplo, os sistemas baseados em agentes têm sido usados para estudar a dinâmica de mercados financeiros através da geração de dados de séries temporais que se assemelham a evolução dos preços das ações (Hoffmann et al. 2006, Neri 2012).

Em ABS um cenário de agentes que interagem uns com os outros e com seu ambiente, está sendo modelado e simulado, como um Sistema Multiagente (SMA). Um SMA é composto de múltiplos agentes inteligentes interagindo e podem ser utilizados para resolver problemas que são difíceis ou impossíveis de um agente individual ou um sistema monolítico resolver. Seus agentes participantes, sejam estes seres humanos, instituições sociais, sistemas de software ou máquinas, podem executar ações, perceber seu ambiente e reagir a mudanças nele. Os agentes também têm um estado mental que incluem componentes como: conhecimento/opiniões; objetivos; lembranças; e compromissos. Comparado com os métodos tradicionais de simulação, e.g, equações matemáticas, simulação de eventos discretos, autômatos celulares e teoria de jogos, a ABS é menos abstrata e mais próxima da realidade (Wagner 2003).

Ambientes de agentes podem ser organizados de acordo com várias propriedades (Russell and Norvig 2003, Salamon 2011):

- Acessibilidade – o ambiente é totalmente acessível se todas as informações dos estados podem ser percebidas pelos agentes, caso contrário o ambiente é parcialmente acessível;
- Determinismo – se cada ação no ambiente provoca a ocorrência garantida de um único efeito;
- Dinamicidade – o ambiente é estático se os estados do ambiente não se alteram enquanto o agente delibera suas ações, de forma contrária é dinâmico;
- Continuidade – o ambiente é discreto caso possua um número finito de percepções e ações, caso contrário ele é contínuo;
- Episodicidade – se as ações do agente em determinados períodos de tempo influenciam outros períodos;
- Dimensionalidade – se as características espaciais são apresentadas em múltiplas dimensões, i.e., temporais, espaciais.

Conforme descrito por Wooldridge (2009), em um sistema multiagente os agentes possuem características importantes, como:

- Autonomia – os agentes são pelo menos parcialmente autônomos;
- Percepção limitada – nenhum agente tem uma visão completa do sistema ou o sistema é muito complexo para um agente fazer uso prático de tal conhecimento;
- Descentralização – não há nenhum agente designado controlador (ou o sistema é efetivamente reduzido a um sistema monolítico).

2.2.1 Plataformas de simulação baseada em agentes

Para estudar o problema apresentado na Seção 1.3 e para desenvolver a solução proposta, é muito útil ter uma forma de simular a geração de eventos de baixo nível a partir de atividades de alto nível, i.e. ao desenvolver um algoritmo para fazer mineração de processos, é preciso ter alguns logs disponíveis para testá-lo. Utilizar logs reais parece ser a escolha natural. No entanto, os logs de eventos reais geralmente contém imperfeições que podem dificultar o ajuste do algoritmo de mineração. Por exemplo, os logs reais podem ser incompletos e/ou conter ruídos ou variações de comportamento. Assim, uma abordagem mais comum é primeiro testar a precisão dos novos algoritmos de mineração de processo em registros criados através de simulação. Isso permite o pesquisador a ter mais controle sobre as propriedades do log de eventos e assim melhor ajustar o seu algoritmo de mineração. Além disso, o modelo original (o modelo simulado) também poderá ser útil para auxiliar na avaliação da qualidade do algoritmo de mineração.

Ferramentas tradicionais de simulação de processos não são totalmente apropriadas para este propósito, uma vez que é necessário simular não somente processos de alto nível, mas também a forma em que um conjunto de participantes executam o processo, gerando assim uma sequência não-determinística de eventos de baixo nível com o escopo de cada atividade. Plataformas baseadas em agentes são especialmente convenientes para este propósito, uma vez que agentes podem ter características interessantes, como autonomia e pró-atividade (Wooldridge and Jennings 1995), que podem ser utilizadas para imitar a forma como humanos interagem enquanto executam suas tarefas.

A Tabela 2.2 apresenta um estudo comparativo entre algumas plataformas de modelagem e simulação baseada em agentes.

Justificamos a escolha do AOR no contexto deste trabalho por se tratar de uma abordagem que utiliza uma conceitualização epistemológica de representação de todos os modelos possíveis de simulação em alto nível através do uso de ontologias. A princípio, com uma ontologia de alto nível é possível representar processos de qualquer área, como e.g. biologia, economia, jogos, física, ciência sociais, aprendizado por reforço, entre outras. Portanto, a escolha do AOR se deve a toda capacidade de simulação encontrada na plataforma.

O AOR emprega o conceito de regras de reação para modelagem do comportamento para SI organizacionais. Conforme a Tabela 2.2, observa-se que o AOR fornece uma plataforma de simulação baseada em agentes em um meta-modelo ontológico de alto nível e comparado as outras plataformas de simulação, oferece quase uma lista completa de

[9] http://repast.sourceforge.net/
[10] http://ccl.northwestern.edu/netlogo/
[11] http://www.agentisolutions.com/
[12] https://oxygen.informatik.tu-cottbus.de/aor/?q=node/2

Características	RePast[1]	NetLogo[2]	Brahms[3]	AOR[4]
Distinguir entre objetos e agentes		✓	✓	✓
Fornecer uma variedade de modelos espaciais	✓			✓
Fornecer uma ontologia fundamental de eventos				✓
Distinguir entre objetos / agentes físicos e não físicos				✓
Suportar um conceito de atividades			✓	✓
Modelar o comportamento baseado em regras			✓	✓
Fornecer um modelo de percepção			✓	✓
Distinguir entre fatos e crenças			✓	✓
Suportar crenças sobre outras entidades			✓	✓

Tabela 2.2: Plataformas de simulação baseada em agentes. Adaptado de (Wagner and Diaconescu 2009)

funcionalidades específicas para sistemas de simulação com agentes: é independente de uma linguagem específica de programação; é orientado ao modelo; utiliza modelagem de comportamento (declarativa) baseada em regras; suporta um conceito de espaço; aceita a distinção entre objetos e agentes, incluindo um modelo cognitivo de percepções; e apóia a distinção entre fatos e crenças. Além disso, nenhuma das plataformas apresentadas na Tabela 2.2 inclui um tratamento sistemático de meta-conceitos orientados a agentes, tais como ações, eventos, compromissos e reivindicações.

2.2.2 Simulação de processos com o AOR

Neste trabalho, utilizamos uma plataforma baseada em agentes para simular a execução de processos de negócio para gerar um log de eventos com eventos de baixo nível. Embora existam diversas plataformas de simulação baseada em agentes (Railsback et al. 2006), voltamos nossa atenção para a plataforma AOR introduzida por (Wagner 2004, Wagner et al. 2009).

Na plataforma AOR de simulação os agentes respondem a eventos no seu ambiente, executando ações e interagindo uns com os outros, o que por sua vez gera novos eventos. A criação de um cenário de simulação no AOR consiste em um modelo de simulação, uma definição de estado inicial e zero ou mais definições de visão. Este modelo de simulação inclui diversos componentes, como:

1. Modelo de espaço – uma simulação pode usar vários modelos de espaço para agentes físicos caracterizados pela dimensão, discreto/contínuo, geometria (euclideana ou toroidal) e limites de espaço;

2. Tipos de entidade – inclui diferentes categorias ontológicas de eventos, mensagens, objetos e tipos de agentes, conforme a Figura 2.2;

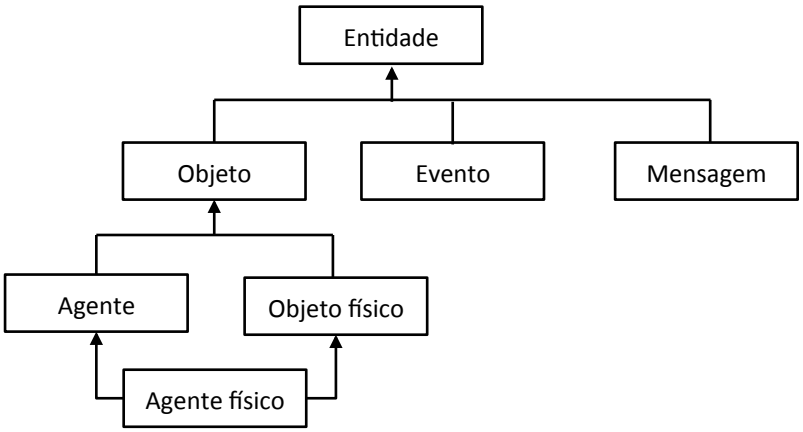


Figura 2.2: Categorias ontológicas de alto nível do AOR. Adaptado de (Wagner 2004).

Um cenário de simulação modelado somente com objetos, i.e. sem o uso de agentes, restringe o protocolo de interação do simulador, prejudicando a troca de mensagens inerente aos agentes.

3. Regras de ambiente – define as regras de causalidade que governam o ambiente e suas mudanças. Tanto o comportamento do ambiente (as suas leis de causalidade) e o comportamento dos agentes são modelados com a ajuda de regras, suportando, assim, modelagem do comportamento declarativo de alto nível. O ambiente de regras é um conjunto de variáveis do tipo *5-tuple* (EvtT, Var, Cond, UpdExpr, ResEvtExpr), onde: *EvtT* denota o tipo de evento que dispara a regra; *Var* expressa um conjunto de declarações de variáveis; *Cond* expressa uma fórmula de condição lógica para variáveis; *UpdExpr* especifica uma atualização do estado do ambiente; e *ResEvtExpr* expressa uma lista de resultado de eventos;

4. Opinião e respostas as consultas – suporte para a distinção entre crenças e opiniões, incluindo a auto-opinião (i.e. opinião do agente sobre si mesmo). Utiliza a linguagem de consulta W3C RDF[5] SPARQL (Wagner and Diaconescu 2009) para expressar consultas que um agente pode solicitar a outro agente sobre suas opiniões.

Um tipo de agente é definido por meio de: um conjunto (objetivo) de propriedades; um conjunto (subjetivo) de propriedades para a auto-opinião; um conjunto (subjetivo) de opiniões de tipos entidades; um conjunto de regras do agentes, que define o comportamento do agente em respostas aos eventos.

Todas estas construções (i.e. tipos de entidades, regras dos agentes, regras de ambiente e as condições iniciais) são especificadas utilizando uma linguagem baseada em *eXtensible Markup Language* (XML) conhecida como AOR *Simulation Language* (AORSL) (Nicolae et al. 2010). Esta linguagem permite também incorporar código Java a fim de implementar

[5]http://www.w3.org/RDF/

as funções e expressões auxiliares. De fato, a especificação do cenário no AORSL é transformada em código Java ou Java script pelo sistema AOR. Executar a simulação equivale a compilar e executar o código Java gerado automaticamente. O AOR apresenta simuladores na forma de um sistema de gerenciamento de simulações Java *standalone* – AOR-JavaSim[6] e na forma de um simulador na Web – Simurena *framework*[7]. Ambas as traduções dos cenários expressadas em AORSL são ilustrados na Figura 2.3.

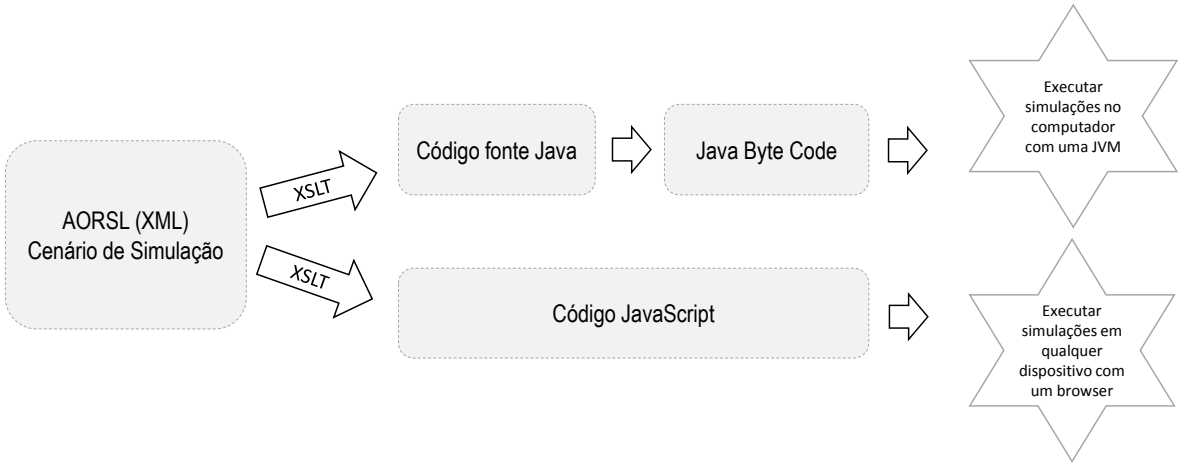


Figura 2.3: Geração de código pelo AOR. Adaptado de Wagner (2004)

Levando em consideração que linguagens de modelagem conceitual, como e.g. UML entre outras citadas na Seção 2.1.1, não suportam conceitos de agentes, pois não conseguem retratar de forma coerente a troca de mensagens entre os agentes e os estados de um agente, Wagner (2002) criou uma extensão de UML chamada *AOR Modeling language* (AORML). Através de AORML consegue-se representar os comportamentos dos agentes, ações, as trocas de mensagens realizadas por cada agente e os objetos. Conforme Wagner (2002), a AORML está organizada em um modelo de análise e um modelo de projeto:

1. O *modelo Externo*, conhecido como modelo de análise, modela a visão externa, de quem esta observando os agentes e suas interações, considerando o domínio do problema de uma forma mais abrangente e focando sempre no ambiente;

2. O *modelo Interno*, como é conhecido como o modelo de projeto, modela o problema na visão de cada agente, como é realizada a comunicação com os outros agentes, a notificação dos eventos ao ambiente por outros agentes e como ele enxerga o ambiente.

Um modelo de simulação no AOR pode ser implementado em diferentes linguagens de programação, mas mantendo a conformidade com a arquitetura e modelo de execução, conforme foi definido em Wagner (2003).

Em resumo, a plataforma AOR nos permitiu implementar plenamente diversos modelos de processo na forma de cenários de simulação baseado em agentes. Neste trabalho

[6] http://code.google.com/p/aor-javasim/
[7] http://www.simurena.de

foram implementados os cenários: processo de compra; processo de indenização de seguro; e processo de empréstimo bancário. Para cada modelo foi especificado um conjunto de agentes e seu comportamento dentro no escopo de cada atividade de alto nível. O AOR por si só tem a habilidade de registrar todos os eventos que são gerados durante uma simulação em um arquivo no formato XML e com *scripts* é possível transformar este arquivo em um log de eventos que é similar ao tipo de log eventos utilizado para mineração de processos. Tendo como entrada o log de eventos e um modelo de alto nível para o processo, a abordagem proposta de mineração descobre o comportamento de baixo nível associado a cada atividade de alto nível.

2.3 Mineração de processos

A mineração de processos (van der Aalst 2011) é uma nova e emergente área de pesquisa, onde o foco principal é a extração de informação sobre o comportamento em tempo de execução de processos de negócio a partir de log de eventos registrados por SI disponíveis em uma organização. Tais informações, podem ser utilizadas para auxiliar no desenvolvimento e implantação de novos sistemas que suportam a execução de processos de negócio ou também no apoio para atividades de análise, auditoria e melhoria de processos de negócio existentes. Em outras palavras, as técnicas de mineração de processos são úteis pois podem ser capazes de traduzir o que realmente está ocorrendo na execução de um processo em uma organização a partir de log de eventos e não o que as pessoas pensam que pode estar ocorrendo nesta organização.

Os primeiros trabalhos sobre a mineração de processo surgiram na década de 90 e consistiram em minerar modelos de processos em logs de eventos no contexto da engenharia de software (Cook and Wolf 1995, Cook 1996, Cook and Wolf 1998a;b). A mineração de processos direcionada a negócios foi introduzida primeiramente em 1998, denominada *workflow mining* (Agrawal et al. 1998). Desde então, diversos grupos de pesquisas concentraram seus esforços na mineração de modelos de processo (van der Aalst et al. 2004, van Dongen et al. 2005, Günther and van der Aalst 2006, Weijters et al. 2006, Medeiros et al. 2007, Ferreira and Gillblad 2009). Tiwari et al. (2008) apresentaram um estudo detalhado sobre a mineração de processos.

A idéia geral da mineração de processos está ilustrada na Figura 2.4. A partir desta figura também podemos ver que as técnicas de mineração de processo podem ser utilizadas para três diferentes propósitos (van der Aalst 2011; 2012): *Discovery*, *Conformance* e *Extension/Enhancement*.

(1) *Discovery* – consiste em descobrir o modelo baseado no comportamento registrado no log de eventos. Os eventos podem ter diversos tipos de atributos (como *timestamp*, informações transacionais e recursos utilizados) que podem ser utilizados para a descoberta do processo. No entanto, para simplificar, muitas vezes representamos os eventos somente pelo tipo da atividade. Desta forma, uma instância do processo pode ser representada por um *trace* descrevendo uma sequência de atividades;

(2) *Conformance* – a descoberta de processos é apenas o ponto de partida para uma análise mais profunda. Para o propósito de verificação da conformidade, o comportamento modelado e o comportamento observado ou o log de eventos é comparado, a fim

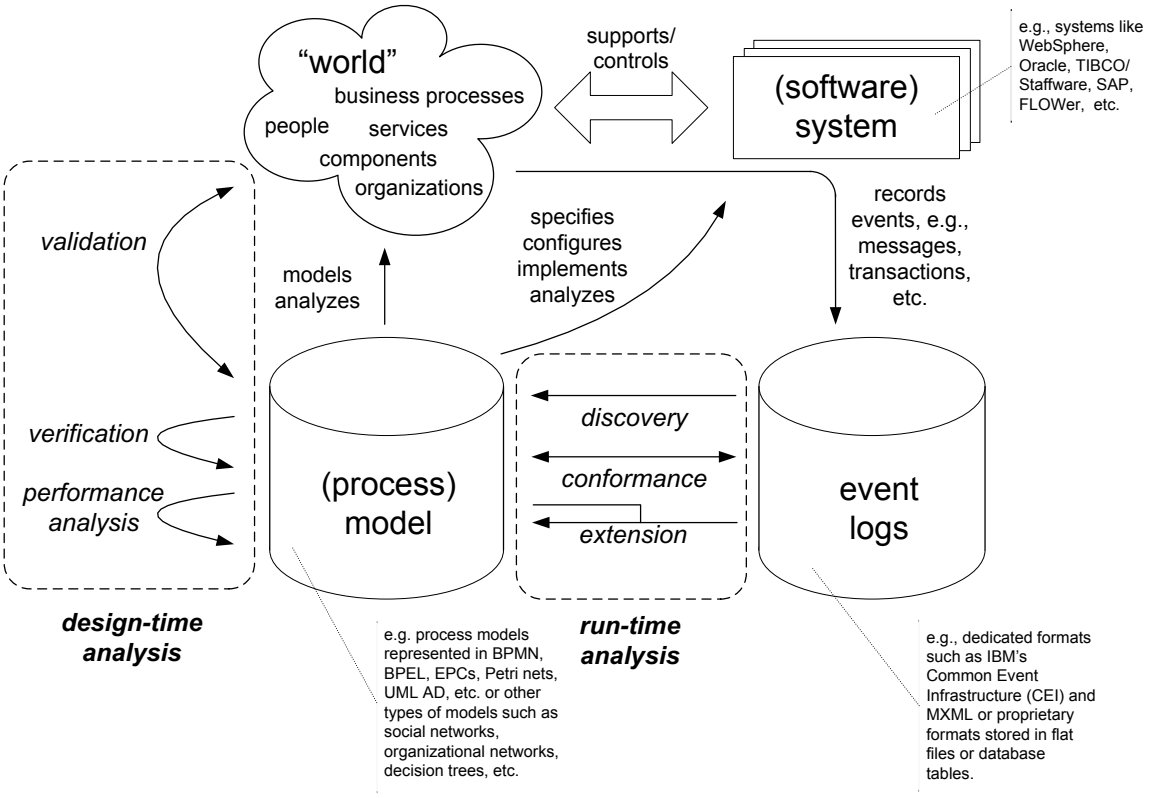


Figura 2.4: Visão geral da mineração de processos (van der Aalst 2011)

de determinar até que ponto o processo está sendo realizado como descrito no modelo existente. Existem métricas especiais para a verificação de conformidade (Rozinat and van der Aalst 2008) e também há maneiras de reproduzir o registro de eventos no modelo de processo a fim de verificar a conformidade (van der Aalst et al. 2012);

(3) *Extension/Enhancement* – que consiste em melhorar ou estender um modelo de processo existente com base no comportamento do tempo de execução registrado em um log de eventos. Por exemplo, quando um processo tem uma decisão entre dois caminhos, em algum ponto, será possível determinar a porcentagem exata de casos que se seguem a cada caminho, ou mesmo para descobrir outros caminhos que não estavam no modelo original. Existem relativamente menos trabalhos relacionados na melhoria do modelo do que na descoberta de processo ou verificação da conformidade.

As técnicas relacionadas a *Discovery* utilizam como entrada um log de eventos para produzir um modelo, que tipicamente é representado por um modelo de processo (como e.g. BPMN, EPC, UML entre outros). As técnicas relacionadas a *Conformance* utilizam como entrada um modelo além do log de eventos e sua saída consiste na informação de diagnóstico indicando as semelhanças ou diferenças entre o modelo e registros que estão no log de eventos. As técnicas relacionadas a *Extension/Enhancement* também requerem como entrada um modelo e o log de eventos e a saída é um modelo estendido.

Embora em todos os propósitos de mineração de processos temos entradas e saídas, em todos os tipos temos como entrada comum um log de eventos, que é normalmente uma lista de eventos em ordem cronológica, onde cada evento se refere a uma atividade que foi executada por algum agente durante a execução de uma instância do processo.

Para facilitar o entendimento do papel da mineração de processos no âmbito deste trabalho, ilustramos um exemplo na Tabela 2.3, onde cada evento no log de eventos contém os seguintes elementos: instância; atividades; agente; e data e hora. Através da aplicação de técnicas de mineração de processos é possível extrair diferentes tipos de modelos comportamentais a partir do log de eventos. Um exemplo é o modelo de controle de fluxo representado na Figura 2.5, que foi obtido através da análise da sequência de atividades registrada na coluna atividade.

instância	*atividade*	*agente*	*data e hora*
1	Requisicao	Funcionario	2012-06-14 10:23
1	Despachar Produto	Deposito	2012-06-14 15:45
2	Requisicao	Funcionario	2012-06-15 11:21
2	Aprovar Compra	Compras	2012-06-19 14:34
3	Requisicao	Funcionario	2012-06-21 09:31
3	Aprovar Compra	Compras	2012-06-23 10:22
3	Comprar Produto	Compras	2012-06-23 11:53
3	Receber Produto	Deposito	2012-06-27 12:07
3	Despachar Produto	Deposito	2012-06-28 08:25
...	...	...	...

Tabela 2.3: Log de eventos

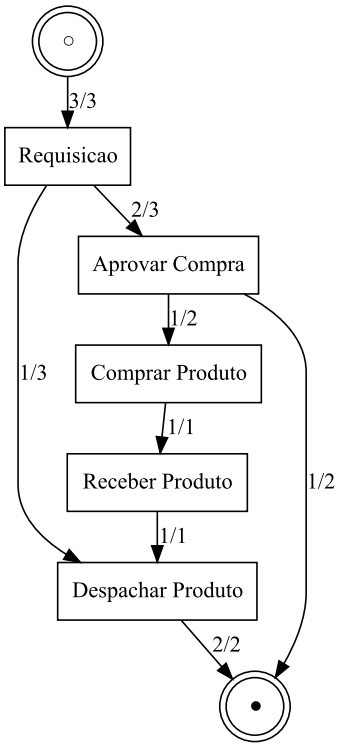


Figura 2.5: Modelo de controle de fluxo

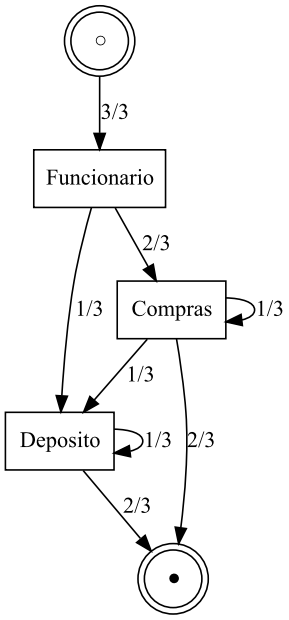


Figura 2.6: Modelo organizacional

Cada estado no modelo de controle de fluxo da Figura 2.5 representa uma atividade diferente e as setas entre os estados representam as transições que devem ser observadas no log de eventos (onde cada transição está entre dois eventos consecutivos com a mesma *instância*). Algumas transições ocorreram somente uma vez, enquanto outras ocorreram múltiplas vezes, em diferentes instâncias do processo. Temos o estado onde o arco inicia, a ser chamado o estado de origem e o estado onde a arco termina, a ser chamado o estado de destino. O rótulo próximo a cada seta indica o número de vezes que a transição ocorreu dividido pelo número total de vezes que o estado de origem foi registrado no log de eventos. Em uma interpretação frequentista, isto pode ser tomado como uma medida de probabilidade indo a partir do estado de origem para o estado de destino e por isso este modelo pode ser considerado como uma forma de cadeia de Markov.

O estados especiais com "$\circ$" (para "aberto") e "$\bullet$" (para "fechado") são utilizados para designar o início e o final da instância do processo, respectivamente. Estes estados não são explicitamente registrados no log de eventos, mas eles podem ser descobertos a partir do ponto onde uma *instância* aparece pela primeira vez e pela última vez, respectivamente. Na prática, os eventos de diferentes instâncias do processo podem ser intercalados no tempo e portanto aparecem intercalados no log de eventos, mas é possível reagrupá-los de acordo com a *instância*.

A Figura 2.6 apresenta o resultado de uma análise similar realizada para coluna *agente*. Aqui, cada estado representa um agente diferente e as transições representam a entrega de trabalho entre agentes, i.e., uma transição representa o fato que um agente executa uma atividade antes de entregar a instância do processo a outro agente. Neste exemplo, existem casos onde o mesmo agente executa duas atividades consecutivas e esta é a razão porque existem auto-loops na Figura 2.6. Em qualquer caso, o método que foi utilizado para construir este modelo foi o mesmo como na Figura 2.5 e novamente os rótulos de transição podem ser tomados como uma medida da probabilidade de ir a partir do estado de origem para um estado de destino.

Assim, a idéia básica da mineração de processos é extrair conhecimento a partir de logs de eventos registrados por um SI. O objetivo é entender e melhorar os processos reais, com base em informação que foi de fato coletada durante a execução destes processos. Por outro lado, a mineração de processos inclui diferentes perspectivas: controle de fluxo, organizacional e performance. Em cada destas perspectivas existem um grande número de técnicas especializadas. A seguir vamos apresentar diversas destas técnicas segundo cada perspectiva.

2.3.1 Perspectiva de controle de fluxo

A perspectiva de controle de fluxo ou processo tem o foco na ordem das atividades, i.e, minerando informações de ordenação em um log de eventos, um algoritmo de mineração pode redescobrir o modelo de processo que reflete o comportamento neste log. O objetivo da perspectiva de controle de fluxo é encontrar uma boa caracterização de todos os caminhos possíveis, e.g. expressados em uma linguagem de modelagem de processos conforme descrito na Seção 2.1.1.

Dentre as diversas abordagens e técnicas de mineração baseadas na perspectiva de controle de fluxo podemos destacar:

- *α-algorithm* (van der Aalst et al. 2004): este algoritmo recebe como entrada um log de eventos que contém as sequências de execução (*traces*) de um modelo de processo. Com base neste registro, pode-se inferir as relações entre as tarefas. Estas relações de ordenação indicam, por exemplo se um uma tarefa é causa para outras tarefas, se duas tarefas são executadas em paralelo e assim por diante. O *α-algorithm* usa essas relações de ordenação para redescobrir um modelo de processo descrito no comportamento do log. O modelo de processo minerado, ou descoberto, é expresso como *Workflow Net* (WF-Net) (van der Aalst 1998). *Workflow Nets* formam um tipo especial de redes de petri.

  A mineração de controle de fluxo explana como conceito básico que: minerando informações de ordenação em um log de eventos, um algoritmo de mineração pode

redescobrir o modelo de processo que reflete o comportamento neste log. O *α-algorithm* é um algoritmo de mineração muito simples, mas também muito poderoso, pois consegue minerar corretamente *SWF nets* sem *short loops* originado de logs sem ruídos (i.e variação de comportamento). Ele se baseia em quatro relações de ordenação, sendo que estas relações também são utilizadas pela heurística da abordagem genética apresentada por (Medeiros 2006).

Entre as limitações do *α-algorithm* podemos citar a sua incapacidade de capturar *short loops*, tarefas invisíveis (i.e não aparecem em nenhum *trace* de eventos), tarefas duplicadas, construções *non-local non-free-choice* e ruídos. Tarefas invisíveis não são suportadas por *Structured Workflow Nets* (SWF-nets). *Short loops* são confundidos com construções paralelas quando são inferidas as relações de ordenação. Construções *non-local non-free-choice* também não são capturadas pela relações de ordenação do log. Existem diversas extensões do *α-algorithm*, mas essas extensões ainda tiveram problemas para lidar com as tarefas invisíveis e tarefas duplicadas. Além disso, elas também são muito sensíveis a ruídos.

- *HeuristicsMiner* (Weijters et al. 2006): esta abordagem de mineração considera somente a ordem dos eventos dentro de um caso. Em outras palavras, a ordem de eventos entre os casos não é importante. Por exemplo, para o registro em em log de eventos apenas os campos *instância*, *data e hora* e *atividade* são considerados durante a mineração. A *data e hora* de uma atividade é usada para calcular esta ordenação. Este algoritmo pode lidar com os ruídos e o baixo comportamento (i.e. com baixa frequência de comportamento registrado no log de eventos). Em situação prática (i.e. com o log de eventos com milhares de *traces*, comportamento frequente baixo e algum ruído) o algoritmo *HeuristicsMiner* pode focar em todo o comportamento no log de eventos ou apenas no comportamento principal.

- *Métodos de inferência* (Cook and Wolf 1995): Métodos de inferência utilizam um conjunto de três diferentes algoritmos para inferir uma máquina de estado finita (*Finite State Machine* – FSM) de um log de eventos, o *RNet*, o *Ktail* e o *Markov*, onde o log é considerado como uma simples sequência de símbolos. Os três algoritmos representam diferentes níveis de compromisso entre precisão e robustez ao ruído. O algoritmo de Markov, inspirado nos modelos Markov, demonstra ser a técnica mais promissora. O algoritmo funciona através da construção de um grafo de eventos como resultado considerando cadeias de Markov com ordem crescente. Na última etapa, o grafo é convertido para uma FSM que representa o processo que foi encontrado.

- *Grafos direcionados acíclicos* (Agrawal et al. 1998): É um algoritmo capaz de gerar um grafo de dependência a partir de um log de sistema de *workflow*. O log deve conter um número relativamente elevado de execuções de um mesmo processo, para que o grafo de dependência para que esse processo possa ser completamente construído. Originalmente, o algoritmo foi proposto para apoiar a adoção de sistemas de *workflow* em vez de realmente buscar mineração processo.

- *Aquisição de* workflow *indutivo* (Herbst and Karagiannis 1998): Esta abordagem tem o objetivo de encontrar um modelo oculto de Markov (*Hidden Markov Model* - HMM) que melhor representa a estrutura do processo original. O HMM pode ser

encontrado por qualquer refinamento *top-down* ou *bottom-up* de uma estrutura de HMM inicial. Estes são conhecidos como algoritmos de modelo *splitting* e *merging*, respectivamente. A estrutura HMM inicial é construída diretamente a partir do registro, que é considerado como uma simples sequência de símbolos. Os resultados apresentados sugerem que o modelo *splitting* é mais rápido e mais preciso do que o modelo *merging*.

- *Grafos de instâncias* (van Dongen and van der Aalst 2004): É uma abordagem que visa retratar representações gráficas de execução do processo, especialmente usando EPC's. Para cada registro (*trace*) de execução encontrado no log, um grafo de exemplo é obtido para essa instância do processo. A fim de identificar o paralelismo possível, cada grafo de instância é construído utilizando as dependências encontradas em todo o log. Diversos Grafos de instâncias podem então ser agregados para se obter o modelo geral para esse log.

- *Clustering Hierárquico* (Greco et al. 2005): Dado um grande conjunto de registros de execução de um único processo, este algoritmo separa tais registros em *clusters* e encontra o grafo de dependência separadamente para cada *cluster*. Os *clusters* de registros de *workflow* são organizados em uma árvore, que origina o conceito de modelo hierárquico. Depois que os modelos de *workflow* para os diferentes *clusters* forem encontrados, uma abordagem *bottom-up* através de uma árvore hierárquica os generaliza em um único *cluster*.

- *Genetic Process Mining* (Medeiros 2006): é um algoritmo em que as várias soluções candidatas são avaliadas por uma função *fitness* que determina como cada solução é consistente com o log. Cada solução é representada por uma matriz causal, i.e. um mapa da entrada e saída de dependências para cada atividade. Soluções candidatas são geradas pelo cruzamento de seleção e mutação como nos típicos algoritmos genéticos. O espaço de busca é o conjunto de todas as possíveis soluções com diferentes combinações das atividades que aparecem no log de eventos. O log deve conter um número relativamente elevado de registros (*traces*) de execução.

Apesar destas abordagens não se proporem resolver o problema de relacionar o comportamento dos processos de negócio no macro e micro níveis, os algoritmos apresentados para mineração de processo sob a perspectiva de controle de fluxo apresentam diversas soluções em vários contextos.

Os resultados de mineração processos normalmente são afetados pela heterogeneidade encontrada nos logs de eventos reais, e.g. os modelos de controle de fluxo descobertos podem ser como *modelos spaghetti* (van der Aalst and Günther 2007). Desta forma, atividades de pré-processamento que trata a filtragem/divisão do log de eventos, tornam-se necessárias na criação de subconjuntos mais homogêneos dos eventos registrados no log de eventos, i.e. o pré-processamento é uma atividade crucial dada a variação e variabilidade encontrada em log de eventos reais, conforme descrito por Bose and van der Aalst (2012).

2.3.2 Perspectiva organizacional

A perspectiva organizacional tem o foco nos atores que originaram o evento (*originators)*, i.e. quais atores estão envolvidos na realização das atividades e como eles estão

relacionados (van der Aalst and Song 2004, van der Aalst et al. 2005, Song and van der Aalst 2008, Reungrungsee et al. 2012). O objetivo é tanto estruturar a organização, classificando os atores (e.g. pessoas, sistemas, funções ou unidades organizacionais) como para mostrar as relações entre os atores individuais, i.e descrever as redes de interação social. Por exemplo, em um hospital e muitas outras organizações em diversas áreas de conhecimento, a definição dos processos está relacionados ao processo decisório dos atores humanos.

A descoberta de conhecimento organizacional, tais como as estruturas organizacionais e as redes sociais, permite aos gerentes compreender as estruturas organizacionais e melhorar os processos de negócios. Portanto, a mineração organizacional auxilia na compreensão e melhoria das estruturas organizacionais e sociais. Por exemplo, as redes sociais mostram as estruturas de comunicação nas empresas. Isso pode ser usado para criar infra-estruturas de comunicação ou *layouts* de escritório.

Song and van der Aalst (2008) desenvolveram métodos de mineração aplicados a três diferentes tipos de mineração organizacional:

- Mineração de modelos organizacionais – visa derivar o modelo organizacional a partir de logs de processos;
- Análise de redes sociais – visa monitorar como casos individuais são roteados entre os autores, como por exemplo a métrica *handover of work*;
- Fluxo de informação entre entidades organizacionais – além de gerar redes sociais, onde os nós são autores, também possibilita construir redes sociais, onde os nós correspondem a entidades organizacionais (i.e. grupos de autores). As redes sociais baseadas em entidades organizacionais, tais como unidades organizacionais ou funções, fornecem informações adicionais em um nível de agregação superior.

Na mineração de modelos organizacionais, Song and van der Aalst (2008) elaboraram quatro métodos de mineração para dois tipos de entidades organizacionais que podem ser extraídas de logs de processos: (1) grupo baseado em tarefas (*task-based team*) – que pode ser relevante para a departamentalização funcional onde os funcionários possuem habilidades e conhecimentos semelhantes para executar as tarefas; (2) grupo baseado em casos (*case-based team*) – que está relacionado com a equipe do projeto onde os funcionários geralmente possuem diferentes habilidades e trabalham juntos no mesmo caso, que é útil para compreender a organização.

No grupo baseado em tarefas foram implementados três métodos de mineração: (1) *default mining* – que é um método simples que mostra claramente o relacionamento entre as tarefas e os atores; (2) *metrics based on joint activities* – que é uma técnica assume que os atores que fazem tarefas semelhantes estão mais intimamente ligados do que os atores que realizam tarefas completamente diferentes; e (3) *hierarchical organizational mining* – que ao contrário dos dois métodos citados anteriormente, pode derivar um modelo em que os modelos organizacionais são hierarquizados através de técnicas de *clustering*.

Para o grupo baseado em casos foi implementado o quarto método de mineração chamado *metrics based on joint cases* – que ao contrário dos métodos anteriores, deriva as estruturas dos grupos baseado nos casos. As métricas contam com qual frequência dois autores realizam atividades no mesmo caso em uma organização. Após a aplicação dos métodos *metrics based on joint activities*, *hierarchical organizational mining* e *metrics*

based on joint cases, são obtidos *clusters* que correspondem a póssiveis entidades organizacionais.

A análise de redes sociais possibilita o entendimento das estruturas sociais na organização pelos gerentes, monitorar como pessoas, grupos ou componentes de software estão trabalhando juntos, foi elaborado um método de mineração que pode utilizar as métricas *handover of work*, *subcontracting*, *working together* e *Pearson's correlation coefficient* para gerar as redes sociais. Com a estrutura de comunicação gerada é possível aplicar diversas técnicas de análise de redes sociais, como: densidade, equivalência, centralidade, coesão, entre outras.

No fluxo de informação entre entidades organizacionais, que é utilizado para investigar o fluxo de informações entre locais físicos, foi definido um método para a rede social das entidades organizacionais, através do cálculo da rede a partir do resultado da mineração de redes sociais.

2.3.3 Perspectiva de performance

A perspectiva de performance ou desempenho tem o foco nas propriedades dos casos do processo (i.e. *case*). Os casos podem ser caracterizados por seu caminho no processo ou pelos executores que estão trabalhando em um caso. No entanto, os casos também podem ser caracterizados pelos valores dos elementos de dados correspondentes. Por exemplo, no log de um reparo do telefone, pode ser interessante saber as diferenças de tempo de *throughput* entre os diferentes tipos de telefone. Sendo que os eventos que são anotados com referências temporais, é possível a descoberta de pontos de estrangulamento, medição dos níveis de serviço, monitorização da utilização de recursos e previsão do tempo de processamento necessário para conclusão dos casos existentes.

Aplicada a esta perspectiva podemos citar a abordagem *Case data extraction* que está implementada no ProM (van Dongen et al. 2005). Ela pode ser utilizada para fazer a interface com diversas ferramentas de descoberta de conhecimento (*knowledge discovering tools*), como e.g. *Viscovery*[8] e *SPSS AnswerTree*[9].

Outra forma inovadora de visualizar evento no contexto de mineração de processos é a *dotted chart analysis* (Song and van der Aalst 2007), que demonstra uma visão geral do processo delineado e sua performance. Através deste paradigma, que também está implementado na plataforma ProM, é possível selecionar diferentes visões através da troca de tipos de componentes, opções de tempo ou ampliação de partes do gráfico.

2.3.4 Ferramentas para mineração de processos

Com o desenvolvimento de algoritmos de mineração de processos, fez-se necessário o desenvolvimento de ferramentas para dar suporte a estas abordagens. Desta forma surgiram as plataformas EMiT (van der Aalst and van Dongen 2002), *Little Thumb* (Weijters and van der Aalst 2003) e Minson (van der Aalst and Song 2004). No entanto elas apresentaram diversas limitações, como e.g. formato próprio para armazenar arquivos de logs, incapacidade de lidar com um número elevado de casos nos logs de eventos, entre outras.

[8] http://www.viscovery.net/

[9] http://www.spss.co.in/answertree.aspx

Estudos abrangentes sobre ferramentas para mineração de processos são apresentados por Ailenei (2011), Claes and Poels (2012). Neste trabalho vamos apresentar algumas plataformas comerciais e livres que estão difundidas na área:

(a) Dentre as ferramentas comerciais, a que vem ganhando destaque é o **Disco**[10], que é a sucessora do **Nitro**[11]. Esta ferramenta dá suporte a área de mineração processos e tem como principal objetivo extrair informações significativas de processos a partir de log de eventos, a fim de melhorar os processos. Nesta ferramenta não há necessidade de decidir qual algoritmo de mineração deve-se usar, dispensando a necessidade de aprender uma nova linguagem de modelagem de processos, pois cada algoritmo apresenta o resultado da mineração em uma notação específica. Ela apresenta algumas das principais estatísticas, como o número de eventos e casos incluídos no registro, informações sobre o período de tempo (data inicial e final) do processo observado e atividades incluídas no processo. A ferramenta proporciona *insights* para os diferentes variantes de casos (i.e a sequência das várias tarefas dos diferentes casos que aparecem). Tempos de *throughput* de tarefas e dos casos podem ser identificados. Ela se concentra em cada atributo que está incluído no log, como *timesptamp*, os recursos, as informações relacionadas com os custos, etc. Alguns resumos para qualquer atributo são dados para que conclusões diferentes possam ser tiradas.

   Outra ferramenta comercial é a **Flow**, desenvolvida pela companhia Norueguesa *Fourspark*[12]. Suporta o formato de importação do *Microsoft Office Excel* 2003 (arquivos .xls). A interface de usuário é construída como um painel de controle, fornecendo flexibilidade para o usuário criar vários *widgets* que exibem diferentes tipos de informação.

   Futura Reflect[13] é um sistema de mineração de processos desenvolvido na Holanda que pode ser usado tanto como uma ferramenta *stand-alone* ou como um componente integrado na suite *Pallas Athena*[14] *BPM/one*. *Futura Reflect* é uma ferramenta baseada na web que suporta para os logs de eventos o formato de entrada CSV. Após a análise, os resultados podem ser exportados e salvos como arquivos de imagem na máquina local ou como modelos *FLOWer* onde posteriormente podem ser carregados na plataforma *BPM/one*. Outro tipo de exportação que o *Futura Reflect* suporta é o formato Excel que pode ser usado para salvar os dados resultantes da análise de desempenho.

   ARIS Performance Manager (PPM)[15] pertence a empresa Software AG e é um componente da Plataforma ARIS. Tem como objetivo analisar os processos de negócios com base em dados históricos. O ARIS PPM suporta uma diversos formatos de entrada, tais como arquivos CSV, logs de eventos extraídos de bancos de dados e sistemas externos (e.g. ERP, CRM, entre outros), o formato gráfico PPM e o formato do sistema de eventos PPM. Os modelos descobertos podem ser exportados como arquivos de imagem, XML e arquivos AML. Enquanto que os dados gerados como

[10] http://fluxicon.com/disco/
[11] http://fluxicon.com/nitro/
[12] http://fourspark.no/
[13] http://www.futuratech.nl/site/
[14] http://www.pallas-athena.com/
[15] http://www.softwareag.com/nl/products/aris platform/aris controlling/aris process performance/

resultado de diferentes tipos de análise podem ser salvos em arquivos do *Microsoft Office Excel*[16].

ProcessAnalyzer[17] é uma ferramenta de mineração de processos baseado no *Excel* produzido pela empresa finlandesa QPR. A ferramenta é instalada como um *Add-on* no *Microsoft Office Excel*. Após a instalação, uma nova aba chamada *QPR ProcessAnalyzer* é criada no ambiente. O sistema é capaz de importar qualquer log de eventos adequado que está aberto no *Excel*, desde que o registro esteja contido em uma planilha com o título "Data". O *QPR ProcessAnalyzer* oferece cinco diferentes tipos básicos de análise de processos: *process*; *variation*; *timeline*; *path*; e *resource analysis*.

(b) Dentre as ferramentas livres, o **ProM** (van Dongen et al. 2005) está a frente das pesquisas de mineração de processos. O ProM é uma plataforma acadêmica extensível para implementar métodos de mineração de processos em um ambiente padrão, pois tem suporte para a implementação de diferentes algoritmos forma de *plug-ins* que servem para diferentes propósitos de mineração de processo, i.e. o ProM é chamado de "*framework*", pois é um aplicativo que pode ser estendido com plug-ins terceiros, onde cada *plug-in* implementa uma técnica de mineração diferente. Desta forma, o ProM permite aos pesquisadores a combinação de seus próprios algoritmos com de outras pessoas e a integração com muitas ferramentas existentes, tanto comerciais como livres.

O *framework* ProM recebe arquivos de entrada no formato *eXtensible Event Stream* (XES) ou *Mining* XML (MXML) e oferece suporte na forma de *plug-ins* para múltiplos formatos e linguagens, como e.g. redes de petri, EPC, redes sociais, entre outras. Eles podem ser utilizados de várias maneiras e combinados para serem aplicados em diversas situações da vida real. Devido a facilidade na instalação de novos *plug-ins*, os autores da plataforma recomendam que desenvolvedores e pesquisadores utilizem esta estrutura para implementação de novas propostas de algoritmos.

O formato extensível MXML suportado pela ferramenta de análise e mineração de processo ProM, é baseado no formato XML para armazenar logs de eventos do processo. Convertendo logs de eventos para o formato MXML, é possível torná-lo acessível para análise em profundidade no ProM, que é uma ferramenta utilizada para inteligência em processos de negócio.

Visando apoiar a conversão de dados de log de eventos para o formato MXML, foi desenvolvida a ferramenta ProM *Import*[18]. A maioria das funcionalidades desta ferramenta precisa ser configurada através de um filtro de importação.

O formato XES, que também é suportado pelo ProM, está em conformidade com o novo padrão desenvolvido pela IEEE *Task Force Process Mining*[19] para mineração de processos. Através da ferramenta de mapeamento XESame, é possível realizar o mapeamento de logs de eventos para o formato XES (Verbeek et al. 2010).

[16] http://office.microsoft.com/pt-br/excel/
[17] http://www.qpr.com/products/qpr-processanalyzer.htm
[18] http://www.promtools.org/promimport/
[19] http://www.win.tue.nl/ieeetfpm

Portanto, as ferramentas comerciais de mineração de processos podem ser a melhor escolha para profissionais que utilizam técnicas de mineração de processos somente como suporte para suas atividades principais. No entanto, o ProM é mais apropriado para pesquisadores de mineração de processos, pois torna mais fácil o desenvolvimento e teste de novos algoritmos, mas em contrapartida requer uma competência específica para utilizá-lo e não tem suporte profissional.

2.3.5 O algoritmo *Expectation-Maximization*

Na mineração de processos, alguns trabalhos foram desenvolvidos através do procedimento EM, demonstrando a aplicabilidade do algoritmo nesta área. Exemplos são dados em Ferreira et al. (2007), onde o algoritmo EM é utilizado para *clustering* de sequências na mineração de processos de negócio; Ferreira and Gillblad (2009) aplicam o procedimento EM para descobrir o modelo de processo quando o log de eventos fornece uma sequência de eventos não rotulados. Rebuge and Ferreira (2012) utilizam o procedimento EM para análise dos processos do serviço de urgência em uma organização de saúde.

O algoritmo *Expectation-Maximization*(EM), é um procedimento iterativo eficiente para calcular a máxima verossimilhança (*maximum-likelihood*) estimada, quando existem dados não observados ou ausentes. Em estatística, a estimativa por máxima verossimilhança é um método para estimar os parâmetros de um modelo estatístico. Desta forma, a partir de um modelo estatístico e um conjunto de dados, a estimativa por máxima verossimilhança estima valores para os diferentes parâmetros do modelo (Dempster et al. 1977, Moon 1996, McLachlan and Krishnan 2008).

Para calcular a máxima verossimilhança é desejado estimar os parâmetros do modelo para os quais os dados não observados são os mais prováveis. A convergência é assegurada uma vez que o algoritmo garante o aumento da verossimilhança a cada iteração. Assim, a cada iteração, o algoritmo executa dois passos:

(1) *Expectation* (*E-step*) – os dados ausentes são estimados de acordo com os dados observados que geraram o modelo e seus parâmetros iniciais.

(2) *Maximization* (*M-step*) – a verossimilhança é maximizada assumindo que os dados ausentes são conhecidos. A estimativa calculada para os dados ausentes no *E-step* são utilizadas para este fim.

Desta forma, o *E-step* pode ser entendido como um passo para contruir um mínimo local para a posterior distribuição, ao passo que o *M-step* melhora este valor, aperfeiçoando a estimativa dos dados faltantes.

O algoritmo EM tem desempenhado um grande papel em diversas áreas de pesquisa, como: perfis de aprendizagem de domínios proteicos (Anders et al. 1994); famílias RNA (Eddy and Durbin 1994); descoberta de módulos transcricionais (Segal et al. 2003); processamento de imagens (Carson et al. 2002); identificação de proteínas (Nesvizhskii et al. 2003); imagiologia médica (De Pierro 1995).

2.3.6 Cadeias de Markov

Atualmente, encontramos aplicações de cadeias de Markov em diversas áreas, como as Ciências Sociais, Biológicas e Administrativas. No entanto, no âmbito da mineração

de processos, os modelos de Markov foram estudados e aplicados em alguns casos: Cook and Wolf (1995) desenvolveram métodos para inferir uma FSM de um log de eventos, onde o algoritmo constrói um grafo de eventos como resultado considerando cadeias de Markov com ordem crescente, ao final o grafo é convertido para uma FSM que representa o processo. Maruster et al. (2002) proporam um método de aprendizagem que utiliza o log de *workflow* como entrada em um modelo de regressão logística para descobrir as conexões diretas entre os eventos de um log de *workflow* incompleto com ruídos. Ferreira et al. (2007) utilizaram os modelos de Markov em um procedimento EM para *clustering* de sequências, demonstrando ser uma técnica poderosa para resolver os diferentes comportamentos de um log de eventos e fornecer informações sobre sua estrutura básica.

Uma cadeia de Markov é um sistema matemático que sofre transições de um estado para outro, entre um número finito ou contável de possíveis estados. Particularmente, é um caso de processo estocástico com estados discretos que também apresenta a propriedade Markoviana, também é chamada de memória Markoviana (Markov 1971). A definição desta propriedade é que os estados anteriores são irrelevantes para a predição dos próximos estados, uma vez que o estado atual seja conhecido.

Os Processos Estocásticos são de interesse para descrever o procedimento de um sistema operando sobre algum período de tempo. Formalmente, a variável randômica $X(t)$ representa o estado do sistema no parâmetro (geralmente tempo) t. Portanto, pode-se afirmar que $X(t)$ é definido em um espaço denominado espaço de estados.

Um Processo Estocástico é dito ser um Processo Markoviano se:

$$\begin{aligned} &P\{X(t_{k+1}) \leq \mathrm{x}_{k+1} | X(t_k) = \mathrm{x}_k, X(t_{k-1}) = \mathrm{x}_{k-1}, ..., X(t_1) = \mathrm{x}_1, X(t_0) = \mathrm{x}_0\} \\ &= P\{X(t_{k+1}) \leq \mathrm{x}_{k+1} | X(t_k) = \mathrm{x}_k\} \\ &\text{para } t_0 \leq t_1 \leq ...t_k \leq t_{k+1} = 0, 1, ... \text{ e toda sequência } k_0, k_1, ..., k_{t-1}, k_t, k_{t+1} \end{aligned} \tag{2.1}$$

A Expressão 2.1 pode ser "traduzida" por: a probabilidade condicional de qualquer evento futuro, dado qualquer evento passado e o estado presente $X(t_k) = \mathrm{x}_k$, é independente do evento passado e depende somente do estado presente. Em outras palavras, um processo estocástico é dito ser um Processo Markoviano se o estado futuro depende apenas do estado presente e não dos estados passados. Este tipo de processo estocástico é também denominado de processo sem memória (*memoryless process*), uma vez que o passado é esquecido ou desprezado.

Portanto, se o espaço de estados é discreto (finito ou contável), então o modelo de Markov é denominado cadeia de Markov. As cadeias de Markov estão presentes em diversos problemas onde existem quantidades aleatórias que variam com o tempo.

Descrevemos uma cadeia de Markov como segue: Temos um conjunto de *estados*, $S = \{s_1, s_2, ..., s_r\}$. O processo inicia em um destes estados e move sucessivamente de um estado para outro. Cada movimento é chamado um *passo*. Se a cadeia corrente está no passo s_i, então ela move do estado s_j ao próximo passo com a probabilidade denotada por p_{ij} e esta probabilidade não dependente sobre o qual estado a cadeia estava antes do atual estado.

As probabilidades p_{ij} são chamadas *probabilidades de transição*. O processo pode permanecer no estado que ele está e isto ocorre com probabilidade p_{ii}. Uma distribuição de probabilidade inicial, definida em S, especifica o estado inicial. Normalmente, isto é

feito especificando um estado particular como o estado inicial. Para demonstrar as probabilidades de transição nos casos finitos é mais conveniente representar como uma matriz quadrada, chamada de *matriz de probabilidades de transição* ou *matriz de transição*. Estas características estão ilustradas na Figura 2.7, conforme o exemplo da matriz **T** abaixo:

$$\mathbf{T} = \begin{array}{c|ccccc} & 1 & 2 & ... & n & \\ \hline 1 & s_{11} & s_{12} & ... & s_{1n} & s_{1(n+1)} \\ 2 & s_{21} & s_{22} & ... & s_{2n} & s_{2(n+1)} \\ ... & ... & ... & ... & ... & ... \\ n & s_{n1} & s_{n2} & ... & s_{nn} & s_{n(n+1)} \end{array}$$

Figura 2.7: Exemplo de uma matriz de transição

Em termos mais resumidos, em uma cadeia de Markov o sistema de mudança de estado são chamadas transições e as probabilidades associadas com várias mudanças de estado são chamadas probabilidades de transição. O processo é caracterizado por um espaço de estado, uma matriz de transição descreve a probabilidade das transições e um estado inicial e final no espaço de estado.

2.4 Métricas para modelos de processo

Neste trabalho, estamos interessados em métricas que podem ser usadas para avaliar a complexidade do modelo de processo encontrado através da mineração de processo. Tais métricas servem de base para a definição e avaliação da abordagem proposta neste trabalho, o modelo hierárquico de Markov e seus componentes.

O termo complexidade é utilizado de diferentes formas, no entanto a definição de complexidade de modelos de processos de negócio foi dada por Vanderfeesten et al. (2007), que afirma que "a complexidade mede a simplicidade e capacidade de compreensão de um projeto". A baixa complexidade geralmente resulta em um modelo simples e extremamente compreensível. A complexidade de um modelo está totalmente relacionada com a compreensibilidade deste modelo (inversamente proporcionais), i.e., se a complexidade de um modelo é alta, a compreensibilidade deste modelo é baixa.

A compreensibilidade é um fator importante, pois analistas de negócio necessitam compreender o modelo para verificar se ele é válido e completo conforme o processo de negócio descrito, e analistas de sistemas necessitam entender o modelo a fim de implementá-lo ou representá-lo (Moody 1998).

A utilização da técnica de mineração descrita na Seção 3.3 produz um modelo hierárquico, onde o comportamento de eventos de baixo nível é separado em vários micro-modelos. Cada um destes micro-modelos corresponde a uma atividade de alto nível com o qual o analista está familiarizado. Na verdade, estas são as atividades de alto nível usadas por especialistas em negócios para descrever o processo de negócio. A separação do comportamento de baixo nível e estas atividades de alto nível resulta em modelos que são muito menos complexos do que se todo o comportamento de baixo nível for capturado em um único modelo.

A complexidade de um modelo pode ser medida de diversas formas. Muitas métricas para medir a complexidade de modelos de processo podem ser encontradas na literatura variando de métricas simples e óbvias como o número de atividades ou número de arcos, até métricas mais complexas, como *cyclomatic number* (McCabe 1976). Na literatura, existem vários indicadores que têm sido propostos para a análise de modelos de processos de negócios (Gruhn and Laue 2007, Vanderfeesten et al. 2007, Mendling 2009, Dijkman et al. 2011). Na maioria dos casos, estas métricas têm sua origem na análise de redes e engenharia de software.

Visando apresentar uma visão geral das métricas para modelos de processos, o restante desta seção está organizado da seguinte forma: na Seção 2.4.1 são apresentados os fundamentos sobre medição, além de discutir os fenômenos empíricos que podem ser capturados em termos de uma medida; na Seção 2.4.2 são apresentadas algumas métricas em disciplinas relacionadas que podem ser promissoras para modelos de processo de negócio, como a análise de redes e engenharia de software; na Seção 2.4.3 são apresentados os trabalhos relacionados sobre métricas para modelos de processo de negócio, bem como os fatores que influenciam a complexidade e compreensibilidade de um modelo.

2.4.1 Fundamentos de medição

As ciências naturais, sociais e engenharias, utilizam a sistemática de medição para formar conceitos abstratos em avaliações empíricas. A medição pode, no mínimo, servir para o seguintes propósitos: controle, compreensão e melhoria. A declaração clássica de DeMarco (1982) "você não pode controlar o que não se pode medir", enfatiza que baseado na compreensão da relação entre os diferentes atributos, pode-se fazer previsões como analisar se as metas serão cumpridas e que ações devem ser tomadas. Nesta definição, um atributo se refere a uma propriedade ou característica de uma entidade, enquanto uma entidade pode ser um objeto ou um evento do mundo real.

As métricas desempenham um papel importante na operacionalização de vários aspectos relacionados a qualidade nas áreas de modelagem de processos, engenharia de software e análise de redes. Diversos autores utilizam métricas para capturar diferentes aspectos de modelos de processo de negócio que são presumivelmente relacionados com a qualidade (Aguilar et al. 2006a, Laue and Gruhn 2006, Cardoso et al. 2006, Mendling 2007, Lassen and van der Aalst 2009, Mendling et al. 2010, Reijers and Mendling 2011, Kreimeyer and Lindemann 2011, SáNchez-GonzáLez et al. 2012).

A modelagem de processos de negócio ainda não tem estabelecida um conjunto geral de métricas comumente aceitas. Portanto, esta seção tem o objetivo de contribuir com uma revisão sucinta das métricas para modelos de processos conforme a literature disponível a partir dos anos 70.

A medição pode ser formalmente definida com um mapeamento, também chamada de *escala*, a partir do domínio empírico para o âmbito de conceitos matemáticos (Fenton and Pfleeger 1998). Na engenharia de software, observamos que os termos métrica e medição são muitas vezes utilizados como sinônimos. Não obstante, existem diversas descrições imprecisas na literatura para as características de uma métrica (Ghezzi et al. 2002, Sommerville 2006, Laird and Brennan 2006). Em Mendling (2007), utiliza-se o termo *métrica* para referenciar o tipo de *medição* que quantifica um certo atributo de uma entidade do mundo real através de um caminho pré-definido para quantificação.

Como tal, uma métrica busca representar uma situação real de uma forma reduzida e eficiente. Já a teoria da medição fornece a base para a concepção de tais métricas, pois descreve como um fenômeno pode ser medido através do estabelecimento de mapeamento de um conceito empírico para um conceito matemático.

De acordo com Zuse (1998), as métricas também podem ser classificadas como *base* e *derivada*. Métricas base normalmente surgem no início da pesquisa e são o ponto de partida para métricas derivadas, enquanto as métricas derivadas agregam uma ou mais métricas base.

Normalmente, conjuntos de métricas relacionadas são organizadas em um sistema de medição com o objetivo de agrupar as métricas com base em suas semelhanças e destinadas a representar uma questão empírica de uma forma bem equilibrada e completa. Na prática, sistemas de medição são usados como meios de análise, como e.g. na melhoria de processos, *benchmarking* ou até como ferramenta de gestão para planejamento e controle de um sistema complexo. De Toni and Tonchia (2001) apresentam em maiores detalhes estudos empíricos concentrando modelos, estruturas, características e indicadores de performance para sistemas de medição.

2.4.2 Métricas em disciplinas relacionadas

Visando construir conhecimento e características importantes aplicáveis na análise de complexidade e compreensibilidade de processos de negócio, foi realizado um estudo sobre as métricas em disciplinas relacionadas disponíveis na literatura. No contexto deste trabalho, métricas aplicadas na análise de redes e engenharia de software apresentam potencial interessante para construção deste conhecimento.

Métricas aplicas na análise de redes

Métricas para descrever a estrutura de redes (Brandes and Erlebach 2005) são geralmente derivadas da teoria dos grafos aplicada e teoria de redes. Como modelos de processos de negócio (e.g EPC's) são uma classe especial de grafos, também pesquisamos as técnicas de análise de redes.

Em Freeman (1979), Newman (2003) e Mendling (2007) observamos algumas características comuns utilizadas para descrever uma estrutura de rede: *Degree* – número de arestas que estão conectadas a um vértice; *Density* – mede a coesão geral do grafo através da relação entre arestas existentes e o número máximo de arestas possíveis; e *Connectivity* – mede o nível de homogeneidade da rede através do número de nós que devem ser removidos para que o grafo seja desconectado. *Centrality* – é uma métrica utilizada para representar a coesão de uma rede em torno de um nó central;

Um modelo de processo de negócio, expressado por exemplo em BPMN, incorpora diferentes aspectos da teoria dos grafos e pode ser visto com um tipo especial de grafos. Portanto, estas técnicas de análise de redes podem ser aplicadas também neste contexto.

Métricas na engenharia de software

Na engenharia de software, existem também diversas métricas para medir diferentes aspectos que são relevantes para a garantia da qualidade (Boger and Lyons 1985). O desafio é estabelecer uma relação entre métricas para atributos internos (que são calculados

de forma direta, como e.g. *Lines of Code*) e atributos externos (calculados indiretamente, como e.g. manutenibilidade, confiabilidade). Por este motivo, geralmente estas métricas são reagrupadas usando um sistema de medição como o *Goal Question Metric* (GQM) (Basili et al. 1994).

Normalmente, as métricas se diferem levando em consideração o paradigma de programação. A programação procedural usa uma série de chamadas de funções, além de divisão e junção de operações (como e.g. *goto*, *for*, *then*, *while... do*, etc) para constituir o controle de fluxo de um programa. Enquanto, a programação orientada a objetos utiliza classes para definir e instanciar objetos que então usam métodos para transformar dados. Assim, seus grafos de controle de fluxo variam significativamente.

As métricas estruturais mais comumente apresentadas na literatura relacionada ao paradigma procedural são baseadas na contagem de chamadas de função ao longo do controle de fluxo (Mendling 2007, Kreimeyer and Lindemann 2011):

(1) *Lines of Code (LOC)* (Azuma and Mole 1994): mede o tamanho de um programa, que pode variar substancialmente dentre as diferentes linguagens de programação, sendo portanto de valor informativo limitado.

(2) *Cyclomatic Number* (McCabe 1976): aplicável na medição de software, tendo sido introduzido para calcular a partir do grafo de controle de fluxo de um programa o número de caminhos independentes lineares. Por esta razão, um baixo número ciclomático indica que o programa é fácil de entender e modificar. O número ciclomático é também um indicador da testabilidade, uma vez que corresponde ao número de casos de teste necessários para atingir a cobertura completa de testes unitários.

(3) *Halstead's metrics* (*Length, vocabulary, volume, difficulty and effort*) (Halstead 1977): fornece definições mensuráveis para a complexidade de um programa de software com base na complexidade da linguagem (número de operadores e operandos distintos). Operadores compreendem comandos e elementos estruturais como parênteses. Operandos referem-se basicamente a elementos que têm um valor como variáveis e constantes. Embora este trabalho teve um impacto duradouro sobre a pesquisa de métricas de software, ela tem sido criticada como "um exemplo de medição confusa e inadequada".

(4) *Information flow metric* (Henry and Kafura 1981): utilizada para medir o *fan-in* e *fan-out* de um módulo, baseia-se no fluxo de dados entre os diferentes módulos de um programa. Esta métrica é comumente usada para estimar os erros mais prováveis.

(5) *Oviedo's data flow complexity* (Oviedo 1993): apresenta o grau em que um programa pode ser dividido em blocos distintos que são executados como uma unidade e tem apenas relação em um nível de definição. Desta forma, demonstra o potencial para a decomposição.

(6) *COnstructive COst MOdel* (Boehm 1981): é uma abordagem de estimativa de custos que usa a métrica de estimativa de esforço de programação como base para a estimativa de custos. Esta métrica é baseada no tamanho de um programa (e.g, *lines of code*) e na linguagem de programação.

(7) *Defect Density* (Zuse 1991): representa o número de defeitos encontrados em cada um dos módulos de um programa. É mais útil para localizar partes do software que são mais propensas a erro e se concentrar no esforço de programação destas partes, como e.g. os *outliers* entre todos os módulos.

Na programação orientada a objetos (OO), a coesão e o acoplamento são importantes e conduzem o desenvolvimento das métricas, conforme apresentado por (Chidamber and Kemerer 1994). Algumas destas métricas são enumeradas a seguir:

(1) *Weighted methods per class*: calculada através da soma dos pesos de complexidade ao longo de todos os métodos.

(2) *Depth of inheritance tree*: define uma hierarquia de classes na qual uma classe individual herda propriedades e métodos. A métrica dá a profundidade de tal herança para uma classe, a fim de descrever a forma como muitos antecessores potencialmente a afetarão.

(3) *Number of children*: fornece o número de sucessores na árvore de herança para uma determinada classe.

(4) *Coupling between object classes*: indica a quantas outras classes, uma classe está acoplada.

(5) *Response for class*: fornece o tamanho do conjunto de métodos que uma classe pode usar para responder a uma mensagem. É calculada através da soma dos métodos locais somados aos métodos chamados por estes métodos locais.

(6) *Lack of cohesion metric*: é o número de conjuntos disjuntos identificados pela comparação de cada par de métodos e as variáveis de instâncias que se relacionam. Um valor alto para essa métrica pode sugerir uma divisão da classe.

Existem diversas outras métricas disponíveis na engenharia de software e uma grande maioria delas que são mais conceituais não evoluiram dentro das práticas de projeto. No entanto, observamos que as métricas mais abstratas muito influenciaram o desenvolvimento de outras.

As métricas de engenharia de software podem ser muito relevantes para a análise de processos, pois o software é similarmente caracterizado por diversas decisões que não podem ser analisadas de forma determinística. Portanto, a transferência destes fundamentos para a gestão de processos se torna viável e adequada.

2.4.3 Métricas para processos de negócio

Existem diferentes formalismos utilizados na modelagem de processos de negócio e configuração de SI orientados a processos, conforme descrito na Seção 2.1.1. Sabe-se que os usuários destas linguagens tem problemas ao entender estes diagramas e que a complexidade dos modelos contribuem para diversos problemas relativos a falhas do projeto destes sistemas, conforme apresentado em Mendling (2007), Mendling et al. (2006; 2007a). O mesmo comportamento de um processo pode ser representado de diversas maneiras, então

se o modelo é muito complexo para ser entendido pelos usuários finais, pode-se decompor ou dividir o modelo.

Por esta razão, é muito útil ter medidas que forneçam informações sobre a compreensibilidade e manutenibilidade de modelos de processos de negócio. Tais medidas podem nos dizer se o modelo tem um tamanho apropriado, se é bem estruturado, fácil de compreender e dividido em módulos de forma sensata.

Vamos considerar neste trabalho a compreensibilidade e complexidade de modelos de processo de negócio como fatores determinantes para a melhoria do processo. Isto é baseado na hipótese de que os modelos de processo são construídos por humanos e que sua concepção está sujeita a uma racionalidade limitada devido a informação que está disponível, as limitações cognitivas de suas mentes e a quantidade finita de tempo que têm para tomar uma decisão (Simon 1996).

A compreensão de qualquer modelo de processo por um indivíduo é influenciada por uma variedade de fatores, incluindo fatores relacionados ao tamanho, fatores pessoais como competência na modelagem, finalidade da modelagem como documentação ou execução, conhecimento do domínio e linguagem de modelagem a ser utilizada. Comumente, as métricas não são independentes umas das outras, mas podem ser organizadas em um sistema de medição (e.g. de acordo com o foco, meta, granularidade). Descrevemos alguns destes fatores encontrados na literatura:

- Tamanho – diversos trabalhos apontam para métricas voltadas ao tamanho como um fator importante para a compreensão de modelos de software e processo. Enquanto o tamanho de software é frequentemente equacionado com linhas de código, o tamanho de um modelo está relacionado com a contagem do número de um tipo particular de elementos de um modelo de processo (e.g. *nodes, arcs, tasks, start-events, end-events, connectors, AND-splits, AND-joins, XOR-splits, XOR-joins, OR-splits, OR-joins*). Isto inclui a contagem do número de arcos (*arcs*) e dos nós (*nodes*). O último pode ser ainda subdividido em tarefas e conectores. As contagens mais específicas são sub-categorias dos diferentes tipos de conectores lógicos, como *AND-splits* and *OR-joins*.

- Controle de fluxo e estrutura – o controle de fluxo pode muito influenciar a compreensibilidade de um modelo de processo, i.e., um modelo sequencial é mais compreensível que um modelo com diversas execuções diferentes em paralelo ou diversos pontos de decisão. Nesta característica podemos incluir métricas relacionadas com os conectores e suas interações. Outras características, como e.g. partes cíclicas de um modelo, são presumivelmente mais difíceis de entender do que as partes sequenciais. A ciclicidade capta o número de nós de um ciclo e os relaciona com o número total de nós.

- Modularidade – está relacionada aos aspectos de um modelo de processo global com relação a divisão/decomposição em módulos ou sub-processos. Vanderfeesten et al. (2007) define a modularidade como "o grau no qual um projeto é dividido em vários módulos". A modularidade implementada de forma racional, pode auxiliar a compreensibilidade de um modelo de forma positiva.

- *Design e layout* – como os modelos de processos são muitas vezes representações gráficas de um processo de negócio, o *design* e *layout* do modelo de processo é de grande influência para a sua compreensibilidade e complexidade.

  Em Gemino and Wand (2003), Laue and Gruhn (2006), Mendling et al. (2007b) as características de projeto e desenho, como por exemplo a topologia (minimização do número de linhas cruzadas), nomenclatura (descrições textuais de elementos do modelo), modularidade (decomposição do modelo) e ferramentas de modelagem (*layout* do modelo), são apontadas como fatores com influência significativa na compreensão dos modelos de processo.

- Pessoal – os modelos apresentados estão relacionados ao modelo em si, mas há outros fatores que influenciam a capacidade de compreensão de modelos de processos, entre os quais existem alguns fatores pessoais que podem ser encontrados na literatura:

  - Mendling et al. (2007b) confirmam que a experiência e conhecimento em modelagem de processos tem influência positiva para a compreensão do modelo.
  - Outro fator que é facilmente esquecido, mas os especialistas afirmam que tem muita influência é o conhecimento do domínio (Mendling et al. 2007b). De fato, é importante a análise da compreensão do domínio e não apenas da compreensão dos elementos do modelo (Gemino and Wand 2003).
  - Gemino and Wand (2004) afirmam que a dificuldade de compreensão devido às diferenças entre a idéia do observador (compreensão) do modelo e o que o modelo representa pode ter três causas diferentes: (1) o usuário interpreta mal o diagrama devido a falta de experiência com o método; (2) o usuário desenvolve uma concepção diferente do domínio representado daquela transmitida pelo diagrama; (3) existe ambiguidade dentro do próprio modelo.

Atualmente, as métricas para avaliação de modelos de processo ainda não estão consolidadas ou compiladas em um corpo coerente de conhecimento. Enquanto medidas quantitativas têm sido usadas há muito tempo, o conhecimento sobre a avaliação qualitativa dos processos e suas estruturas ainda está fragmentado (Mendling 2007). O trabalho pioneiro em métricas qualitativas ocorreu somente nos anos 90 com a avaliação de redes de petri (Soo and Jung-Mo 1992).

Dentre os trabalhos relacionados a métricas para modelos de processo, os autores dificilmente referem-se uns aos outros, tornando difícil agrupar as contribuições por área de aplicação, mas geralmente apresentam dois tipos de métricas: (1) aquelas que são principalmente usadas para a predição do comportamento de processos; e (2) aquelas que são usadas para estimar erros em modelos de processo. No entanto, a fronteira entre os dois tipos não pode ser claramente definida, pois frequentemente os modelos de processos são concebidos de uma forma que não estão completamente livre de erros, enquanto os desvios de um modelo correto semânticamente e semioticamente são destinados a transmitir uma certa finalidade ou significado (Mendling 2007).

No geral, muitas métricas são semelhantes e utilizam conceitos semelhantes que foram, em parte, incorporados de forma independente. Visões abrangentes sobre métricas relacionadas a *workflows* e modelos de processo de negócio inicialmente foram realizadas por Cardoso (2006), Laue and Gruhn (2006). Posteriormente contribuições relevantes foram realizadas nos trabalhos de Mendling (2007), Kreimeyer and Lindemann (2011).

Na literatura, diversas métricas com o foco em modelos de processos de negócio foram criadas e pesquisadas, diversas foram adaptadas de métricas utilizadas na análise de redes e engenharia de software, conforme apresentado na Seção 2.4.2. Neste contexto, descrevemos a seguir de forma cronológica diversas abordagens encontradas na literatura aplicadas na medição de processos de negócio:

(1) *Maximum/Mean nesting depth* (Schroeder 1984). São métricas que tem influência sobre a complexidade global do modelo: quanto maior o *nesting depth* maior a complexidade. Ambas as métricas demonstram ter uma influência sobre estruturas relacionadas a outras métricas de complexidade. A definição destas métricas para modelos de processo de negócio é simples: O *nesting depth* de uma ação é o número de decisões do controle de fluxo que são necessárias para executá-la. Esta métrica pode ser calculada adicionalmente a *Cardoso Metric* (Cardoso 2005).

(2) *Structural effectiveness metrics – simplicity, flexibility, integration and efficiency* (Tjaden et al. 1996). Este trabalho operacionalizou quatro características de um processo de negócio que precisam ser equilibradas: *Simplicity* representa basicamente o tamanho do processo; *Flexibility* e *Integration level* não são calculadas, mas determidas usando uma pontuação semelhante a abordagem de análise de pontos por função; *Efficiency* é uma característica que embora mencionada no trabalho, os autores não forneceram um conceito detalhado. Esta abordagem é criticada por Balasubramanian and Gupta (2005) como demasiadamente abstrata.

(3) *Daneva's cohesion metrics – Cohesion of functions, events, and data objects* (Daneva et al. 1996). Neste trabalho é introduzido um conjunto de métricas de complexidade para EPC's com base nos elementos de notação do modelo: *function cohesion*, *event cohesion* e *cohesion of a logical connector*. A partir de uma validação limitada a onze modelos EPC, os autores concluem que essas métricas são apropriadas para identificar fragmentos propensos a erros do processo.

(4) *Nissen's metrics for designing engineering – Distinct paths, hierarchy levels, cyles, diameter, parallelism* (Nissen 1998). Também como um dos primeiros conceitos, destina-se a facilitar o (re)*design* de um processo. Ela conta o número de caminhos independentes, os níveis de hierarquia encontrados no modelo, o número de ciclos independentes, o diâmetro total (i.e., o caminho mais longo) e o grau de paralelismo entre atividades.

(5) *Morasca's metrics for concurrent software specifications – size, length, structural complexity and coupling (concurrent and sequential contribution, arcs of a subnet)* (Morasca 1999). A partir de esforços de medição para programas concorrentes, esta abordagem propõem um conjunto de métricas para redes de petri e uma base teórica. *Size, lenght, structural complexity and coupling* são identificados como atributos de uma rede de petri e para cada atributo define-se um conjunto de propriedades axiomáticas que uma respectiva métrica teria que cumprir. *Size* usa o número de lugares e transições. *Length* usa a distância máxima e mínima (i.e. o diâmetro da rede). *Structural complexity* usa o número caminhos de base. *Concurrent contribution* usa o número de arcos menos duas vezes o número de transições. *Sequential contribution* usa o número de transições com um lugar de entrada. *Coupling* usa o número de

arcos de entrada ou saída de uma sub-rede. No entanto, esta contribuição é somente teórica e não possui uma validação empírica.

(6) *Latva-Koivisto's complexity metrics – Coefficient of Network Connectivity, Cyclomatic Number, Reduction Complexity Index, Restrictiveness Estimator and Number of Trees in a Graph* (Latva-Koivisto 2001). Propõe métricas de complexidade para vários modelos de processos de negócios, incluindo *Coefficient of Network Connectivity, Cyclomatic Number, Reduction Complexity Index, Restrictiveness Estimator and Number of Trees*. Segundo (Mendling 2009), este trabalho possui dois pontos fracos básicos: Primeiro, não há distinção entre os diferentes tipos de elementos de roteamento; Segundo, está faltando a motivação para considerar o *Restrictiveness estimator* e *Number of Trees* em um grafo como uma métrica de complexidade. Este trabalho não recebeu muita atenção da comunidade deste que foi publicado mantendo-se apenas como um relatório técnico.

(7) *Cardoso's control-flow complexity metric* (Cardoso 2005). Esta métrica é uma adaptação de *McCabe's Cyclomatic Number*, destinada a análise de complexidade de *workflow* através da medição do número possível de decisões de controle de fluxo, i.e., ela é utilizada em linguagens com as estruturas *XOR-*, *OR-* e *AND-splits* (como e.g. Petri nets) através da medição destes tipos de conectores na rede e atribuição para cada um de uma certa penalidade. A base lógica para cada *split*, é atribuir uma penalidade dependendo de quantos sub-estados diferentes ele induz quando executado. Esta métrica é bem adequada para a medição do número de casos de teste necessários em um modelo, não levando em consideração outras informações de estruturas relacionadas, mas não foi projetada para prever erros.

(8) *Metrics for process planning – Degree of automatization, activity automation, role integration, role dependency, activity parallelism, delay risk* (Balasubramanian and Gupta 2005). Inspirada no trabalho apresentado por Tjaden et al. (1996), este conjunto de métricas suporta o planejamento do processo, apoiando a avaliação de diferentes alternativas para um determinado processo. Para este fim, é estendido um conjunto existente de métricas por algumas métricas que abordam a possibilidade de execução automatizada do processo e tomada de decisão sempre que possível, a integração proposital de recursos, a medida do grau de paralelismo entre as tarefas, bem como a identificação de gargalos estruturais que possivelmente causam atrasos.

(9) *Maintainability metric – Analyzability, understandability, modifiability, all using the size, complexity, coupling of a process* (Aguilar et al. 2006a). Métrica para estimar a manutenabilidade de um processo através da divisão em três conceitos: (1) *Analyzability* – aborda a probabilidade de descobrir erros no processo; (2) *Understandability* – estima a facilidade na compreensão do modelo; (3) *Modifiability* é uma medida para a probabilidade de fazer a alteração correta aos modelos errados. As métricas são baseadas no tamanho (contando todas as entidades no processo), complexidade (com base no número de dependências) e acoplamento (entre atividades).

(10) *Number of activities* (Cardoso et al. 2006) – é equivalente a métrica mais simples de complexidade para software *Lines of code*. Por esta razão, o "número de atividades" é uma medida fácil de compreender para o tamanho de um modelo de processo de

negócio. Não obstante, esta métrica não leva em consideração a estrutura do modelo: um diagrama de um modelo de processo de negócio pode ser elaborado com um controle de fluxo bem estruturado, que é fácil de entender ou de forma não estruturada, tornando complexo o entendimento. Por esta razão, também são discutidas neste trabalho outras métricas que medem a complexidade do controle de fluxo.

(11) *Cardoso's log-based complexity metric* (Cardoso 2007) – Extensão da métrica *Cardoso's control-flow complexity*. Tem um foco estrutural para medição em tempo de execução que avalia a complexidade de um processo com base nos dados de registro de como o processo é executado. Portanto, é adequada apenas para *workflows* formalizados.

(12) *Cognitive weights* (Gruhn and Laue 2007) – A idéia chave desta abordagem, é atribuir um peso cognitivo (*cognitive weight*) para as estruturas básicas de controle de software: o mais difícil de uma estrutura de controle é entender seu maior peso cognitivo. A abordagem é baseada em coeficientes empíricos consolidados para os padrões básicos de controle de fluxo.

(13) *Log-based process metrics – Unary significance, binary significance, and binary correlation* (Günther and van der Aalst 2007). Estas métricas foram desenvolvidas para um trabalho de mineração de processos que trata modelos de processos pouco estruturados, que tendem a ser bastante confusos e difícil de entender, normalmente referenciados como *Spaghetti models*. O objetivo desta técnica de mineração é apresentar graficamente o comportamento mais relevante, através de um cálculo de relevância das atividades e suas relações. Para alcançar isso três tipos de métricas primárias foram implementadas: (1) *Unary significance* descreve a importância relativa de uma classe de eventos, a qual será representada como um nó no modelo de processo. Como esta abordagem é baseada na remoção de comportamento menos significativo, e como a remoção de um nó implica a remoção de todos os seus arcos conectados, a métrica *Unary significance* é o principal condutor desta simplificação. Em outras palavras, mede o nível de interesse relacionado aos eventos (e.g. através do cálculo de suas frequências no log); (2) *binary significance* descreve a importância relativa de uma relação de precedência entre duas classes de eventos, i.e. uma aresta no modelo de processo. O seu objetivo é o de amplificar e isolar o comportamento observado que é suposto ser do maior interesse. Esta abordagem de simplificação principalmente influencia a seleção de arestas que serão incluídas no modelo de processo simplificado; e (3) *Binary correlation* que determina qual a relação de dois eventos que seguem um ao outro, de modo que eventos altamente relacionados possam ser agregados.

(14) *Cognitive cross-connectivity* (Vanderfeesten et al. 2008). Esta métrica força as ligações entre os pares de entidades no processo com base na hipótese de que "a relação estrutural entre qualquer par de elementos de modelo" é uma operação mental complexa, i.e. seu objetivo é capturar o quão difícil é entender como dois nós em um modelo de processo se relacionam entre si. Esta métrica é validada para prever erros e compreensão de modelos de processos.

(15) *Understandability* (Ghani et al. 2008). Além de outras medidas de complexidade comuns, este trabalho introduz a capacidade de compreensão de um modelo de processo

como uma contribuição para a sua manutenção. É proposto um cálculo baseado em anti-padrões, para o qual no entanto, os autores não fornecem uma evidência empírica.

(16) *Conformance checking – Fitness and Appropriateness* (Rozinat and van der Aalst 2008). O alinhamento de negócio na área de TI está relacionado a uma estado dinâmico de negócios que permite que as organizações utilizem TI de forma mais eficaz, o que significa geralmente se concentrar mais nos resultados, i.e. a capacidade de TI para produzir valor de negócio.

Este trabalho responde uma importante questão nas organizações:Nosso processo está sendo executado conforme foi planejado? Através de abordagem incremental, este trabalho mede a conformidade entre o comportamento modelado (i.e. que foi definido no modelo de processo) e o comportamento observado (i.e. o processo que foi executado presente nos logs de eventos).

Diversos SI produzem diversos tipos de log de eventos (ERP, CRM, SCM, B2B), que são logs de transação ou trilhas de auditoria, tipicamente com atributos para identificar as atividades realizadas (início ou fim), *originator* (quem criou/originou certo evento) e *timestamp* (quando certo evento ocorreu). Desta forma, é possível realizar uma análise de correspondência entre o log de eventos e o modelo de processo possibilitando a melhoria de processos através do diagnóstico de níveis de granularidade do modelo de processo, identificação de gargalos de performance, probabilidades e frequências.

Esta abordagem se baseou no desenvolvimento de métricas com o objetivo de detectar as inconsistências entre o modelo de processo e o log de execução correspodente. Para medir esta conformidade foram desenvolvidas métricas através de duas dimensões: (1) *Fitness* que verifica quais traços do log podem ser associados a uma execução válida especificada no modelo do processo; e (2) *Appropriateness* que verifica o grau de precisão em que o modelo de processo descreve o comportamento observado, combinado com o grau de clareza no que ele é representado.

(17) *Metrics for business process models – Size, density, partitionability, connector interplay, cyclicity, concurrency* (Mendling 2009). O autor coleta e estende métricas e suas bases empíricas para verificar modelos de processos e prever possíveis erros. Essas métricas fazem uso de um forte formalismo orientado a linguagem de modelagem processo EPC, incorporando este formalismo para avaliação dos modelos de tais modelos para sua correção formal.

(18) *Extended cyclomatic metric* (Lassen and van der Aalst 2009). É uma adaptação da métrica *McCabe's Cyclomatic Number* (McCabe 1976). Apenas as redes de *workflow* são redes de máquina de estado e se comportam com um grafo de controle de fluxo, i.e. que não permite a concorrência/simultaneidade. Neste sentido esta métrica foi extendida para resolver esta questão considerando quais estados podem estar no processo em que as transições podem ocorrer.

(19) *Extended cardoso metric* (Lassen and van der Aalst 2009) – É uma generalização e melhoria da *Cardoso Metric* (Cardoso 2005) com o objetivo de trabalhar com *Non-free Choice Petri nets*, onde decisões mais complexas são possíveis.

(20) *Structuredness metric* (Lassen and van der Aalst 2009) – Esta métrica foi desenvolvida a partir dos problemas encontrados nas métricas *Cardoso Metric* (Cardoso 2005) e *McCabe's Cyclomatic Number* (McCabe 1976). A *Cardoso Metric* tem somente o foco na sintaxe do modelo e ignora a complexidade do comportamento. O *AND-split* é considerado mais simples que *XOR-split* enquanto o número de estados possíveis em uma rede de concorrência pode ter um número exponencial de tarefas. Isto pode ser resolvido atribuindo ao *AND-split* uma alta penalidade. No entanto, isto não resolve o problema uma vez que a complexidade do comportamento resulta da interação entre os componentes de modelagem diferentes. *McCabe's Cyclomatic Number* tem o foco somente no comportamento resultante e ignora a complexidade do modelo em si. Variantes de ambas as métricas tem dois problemas em comum: (i) focam em um único aspecto (comportamento x sintaxe); (ii) não consideram a interação entre os elementos. A rede consiste somente em *AND-splits*, *AND-joins* ou *XOR-splits* e *XOR-joins* que são comportamentos simples se comparados a uma rede com todos os tipos de escolhas e concorrências. Existem certos tipos de padrões que são mais complicados que outros, e.g. construções envolvendo *arbitrary loops*, *milestones*, e *syncronizing merges* são identificados como complexos.

    A idéia por trás da *Structuredness Metric* decorre da observação de que redes de *workflow* são muitas vezes estruturadas em termos de *Design Patterns* (Beck and Cunningham 1987). Partes do modelo de processo podem implementar padrões básicos, como sequência, escolha, iteração, etc, ou padrões mais avançados de controle de fluxo (van der Aalst et al. 2003a). Estes têm um grau variável de quão bem eles são compreendidos e comunicados pelas pessoas. Portanto, esta métrica foi desenvolvida para captar essa visão que reconhece diferentes tipos de estruturas e resultados de cada estrutura, dando-lhe alguns valores de penalidade. A soma destes valores é usado para definir a *Structuredness Metric*.

(21) *Behavioural profiles* (Weidlich et al. 2011) – *Trace Consistency and Profile Consistency*. Este conceito captura as restrições comportamentais essenciais de um modelo de processo através da derivação de um perfil comportamental e da medição do um grau de consistência do perfil. A estrutura de um perfil comportamental fornece uma maneira simples e direta para definir um grau de consistência variando de 0 a 1.0, conhecido como o grau de consistência do perfil. Desta forma, pode-se coletar informações detalhadas dos analistas de negócios e projetistas de software de quanto e onde os modelos de processo podem desviar um do outro.

A Tabela 2.4 fornece uma lista das métricas que podem ser aplicadas na análise de modelos de processo. A partir de uma ordem cronológica destes trabalhos, podemos observar que nos últimos anos cresceu o interesse em derivar métricas especialmente para o propósito de analisar modelos de processos de negócio.

2.5 Trabalhos correlatos

A lacuna entre atividades de alto nível e eventos de baixo nível é um problema muito conhecido na área de mineração de processo (Greco et al. 2005, Günther and van der Aalst 2007, Günther et al. 2010, Bose et al. 2012, Baier and Mendling 2013). Apesar

Métrica	Referência
Maximum/Mean nesting depth	(Schroeder 1984)
Structural effectiveness metrics	(Tjaden et al. 1996)
Daneva's cohesion metrics	(Daneva et al. 1996)
Nissen's metrics for designing engineering	(Nissen 1998)
Morasca's metrics for concurrent soft. spec.	(Morasca 1999)
Latva-Koivisto's complexity metrics	(Latva-Koivisto 2001)
Cardoso's control-flow complexity metric	(Cardoso 2005)
Metrics for process planning	(Balasubramanian and Gupta 2005)
Maintainability metric	(Aguilar et al. 2006b)
Number of activities	(Cardoso 2006)
Cardoso's log-based complexity metric	(Cardoso 2007)
Cognitive weights	(Gruhn and Laue 2007)
Log-based process metrics	(Günther and van der Aalst 2007)
Cognitive cross-connectivity	(Vanderfeesten et al. 2008)
Understandability	(Ghani et al. 2008)
Conformance checking	(Rozinat and van der Aalst 2008)
Metrics for business process models	(Mendling 2009)
Structuredness metric	(Lassen and van der Aalst 2009)
Behavioural profiles	(Weidlich et al. 2011)

Tabela 2.4: Metricas para análise de modelos de processos de negócio

do desenvolvimento de uma ampla variedade de técnicas de mineração de processo, a maioria delas é capaz de descobrir modelos de comportamento que estão ao mesmo nível de abstração dos eventos registados no log de eventos. No entanto, com o crescente interesse geral na mineração de processos, os usuários finais estão procurando soluções para analisar os dados do evento e visualizar os resultados em um nível mais alto de abstração, de preferência no mesmo nível em que eles estão acostumados a modelar seus processos de negócio.

Recentemente, a comunidade acadêmica tem olhado para este problema e, enquanto tema corrente de pesquisa, algumas abordagens de mineração de processos foram propostas para serem capazes de produzir modelos mais abstratos a partir de um log de eventos. Estas abordagens podem ser divididas em dois grupos principais:

(a) Existem técnicas que funcionam com base em modelos, através da extração de um modelo de baixo nível do log de eventos e, em seguida, criam representações mais abstratas do que este modelo. Exemplos são apresentados por Greco et al. (2005) e Günther and van der Aalst (2007). Basicamente, estas técnicas funcionam por agregação de nós no modelo, a fim de produzir uma imagem simplificada e mais abstrata do processo. Em geral, estas abordagens permitem uma simplificação do processo passo a passo, até que, no limite, tudo é agregado em um único nó. É o usuário final, que deve saber até onde deve realizar a simplificação, a fim de obter resultados significativos. A desvantagem destas abordagens é que não é possível identificar automaticamente os nós agregados como atividades significativas de negócio (eles são

simplesmente rotulados como "*Cluster* A", "*Cluster* B", etc), por isso pode ser difícil para o usuário final compreender e analisar os resultados.

(b) Há técnicas que funcionam na base dos eventos, traduzindo o log de eventos em uma sequência mais abstrata de eventos e, em seguida, produz um modelo em que traduz o log de eventos. Exemplos destas técnicas são descritos por Günther et al. (2010), Bose et al. (2012) e Baier and Mendling (2013). Basicamente, as duas primeiras técnicas trabalham identificando padrões frequentes de log de eventos e, em seguida, substituindo cada um desses padrões por um único evento, de nível superior. Depois que todas as substituições foram feitas, o log de eventos torna-se uma sequência de eventos mais abstratos. A terceira faz o pré-processamento do log de eventos através do mapeamento entre os eventos e atividades utilizando como entrada, além do log de eventos, a documentação do processo, i.e. instruções de trabalho. Como etapa final nestas abordagens, é possível extrair um modelo por meio de técnicas usuais, como o α-*algorithm* (van der Aalst et al. 2004), o *heuristics miner* (Weijters et al. 2006) e o *genetic miner* (Medeiros et al. 2007). Tal como acontece com as outras abordagens, é possível realizar essa abstração em múltiplos estágios, no entanto nos trabalhos realizados por Günther et al. (2010), Bose et al. (2012) não há garantia de que os padrões de eventos que são identificados no log de eventos correspondem a atividades significativas de negócio, por isso é o usuário final que determina se tal correspondência realmente existe. No trabalho apresentado por Baier and Mendling (2013) a ausência de documentação do processo pode inviabilizar a anotação dos relacionamentos.

Por outro lado, diversos autores também têm focado no uso de modelos hierárquicos como um meio para reduzir a complexidade e melhorar o entendimento dos processos de negócio. Mesmo que aqui estamos utilizando um tipo específico de modelo hierárquico, i.e. o modelo hierárquico de Markov, o conceito de comportamento aninhado em sub-processos se aplica a muitas linguagens de modelagem de processos. Na literatura, esse conceito é muitas vezes referido por termos diferentes, como *modularidade* (Reijers and Mendling 2008), *sub-processos* (Reijers et al. 2010) ou *hierarquia* (Zugal et al. 2012). Em particular:

- Reijers and Mendling (2008) conduziu um experimento envolvendo 28 consultores profissionais e experientes modeladores de processo. Esses consultores foram apresentados a dois processos de negócios, que eram bastante complexos (mais de 100 tarefas). Para cada um destes processos, havia dois modelos: um modelo que utilizou modularidade (sob a forma de sub-processos aninhados), e outro completamente sem modularidade (*flattened*). Os consultores foram convidados a responder a um conjunto de perguntas sobre esses processos. Observou-se que o número de respostas corretas para os processos modulares foram maiores do que para os processos *flattened*. A diferença foi estatisticamente significativa e os autores concluem que a modularidade tem um impacto positivo sobre a compreensibilidade de modelos de processos.
- Baseado no mesmo experimento, Reijers et al. (2010) foram ainda mais longe, aplicando um conjunto de métricas para medir a complexidade das versões modulares

de ambos os processos de negócios. Para este efeito, foram utilizadas várias métricas, incluindo o número de sub-processos em cada modelo. Eles descobriram que um desses modelos tem um maior número de sub-processos do que o outro e é esse modelo que leva o maior número de respostas corretas por parte dos consultores. Além disso, os autores propõem diversos critérios para ajudar o analista a identificar quais nós em um modelo de processo devem ser colocados juntos em um sub-processo. É interessante notar que nossa abordagem proposta neste trabalho, este passo é realizado automaticamente por nossa técnica de mineração.

- Em Zugal et al. (2012) encontra-se uma discussão do problema a partir de uma perspectiva cognitiva/psicológica e sugere que há evidências contraditórias sobre o impacto da hierarquia na inteligibilidade de modelos de processos. Por um lado, a hierarquia fornece abstração e ocultação de informações, e se torna mais fácil de responder a certas perguntas quando alguns dos detalhes são escondidos, de modo que há menos elementos a serem considerados. Por outro lado, faz com que uma hierarquia cause uma atenção ao efeito da divisão quando a resposta à pergunta pode somente ser encontrada considerando vários sub-processos ao mesmo tempo. Os autores concluem que o impacto da hierarquia na compreensibilidade depende das perguntas feitas. Eles também argumentam que para um impacto positivo ser perceptível, é necessário presumivelmente um determinado tamanho do modelo.

Em geral, estes trabalhos sugerem que o uso de hierarquia traz benefícios em termos de compreensibilidade de modelos de processos. Neste trabalho, também propomos um modelo de melhoria através de um conjunto de métricas para medir a complexidade dos modelos hierárquicos. Enfim, estas métricas podem também servir como um indicador para a compreensibilidade de tais modelos.

2.6 Considerações

Neste capítulo, introduzimos os fundamentos teóricos relacionados ao nosso trabalho. Fundamentos sobre processos de negócio, modelagem e execução de processos foram apresentados no intuito de introduzir ao tema de pesquisa. Foram explorados conceitos de simulação baseada em agentes através de uma pesquisa sobre as plataformas existentes, configuração e modelagem de cenários de simulação. Foi discutida a mineração de processos, técnicas, perspectivas e ferramentas. Também foram pesquisadas métricas que podem ser usadas para avaliar a complexidade de modelos de processo, bem como métricas em áreas correlatas. Todos os resultados teóricos apresentados nesta seção foram discutidos focalizando o tema desta pesquisa.

Trabalhos desenvolvidos por diversos autores sugerem que a maturidade nos processos de negócio (Hammer and Champy 1993, Aguilar-Savén 2004, Mcafee and Brynjolfsson 2008) contribui para atingir os objetivos de negócio nas organizações. Neste contexto, a modelagem dos processos de negócio fornece a compreensão e análise dos processos. Observamos diversas linguagens para modelagem de processos e um estudo comparativo através de diversas perspectivas foi realizado no trabalho de List and KorList (2006). Aspectos relacionados a execução de processos (van der Aalst 2000), tanto através através de sistemas de *workflow* aplicados em SI empresariais, como através de simulação também

foram discutidos, pois facilitam a análise e diagnóstico do processo visando a identificação de falhas para melhoria no modelo.

Com respeito a mineração de processos, diversos autores propõem técnicas para minerar modelos em log de eventos para tratar diferentes problemas através de três perspectivas: *controle de fluxo* (van der Aalst et al. 2004, Medeiros et al. 2008, Medeiros 2006)); *organizacional* (Song and van der Aalst 2008, van der Aalst et al. 2005, van der Aalst and Song 2004, Reungrungsee et al. 2012); e *performance* (Song and van der Aalst 2007). Também foi apresentado um estudo sobre ferramentas para dar suporte às abordagens de mineração de processos, como e.g. ProM (van Dongen et al. 2005), Disco, entre outras. Estudos exploratórios sobre ferramentas de mineração e suas características são apresentadas nos trabalhos de Ailenei (2011), Claes and Poels (2012).

Com o objetivo de selecionar métricas aplicáveis no contexto de modelos de processos de negócio, foi realizado um estudo da literatura pertinente a esta temática (conforme resumo da Tabela 2.4) e em áreas correlatas, como a análise de redes e engenharia de software, conforme descrito na Seção 2.4.2. Através deste estudo foi possível identificar as métricas que são aplicáveis na análise de complexidade de modelos hierárquicos, que é o foco deste trabalho.

Ao final, os trabalhos correlatos foram discutidos e comparados com a abordagem proposta neste trabalho. Encontramos abordagens de mineração baseadas em modelos (Günther et al. 2010, Bose et al. 2012, Greco et al. 2005, Günther and van der Aalst 2007) e técnicas baseadas em eventos (van der Aalst et al. 2004, Weijters et al. 2006, Medeiros et al. 2007). Também focamos o estudo no uso de modelos hierárquicos como um meio para reduzir a complexidade e melhorar o entendimento dos processos de negócio, onde encontramos os trabalho referentes a *modularidade* (Reijers and Mendling 2008), *subprocessos* (Reijers et al. 2010) e *hierarquia* (Zugal et al. 2012). Finalmente, aspectos que diferenciam a abordagem proposta discutida nesta pesquisa foram apresentados.

No Capítulo 3 apresentamos uma abordagem de solução baseada em um ciclo iterativo de melhoria de processos de negócio. A estratégia adotada busca proporcionar uma melhor compreensão aos gestores no direcionamento preciso do processo e como ele deve ser melhorado. A abordagem fornece uma base construtiva para a análise de processos, auxiliando os profissionais na avaliação sistemática dos resultados de iniciativas de melhoria nas organizações.

Capítulo 3

Abordagem proposta

Este capítulo descreve a abordagem integrada de simulação baseada em agentes e mineração de processos para a melhoria de modelos de processo de negócio. No âmbito de cada etapa da abordagem, apresentamos em detalhes as contribuições que foram desenvolvidas no decorrer deste trabalho. Nesta direção, o ciclo de melhoria proposto pode apoiar uma série de atividades essenciais contempladas na reengenharia e melhoria de processos de negócio, i.e. a abordagem de melhoria pode proporcionar uma melhor compreensão aos gestores no direcionamento preciso do processo e como ele pode ser melhorado. A abordagem fornece uma base construtiva para a análise de processos, auxiliando os profissionais na avaliação dos resultados de forma mais sistemática, bem como nas iniciativas de melhoria nas organizações.

3.1 Abordagem iterativa de melhoria de processos

Como premissa deste trabalho assume-se que especialistas de negócios sejam capazes de fornecer uma descrição de alto nível do processo de negócio. Tipicamente, o processo é descrito em termos de um modelo de processo, com um conjunto de atividades de alto nível. Por outro lado, existem as técnicas de mineração de processos para extrair o comportamento de um processo de negócio a partir de logs de eventos, mas os eventos de baixo nível que são registradas no log de eventos podem não ter uma relação clara com as atividades de alto nível definidas no modelo de processo.

Neste contexto, propomos um ciclo de melhoria, conforme ilustrado na Figura 3.1, que descreve uma abordagem com o objetivo de melhorar modelos de processo através de uma estratégia de mineração hierárquica. A idéia principal é desenvolver uma abordagem baseada em ciclos repetidos (i.e. iterações), permitindo analistas de negócio identificar oportunidades de melhoria do macro-modelo a partir dos modelos que são descobertos durante as iterações. Em cada iteração, modificações de projeto dos modelos de processo são realizadas e novos modelos mais "equilibrados" em termos de complexidade e compreensibilidade podem ser obtidos. Passaremos a detalhar cada estágio do ciclo de melhoria conforme apresentado na Figura 3.1.

Como em qualquer projeto, a melhoria de modelos de processo necessita ser cuidadosamente planejada. Antes de proceder com o esforço para o estágio de mineração de processo, deve-se verificar as fontes de dados que irão apoiar o projeto (Estágio 1). Desta forma, um importante passo é planejar os dados que serão fornecidos com entrada, e.g. a

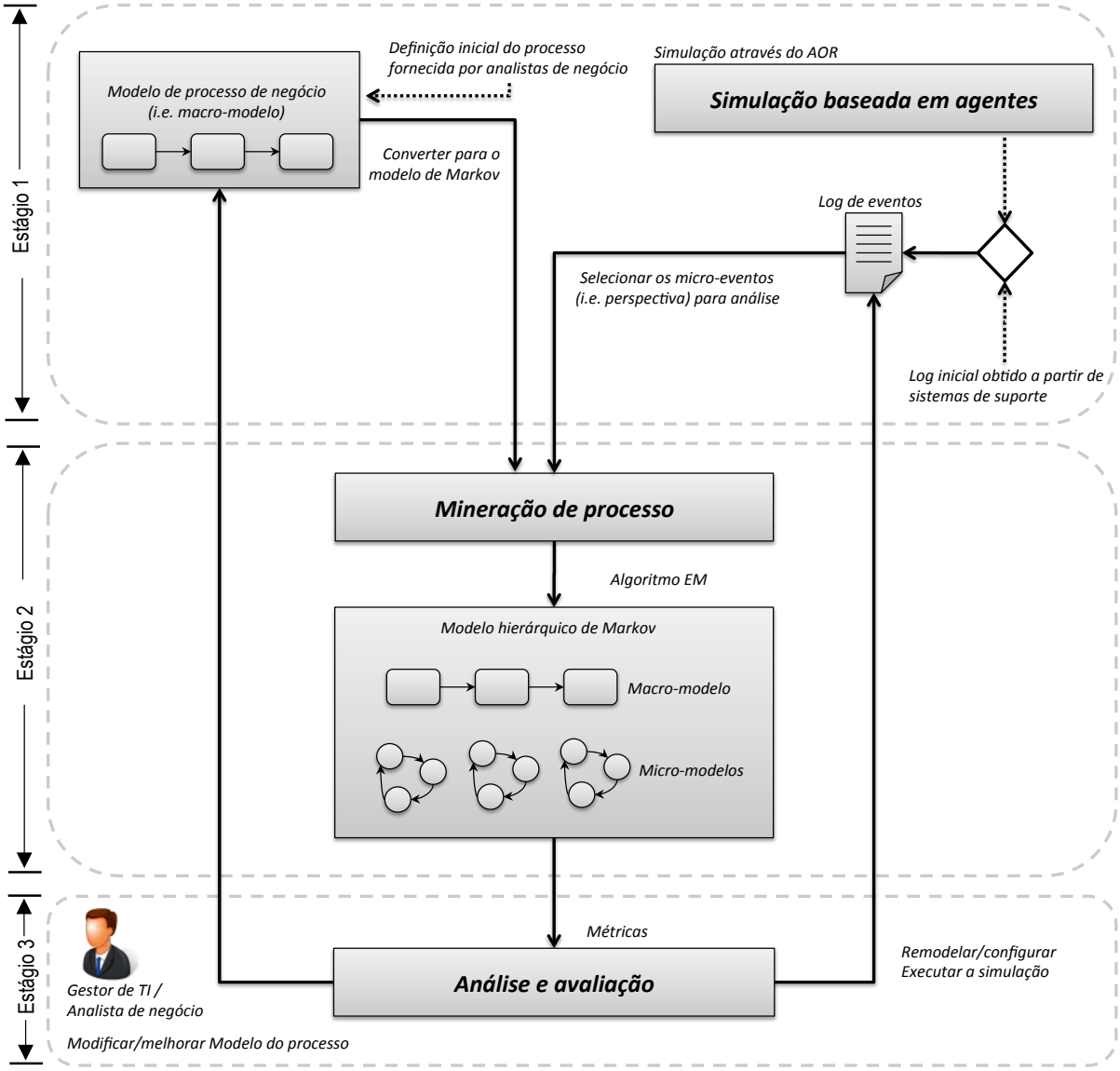


Figura 3.1: Ciclo de melhoria de modelos de processo de negócio

descrição de alto nível do processo de negócio e o log de eventos que fornece o comportamento na execução do processo. O modelo de alto nível do processo (i.e. o macro-modelo) pode ser fornecido por analistas de negócio. No entanto, o log de eventos pode ser obtido tanto por sistemas de suporte como através de simulação. Levando em consideração a dificuldade em se obter um log de eventos a partir de sistemas de suporte devido a diversos fatores como tempo, custo e tempo envolvido para se obter um número elevado de instâncias que representam o comportamento da execução do processo, neste trabalho em particular utilizamos um sistema de simulação baseado em agentes chamado AOR, que está descrito em detalhes na Seção 3.2.1.

A simulação baseada em agentes foi uma estratégia adotada no contexto deste trabalho para gerar os logs de eventos visando a melhoria no modelo de processo de negócio. Nesta direção, no Estágio 2 utilizamos o sistema AOR para essa finalidade, possibilitando assim

configurar os cenários de simulação que compreendem múltiplos agentes que interagem. Ao executar uma simulação, o AOR registra um log de eventos destas interações que serve como entrada para o Estágio 2 que trata a mineração de processo.

O Estágio 2 trata uma das principais questões abordadas neste trabalho, que é como mapear os eventos de baixo nível registrados em um log de eventos para as atividades de alto nível definidas em um modelo de processo. Neste trabalho, isto é realizado com o auxílio de uma técnica de mineração que é capaz de extrair um modelo hierárquico de Markov a partir de um log de eventos e de uma representação do modelo de Markov do modelo do processo de alto nível. A Seção 3.3 apresenta de forma intuitiva e formal o modelo hierárquico de Markov, que é a abordagem de mineração desenvolvida neste trabalho. Em conjunto com a descrição de alto nível do processo de negócio, o log de eventos é utilizado para extrair um modelo hierárquico para posteriormente ser objeto de análise e avaliação, a fim de verificar as possibilidades de alterações para melhoria do modelo (Estágio 3).

No Estágio 3, a partir do modelo hierárquico descoberto com a técnica de mineração, é possível avaliar se o modelo de processo de alto nível é realmente uma boa representação para o comportamento de baixo nível observado. Em particular, é possível medir a qualidade de tal modelo por meio de um conjunto de métricas. A utilização de métricas para avaliar a qualidade dos modelos de processos tem sido exaustivamente investigada na literatura (Mendling 2009, Vanderfeesten et al. 2007, Dijkman et al. 2011, Gruhn and Laue 2007), mas aqui nos concentramos em um conjunto de métricas que são mais adaptadas ao modelo hierárquico que está no ponto central da nossa abordagem. Especificamente, tal modelo hierárquico deve ser "equilibrado" no sentido de que todos os seus componentes – i.e. os micro-modelos e o macro-modelo – deverão ter uma complexidade semelhante, ao invés de ter alguns componentes que são mais complexos e outros extremamente simplificados. A Seção 3.4 apresenta as métricas definidas para avaliação da complexidade dos modelos hierárquicos obtidos a partir da abordagem de mineração.

Ainda no estágio 3, ao prosseguir com a análise e avaliação do modelo hierárquico através de um conjunto de métricas, o analista pode alterar a descrição de alto nível do processo, a fim de criar uma melhor representação do comportamento que ocorre na prática. No entanto, a alteração do processo de alto nível pode causar uma percepção diferente de como os eventos de baixo nível são mapeados para as atividades de alto nível. Por exemplo, se uma atividade é dividida em duas, ou se duas atividades são unidas, a relação entre os eventos de baixo nível e as atividades de alto nível irá mudar. Neste momento, podemos retornar ao Estágio 1 e este ciclo de melhoria baseado na simulação e mineração pode ser repetido até que se encontre um modelo de processo satisfatório. Ao contrário do que pode parecer à primeira vista, essa mudança não é inteiramente previsível, uma vez que os agentes se organizam de uma forma não-determinística para a execução do processo. Além disso, as técnicas de mineração de processo não fornecem uma precisão perfeita, para que se possa obter maiores conhecimentos sobre o comportamento do processo em tempo de execução pela experimentação de diferentes configurações para o modelo de processo.

Idealmente, uma nova versão do processo seria implantada na organização e seria capaz de coletar novos logs de eventos. Em seguida, pode-se executar o algoritmo de mineração de processo novamente, a fim de verificar quais foram as alterações introduzidas no comportamento em tempo de execução e se o novo processo é uma boa representação do

comportamento. Na prática, talvez não seja possível realizar tais experimentos no ambiente do mundo real, devido aos riscos, custos e tempo envolvidos. Por isso, introduzimos a possibilidade de realizar uma simulação baseada em agentes para o novo modelo de processo (Estágio 1). Esta simulação será configurada com o conhecimento que foi coletado até o momento sobre o processo.

Este processo iterativo de simulação e mineração pode facilitar a identificação e introdução de melhorias no modelo do processo, pois os micro-modelos (i.e. comportamento de baixo nível) com grande complexidade sugerem que sejam introduzidas outras atividades ou caminhos no macro-modelo (i.e. comportamento de alto nível).

3.2 Estágio 1: obtendo os dados de entrada

Atualmente, os processos e SI estão cada vez mais interligados às atividades organizacionais. Nesta direção, as organizações estão "projetando" o negócio em torno de processos de negócios. Portanto, há um interesse crescente nos SI, onde sua execução ocorre com base em um modelo de processo de negócio, chamados *Process-Aware Information Systems* (PAIS), como forma de fazer uma ponte entre as pessoas e software através da tecnologia de processo. ERP, WFM, CRM, entre outros podem ser utilizados para criar PAIS. Não apenas os sistemas WFM tem um mecanismo de processo; ERP modernos e CRM também oferecem componentes integrados de *workflow*. Este componente permite o projeto e melhoria de processos, i.e. processos estruturados que lidam com casos (e.g. solicitações de emprego, manipulação de sinistros, pedidos de clientes, viagens de negócios, entre outros).

A mineração de processos é motivada pelo fato de que muitos processos de negócios deixam seus "rastros" em SI transacionais (e.g WFM, ERP, CRM), i.e. eventos de negócios são registrados nos chamados logs de eventos. Até recentemente, as informações contidas nestes registros raramente eram utilizadas para analisar os processos subjacentes. A idéia básica da abordagem de melhoria proposta neste trabalho visa melhorar este procedimento, fornecendo uma abordagem integrada de simulação baseada em agentes, mineração e avaliação de processos, i.e. descobrir e melhorar os modelos de processos de negócio utilizando os logs de eventos gerados por SI transacionais.

Nesta direção, a abordagem de melhoria proposta recebe como entrada dois tipos de informação:

(1) Modelo de processo de alto nível – os processos de negócio são normalmente modelados por analistas de negócio em um alto nível de abstração, que no contexto deste trabalho é denominado de macro-modelo.

(2) Log de eventos – reflete a execução de um processo de negócio, que no contexto deste trabalho é denominada micro-sequência.

A primeira entrada pode ser fornecida por analistas de negócio ou obtida na documentação do processo. A segunda entrada pode ser obtida de duas formas:

(1) Sistemas transacionais – através do registro dos eventos de negócio. Este comportamento reflete a execução do SI baseado no processo de negócio.

(2) Simulação – no caso de não haver possibilidade de obter um log de eventos de SI transacionais, pode-se gerar o log de eventos via simulação. Particularmente, neste trabalho utilizamos o sistema AOR, que é uma plataforma de simulação baseada em agentes, onde os agentes operam sobre uma infraestrutura de sistemas ou em um ambiente possibilitando registrar suas interações na forma de eventos de comunicação. Tipicamente, cada evento se refere a uma operação que pode ser executada por algum agente durante a execução de uma instância do processo e estes eventos de baixo nível são registrados em um log de eventos.

A Seção 2.2 descreve de forma conceitual o processo de modelagem e simulação na plataforma AOR para obter o comportamento de baixo nível modelos de processo de negócio. No entanto, na Seção 3.2.1 vamos descrever de forma prática como declarar um cenário ou um modelo de simulação em AORSL.

3.2.1 O modelo de simulação no AOR

Na plataforma AOR existem basicamente dois tipos diferentes de eventos. O evento exógeno (*exogenous event*) que é um evento externo (como e.g. a chegada de um novo cliente) que não depende das ações dos agentes. Normalmente, é a ocorrência de um evento exógeno que dispara uma execução de simulação. Para executar múltiplas instâncias de um processo de negócio, o sistema AOR programa a ocorrência de eventos exógenos para disparar todo o processo em diferentes lapsos de tempo.

O segundo tipo de evento é uma mensagem (*message*), que é a base da simulação no sistema AOR. Agentes trocam mensagens entre si, por exemplo, se o agente X envia uma mensagem M1 para agente Y, então isso pode resultar em uma nova mensagem M2 sendo enviada a partir de Y para Z. Esse encadeamento de mensagens mantém a execução da simulação até que não haja mais mensagens a ser trocadas. Neste ponto, um novo evento exógeno é necessário para desencadear uma nova execução de simulação. Neste trabalho, nós representamos a troca de uma mensagem M sendo enviada do agente X para agente Y como:

$$\mathsf{X} \xrightarrow{\mathsf{M}} \mathsf{Y}$$

No sistema AOR, a especificação de um novo cenário de simulação começa pela definição de um conjunto de tipos de entidade (*entity types*). Estes tipos de entidade incluem os tipos de agentes, as mensagens e os eventos que serão utilizados no cenário. O comportamento dos agentes é especificado por meio de regras dos agentes (*agent rules*). Tipicamente, uma regra de um agente define quando uma determinada mensagem é recebida, uma outra mensagem é produzida e enviada para algum outro agente. Uma vez que as regras de cada agente são definidas separadamente, o cenário de simulação é efetivamente implementado de forma descentralizada pelo comportamento combinado de agentes.

Existem ainda no sistema AOR as regras de ambiente (*environment rules*). Basicamente, estas têm a ver com a ocorrência de eventos exógenos e elas definem o que deve ser feito quando esses eventos ocorrerem. Tipicamente, uma regra de ambiente especifica o que ocorre no ambiente quando um determinado evento ocorre, como por exemplo uma mensagem a ser enviada para algum agente. Enviando esta mensagem, em seguida, desencadeia uma regra do agente receptor, o que cria uma cadeia de troca de mensagens que coloca toda a simulação em movimento.

As regras de ambiente também tem a capacidade de criar e destruir os agentes. Isto é especialmente útil para simular, por exemplo, a chegada (ou saída) de novos clientes. Os agentes que são criados dinamicamente em tempo de execução deve ser de um certo tipo que já foi definido antes. Esses agentes também têm regras e participam na simulação através da troca de mensagens com outros agentes. Um conjunto de condições iniciais (*initial conditions*) para o cenário de simulação, especifica por exemplo quais agentes existem no início da simulação. As condições iniciais incluem a programação para a ocorrência de pelo menos um evento exógeno para disparar a simulação.

O AOR possibilita a configuração de parâmetros em um cenário de simulação. A Figura 3.2 ilustra uma captura de tela resultante da especificação AORSL no ambiente de simulação AOR. Na parte superior da Figura 3.2, o menu "*Build*" é utilizado para gerar o código Java a partir da especificação AORSL mostrada no painel do meio. O botão "*play*" na barra de ferramentas é usado para iniciar a simulação com os parâmetros que podem ser configurados no lado direito deste botão. A partir disso, o parâmetro mais importante é o número de passos de simulação (*simulation steps*), uma vez que controla o tempo de execução da simulação (em escala de tempo dos agentes). Os parâmetros restantes, iterações de simulação (*simulation iterations*) e atraso do intervalo de tempo (*step time delay*), são usados para executar a simulação múltiplas vezes e para inserir um atraso de tempo entre etapas consecutivas, respectivamente.

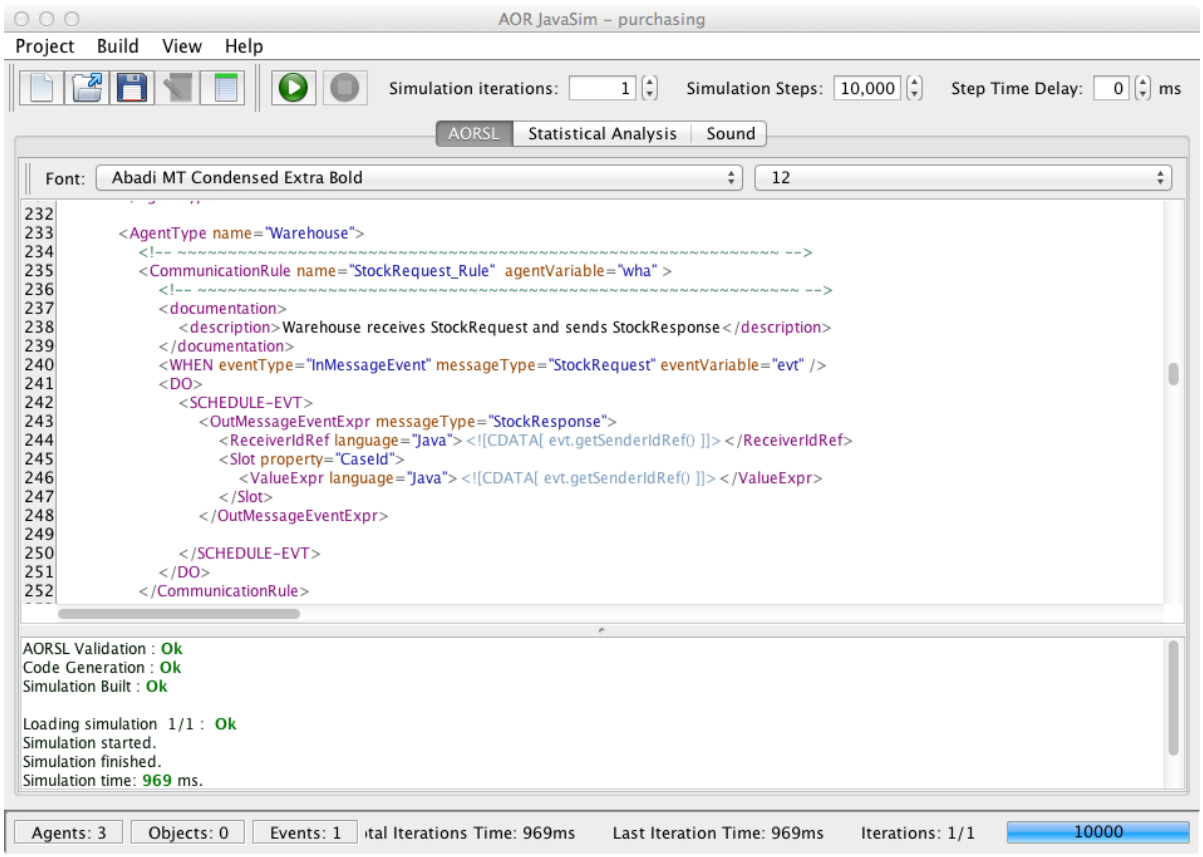


Figura 3.2: O Simulador AOR com um cenário de simulação

3.2.2 Log de eventos com o AOR

Quando um processo é simulado no AOR, um log de eventos é produzido utilizando uma estrutura específica do AOR no formato da linguagem XML. Basicamente, este log de eventos baseado em XML tem uma entrada para cada passo da simulação. Cada um desses registros de entrada contém: quais agentes estavam ativos naquele momento; quais *exogenous events* (se houver) foram recebidos pelo ambiente; quais *environment rules* foram ativadas por esses eventos; quais mensagens (se houver) foram recebidas por cada agente; quais *agent rules* foram ativadas por aquelas mensagens; e, finalmente, quais as mensagens que foram produzidas como resultado de ativação de qualquer destas regras.

No log de eventos, é possível reconhecer cada instância do processo associada com uma instância diferente do respectivo agente. Quando o sistema AOR cria um novo agente, ele atribui um número único para esse agente e este número é incrementado automaticamente para cada novo agente do mesmo tipo que está sendo criado. Os números são geralmente negativos para os agentes criados dinamicamente, a fim de distinguir os agentes pré-existentes que são normalmente atribuídos números positivos. Por exemplo, para agentes pré-definidos no cenário de simulação atribuímos números positivos (como o agente No.1, agente No.2, agente No.3, etc...) etc. Por outro lado, os agentes que são criados dinamicamente têm números negativos (como -1, -2, -3, etc...) e estes valores são utilizados como o identificador da instância para as instâncias correspondentes do processo.

O sistema AOR registra no log de eventos qual agente (No.) envia cada mensagem e qual agente a recebe. Uma vez que agentes dinâmicos não têm comunicação uns com os outros neste processo, sempre que uma mensagem está sendo enviada ou recebida a partir de um agente dinâmico, é possível descobrir imediatamente a instância do processo ao qual a mensagem pertence. Nas mensagens que são trocadas entre diferentes agentes dinâmicos, a nossa implementação especifica que essas mensagens devem levar uma propriedade chamada instância que contém o número do agente dinâmico correspondente a essa instância específica do processo. Desta forma, é possível determinar a instância do processo para cada mensagem.

O comportamento dos agentes na execução do cenário de simulação é a base para mineração de processos. No entanto, a técnica proposta neste trabalho para mineração de processos, o modelo hierárquico de Markov, utiliza como entrada um log de eventos no formato da Tabela 1.1. Assim, foi implementado na linguagem de programação Python[1] (versão 2.7) um conversor dos logs que analisa os eventos gerados pelo AOR no formato XML e os transforma em eventos no formato padrão de mineração.

Como resultado das atividades de conversão, temos um arquivo de log de eventos com dados correspondentes às instâncias de processos que teriam sido executados na organização, os *timestamps* dos eventos executados obtido através do passo de simulação, os agentes que executaram esses eventos, entre outros tipos de dados. Estas informações são o ponto de partida para a mineração de processos.

[1] http://www.python.org/

3.3 Estágio 2: mineração com o modelo hierárquico de Markov

Nesta seção introduzimos primeiramente de forma intuitiva e depois formal a técnica de mineração de processos denominada modelo hierárquico de Markov. O desenvolvimento e uso desta técnica faz parte da contribuição deste trabalho que está sendo utilizada no contexto da melhoria de modelos de processos de negócio. Nesta direção, serão apresentados os problemas em questão e os algoritmos que foram desenvolvidos para capturar o macro comportamento do processo de negócio, o micro comportamento encontrado no log de eventos e a relação entre os dois. Ao final, ilustramos o funcionamento do algoritmo proposto através de um exemplo de execução.

3.3.1 Apresentação do modelo

Um modelo hierárquico de Markov é uma cadeia de Markov onde cada estado contém outra cadeia de Markov. Enquanto a cadeia de Markov de alto nível está em um certo estado (estado "macro"), a cadeia de Markov de baixo nível para cada estado macro pode ser a mudança entre estados "micro". Um modelo hierárquico de Markov pode ter um número arbitrário de camadas, mas para nossso propósito é suficiente utilizar um modelo somente com duas camadas: uma para representar o processo de alto nível e outra para representar o comportamento de baixo nível associado com cada atividade de alto nível.

Para ilustrar um exemplo simples do modelo hierárquico de Markov proposto, vamos considerar um processo de negócio que pode ser descrito em alto nível que compreende a sequência de atividades A, B, e C. Esta sequência de atividades representa o modelo de controle de fluxo para o processo de alto nível que está ilustrado na Figura 3.3. Por outro lado, vamos considerar que quando cada uma destas atividades são executadas, isto resulta em alguma sequência de eventos de baixo nível sendo registradas no log de eventos. Vamos nos referir a estes eventos de baixo nível como X, Y e Z. Estes eventos de baixo nível podem representar uma de várias coisas diferentes: por exemplo, eles podem representar um conjunto de tarefas de baixo nível sendo executada por agentes; podem representar os próprios agentes; ou podem representar um conjunto de mensagens de baixo nível trocadas entre os agentes.

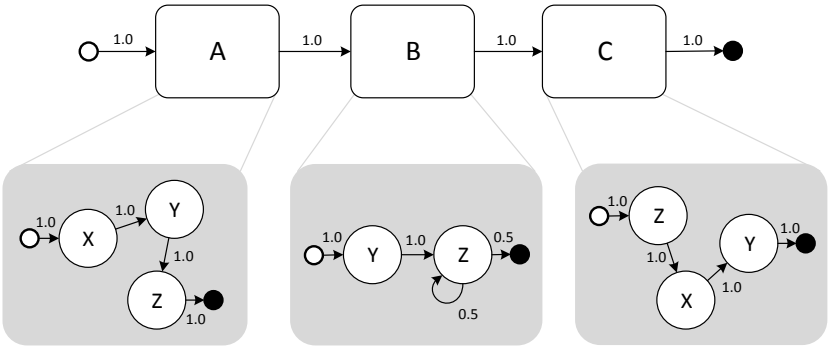


Figura 3.3: Um simples exemplo de um modelo de processo hierárquico

Em um modelo hierárquico de Markov, tal como o descrito na Figura 3.3, a cadeia de Markov na camada superior capta o controle de fluxo do processo de negócio em termos das transições possíveis entre um conjunto de atividades de alto nível. Esta cadeia de Markov é conhecida como o *macro-modelo*. Por outro lado, as cadeias Markov na camada inferior capturam a sequência de eventos de baixo nível, obtidos durante a execução de cada uma das atividades de alto nível. Estas Cadeias de Markov são referenciadas como os *micro-modelos*.

Cada uma dessas cadeias de Markov é capaz de passar por uma sequência de estados. No macro-modelo, os estados são atividades de alto nível, enquanto que nos micro-modelos os estados são eventos de baixo nível. No exemplo específico da fig 3.3, o macro modelo pode gerar apenas uma única sequência (i.e. ABC). Isso é conhecido como a *macro-sequência*. Pode-se imaginar que, se houvesse outras transições com probabilidade diferente de zero no macro modelo, poderia haver outras macro-sequências possíveis.

Toda vez que o processo de negócio é instanciado, isso é equivalente a geração de uma nova macro-sequência do macro-modelo. No entanto, esta macro-sequência não pode ser observada diretamente, já que o log de eventos registra os eventos de baixo nível, mas não é a atividade real que produziu esses eventos. Portanto, mesmo no caso em que apenas uma única macro-sequência é possível, como no exemplo da Figura 3.3, ainda é incerto qual atividade produziu cada evento. Ou seja, a relação entre a sequência de eventos XYZYZZZXY e as atividades na macro-sequência ABC não é clara. A sequência de eventos é conhecida como a *micro-sequencia*.

Uma inspeção na Figura 3.3 é suficiente para concluir que o mapeamento entre macro-sequência e micro-sequência pode ser como se segue:

AAABBBCCC
| | | | | | | | |
XYZYZZZXY

Acima, a sequência AAABBBCCC é uma *expansão* da macro-sequência ABC para o mesmo tamanho da micro-sequência XYZYZZZXY. Tal expansão fornece um mapeamento de cada evento para uma atividade no macro-modelo. Para simplificar, a partir deste ponto em diante, vamos nos referir a expansão da macro-sequência (i.e. AAABBBCCC) como sendo a atual macro-sequência que estamos procurando.

Seja qual for o significado dos eventos de baixo nível X, Y e Z, consideramos que a atividade A resulta na sequência de eventos XYZ. De forma semelhante, a atividade B resulta em uma sequência de eventos no formato YZZ..., onde pode haver múltiplos Z's até que uma certa condição se torne verdadeira. Finalmente, a atividade C resulta em uma sequência de eventos no formato ZXY. Estas sequências de eventos são representadas na Figura 3.3 como cadeias de Markov de baixo nível.

Executando esse modelo corresponde à execução da sequência de atividades ABC. No entanto, no log de eventos encontramos *traces* como XYZYZZZXY sem ter nenhuma idéia de como essa sequência de eventos pode ser mapeada para a sequência de atividades ABC. A sequência ABC será chamada de *macro-sequência* e o modelo de alto nível para o processo de negócio em termos das atividades A, B e C é referenciado como o *macro-modelo*. Por outro lado, a sequência de eventos XYZYZZZXY será chamada como *micro-sequência*, e a cadeia de Markov de baixo nível que descreve o comportamento de cada macro-atividade, em termos dos eventos X, Y e Z é referenciado como um *micro-modelo*.

Em geral, o macro-modelo pode ser derivado a partir de uma descrição do processo de negócio, mas a macro-sequência real que é produzida por uma instanciação do processo é desconhecida, porque esta sequência não pode ser observada. Isto é especialmente problemático se o modelo de processo permite vários caminhos diferentes de execução. Em contraste, uma micro-sequência pode ser recuperada (como um *trace*) a partir do registro do evento, mas não está claro como tais micro-sequências devem ser divididas de acordo com as atividades de alto nível do processo (i.e. o macro-modelo). Isso ocorre porque os micro-modelos que descrevem o comportamento dos agentes dentro de cada atividade de alto nível são desconhecidos. No entanto, este comportamento pode ser extraído de log de eventos.

O problema abordado é *como descobrir a macro-sequência e os micro-modelos com maior probabilidade a partir de um determinado macro-modelo e micro-sequência.* Em outras palavras:

- As entradas são o macro-modelo e a micro-sequência. O macro-modelo representa o conhecimento prévio sobre o processo de negócio em termos de um conjunto de atividades de alto nível; é um macro-modelo de controle de fluxo expresso como uma cadeia de Markov. Por outro lado, a micro-sequência representa um *trace* de baixo nível que pode ser observado em um log de eventos. Se houver várias instâncias do processo, haverá várias micro-sequências no log de eventos.
- As saídas são a macro-sequências e os micro-modelos. A macro-sequência representa o caminho atual no macro-modelo (i.e. a sequência de macro-estados) que explica como uma dada micro-sequência foi gerada durante a execução. Se existem múltiplas instâncias do processo, haverão várias micro-sequências e haverá uma macro-sequência separada para cada micro-sequência. Por outro lado, os micro-modelos representam o comportamento de eventos de baixo nível para cada atividade de nível macro. Cada micro-modelo é expressado como uma cadeia de Markov.

Para tornar mais claro, o problema está ilustrado na Figura 3.4.

É importante notar que, se os micro-modelos são conhecidos, então a macro-sequência pode ser facilmente derivada, como no mapeamento acima. Por outro lado, se a macro-sequência é conhecida, então a partir da macro-sequência e micro-sequência será possível derivar os micro-modelos. Estas podem ser consideradas como sendo dois diferentes sub-problemas. O que faz com que este problema de mineração de processos seja especialmente difícil de resolver é que tanto a macro-sequência e micro-modelos são, simultaneamente, desconhecidos. No entanto, na Seção 3.3.2 veremos que é possível conceber uma solução para este problema com base em uma solução para esses dois sub-problemas.

Definições

Seja $\mathbf{S}$ o conjunto de estados possíveis em uma cadeia de Markov e sejam i e j quaisquer dois estados desses. Então $\mathbb{P}(j \mid i)$ é a probabilidade que o próximo estado será j dado que o estado atual é i. Para conveniência, isto será referenciado como a probabilidade de transição a partir de um estado corrente i para um estado subsequente j. Neste trabalho, como em (Veiga and Ferreira 2010), estendemos o conjunto $\mathbf{S}$ com dois estados especiais – o estado inicial ($\circ$) e o estado final ($\bullet$) – de modo a incluir a probabilidade inicial e final

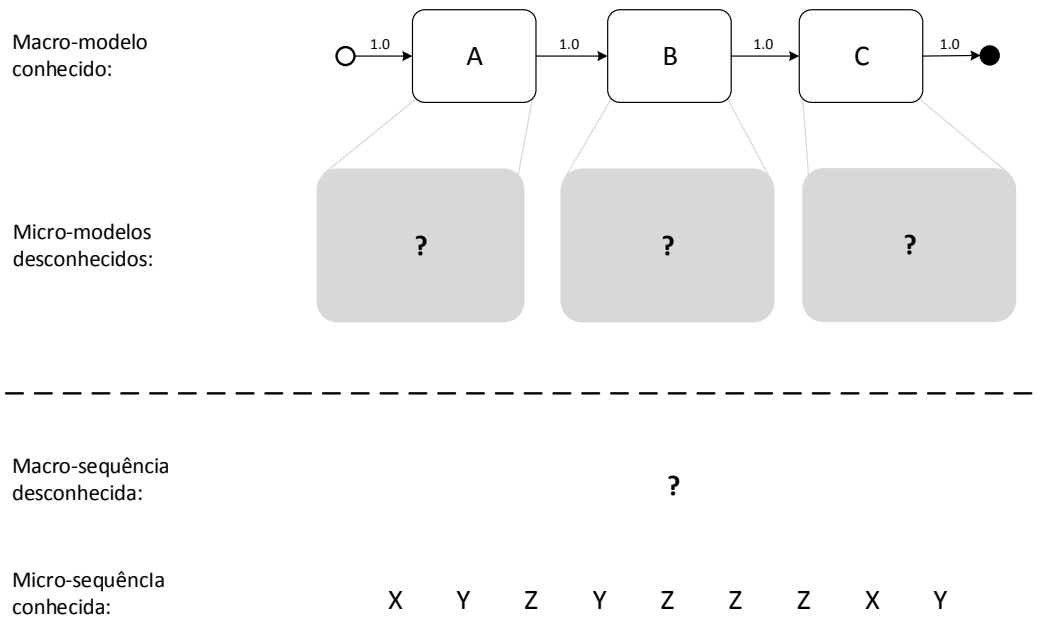


Figura 3.4: O problema de minerar um modelo hierárquico de Markov

da cadeia de Markov em determinado estado. Nós representamos esse conjunto ampliado de estados como $\overline{\mathbf{S}} = \mathbf{S} \cup \{\circ, \bullet\}$. Por exemplo, $\mathbb{P}(i \mid \circ)$ é a probabilidade da cadeia de Markov iniciar no estado i. Similarmente, $\mathbb{P}(\bullet \mid i)$ é a probabilidade da cadeia de Markov terminar no estado i.

Por definição, $\mathbb{P}(\circ \mid i) \triangleq 0, \forall_{i \in \overline{\mathbf{S}}}$ uma vez que nada pode vir antes do estado inicial. Da mesma forma, $\mathbb{P}(i \mid \bullet) \triangleq 0, \forall_{i \in \overline{\mathbf{S}}}$ uma vez que nada pode vir depois do estado final. Também, $\mathbb{P}(\bullet \mid \circ) \triangleq 0$ uma vez que a cadeia de Markov não pode começar e terminar imediatamente, sem passar por qualquer estado observável.

Uma cadeia de Markov é representada por uma matriz $\mathbf{T} = \{p_{ij}\}$ de probabilidades de transição, onde $p_{ij} = \mathbb{P}(j \mid i), \forall_{i,j \in \overline{\mathbf{S}}}$. Mais formalmente, uma cadeia de Markov $\mathcal{M} = \langle \overline{\mathbf{S}}, \mathbf{T} \rangle$ é definida por uma tupla onde $\overline{\mathbf{S}}$ é o conjunto ampliado de estados e $\mathbf{T}$ é a matriz de probabilidade de transição entre os estados.. A cadeia de Markov está sujeita à restrição estocástica $\sum_{j \in \overline{\mathbf{S}}} \mathbb{P}(j \mid i) = 1$ para todos estados $i \in \overline{\mathbf{S}} \setminus \{\bullet\}$. Em outras palavras, existe sempre algum estado subsequente para o estado corrente i, exceto quando o estado final foi alcançado. Para o caso particular do estado final, temos $\sum_{j \in \overline{\mathbf{S}}} \mathbb{P}(j \mid \bullet) = 0$.

O fato que $\forall_{j \in \overline{\mathbf{S}}} : \mathbb{P}(j \mid \bullet) = 0$ significa que a última linha da matriz $\mathbf{T}$ é zero. Também o fato que $\forall_{i \in \overline{\mathbf{S}}} : \mathbb{P}(\circ \mid i) = 0$ significa que a primeira coluna da matriz $\mathbf{T}$ é zero. Finalmente, o fato que $\mathbb{P}(\bullet \mid \circ) = 0$ significa que o último elemento na primeira linha da matriz é zero. Estes fatos são ilustrados na Figura 3.5, nos elementos marcados como $\mathbf{0}$.

Em um modelo hierárquico de Markov, há uma cadeia de Markov para descrever o macro-modelo (nível mais alto na Figura 3.3) e há um conjunto de cadeias de Markov para descrever o micro-modelo para cada atividade (nível mais baixo na Figura 3.3).

O macro-modelo é definido como uma cadeia de Markov $\mathcal{M}' = \langle \overline{\mathbf{S}'}, \mathbf{T}' \rangle$ onde $\mathbf{S}'$ é o conjunto de estados que representam as atividades na descrição alto nível do processo de negócio e o $\mathbf{T}'$ é a matriz de probabilidade de transição entre os estados. Por outro lado,

$\mathbf{T} =$

	$\circ$	1	2	...	n	$\bullet$	
$\circ$	$\mathbf{0}$	p_{01}	p_{02}	...	p_{0n}	$\mathbf{0}$	$(\Sigma_j p_{0j} = 1)$
1	$\mathbf{0}$	p_{11}	p_{12}	...	p_{1n}	$p_{1(n+1)}$	$(\Sigma_j p_{1j} = 1)$
2	$\mathbf{0}$	p_{21}	p_{22}	...	p_{2n}	$p_{2(n+1)}$	$(\Sigma_j p_{2j} = 1)$
...	...	...	...	...	...	...	...
n	$\mathbf{0}$	p_{n1}	p_{n2}	...	p_{nn}	$p_{n(n+1)}$	$(\Sigma_j p_{nj} = 1)$
$\bullet$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	...	$\mathbf{0}$	$\mathbf{0}$	$(\Sigma_j p_{(n+1)j} = 0)$

Figura 3.5: Forma geral da matriz de transição

$\mathcal{M}' = \langle \overline{\mathbf{S}'}, \mathbf{T}' \rangle \qquad \overline{\mathbf{S}'} = \{\circ, \mathsf{A}, \mathsf{B}, \mathsf{C}, \bullet\} \qquad \mathbf{T}' =$

	$\circ$	A	B	C	$\bullet$
$\circ$	0	1	0	0	0
A	0	0	1	0	0
B	0	0	0	1	0
C	0	0	0	0	1
$\bullet$	0	0	0	0	0

$\mathcal{M}''_{\mathsf{A}} = \langle \overline{\mathbf{S}''_{\mathsf{A}}}, \mathbf{T}''_{\mathsf{A}} \rangle \qquad \mathcal{M}''_{\mathsf{B}} = \langle \overline{\mathbf{S}''_{\mathsf{B}}}, \mathbf{T}''_{\mathsf{B}} \rangle \qquad \mathcal{M}''_{\mathsf{C}} = \langle \overline{\mathbf{S}''_{\mathsf{C}}}, \mathbf{T}''_{\mathsf{C}} \rangle$

$\overline{\mathbf{S}''_{\mathsf{A}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \overline{\mathbf{S}''_{\mathsf{B}}} = \{\circ, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \overline{\mathbf{S}''_{\mathsf{C}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\}$

$\mathbf{T}''_{\mathsf{A}} =$

	$\circ$	X	Y	Z	$\bullet$
$\circ$	0	1	0	0	0
X	0	0	1	0	0
Y	0	0	0	1	0
Z	0	0	0	0	1
$\bullet$	0	0	0	0	0

$\mathbf{T}''_{\mathsf{B}} =$

	$\circ$	Y	Z	$\bullet$
$\circ$	0	1	0	0
Y	0	0	1	0
Z	0	0	$\frac{1}{2}$	$\frac{1}{2}$
$\bullet$	0	0	0	0

$\mathbf{T}''_{\mathsf{C}} =$

	$\circ$	X	Y	Z	$\bullet$
$\circ$	0	0	0	1	0
X	0	0	1	0	0
Y	0	0	0	0	1
Z	0	1	0	0	0
$\bullet$	0	0	0	0	0

Figura 3.6: Um exemplo do modelo hierárquico de Markov

os micro-modelos são definidos com um conjunto de cadeias de Markov $\{\mathcal{M}''_i : i \in \mathbf{S}'\}$ onde cada $\mathcal{M}''_i = \langle \overline{\mathbf{S}''_i}, \mathbf{T}''_i \rangle$ é uma cadeia de Markov que descreve o comportamento dos agentes quando executam atividades $i \in \mathbf{S}'$.

Para o exemplo da Figura 3.3, um possível modelo é mostrado na Figura 3.6. Nesta figura, o macro-modelo é denotado por$\mathcal{M}'$ e os micro-modelos são denotados por $\mathcal{M}''_{\mathsf{A}}$, $\mathcal{M}''_{\mathsf{B}}$ e $\mathcal{M}''_{\mathsf{C}}$, respectivamente. Em particular, em $\mathbf{T}''_{\mathsf{B}}$ presume-se que a probabilidade de ir do estado Z para o mesmo estado Z é igual para a probabilidade de terminação da cadeia de Markov neste estado, uma vez que ambos são $\frac{1}{2}$.

Semântica de Execução

Considerando o exemplo da Figura 3.3, a semântica de execução para o modelo hierárquico de Markov é descrita da seguinte forma:

(a) Executar o macro-modelo $\mathcal{M}' = (\overline{\mathbf{S}'}, \mathbf{T}')$ como cadeia de Markov, iniciando com o estado inicial ($\circ$) e indo através da mesma sequência de estados de acordo com as probabilidades de transição em $\mathbf{T}'$, até o estado final ($\bullet$) ser alcançado.

(b) Para cada estado i que o macro-modelo $\mathcal{M}'$ entra, executar o micro-modelo correspondente $\mathcal{M}''_i$ como uma cadeia de Markov, novamente iniciando com o estado incial ($\circ$) e indo atravéas da mesma sequência de estados de acordo com as probabilidades de transição em $\mathbf{T}''_i$, até o estado final ($\bullet$) ser alcançado. Apenas assim o macro-modelo pode avançar para o próximo estado.

(c) A *micro-sequencia* $\mathbf{s}''$ é obtida pela concatenação de todos os estados observados no nível micro. Um exemplo é $\mathbf{s}'' = \mathsf{XYZYZZZXY}$. Claramente, cada um desses micro-estados pertencem a algum macro-estado, no sentido de que cada micro-estado foi produzido por algum micro-modelo associado com um macro-estado. Em $\mathbf{s}'' = \mathsf{XYZYZZZXY}$, o primeiro micro-estado XYZ pertence a A; o meio YZZ pertence a B; e o último ZXY pertence a C. Seja $\mathbf{s}'$ a *macro-sequencia* definida como uma sequência de estados que o macro-modelo estava no momento em que cada micro-estado foi gerado. Então $\mathbf{s}' = \mathsf{AAABBBCCC}$.

Nosso objetivo é encontrar a macro-sequência $\mathbf{s}'$ e o conjunto de micro-modelos $\{\mathcal{M}''_i\}$ para cada macro-estado $i \in \mathbf{S}'$. Para este propósito, somente a micro-sequência $\mathbf{s}''$ e o macro-modelo $\mathcal{M}'$ são conhecidos. Note que conhecer somente o macro-modelo $\mathcal{M}'$ não resolve o problema, já que em geral, o macro-modelo pode gerar várias macro-sequências possíveis e não se sabe qual macro-sequência realmente ocorreu para uma determinada micro-sequência. Por outro lado, conhecer somente a micro-sequência $\mathbf{s}''$ não resolve o problema, uma vez que não existe nenhuma idéia sobre como a micro-sequência observada deve ser particionada em um conjunto de macro-atividades. Um algoritmo para encontrar e estimar para $\mathbf{s}'$ e $\{\mathcal{M}''_i\}$ a partir de $\mathcal{M}'$ e $\mathbf{s}''$ é desenvolvido na próxima seção.

3.3.2 Algoritmos de mineração

O problema apresentado na Seção 3.3.1 é equivalente ao de encontrar parâmetros desconhecidos $\{\mathcal{M}''_i\}$ para um modelo que produz tanto os dados observados ($\mathbf{s}''$) como os dados não observados ($\mathbf{s}'$). Tal tipo de problema se encaixa bem no EM *framework* (Dempster et al. 1977, Moon 1996, McLachlan and Krishnan 2008), que foi descrito na Seção 2.3.5. Neste trabalho, se os dados ausentes $\mathbf{s}'$ são conhecidos (neste caso, os micro-modelos), então seria possível calcular $\{\mathcal{M}''_i\}$ diretamente a partir de $\mathbf{s}'$ e $\mathbf{s}''$. Por outro lado, se os parâmetros do modelo $\{\mathcal{M}''_i\}$ são conhecidos, então seria possível determinar os dados ausentes $\mathbf{s}'$. O que torna o problema particularmente difícil é o fato de que tanto o $\{\mathcal{M}''_i\}$ e $\mathbf{s}'$ não são conhecidos.

Para este tipo de problema (ilustrado na Figura 3.4), é possível desenvolver um procedimento utilizando a técnica EM da seguinte forma:

(a) Obter, por algum meio, a estimativa inicial para os dados não observados $\mathbf{s}'$. Podemos gerar randomicamente a macro-sequência definindo uma sequência randômica a partir do macro-modelo e expandindo para o comprimento da micro-sequência. Por exemplo, para o macro-modelo da Figura 3.4 há somente uma sequência possível (i.e. ABC)

e esta sequência pode ser expandida de diferentes formas, tal como AAABBBCCC, AAAABBBBC ou ABCCCCCC.

(b) Com a estimativa atual para os dados ausentes $\mathbf{s}'$, obter uma melhor estimativa para os parâmetros do modelo desconhecido $\{\mathcal{M}_i''\}$, i.e usando a macro-sequência corrente, estimar os micro-modelos de acordo com o procedimento descrito na Seção 2.

(c) Com a estimativa corrente para os parâmetros do modelo, obter uma estimativa melhorada dos dados ausentes $\mathbf{s}'$, i.e usando a estimativa corrente para os micro-modelos, determinar a sequência mais provável para a macro-sequência de acordo com o procedimento descrito pelo Algoritmo 3.

(d) Repita a sequência de passos (b) e (c) até que os dados ausentes $\mathbf{s}'$ e os parâmetros do modelo convirjam $\{\mathcal{M}_i''\}$, i.e até a macro-sequência não mudar mais. Neste ponto, os micro-modelos também não mudarão mais. A macro-sequência e os micro-modelos assim obtidos são a solução para o problema de mineração.

O Algoritmo 1 descreve uma adaptação do procedimento EM descrito para resolver o nosso problema principal. Iniciamos pela randomização da macro-sequência $\mathbf{s}'$ (Passo 1) e, em seguida, usamos essa sequência para obter uma estimativa para os micro-modelos $\{\mathcal{M}_i''\}$ (Passo 2). Depois disso, usamos a estimativa atual de $\{\mathcal{M}_i''\}$ para obter uma estimativa melhor para $\mathbf{s}'$ (Passo 3), e, em seguida, usamos esse $\mathbf{s}'$ para obter uma estimativa melhor para $\{\mathcal{M}_i''\}$ (Passo 2), e assim por diante, até ambas as estimativas convergirem.

Algoritmo 1 Estimar os micro-modelos $\{\mathcal{M}_i''\}$ e a macro-sequência $\mathbf{s}'$ a partir do macro-modelo $\mathcal{M}'$ e a micro-sequência $\mathbf{s}''$

---

1. Obter uma sequência randômica $\tilde{\mathbf{s}}$ a partir da cadeia de Markov $\mathcal{M}'$ e use esta sequência como base para construir a macro-sequência $\mathbf{s}'$ com o mesmo tamanho de $\mathbf{s}''$ (por exemplo, se $\tilde{\mathbf{s}} = \mathsf{ABC}$ e $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ então $\mathbf{s}' = \mathsf{AAABBBCCC}$)

2. Dada a micro-sequencia $\mathbf{s}''$, o macro-modelo $\mathcal{M}'$ e a estimativa corrente para $\mathbf{s}'$, encontrar e estimar $\{\mathcal{M}_i''\}$ (ver Algoritmo 2)

3. Dada a micro-sequencia $\mathbf{s}''$, o macro-modelo $\mathcal{M}'$ e a estimativa corrente para $\{\mathcal{M}_i''\}$, encontrar e estimar $\mathbf{s}'$ (ver Algoritmo 3)

4. Volte ao passo 2 e repita a partir de lá até as estimativas para $\mathbf{s}'$ e $\{\mathcal{M}_i''\}$ convergirem.

O problema agora é como realizar os Passos 2 e 3 do Algoritmo 1. A solução para esses sub-problemas é descrito ao longo desta seção.

Expansão da sequência

No exemplo do Passo 1 do Algoritmo 1, a sequência randômica $\tilde{\mathbf{s}} = \mathsf{ABC}$ extraída do macro-modelo $\mathcal{M}'$ foi expandida para $\mathbf{s}' = \mathsf{AAABBBCCC}$, que inclui um número igual de A's, B's e C's. Nada estabelece que $\tilde{\mathbf{s}}$ deve ser expandido desta forma. Outras expansões são possíveis (e.g. $\mathbf{s}' = \mathsf{AAAABBBBC}$ ou $\mathbf{s}' = \mathsf{ABCCCCCCC}$), enquanto a expansão em

conformidade com a dada sequência de macro-estados (i.e. $\tilde{\mathbf{s}} = \mathsf{ABC}$). Neste trabalho expandimos a sequência $\tilde{\mathbf{s}}$ selecionando repetidamente um estado de forma randômica a partir da sequência e inserindo um símbolo igual ao lado, até que o comprimento da sequência atinge o mesmo comprimento que a dada micro-sequência. Por exemplo, para a sequência $\tilde{\mathbf{s}} = \mathsf{ABC}$, nós escolhemos um número randômico de A's, B's e C's para expandir esta sequência em uma macro-sequência com o mesmo comprimento que $\mathbf{s}'' = \mathsf{XYZYZZZXY}$. O procedimento geral pode ser descrito da seguinte forma:

- Para uma micro-sequência $\mathbf{s}''$ de tamanho $m = |\mathbf{s}''|$ e uma sequência de macro-estados $\tilde{\mathbf{s}}$ de tamanho $n = |\tilde{\mathbf{s}}|$, obter uma sequência de números randômicos $\mathbf{r} = \langle r_1, ..., r_n \rangle$ tal que a sua soma total seja igual ao comprimento de $\mathbf{s}''$ (i.e. $\sum_{i=1}^{n} r_i = m$).
- Montar a macro-sequência $\mathbf{s}'$ a partir de $\tilde{\mathbf{s}}$ e $\mathbf{r} = \langle r_1, ..., r_n \rangle$ concatenando r_1 cópias de $\tilde{\mathbf{s}}[1]$, com r_2 cópias de $\tilde{\mathbf{s}}[2]$, com r_3 cópias $\tilde{\mathbf{s}}[3]$, e assim por diante.

Encontrando $\{\mathcal{M}_i''\}$ quando $\mathbf{s}'$ é conhecido

Se a macro-sequência é conhecida, então é possível identificar quais eventos na micro-sequência pertencem a quais atividades no macro-modelo. Então, a partir da sequência dos eventos que pertencem a cada atividade, é possível computar o micro-modelo associado a cada atividade. A Figura 3.7 ilustra como isto pode ser feito.

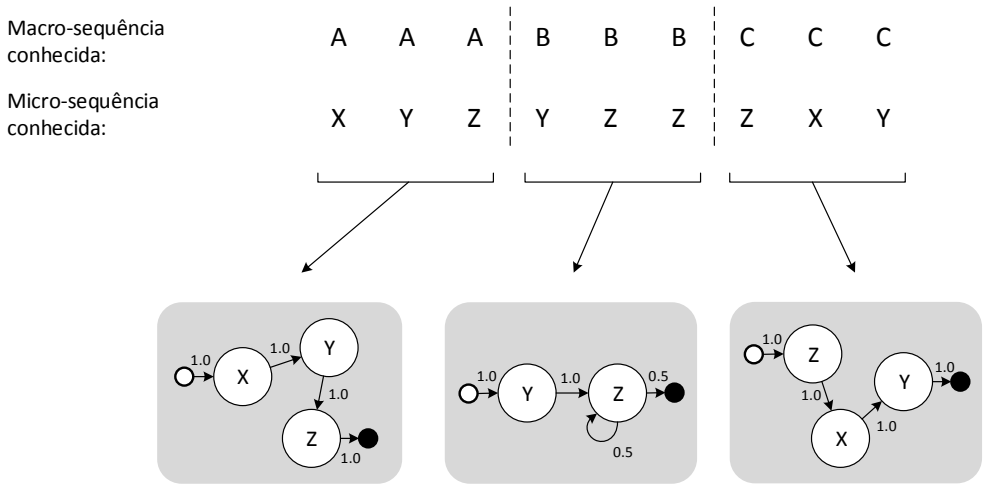


Figura 3.7: Estimando os micro-modelos a partir de uma macro-sequência

A partir da sub-sequência de eventos que pertencem a cada uma das atividades, é possível calcular uma cadeia de Markov, como mostrado na Figura 3.7. Por exemplo, considere a sequência de eventos YZZ produzida pela atividade B. A sequência começa com Y (nesse caso a transição do estado inicial "$\circ$" para o estado Y na cadeia de Markov) e é seguido por Z (por isso a transição do estado Y para Z). Agora, há dois Z's na sequência. Em um caso (o primeiro Z) que é seguido por um outro Z, ao passo que no outro caso (o segundo Z) ele termina a sequência. Isto significa que a partir do estado Z é possível ir para qualquer Z (um auto-loop) ou para o fim da sequência ("$\bullet$").

No exemplo da Figura 3.7 existe uma macro-sequência única e uma única micro-sequência e não há nenhuma informação adicional sobre o comportamento da atividade B. Por isso, assumimos que B sempre produz a sequência de eventos YZZ. Partindo deste pressuposto, podemos atribuir as seguintes probabilidades de transição:

- A sequência começa sempre com um Y (probabilidade de transição 1.0), que é seguida por um Z (mais uma vez, com probabilidade de transição 1.0).
- Sempre que um Z surge, uma das duas coisas podem acontecer: ou Z é seguido por outro Z (o que acontece uma vez em cada dois casos, resultando em uma probabilidade de transição 0.5) ou Z termina a sequência (o outro 0.5).

Mais formalmente, supomos que a macro-sequência $\mathbf{s}'$ é conhecida, por exemplo $\mathbf{s}'$= AAABBBCCC. Então, o que resta a descobrir é $\mathcal{M}_i''$ para todos os estados $i \in \mathbf{S}'$. Isto é descrito no Algoritmo 2. Basicamente, se considera as transições que ocorrem na micro-sequência $\mathbf{s}''$ dentro de cada estado na macro-sequência $\mathbf{s}'$. Para $\mathbf{s}''$= XYZYZZZXY e $\mathbf{s}'$= AAABBBCCC, temos o seguinte mapeamento entre os micro-estados e macro-estados:

XYZYZZZXY
| | | | | | | | |
AAABBBCCC

O Algoritmo 2 começa com a busca de substrings $subs(i)$ para cada macro-estado i. Por exemplo, a substring para o estado A é ∘XYZ•; a substring para o estado B é ∘YZZ•; e a substring para o estado C e ∘ZXY•. (Note que se o estado A aparecer novamente em $\mathbf{s}'$ então uma segunda substring estaria associada com A, e similarmente para outros estados.) A partir do conjunto de substrings associadas com cada macro-estado, o Algoritmo 2 conta o número de transições (passo 2b) e, após a normalização (passo 2c), o resultado origina $\mathcal{M}_i''$.

Este procedimento pode ser facilmente estendido para o caso quando há várias micro-sequências (i.e. várias instâncias do processo de negócio, cada uma correspondendo a um *trace* separado no log de eventos). Basicamente, cada micro-sequência terá a sua própria macro-sequência, e para as dadas micro-sequências e macro-sequências é possível recuperar um conjunto de sub-sequências para cada atividade macro-nível. Para cada evento nestas subsequências, é possível contar as transições possíveis para outros eventos e depois normalizar para obter as probabilidades de transição do micro-modelo.

Encontrando s′ quando $\{\mathcal{M}_i''\}$ é conhecido

O problema de encontrar o macro-sequência para uma determinada micro-sequência quando os micro-modelos são conhecidos é mais difícil, porque pode haver várias possíveis macro-sequências, ou mesmo nenhuma. Se existem várias macro-sequências possíveis, em seguida, estamos interessados em encontrar a que é *mais provável*, no sentido de maximizar a probabilidade de ocorrência da micro-sequência observada.

A Figura 3.8 apresenta como a macro-sequência mais provável para a micro-sequência XYZYZZZXY pode ser determinada se os micro-modelos são conhecidos. É importante notar que, para determinar a macro-sequência, é necessário considerar as transições que são permitidas nos micro-modelos, bem como as transições permitidas no macro-modelo:

Algoritmo 2 Estimar os micro-modelos $\{\mathcal{M}_i''\}$ a partir da micro-sequência $\mathbf{s}''$ e da macro-sequência $\mathbf{s}'$

1. Separar a micro-sequência $\mathbf{s}''$ em um conjunto de substrings correspondentes aos diferentes macro-estados $\mathbf{s}'$. Seja $\mathbf{s}''[n_1 \colon n_2]$ uma substring de $\mathbf{s}''$ a partir da posição n_1 para a posição n_2. Então, para $\mathbf{s}'$ na forma,

$$\mathbf{s}' = \underbrace{aa...a}_{n_a}\underbrace{bb...b}_{n_b}...\underbrace{cc...c}_{n_c}$$

   escolher o primeiro elemento na micro-sequência $\mathbf{s}''$ e criar uma substring ($\mathbf{s}''[1 \colon n_a]$) associada com o estado a, escolher o elemento seguinte n_b de $\mathbf{s}''$ e criar uma substring ($\mathbf{s}''[n_a{+}1 \colon n_a{+}n_b]$) associada com o estado b, e assim por diante. Cada substring deve incluir o estado inicial ($\circ$) e o final ($\bullet$). No próximo passo, $subs(i)$ é utilizado para denotar o conjunto de substrings associados com o estado i.

2. Para cada estado distinto i encontrado em $\mathbf{s}'$, proceda da seguinte forma:

   (a) Inicialize o micro-modelo correspondente $\mathcal{M}_i'' = (\overline{\mathbf{S}_i''}, \mathbf{T}_i'')$ onde $\overline{\mathbf{S}_i''}$ é o conjunto de estados distintos encontrados nas substrings de $subs(i)$ e $\mathbf{T}_i''$ é a matriz de transição inicializada com zeros.

   (b) Para cada par consecutivo de micro-estados $\mathbf{s}''[k \colon k{+}1]$ em cada substring de $subs(i)$, contar a transição a partir do micro-estado $\mathbf{s}''[k]$ para o micro-estado $\mathbf{s}''[k{+}1]$ incrementando a posição correspondente na matriz $\mathbf{T}_i''$. Esta contagem inclui também o estado inicial ($\circ$) e o final ($\bullet$).

   (c) Normalize cada linha da matriz de transição $\mathbf{T}_i''$ tal que a soma dos valores em cada linha seja igual a 1, exceto para a última linha que representa o estado final e, portanto, seu valor deve ser zero, como na Figura 3.5.

---

1. Na primeira posição, na micro-sequência existe um X e, portanto a macro-sequência deve ser iniciada pelo macro-estado cujo micro-modelo pode começar com X. Tal como acontece, há apenas um macro-estado e ele é A.

2. Na segunda posição na micro-sequência, há duas possibilidades a considerar. Ou o evento Y é gerado pela atividade A, ou pode ser o caso que Y marca o início da atividade B. No entanto, mesmo que o micro-modelo de B começa com um Y, esta segunda possibilidade deve ser descartada, porque o micro-modelo de A não pode terminar com o evento anterior X. Portanto, a única possibilidade é para este Y ser produzido por A. De fato, no micro-modelo de A existe uma transição com uma probabilidade de 1.0 a partir de X para Y.

3. Um raciocínio semelhante nos permite estabelecer que o Z na terceira posição da micro-sequência também deve ser produzido por A. A única alternativa seria a de considerar a atividade C. No entanto, o macro-modelo não permite uma transição da atividade A para C e também o micro-modelo de A não permite terminar com o evento anterior Y. Assim, a única possibilidade é o Z ser produzido por A.

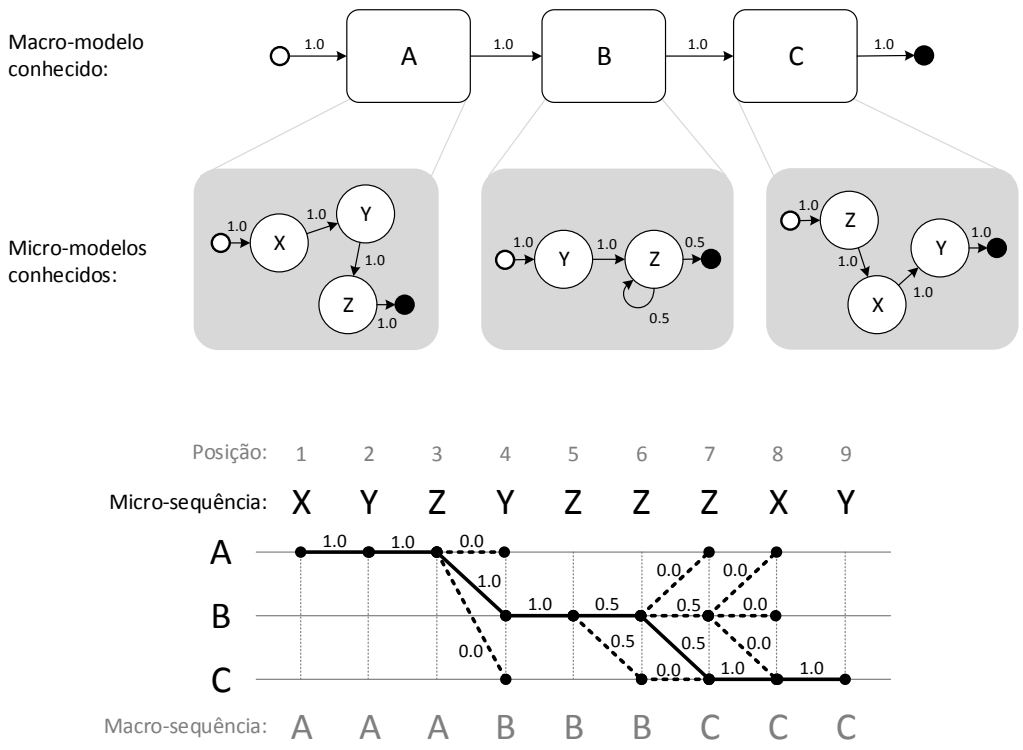


Figura 3.8: Determinando a macro-sequência mais provável

4. Na quarta posição da micro-sequência, há um Y. Não há nenhuma maneira deste Y poder ser produzido pela atividade A, porque o micro-modelo de A não está em um estado onde ele poderia produzir Y. Na verdade, o micro-modelo de A acaba de atingir seu estado final, e também não pode começar com Y, por isso temos que procurar outra atividade. De acordo com o macro-modelo, existe apenas uma atividade que pode seguir A e que é B. Felizmente, o micro-modelo de B começa com um Y. (Se este não for o caso, então, neste ponto chegaríamos à conclusão de que não é possível encontrar uma macro-sequência para uma determinada micro-sequência.) Assim, o Y na quarta posição do micro-sequência deve ter sido produzido por B.

5. Na quinta posição, existe um Z que apenas pode ser produzido por B. A possibilidade deste Z ter sido produzido no início da atividade C não pode ser considerado, pois o micro-modelo de B não pode acabar com o anterior Y.

6. Para o Z que aparece na sexta posição, há duas possibilidades. Ou este Z é produzido pelo auto-loop de Z no micro-modelo de B, ou é produzido no início da atividade C. O último é possível porque o micro-modelo de B pode acabar com o anterior Z e o macro-modelo permite atividade C ocorrer após B. Além disso, ambas as possibilidades são igualmente suscetíveis de ocorrer, uma vez que a probabilidade de B gerar o novo Z (i.e. 0.5) é igual à probabilidade de B terminar, o macro-modelo indo de B para C e C começando com Z (i.e. $0.5 \times 1.0 \times 1.0 = 0.5$). No entanto, se escolher esta segunda possibilidade, vamos chegar a um impasse quando o próximo evento (outro Z) aparecer, porque a atividade C não pode explicar a ocorrência de

dois consecutivos Z's. Por isso, ficamos com a primeira hipótese, i.e. o Z que aparece na sexta posição deve ter sido produzido por B.

7. Para o Z que aparece na sétima posição, há novamente as mesmas duas possibilidades. No entanto, se escolher B, novamente, vamos chegar a um impasse no momento em que o próximo evento X aparece, porque não haverá nenhuma atividade que pudesse explicar o próximo evento X. As razões para isso são que B não pode produzir qualquer X; C pode não começar com um X; e embora A pode começar com um X, o macro-modelo não permite uma transição da atividade B voltar para A. Portanto, a única possibilidade é para C produzir este Z, na sétima posição.

8. O X na oitava posição deve ter sido produzido por C, porque o macro-modelo não permite uma transição para qualquer outra atividade. Felizmente, existe uma transição de probabilidade 1.0 a partir de Z para X no micro-modelo C.

9. Pelas mesmas razões, o último Y deve ter sido produzido por C.

A macro-sequência resultante é AAABBBCCC e é a única macro-sequência possível neste exemplo. A probabilidade de se observar a micro-sequência XYZYZZZXY dada esta macro-sequência e esses micro-modelos pode ser computado multiplicando as probabilidades de transição ao longo do caminho sólido na Figura 3.8.

Se houvessem várias macro-sequências possíveis, então não haveria vários caminhos sólidos na Figura 3.8. Multiplicando as probabilidades de transição ao longo destes caminhos, pode-se computar a probabilidade de ocorrência de uma determinada micro-sequência. A mais provável macro-sequência, i.e. a macro-sequência a ser escolhida, é aquela que produz com a maior probabilidade para a micro-sequência observada.

Mais formalmente, supomos que o micro-modelo $\mathcal{M}_i''$ para cada estado $i \in \mathbf{S}'$ está disponível, mas a macro-sequência $\mathbf{s}'$ é desconhecida, então queremos determinar $\mathbf{s}'$ a partir $\mathbf{s}''$ e $\{\mathcal{M}_i''\}$ a partir de $\mathcal{M}'$. Note que a macro-sequência $\mathbf{s}'$ é produzida pelo macro-modelo $\mathcal{M}'$, que é uma cadeia de Markov, portanto, pode haver várias possibilidades para $\mathbf{s}'$. Em geral, estamos interessados em encontrar a solução mais provável para $\mathbf{s}'$.

O mais provável $\mathbf{s}'$ é dado pela sequência de macro-estados que é capaz de produzir $\mathbf{s}''$ com maior probabilidade. No exemplo da Figura 3.9, tínhamos $\mathbf{s}'' =$ XYZYZZZXY. Sabemos que $\mathbf{s}''$ inicia com X e portanto a macro-sequência $\mathbf{s}'$ deve ser iniciada por um macro-estado cujo micro-modelo pode começar com X. Quando isso acontece, há um único macro-estado tal como na Figura 3.6, e ele é A. Então, agora que começamos com A, tentamos analisar os seguintes símbolos $\mathbf{s}''$ com o micro-modelo $\mathcal{M}_{\mathsf{A}}''$. Descobrimos que este micro-modelo pode explicar a substring XYZ, depois que um novo macro-estado deve ser escolhido para explicar o segundo Y em $\mathbf{s}''$.

Na Figura 3.6, o único micro-modelo que começa com Y é $\mathcal{M}_{\mathsf{B}}''$. Portanto, o segundo macro-estado é B. Agora vamos usar $\mathcal{M}_{\mathsf{B}}''$ para analisar os seguintes símbolos de $\mathbf{s}''$, nos levando todo o caminho através de YZZZ, quando $\mathcal{M}_{\mathsf{B}}''$ não é possível analisar o seguinte X. O terceiro macro-estado é necessário para analisar o final XY mas nenhuma solução adequada pode ser encontrada, porque os micro-modelos $\mathcal{M}_{\mathsf{A}}''$ iniciam com X mas não terminam em Y.

O problema é que a análise do micro-modelo $\mathcal{M}_{\mathsf{B}}''$ foi longe demais. Ele deveria ter parado em YZZ e deixar o final ZXY ser analisado pelo micro-modelo $\mathcal{M}_{\mathsf{C}}''$. Neste caso, teríamos $\mathbf{s}' =$ AAABBBCCC.

$$
\begin{array}{llll}
\mathbf{s}'[1] = \mathsf{A} & \mathbf{s}''[1] = \mathsf{X} & \mathbf{T}'(\circ, \mathsf{A}) \times \mathbf{T}''_{\mathsf{A}}(\circ, \mathsf{X}) & = 1.0 \times 1.0 \\
\mathbf{s}'[2] = \mathsf{A} & \mathbf{s}''[2] = \mathsf{Y} & \mathbf{T}''_{\mathsf{A}}(\mathsf{X}, \mathsf{Y}) & = 1.0 \\
\mathbf{s}'[3] = \mathsf{A} & \mathbf{s}''[3] = \mathsf{Z} & \mathbf{T}''_{\mathsf{A}}(\mathsf{Y}, \mathsf{Z}) & = 1.0 \\
\mathbf{s}'[4] = \mathsf{B} & \mathbf{s}''[4] = \mathsf{Y} & \mathbf{T}''_{\mathsf{A}}(\mathsf{Z}, \bullet) \times \mathbf{T}'(\mathsf{A}, \mathsf{B}) \times \mathbf{T}''_{\mathsf{B}}(\circ, \mathsf{Y}) & = 1.0 \times 1.0 \times 1.0 \\
\mathbf{s}'[5] = \mathsf{B} & \mathbf{s}''[5] = \mathsf{Z} & \mathbf{T}''_{\mathsf{B}}(\mathsf{Y}, \mathsf{Z}) & = 1.0 \\
\mathbf{s}'[6] = \mathsf{B} & \mathbf{s}''[6] = \mathsf{Z} & \mathbf{T}''_{\mathsf{B}}(\mathsf{Z}, \mathsf{Z}) & = 0.5 \\
\mathbf{s}'[7] = \mathsf{C} & \mathbf{s}''[7] = \mathsf{Z} & \mathbf{T}''_{\mathsf{B}}(\mathsf{Z}, \bullet) \times \mathbf{T}'(\mathsf{B}, \mathsf{C}) \times \mathbf{T}''_{\mathsf{C}}(\circ, \mathsf{Z}) & = 0.5 \times 1.0 \times 1.0 \\
\mathbf{s}'[8] = \mathsf{C} & \mathbf{s}''[8] = \mathsf{X} & \mathbf{T}''_{\mathsf{C}}(\mathsf{Z}, \mathsf{X}) & = 1.0 \\
\mathbf{s}'[9] = \mathsf{C} & \mathbf{s}''[9] = \mathsf{Y} & \mathbf{T}''_{\mathsf{C}}(\mathsf{X}, \mathsf{Y}) \times \mathbf{T}''_{\mathsf{C}}(\mathsf{Y}, \bullet) \times \mathbf{T}'(\mathsf{C}, \bullet) & = 1.0 \times 1.0 \times 1.0
\end{array}
$$

Figura 3.9: Exemplo do cálculo de probabilidade total para produzir tanto $\mathbf{s}'$ e $\mathbf{s}''$

Este exemplo simples ilustra a possibilidade de retornar a etapa anterior, uma vez que podem haver diferentes soluções para $\mathbf{s}'$. Com ambos $\mathbf{s}'$ e $\mathbf{s}''$, em conjunto com $\mathcal{M}'$ e $\{\mathcal{M}''_i\}$, é possível calcular a probabilidade de observar uma determinada micro-sequência $\mathbf{s}''$. Este é o produto de todas as probabilidades de transição no macro- e micro-modelos. Seja $\mathbf{T}(i, j)$ a probabilidade de transição do estado i para o estado j em uma matriz transição $\mathbf{T}$. Então, para o exemplo da figura 3.3, temos a sequência de cálculo apresentada na Figura 3.9. Note que o produto de todas probabilidades é $p = 0.25$. Por razões computacionais, utilizamos um log de probabilidade (*log-probability* – $log(p)$) como alternativa. Em geral, escolhemos a solução para $\mathbf{s}'$ que produz o maior valor para *log-probability*. O procedimento é descrito no Algoritmo 3.

Em particular, o Passo 2 do Algoritmo 3 é uma função recursiva que explora todas as possibilidades para $\mathbf{s}'$ com probabilidade não nula. Tal exploração recursiva tem a forma de uma árvore, uma vez que as possibilidades de $\mathbf{s}'[k{+}1]$ são construídas sobre as possibilidades de $\mathbf{s}'[k]$. Cada caminho da raiz ($k = 1$) para uma folha ($k = n$) nesta árvore representa um candidato diferente para $\mathbf{s}'$.

No Passo 3, o algoritmo retorna o candidato com maior *log-probability*, onde *log-probability* é a soma dos *log-probabilities* ao longo do caminho da árvore.

Para melhorar a eficiência, o melhor candidato encontrado até o momento e o seu correspondente *log-probability* podem ser mantidos em variáveis globais. Ao descer um caminho na árvore (i.e. quando a construção de um novo candidato, através da recursividade no Passo 2), assim como *log-probability* para que o candidato fique abaixo do *log-probability* para o melhor candidato encontrado até o momento, esse ramo pode ser podado e a busca pode avançar imediatamente para o próximo ramo.

3.3.3 Extensão do modelo para múltiplas sequências

Os procedimentos e algoritmos apresentados na Seção 3.3.2 podem ser estendidos para o caso de múltiplas instâncias (i.e. múltiplos *traces* no log de eventos, múltiplas micro-sequências) através do cálculo da mais provável macro-sequência para cada micro-sequência. A seguir descrevemos como trabalhar com múltiplas micro-sequências e executar o algoritmo múltiplas vezes.

Algoritmo 3 Determinar a macro-sequência mais provável $\mathbf{s}'$ para uma dada micro-sequência $\mathbf{s}''$ quando $\mathcal{M}'$ e $\{\mathcal{M}''_i\}$ são conhecidos

1. Seja $\mathbf{s}''[k]$ o micro-estado na posição k da micro-sequência $\mathbf{s}''$ e seja $\mathbf{s}'[k]$ sendo o macro-estado correspondente que é para ser determinado. Ambas as sequências começam em $k = 1$ e terminam em $k = n$. Execute o passo 2 recursivamente, a partir de $k = 1$.

2. Considere as seguintes possibilidades de $\mathbf{s}'[k]$:

   (a) Se $k = 1$ então $\mathbf{s}'[k]$ pode ser qualquer macro-estado i tal que $\mathbf{T}'(\circ, i) > 0$ e $\mathbf{T}''_i(\circ, \mathbf{s}''[k]) > 0$. Para cada um destes macro-estados i, definir $\mathbf{s}'[k] \coloneqq i$ e executar o passo 2 para $k \coloneqq k{+}1$.

   (b) Se $1 < k \leq n$ então considere os seguintes casos:

      i. Se ambos $\mathbf{s}''[k{-}1]$ e $\mathbf{s}''[k]$ vêm do mesmo micro-modelo $\mathcal{M}''_i$ então $\mathbf{s}'[k{-}1] = \mathbf{s}'[k] = i$. Considerar este caso somente se $\mathbf{T}''_i(\mathbf{s}''[k{-}1], \mathbf{s}''[k]) > 0$. Se for assim, definir $\mathbf{s}'[k] \coloneqq i$ e executar o passo 2 para $k \coloneqq k{+}1$.

      ii. Se $\mathbf{s}''[k{-}1]$ vem de $\mathcal{M}''_i$ e $\mathbf{s}''[k]$ vem de $\mathcal{M}''_j$ (with $i \neq j$) então $\mathbf{s}'[k{-}1] = i$ e $\mathbf{s}'[k] = j$. Considerar todos os possíveis macro-estados j para os quais $\mathbf{T}''_i(\mathbf{s}''[k{-}1], \bullet) \times \mathbf{T}'(i, j) \times \mathbf{T}''_j(\circ, \mathbf{s}''[k]) > 0$. Para cada um desse macro-estado j, definir $\mathbf{s}'[k] \coloneqq j$ e executar o passo 2 para $k \coloneqq k{+}1$.

   (c) Se $k = n$ então chegamos ao final de $\mathbf{s}''$ e agora temos um candidado completo para $\mathbf{s}'$. Aceitar este candidato se ele termina corretamente, i.e. somente se $\mathbf{T}''_i(\mathbf{s}''[n], \bullet) \times \mathbf{T}'(i, \bullet) > 0$ onde $i = \mathbf{s}'[n]$. Se aceito, armazenar o candidato em uma lista de candidatos.

3. A partir de todos os candidatos para $\mathbf{s}'$ coletados no passo 2(c), retornar o candidato que fornece maior *log-probability* para $\mathbf{s}''$.

Múltiplas micro-sequências

Na Seção 3.3.2 consideramos o uso de uma única micro-sequência $\mathbf{s}''$ (e um macro-modelo $\mathcal{M}'$) para determinar os micro-modelos $\{\mathcal{M}''_i\}$ e a macro-sequência $\mathbf{s}'$. No entanto, um log de eventos contém eventos de múltiplas instâncias do processo e cada instância do processo corresponde a uma micro-sequência separada. Além disso, cada micro-sequência está associada com a sua própria macro-sequência. Seja Ω'' o multiconjunto de micro-sequências (i.e. um conjunto de micro-sequências que podem incluir elementos repetidos) e seja Ω' o multiconjunto das macro-sequências correspondentes. Então, os Algoritmos 1 - 3 podem ser facilmente estendidos, a fim de lidar com múltiplas micro-sequências, através das seguintes adaptações:

(a) No Passo 1 do Algoritmo 1 obtemos uma sequência randômica $\tilde{\mathbf{s}}$ a partir $\mathcal{M}'$ para cada micro-sequência $\mathbf{s}'' \in \Omega''$. Também, nos Passos 2 e 3 do Algoritmo 1 consideramos o conjunto de micro-sequências Ω'' ao invés de uma única micro-sequência $\mathbf{s}''$ (ver observações sobre os Algoritmos 2 e 3 abaixo). Reinterpretar o Passo 4 do Algoritmo 1

quer dizer "repetir os Passos 2 e 3 até que as estimativas para cada $\mathbf{s}' \in \Omega'$ e $\{\mathcal{M}''_i\}$ convergirem".

(b) No Algoritmo 2, Passo 1, construir as substrings $subs(i)$ para cada macro-estado i separando cada micro-sequência $\mathbf{s}'' \in \Omega''$ de acordo com a macro-sequência correspondente $\mathbf{s}' \in \Omega'$. Uma vez que todas as substrings foram coletadas em $subs(i)$, o Passo 2 do Algoritmo 2 pode ser executado sem mudança.

(c) O Algoritmo 3 precisa ser executado para cada micro-sequência $\mathbf{s}'' \in \Omega''$ a fim de determinar a macro-sequência correspondente $\mathbf{s}'$. Não são necessárias alterações para este algoritmo.

Uma questão importante quando se trabalha com várias micro-sequências tem a ver com a randomização inicial da macro-sequência correspondente. Suponha que um macro-modelo é capaz de gerar duas sequências AB e ABC com igual probabilidade. Além disso, suponha que as micro-sequências observados são XXXYYY e XXXYYYZZZ. Quando randomizar da macro-sequência para cada um dessas micro-sequências, parece mais adequado para relacionar as micro-sequências XXXYYY e XXXYYYZZZ com as macro-sequências AB e ABC, respectivamente. No entanto, a macro-sequência será definida randomicamente a partir de AB e ABC portanto, qualquer atribuição é possível. Os cenários possíveis são os seguintes:

(a) As macro-sequências para XXXYYY e XXXYYYZZZ são ambas expansões de AB.

(b) A macro-sequência para XXXYYY é uma expansão de AB e a micro-sequência para XXXYYYZZZ é uma expansão de ABC.

(c) A macro-sequência para XXXYYY é uma expansão de ABC e a micro-sequência para XXXYYYZZZ é uma expansão para AB.

(d) A macro-sequência para XXXYYY e XXXYYYZZZ são ambas expansões de ABC.

Claramente, (b) é a opção mais desejável, uma vez que leva ao modelo mais simples de todos, e aquele que é capaz de gerar as micro-sequências observadas com maior *log-probability*. Neste caso, o micro-modelo para A apenas emite X's, o micro-modelo para B apenas emite Y's, e o micro-modelo C apenas emite Z's.

Na prática, esta questão se manifesta nos diferentes comprimentos de *traces*, onde um *trace* é a (micro-)sequência de eventos registrados para uma determinada instância no log de eventos. Em geral, as maiores micro-sequências devem ser atribuídas às sequências mais longas, que são extraídas do macro-modelo $\mathcal{M}'$. Portanto, no Passo 1 do Algoritmo 1, ao invés de definir uma sequência $\tilde{\mathbf{s}}$ para cada micro-sequência $\mathbf{s}'' \in \Omega''$, define $N = |\Omega''|$ sequências de uma só vez, onde N é o número das micro-sequências. Em seguida, classificam-se as micro-sequências e as sequências randômicas pelo comprimento e só então executa a atribuição entre elas. Esta simples otimização permite o Algoritmo 1 proporcionar melhores resultados, encontrando micro-modelos que são capazes de produzir as micro-sequências com maior *log-probability*.

Executando o algoritmo múltiplas vezes

O Passo 1 no Algoritmo 1 é obter uma inicialização randômica da macro-sequência (ou macro-sequências). Uma vez que esta inicialização é randômica, o algoritmo pode produzir soluções diferentes em várias execuções. Isto pode produzir micro-modelos um pouco diferentes, onde uma determinada parte do comportamento de baixo nível observado termina sendo atribuído a uma macro-atividade, em vez de outra.

Por exemplo, para a micro-sequência $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ o passo de inicialização randômica no Algoritmo 1 pode produzir a macro-sequência $\mathbf{s}' = \mathsf{AAABBBCCC}$. Então, torna-se claro que o evento pertence a qual macro-atividade:

XYZYZZZXY
| | | | | | | | |
AAABBBCCC

A partir do mapeamento acima pode-se derivar a solução de imediato (i.e. os micro-modelos) na Figura 3.6 seguem o procedimento descrito no Algoritmo 2.

No entanto, se a macro-sequência é inicializada com $\mathbf{s}' = \mathsf{AABBBBBCC}$ então teremos o seguinte mapeamento:

XYZYZZZXY
| | | | | | | | |
AABBBBBCC

Este mapeamento origina uma solução diferente, onde os micro-modelos $\mathcal{M}''_{\mathsf{A}}$ e $\mathcal{M}''_{\mathsf{C}}$ serão dois modelos simples que produzem apenas XY e $\mathcal{M}''_{\mathsf{B}}$ será um modelo mais complexo, que começa com Z e a partir Z pode ir para Y, ou para Z, ou para o fim. O *log-probability* da micro-sequência $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ nesta solução é -4.16, enquanto que na primeira solução o *log-probability* é -1.39. Portanto, a primeira solução é a preferida. Novamente, a solução preferida é aquela que é capaz de produzir uma determinada micro-sequência com mais alto *log-probability*.

Este conceito pode ser facilmente estendido para o caso de várias micro-sequências. Assim é obtida uma solução, somando-se o *log-probability* para todas as micro-sequências produz uma medida da qualidade relativa desta solução.

Portanto, uma outra otimização que usamos é executar Algoritmo 1 várias vezes, a fim de escolher o melhor resultado, i.e. o conjunto de micro-modelos que são capazes de produzir as micro-sequências com maior *log-probability* total. O número de execuções é arbitrário e pode ser escolhido pelo usuário. Na prática, descobrimos que um número de execuções K em algum lugar entre $\sqrt{N} \leq K \leq N$, onde N é o número de micro-sequências de entrada, geralmente é suficiente para obter o melhor solução possível.

Antes de prosseguirmos, o Algoritmo 4 resume as adaptações que foram feitas para o Algoritmo 1, a fim de descrever as otimizações da Seção 3.3.3. Ele inclui as adaptações para lidar com múltiplas micro-sequências dos Passos 1 a 4, que inclui a classificação por comprimento da micro-sequência e macro-sequências das etapas 1(a)–1(d) e inclui várias execuções, além da escolha da melhor solução nas etapas A-B.

3.3.4 Descobrindo padrões básicos de *workflow*

A macro-modelo que foi usado como um exemplo nas seções anteriores é capaz de gerar apenas a sequência simples ABC. No entanto, na prática, existem muitos tipos de

Algoritmo 4 Estimar os micro-modelos $\{\mathcal{M}''_i\}$ e as macro-sequências Ω' a partir do macro-modelo $\mathcal{M}'$ e as micro-sequências Ω'' (substitui o Algoritmo 1)

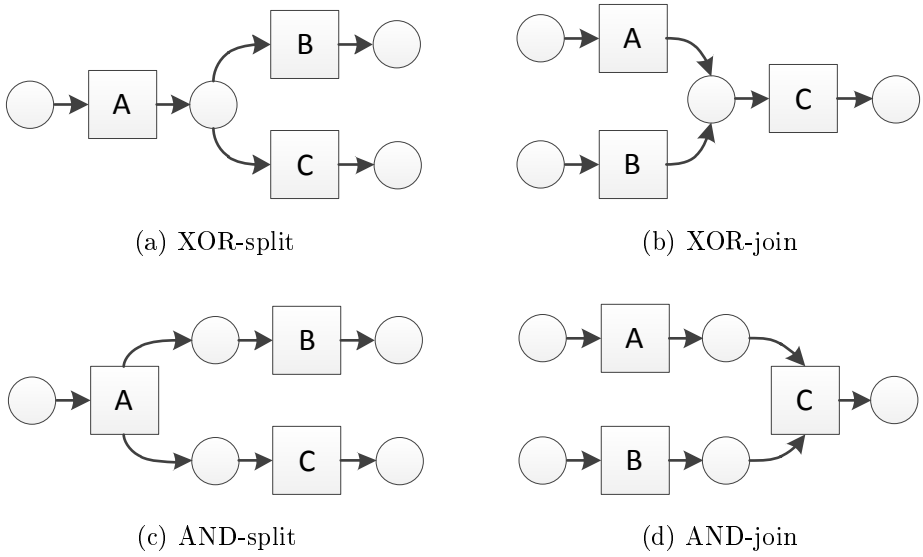
A. Execute os seguintes passos K vezes, onde K deve ser escolhido pelo usuário:

1. Inicialize as macro-sequências Ω' de acordo com o seguinte procedimento:
   (a) Definir $N = |\Omega''|$ sequências randômicas a partir do modelo Markov $\mathcal{M}'$, onde N é o número de micro-sequências em Ω''. Seja Ω' o multiconjunto de sequências randômicas apenas definidas.
   (b) Ordenar as micro-sequências Ω'' e as sequências em Ω' pelo tamanho.
   (c) Após a ordenação, pegar cada sequência em Ω' para ser a sequência que corresponde a micro-sequência na mesma posição em Ω''.
   (d) Expandir cada sequência em Ω' para se tornar a macro-sequência para a micro-sequência correspondente em Ω'' (o procedimento de expansão é descrito na Seção 3.3.2)
2. Alimentar com todas as micro-sequências Ω'' e suas macro-sequências correspondentes Ω' o Algoritmo 2. No passo 1 do Algoritmo 2, separar todas as micro-sequências de acordo com sua respectiva macro-sequência.
3. Para cada micro-sequência $\mathbf{s}'' \in \Omega''$, executar o Algoritmo 3 de modo a determinar a mais provável macro-sequência $\mathbf{s}'$. Atualizar Ω' para que Ω' contenha as macro-sequências obtidas a partir do Algoritmo 3.
4. Volte ao passo 2 e repita até que ambos Ω' e $\{\mathcal{M}''_i\}$ convergirem.
5. Use o macro-modelo $\mathcal{M}'$ e os micro-modelos $\{\mathcal{M}''_i\}$ para computar o *log-probability* de cada micro-sequência produzida $\mathbf{s}'' \in \Omega''$. Somar os *log-probabilities* para todas as micro-sequências em Ω''.

B. Ao final das K soluções definidas no passo A, retornar a solução que teve o maior valor para a soma de *log-probabilities*.

comportamento que um modelo de processo de negócio pode conter, como *XOR-splits*, *AND-splits*, *loops*, etc. Estes tipos de comportamento foram identificados e classificados como um conjunto de padrões de *workflow* (*workflow patterns*) (van der Aalst et al. 2003a). Dado que estes padrões são muito comuns na modelagem de processos, é provável que qualquer modelo do processo de nível elevado irá conter, pelo menos, alguns dos padrões mais básicos.

Nesta seção, nosso objetivo é realizar uma verificação de integridade da abordagem proposta para assegurar que é possível descobrir o comportamento dos agentes em processos de nível macro que envolvem mais do que apenas uma sequência linear de passos. Em particular, gostaríamos de verificar se a abordagem proposta é capaz de lidar com, pelo menos, os padrões mais básicos, ou seja, *XOR-splits*, *XOR-joins*, *AND-splits*, e *AND-joins*, conforme apresentado na Figura 3.10.



(a) XOR-split (b) XOR-join

(c) AND-split (d) AND-join

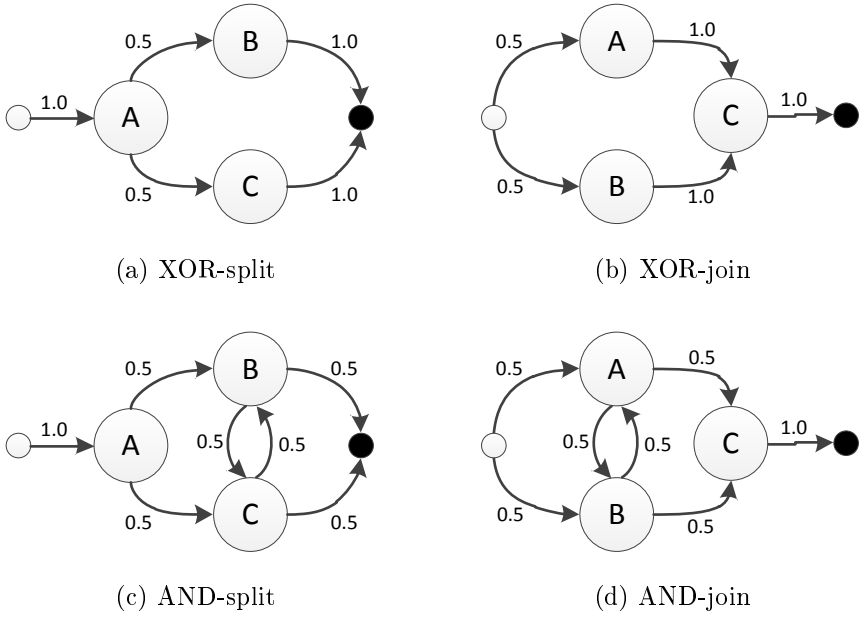
Figura 3.10: Padrões básicos de controle de fluxo expressados com uma rede de petri

Descrevendo padrões como modelos de Markov

Não há nenhum problema em representar um *XOR-split* ou um *XOR-join* por meio de um modelo de Markov, como nas Figuras 3.10(a) e 3.10(b), respectivamente. No entanto, a natureza paralela de *AND-splits* e *AND-joins* não podem ser capturadas exatamente por uma cadeia de Markov de primeira ordem. Na Figura 3.10(c), o AND-split é capaz de gerar as sequências ABC e ACB. Assumindo que estas duas sequências podem ocorrer com igual probabilidade, podemos derivar um modelo de Markov a partir das sequências ∘ABC• e ∘ACB•. O modelo de Markov resultante é apresentado na Figura 3.11(c). O modelo começa sempre com um A, seguido por B ou C. Depois de B, um C pode seguir ou a sequência pode acabar. De maneira semelhante, depois de C um B pode seguir ou a sequência pode terminar.

O problema com o modelo da Figura 3.11(c) é que ele permite mais comportamentos do que apenas as sequências ABC e ACB. De fato, este modelo pode produzir várias iterações envolvendo B e C antes do fim da sequência. O modelo também permite sequências mais curtas, isto é ∘AB• e ∘AC•. No caso particular de um *AND-split* como este, que envolve apenas duas atividades em paralelo, o problema poderia ser resolvido por meio da utilização de um modelo de Markov de segunda ordem. Esse modelo levaria em conta os dois estados anteriores ao determinar o próximo estado. No entanto, o aumento da ordem do modelo de Markov não é uma abordagem escalável uma vez que, em geral, um modelo de processo pode incluir um número arbitrário de atividades em paralelo.

Uma situação similar ocorre com o padrão *AND-join* apresentado na Figura 3.10(d). Neste caso, o modelo especifica que tanto ABC ou BAC podem ocorrer. Em qualquer caso, tanto A e B deve ser concluído antes de C ocorrer, por isso também conhecido como o padrão *synchronization* (van der Aalst et al. 2003a). Supondo-se que ambas as sequências podem ocorrer com igual probabilidade, pode-se derivar a partir de ∘ABC• e ∘BAC• o modelo de Markov ilustrado na Figura 3.11(d). Novamente, este modelo permite mais



(a) XOR-split (b) XOR-join

(c) AND-split (d) AND-join

Figura 3.11: Padrões básicos de controle de fluxo expressados como um modelo de Markov

comportamento do que inicialmente previsto, a saber: vários A's e B's podem preceder C e as sequências mais curtas AC e BC também são possíveis.

Tanto o modelo *AND-split* da Figura 3.11(c) e o modelo *AND-join* da Figura 3.11(d) sofrem de um problema de *underfitting* (van der Aalst et al. 2010) pois permitem um comportamento além do que realmente pode ser produzido por um verdadeiro *AND-split* ou *AND-join*, respectivamente. De qualquer modo, para a finalidade das nossas experiências este não é um problema impeditivo. Pelo contrário, se a nossa abordagem consegue descobrir o comportamento produzido pelos modelos da Figura 3.11(c) e Figura 3.11(d), então ele certamente terá êxito na descoberta do comportamento de um *AND-split* e um *AND-join*, uma vez que o último é um subconjunto do primeiro. Portanto, em nossos experimentos foram utilizados os quatro modelos apresentados na Figura 3.11. Utilizando estes modelos, torna-se possível testar a descoberta não apenas para os padrões básicos, mas também do comportamento *loop*.

Descobrindo os micro-modelos $\{\mathcal{M}''_i\}$

Neste experimento, o objetivo é verificar se o Algoritmo 4 é capaz de descobrir os micro-modelos a partir de um conjunto de micro-sequências de entrada e um determinado macro-modelo, onde o modelo macro é um dos modelos da Figura 3.11. Para simplificar, vamos usar os mesmos micro-modelos $\{\mathcal{M}''_{\mathsf{A}}, \mathcal{M}''_{\mathsf{B}}, \mathcal{M}''_{\mathsf{C}}\}$ como na Figura 3.6, juntamente com cada macro-modelo da Figura 3.11 para obter um modelo hierárquico de Markov diferente. A partir de cada um desses modelos hierárquicos, geramos $N = 100$ micro-sequências por simulação. Então executamos Algoritmo 4 sobre essas micro-sequências e o macro-modelo fornecido para re-descobrir os micro-modelos $\{\mathcal{M}''_{\mathsf{A}}, \mathcal{M}''_{\mathsf{B}}, \mathcal{M}''_{\mathsf{C}}\}$. Como

explicado na Seção 3.3.3, o passo 'A' do Algoritmo 4 pode ser executado um número arbitrário de vezes K. Para este experimento, consideramos a escolha $K = 10$.

Em todos os casos, o Algoritmo 4 descobriu corretamente os micro-modelos, onde $\mathcal{M}''_{\mathsf{A}}$ geram XYZ, $\mathcal{M}''_{\mathsf{B}}$ geram $\mathsf{YZ(Z)}$, e $\mathcal{M}''_{\mathsf{C}}$ geram ZXY. As probabilidades de transição $\mathbf{T}''_{\mathsf{B}}(\mathsf{Z},\mathsf{Z})$ e $\mathbf{T}''_{\mathsf{B}}(\mathsf{Z},\bullet)$ não são exatamente iguais para $\frac{1}{2}$ como na Figura 3.6 uma vez que em uma execução randômica de $N = 100$ sequências podem acontecer que não são exatamente como muitas transições de Z para Z, pois há de Z para $\bullet$. Em qualquer caso, o Algoritmo 4 é capaz de capturar as probabilidades de transição exatas que podem ser encontradas na entrada das micro-sequências.

No entanto, existem diferenças marcantes em termos de número de iterações e tempo de computação entre padrões *XOR* e *AND*. Neste sentido, estas diferenças devem ser esperadas uma vez que os padrões *AND* da Figura 3.11 permitem mais comportamentos não-determinísticos que os padrões *XOR*. A Tabela 3.1 apresenta os resultados da experiência. O número de iterações do algoritmo EM (Passos 2–4 do Algoritmo 4) são relatados para cada execução da etapa 'A '. Aqui é evidente que os padrões *AND* requerem mais iterações para a convergência, que é atribuído ao fato de que as micro-sequências geradas por esses modelos tendem a ser mais longas (9 símbolos em média) do que os micro-sequências obtidas a partir de padrões *XOR* (6 símbolos em média).

	Número de iterações EM para cada execução											
Padrão	1	2	3	4	5	6	7	8	9	10	Média	Tempo total
XOR-split	3	2	3	3	3	3	3	3	3	3	2.9	$<$ 1s
XOR-join	3	2	4	3	4	2	3	3	4	3	3.1	$<$ 1s
AND-split	4	4	3	5	6	5	4	4	3	4	4.2	$\sim$ 41s
AND-join	3	5	4	4	7	2	5	6	5	7	4.8	$\sim$ 37s

Tabela 3.1: Resultados da amostra nos quatro modelos da Figura 3.11 com $N = 100$ sequências e $K = 10$ execuções

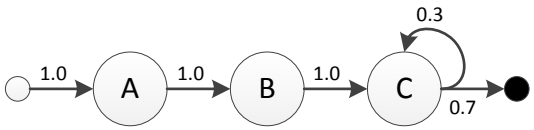
A diferença mais notável é o tempo de execução total. Embora os micro-modelos para um macro-modelo que contém um padrão *OR* possa ser descoberto rapidamente, descobrindo os mesmos micro-modelos para um macro-modelo que contém um padrão *AND* leva muito mais tempo, conforme observado nos resultados que são apresentados na Tabela 3.1. Os números variam significativamente: o tempo total de execução de padrões *AND* estão no intervalo de alguns segundos a alguns minutos, para o caso diferentes permutações de execuções e quantidade de sequências, ao passo que para os padrões *XOR* é sempre menos de um segundo.

A diferença no número de iterações não é grande o suficiente para explicar a diferença em tempo de execução, portanto, que explica os períodos relativamente longos de execução para o padrão *AND*. Aprofundando a análise dos resultados experimentais revela que a maior parte desse tempo de execução é gasto durante a primeira execução do Passo 3 do Algoritmo 4. Quando uma macro-sequência está sendo calculada pela primeira vez, a árvore de recursão que o Algoritmo 3 atravessa é muito grande, e, portanto, a maior parte do tempo gasto na determinação da melhor macro-sequência candidata, já que há muitos candidatos para escolher. Mas, uma vez que a primeira macro-sequência é obtida, as iterações dos Passos 2–4 no Algoritmo 4 executam rapidamente, quase tão rapidamente

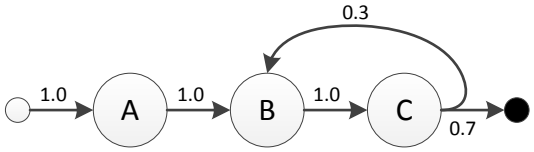
como no caso do padrão *XOR*. Finalmente, olhando a Figura 3.11(c) e 3.11(d), pode-se ver que os padrões *AND*, quando expressados como cadeias de Markov, contêm uma espécie de *loop*, que é a razão pela qual se demorou mais tempo para encontrar a solução.

Macro-modelos com o padrão *loop*

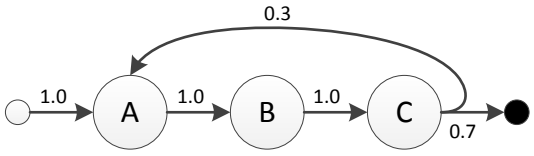
Nos experimentos anteriores, verificou-se um *loop* em um dos micro-modelos (especificamente, no $\mathcal{M}''_{\mathsf{B}}$ que pode produzir YZZ..., tal como apresentado na Figura 3.3) mas não no macro-modelo. Aqui vamos investigar o que acontece quando há um *loop* no macro-modelo. Tal *loop* pode incluir um número arbitrário de atividades, quando se inclui apenas uma ou duas atividades, ele é chamado de *short loop* (Medeiros et al. 2004). Para simplificar, utilizando três atividades A, B, e C como no exemplo da Figura 3.3, é possível ter *loops* de comprimento 1, comprimento 2, e comprimento 3, conforme ilustrado na Figura 3.12 apresenta um exemplo de cada um. Da mesma forma como foi realizado na Seção 3.3.4, usamos os micro-modelos $\{\mathcal{M}''_{\mathsf{A}}, \mathcal{M}''_{\mathsf{B}}, \mathcal{M}''_{\mathsf{C}}\}$ da Figura 3.6 juntamente com cada macro-modelo na Figura 3.12 para obter um modelo hierárquico de Markov diferente. De cada um destes modelos, geramos $N = 100$ micro-sequências e executamos o Algoritmo 4 com $K = 10$.



(a) Loop-1



(b) Loop-2



(c) Loop-3

Figura 3.12: Padrões básicos de controle de fluxo expressados com modelos de Markov

Os resultados são apresentados na Tabela 3.2. O Algoritmo 4 descobre os micro-modelos corretos rapidamente para o caso do Loop-1, com número constante de iterações em cada execução (2); enquanto que para o Loop-2 e Loop-3 o tempo é 32 a 95 vezes maior, respectivamente. Isto pode ser explicado pelo fato de que $\mathcal{M}''_{\mathsf{C}}$ produz a sequência simples ZXY, enquanto $\mathcal{M}''_{\mathsf{B}}$ contém seu próprio *loop* (um loop de baixo nível de comprimento 1

	Número de iterações EM para cada execução											
Padrão	1	2	3	4	5	6	7	8	9	10	Média	Tempo total
Loop-1	2	2	2	2	2	2	2	2	2	2	2.0	$<$ 1s
Loop-2	4	3	5	5	4	5	5	5	2	3	4.1	$\sim$ 32s
Loop-3	3	2	2	2	2	2	2	3	3	2	2.3	$\sim$ 95s

Tabela 3.2: Resultados da amostra nos três *loops* da Figura 3.12 com $N = 100$ sequências e $K = 10$ execuções

onde Z repete em YZZ...). Uma vez que tanto o Loop-2 e Loop-3 incluem a atividade B, todas as vezes que B executa, introduz as suas próprias repetições de Z na micro-sequência. Encontrar a melhor macro-sequência para tal micro-sequência leva mais tempo, já que há mais candidatos para escolher. Finalmente, esta é a mesma razão pela qual demorou mais tempo para encontrar a solução para os padrões *AND* da Tabela 3.1: olhando a Figura 3.11(c) e 3.11(d), pode-se ver que os padrões *AND*, quando expressados como cadeias de Markov, contêm uma espécie de *loop* também.

Portanto, o caso de uma matriz de transição esparsa, que são matrizes com um grande número de elementos desprezíveis ou próximos a zero, pode ser uma restrição da técnica de mineração. No entanto, existem soluções computacionais para este problema. Outra restrição é o caso particular do modelo de processo incluir um número arbitrário de atividades em paralelo, onde a cadeia de Markov deveria levar em consideração os estados anteriores, aumentando a ordem da cadeia. Assim, de fato a escalabilidade da abordagem ainda é um problema que está sendo avaliado.

3.4 Estágio 3: avaliação da complexidade do modelo hierárquico

Ao longo da seção anterior vimos como estimar o modelo de processo hierárquico a partir do log de eventos. A partir de agora veremos como avaliar a complexidade deste modelo hierárquico encontrado através de métricas de complexidade. Ao longo da Seção 2.4, diversas métricas com o foco em modelos de processos de negócio foram apresentadas e vários fatores que influenciam a capacidade de compreensão de modelos de processos foram discutidos. Muitos fatores podem ser encontrados na literatura existente, embora estes fatores baseiam-se essencialmente em uma motivação teórica e principalmente lidam com o tamanho e controle de fluxo do modelo de processo.

Uma pergunta que surge naturalmente a partir deste trabalho refere-se à maneira pela qual os eventos de baixo nível são agrupados em atividades de alto nível. Na Figura 3.3, podemos observar que os eventos X, Y, Z foram divididos em modelos de baixo nível de complexidade similar e também que a complexidade destes modelos de nível baixo é semelhante a do modelo de alto nível, expresso em termos de atividades A, B, C. Por estas razões, podemos dizer que o modelo hierárquico é "equilibrado" no sentido de que todas as cadeias de Markov em seu modelo são de complexidade similar. Mas isso é o que acontece normalmente? Como podemos determinar se um determinado modelo hierárquico é equilibrado e como podemos comparar dois modelos hierárquicos e dizer

que um é mais complexo e equilibrado do que o outro? Esta também é uma questão que é abordada neste trabalho, onde propomos e ilustramos o uso de um conjunto de métricas para avaliar modelos hierárquicos.

Como foi observado que um modelo hierárquico de Markov pode ser extraído a partir de um log de eventos (i.e. uma micro-sequência ou um conjunto de micro-sequências) e uma descrição de alto nível do processo de negócio (i.e. o macro-modelo), podemos voltar nossa atenção para a a forma de avaliar e comparar modelos hierárquicos em uma base quantitativa. Em particular, estamos interessados em medir a complexidade dos modelos hierárquicos e de seus componentes individuais (i.e. o macro-modelo e cada um dos micro-modelos).

Neste trabalho, estamos interessados em métricas que podem ser usadas para avaliar a complexidade de um modelo hierárquico de Markov e seus componentes. Como sugerido na Tabela 2.4, na literatura existe uma diversidade muito grande de métricas de complexidade para modelos de processo, conforme estudos realizados por Cardoso (2006), Laue and Gruhn (2006), Mendling (2007), Kreimeyer and Lindemann (2011). Neste trabalho vamos focar em um subconjunto destas métricas que estão mais alinhadas em medir fatores como: tamanho *(size)*, densidade *(density)*, modularidade *(modularity)*, controle de fluxo *(control-flow)*. Estes tipos de métricas nos permitem medir a complexidade de um modelo hierárquico, para comparar a complexidade de dois modelos e determinar se um dado modelo hierárquico é mais "equilibrado" no sentido que o micro-modelo e o macro-modelo tem complexidade similar. Ao final, as referências disponíveis explicam algoritmos e evidências empíricas relevantes para a métrica. Este conjunto de métricas que foram selecionadas estão sumarizados na Tabela 3.3.

Métrica	**Abrev.**	**Foco**
No. de arcos por nó (Mendling 2009)	NAN	Densidade
Densidade Relacional (Vanderfeesten et al. 2008)	RD	Densidade
No. de caminhos (McCabe 1976)	NP	Controle de fluxo
Tamanho do caminho (Newman 2003)	PL	Tamanho
Complexidade ciclomática (McCabe 1976)	CC	Controle de fluxo
Fan-in/Fan-out (Gruhn and Laue 2007)	FIO	Modularidade
No. de sub-processos (Reijers and Mendling 2008)	SUB	Modularidade

Tabela 3.3: Métricas para avaliar os componentes de um modelo hierárquico

Todas as métricas (exceto SUB) podem ser calculadas separadamente para o macro-modelo e para cada micro-modelo. Para este cálculo, foi realizada uma atualização do programa que foi implementado para abordagem de mineração de processos (modelo hierárquico de Markov), onde foi feita a inclusão de um módulo para suporte no cálculo das métricas que estão listadas na Tabela 3.3. O código fonte do módulo de cálculo das métricas está disponibilizado no Apêndice B.3.

Isto proporciona uma avaliação de cada componente no modelo hierárquico. Por exemplo, no modelo hierárquico da Figura 3.3 tem quatro componentes e as métricas podem ser calculadas para cada um deles. Para se ter uma avaliação do modelo hierárquico como um todo, para cada métrica nós pegamos a média simples dos valores obtidos em todos os

micro-modelos. Ressaltamos que a avaliação do modelo como um todo é visa a avaliação geral da complexidade do modelo hierárquico, no entanto é possível também avaliar especificamente cada macro e micro-modelo descoberto. Como não existe prioridade entre o macro e micro-modelos (i.e. não se aplica a média ponderada), optamos por fazer a segunda média simples deste resultado junto com o valor obtido para o macro-modelo.

Seja m_a o valor da métrica m computada para o macro-modelo, e seja m_i o valor da mesma métrica computada para cada micro-modelo i. Assumindo que existem K micro-modelos, a métrica para o modelo completo m_c é computada como:

$$m_c = \frac{1}{2} \cdot \left(m_a + \frac{1}{K} \cdot \sum_{i=1}^{K} m_i \right) \tag{3.1}$$

Por exemplo, se a complexidade ciclomática pode ser computada para o macro-modelo e para cada um dos micro-modelos, então a complexidade ciclomática para o modelo hierárquico completo pode ser calculada de acordo com Eq. (3.1).

A partir do resultado da aplicação das métricas definidas na Tabela 3.3 temos um ponto de partida para identificar melhorias e introduzi-las no modelo do processo, i.e micro-modelos com grande complexidade sugerem que sejam introduzidas outras atividades/caminhos no macro-modelo de forma a reduzir esta complexidade encontrada.

Visando formar uma base para a análise de complexidade dos modelos de processo, descrevemos todas as métricas listadas na Tabela 3.3, que visam capturar diversos aspectos relacionados a estrutura do modelo de processo. Esta descrição tem o objetivo de apoiar na recomendação de como lidar com as métricas para avaliação de complexidade e justificar a escolha. Nesta direção, as métricas (quando possível) são representadas graficamente na forma de um modelo de Markov para apresentar um exemplo do conjunto de arcos (*arcs*) e nós (*nodes*). Depois, cada modelo de Markov serve de exemplo para a descrição da métrica onde possível. Em seguida, a definição explica como o algoritmo segue para o cálculo da métrica, onde cada métrica tem um significado básico que relata a importância da métrica para a análise de complexidade.

– **Número de arcos por nó**: avalia a densidade de um modelo, i.e. através da razão entre o número total de arcos em um modelo de processo para o número total dos seus nós, podemos ter noção da dimensão (complexidade) de um modelo, conforme ilustrado na Figura 3.13. Um alto valor para esta métrica indica maior complexidade do modelo.

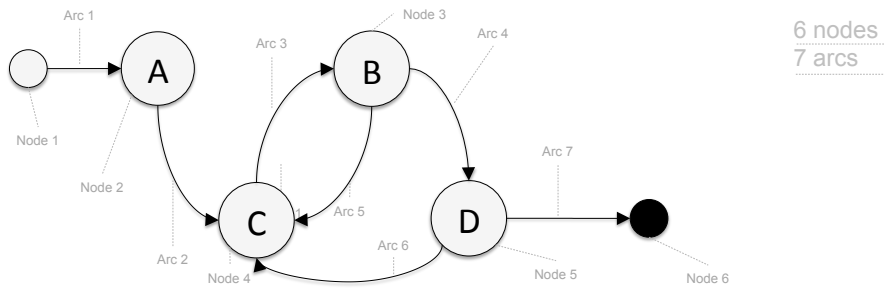


Figura 3.13: Número de arcos por nó expressado como um modelo de Markov

Definição: Razão entre o número de arcos e número de nós, incluindo arcos e nós de início e fim.

Significado: Avaliação da densidade no interior do processo; Descrição do nível de *cross-linking* dentro de um modelo.

– **Densidade relacional**: é um tipo de métrica que além de proporcionar uma visão da dimensão também possibilita a avaliação da intensidade de um modelo de processo, conforme ilustrado na Figura 3.14. Um alto valor para esta métrica indica maior complexidade do modelo.

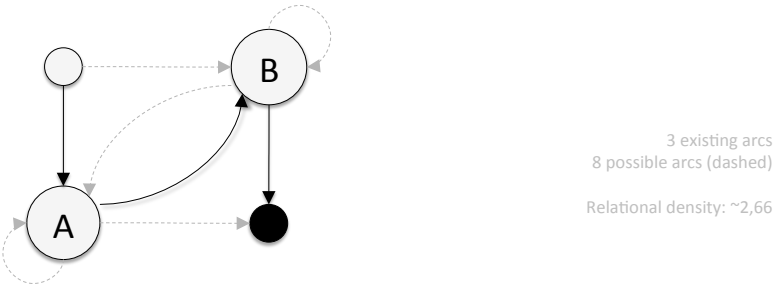


Figura 3.14: Densidade relacional expressada como um modelo de Markov

Definição: Razão entre o número de arcos de um modelo e o número total de arcos possíveis, incluindo arcos de início e fim.

Significado: Avaliação da densidade e intensidade de um processo; Intensidade de *cross-linking*.

– **No. de caminhos**: é um tipo de métrica que pode ser importante para medir a complexidade, pois em um modelo que apresenta uma sequência linear tem a complexidade mínima (independentemente da sequência ser curta ou longa). Não obstante, um modelo que tenha vários caminhos possíveis tem maior complexidade, pois um modelo em que seja possível ir de qualquer atividade para qualquer outra atividade tem complexidade máxima (independentemente do número de atividades), conforme ilustrado na Figura 3.15. Um alto valor para esta métrica indica maior complexidade do modelo.

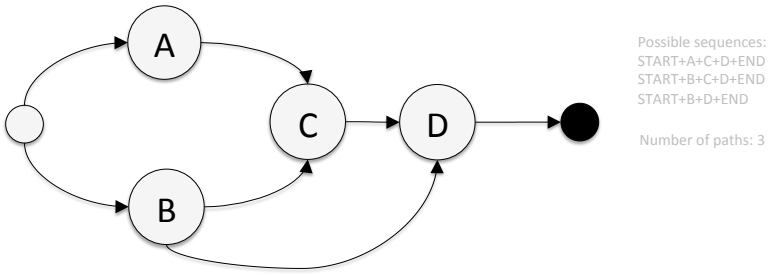


Figura 3.15: Número de caminhos expressado como um modelo de Markov

Definição: Número de todos os possíveis caminhos distintos entre o nó inicial e o nó final.

Significado: Avaliação dos caminhos redundantes através do processo; Determinação de clareza de processamento do processo; Determinação de início e fim críticos.

– **Tamanho do caminho**: é um tipo de métrica importante para verificar o tamanho dos caminhos possíveis em um modelo, conforme ilustrado na Figura 3.16. Um alto valor para esta métrica indica maior complexidade do modelo e que ele pode ser modularizado

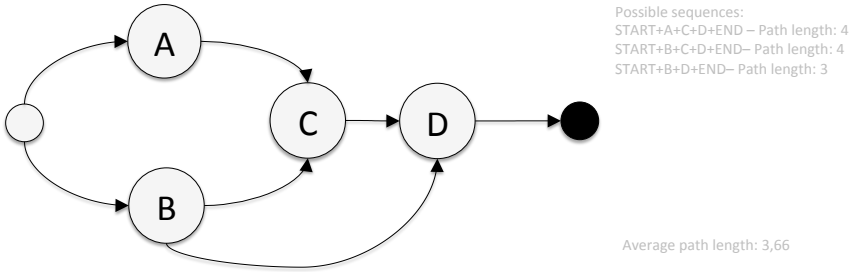


Figura 3.16: Número de caminhos expressado como um modelo de Markov

Definição: Tamanho dos caminhos distintos possíveis a partir do nó inicial até o final obtido através da contagem do número de nós de um caminho, excluindo os nós de início e fim. (Para um determinado modelo, isto é calculado como a média de todos os caminhos possíveis.)

Significado: Dificuldade para chegar a outro nó designado no modelo. Descrição do tamanho do modelo.

– **Complexidade ciclomática**: é uma métrica para avaliar a complexidade do comportamento de modelos considerando os *loops* (ciclicidade) dos caminhos possíveis. Esta métrica apresenta características importantes especialmente por representar processos com iterações, i.e. ela é adaptada para caracterizar os ciclos em geral, o envolvimento de diferentes entidades, relacionamentos nos ciclos e possíveis pontos de decisão que iniciam e reiniciam iterações dentro de um processo. Um modelo de processo com uma elevada ciclicidade pode ser mais suscetível a ser complexo, como e.g. em um modelo sequencial (baixa complexidade) a ciclicidade é 0, conforme ilustrado na Figura 3.17.

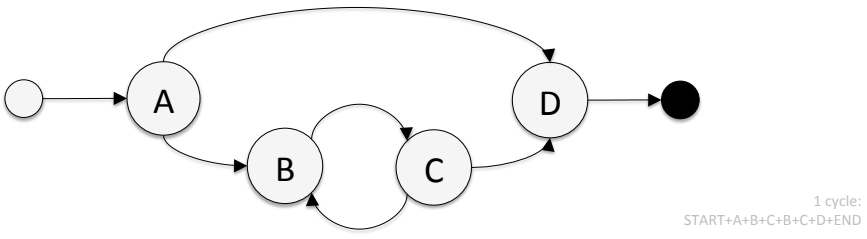


Figura 3.17: Número de caminhos expressado como um modelo de Markov

Definição: Número de caminhos linearmente independentes. Pode ser mensurado como $M = E - N + 2P$ onde E é o número de arcos, N é o número de nós e P é o número de nós de saída.

Significado: Avaliação do nível global de incerteza do processo; Determinação do grau de possível re-trabalho durante a execução do processo.

- **Fan-in/Fan-out**: é baseada no fluxo de dados entre os diferentes módulos de um programa. A informação pode ser passada entre os módulos de três maneiras: por um módulo ao invocar um outro; retornando um resultado do módulo invocado para o módulo que chamou; e por meio da troca de informação através de um estrutura de dados global.

  Definição: $(f_{in} \cdot f_{out})^2$ onde f_{in} é o número de nós que precedem um nó, e f_{out} é o número de nós que se seguem um nó. (Para um modelo, f_{in} e f_{out} são calculados separadamente como a média entre todos os nós.)

  Significado: utilizada para prever quais módulos são suscetíveis a erros; indica que o modelo está sobrecarregado; necessidade de decomposição do modelo em sub-modelos (hierarquização).

- **No. de sub-processos**: é uma métrica de modularidade para auxiliar a compreensão de um modelo de processo. Um alto valor para esta métrica indica menor complexidade do modelo.

  Definição: Número de sub-processos aninhados em um modelo.

  Significado: este tipo de decomposição melhora a compreensão de modelos onde é necessária uma visão em partes específicas; níveis de abstração em modelos de processo de negócio

3.5 Considerações

Neste capítulo foi apresentado a abordagem proposta para melhoria de modelos de processos. Na prática, o estratégia adotada baseado em um ciclo iterativo proporciona uma melhor compreensão aos gestores no direcionamento preciso do processo e como ele deve ser melhorado. A abordagem fornece uma base construtiva para a análise de processos, auxiliando os profissionais na avaliação de forma mais sistemática dos resultados, bem como nas iniciativas de melhoria nas organizações.

Nesta direção, foi descrito em detalhes os estágios que incluem a integração de simulação baseada em agentes, a mineração de processos e a avaliação de complexidade de modelos de processo de negócio através de métricas de complexidade. Foi apresentada a definição formal e algoritmica de um modelo hierárquico de Markov como uma técnica de mineração de processos para capturar a relação entre as atividades de nível macro (existente no modelo de processo de negócio) e os eventos de nível micro (registrados em um log de eventos). Também foi desenvolvido um procedimento EM para estimar os parâmetros de tal modelo. Esta abordagem pode ser utilizada como uma técnica de mineração de processo em cenários onde um log de eventos está disponível, juntamente com uma descrição de alto nível do processo de negócio. Todos os algoritmos descritos nesta seção encontram-se implementados na linguagem de programação Python[2] (versão 2.7). O código fonte do modelo hierárquico de está disponibilizado nos Apêndices B.1 e B.2.

[2] http://www.python.org/

Também demonstramos que a técnica proposta (Algoritmo 4) é capaz de descobrir os micro-modelos para cada macro-atividade. Preliminarmente, isto foi demonstrado em experimentos com um conjunto de padrões básicos de *workflow*. No entanto, no Capítulo 4 descrevemos cenários de aplicação, onde foi utilizada uma plataforma baseada em agentes para implementar processos de negócio e gerar o log de eventos por meio de simulação. Como ilustrado nestes cenários, a abordagem pode ser utilizada para analisar tanto a perspectiva de controle de fluxo como a perspectiva organizacional.

Este processo iterativo de simulação e mineração pode facilitar a identificação e introdução de melhorias no modelo do processo, pois os micro-modelos (i.e. comportamento de baixo nível) com grande complexidade sugerem que sejam introduzidas outras atividades ou caminhos no macro-modelo (i.e. comportamento de alto nível). No Capítulo 4 deste trabalho conduzimos diversos estudos de caso para verificação da aplicabilidade da abordagem proposta.

Capítulo 4

Estudos Exploratórios

Este capítulo descreve três cenários de aplicação e um estudo de caso com um log de eventos real para avaliação experimental da abordagem proposta, onde utilizamos a técnica de mineração de processos descrita na Seção 3.3 para extrair modelos hierárquicos no âmbito da melhoria de modelos de processo proposta no Capítulo 3. Os cenários envolvem: (1) processo de compra (Seção 4.1); (2) processo de indenização de seguro - manipulação de sinistro (Seção 4.2); (3) processo de empréstimo bancário (Seção 4.3); e (4) estudo de caso real com o processo de gestão de incidentes (Seção 4.5). Os modelos de processos de negócio apresentados neste capítulo possuem diferentes características visando assim alcançar diferentes resultados de complexidade.

4.1 Processo de compra

O objetivo deste cenário de aplicação é ilustrar como a abordagem proposta pode ser utilizada para facilitar o refinamento de um modelo de processo, de modo que este modelo se torne não só uma representação mais precisa do comportamento observado, mas também um modelo que alcança melhores resultados em termos das métricas descritas na Seção 3.4.

O cenário envolve um processo de compra, o qual é descrito em alto nível (i.e. o macro-modelo), tanto de forma textual como por meio de um diagrama BPMN (Figura 4.1). No nível micro, o processo é implementado como um conjunto de interações entre os agentes na plataforma AOR (Wagner 2004, Wagner et al. 2009). Durante a simulação, um log de eventos é registrado, o log de eventos contendo as micro-sequências que podem ser utilizadas para descobrir os micro-modelos associados com o comportamento dos agentes ao executar o processo de compra. Neste cenário, o Algoritmo 4 foi utilizado para re-descobrir os micro-modelos originais que foram utilizados para implementar o processo de compra na plataforma AOR. A Figura 4.2 apresenta a cadeia de Markov com probabilidades de transição que representa o diagrama BPMN da Figura 4.1. Para efeitos de comparação, vamos demonstrar uma versão inicial (versão 1) e a versão final (versão 2), entretanto poderia haver outras versões. Inicialmente, o processo é descrito como segue:

> Em uma empresa, um funcionário precisa de um certo produto (e.g. um cartucho de impressora) e envia uma solicitação do produto para o depósito. Se o produto está disponível no depósito, o produto é enviado para o funcionário. Caso contrário, o departamento de compras adquire o produto e o envia para o funcionário.

Note na Figura 4.2, que o macro-modelo abrange três atividades e uma decisão. Por falta de maiores informações, vamos supor que os dois resultados possíveis a partir desta decisão são igualmente prováveis, então a probabilidade de transição da atividade "Requisição" para "Despachar Produto" e "Comprar Produto" são ambas 0.5 (50%) na Figura 4.2. Como característica deste modelo de processo, ressaltamos a utilização dos padrões básicos de *workflow XOR-split* e *XOR-join*.

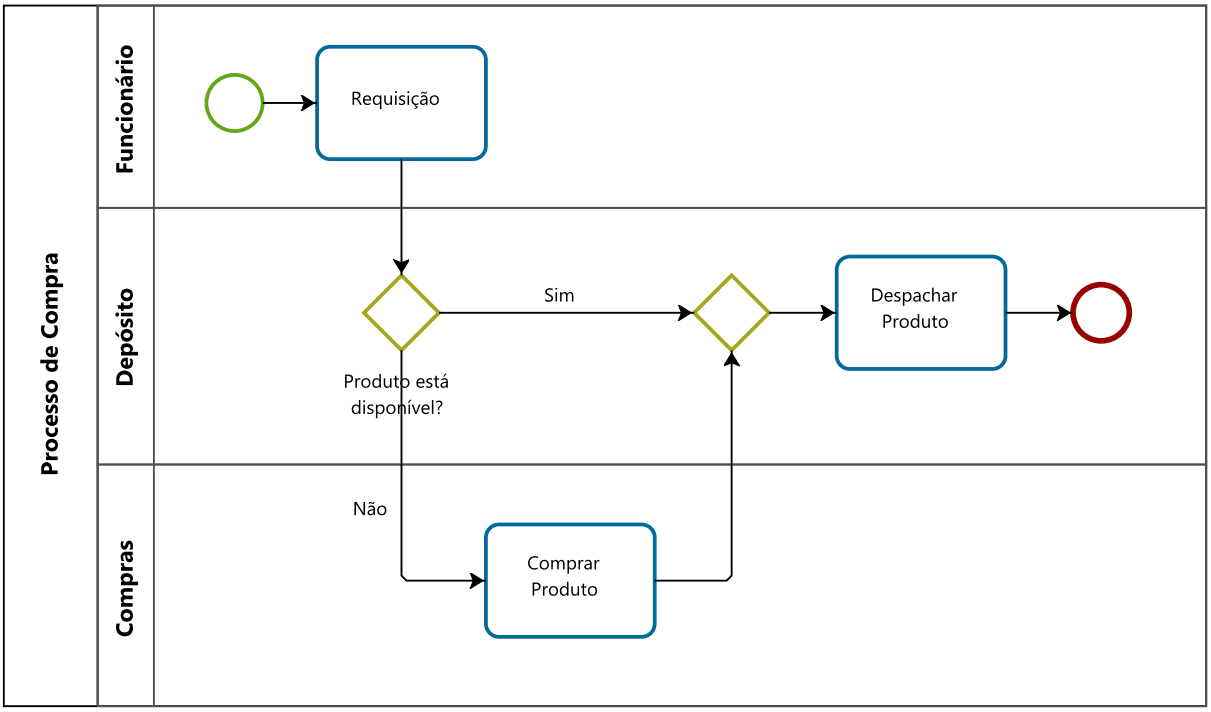


Figura 4.1: Diagrama BPMN para o processo de compra (versão 1)

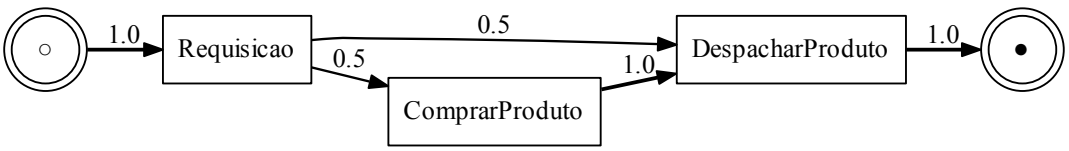


Figura 4.2: Representação do macro-modelo para o processo de compra (versão 1)

4.1.1 Simulação baseada em agentes

Em um cenário real, um log de eventos inicial pode ser obtido a partir de um sistema transacional, a fim de capturar as operações de baixo nível realizadas pelos participantes

no processo ou as interações (i.e. troca de mensagens) entre os participantes. Neste cenário de aplicação, para obter estes eventos de baixo nível o processo de compra foi modelado na plataforma AOR através da linguagem AORSL para especificar as trocas de mensagens entre os agentes. O cenário completo de simulação está disponibilizado no Apêndice C.1. Existem quatro tipos de agentes: o Funcionario, o Deposito, o departamento de compras representado como Compras e o Fornecedor. Há uma instância pré-existente do agente Deposito e uma instância pré-existente do agente Compras. No entanto, existem várias instâncias do agente Funcionario criadas em tempo de execução (cada instância é criada no início da execução da simulação e destruída quando a execução terminar). Poderíamos ter feito o mesmo para o agente Fornecedor, mas pela simplicidade consideramos apenas uma instância do Fornecedor, uma vez que este não tem efeito nos resultados. Cada troca de mensagens é representado sob a forma:

$$remetente \xrightarrow{mensagem} destinatário$$

O processo inclui as seguintes trocas de mensagens:

$$\text{Funcionario} \xrightarrow{\text{SolicitacaoEstoque}} \text{Deposito}$$
$$\text{Deposito} \xrightarrow{\text{RespostaEstoque}} \text{Funcionario}$$
$$\text{Funcionario} \xrightarrow{\text{BuscarProduto}} \text{Deposito}$$
$$\text{Deposito} \xrightarrow{\text{ProdutoPronto}} \text{Funcionario}$$
$$\text{Funcionario} \xrightarrow{\text{ProdutoRecebido}} \text{Deposito}$$
$$\text{Funcionario} \xrightarrow{\text{SolicitacaoCompra}} \text{Compras}$$
$$\text{Compras} \xrightarrow{\text{SolicitaInfo}} \text{Funcionario}$$
$$\text{Funcionario} \xrightarrow{\text{RespostaInfo}} \text{Compras}$$
$$\text{Compras} \xrightarrow{\text{ResultadoAprovacao}} \text{Funcionario}$$
$$\text{Compras} \xrightarrow{\text{PedidoCompra}} \text{Fornecedor}$$
$$\text{Fornecedor} \xrightarrow{\text{TermosPagamento}} \text{Compras}$$
$$\text{Compras} \xrightarrow{\text{ComprovantePagamento}} \text{Fornecedor}$$
$$\text{Fornecedor} \xrightarrow{\text{NotaEntrega}} \text{Deposito}$$
$$\text{Deposito} \xrightarrow{\text{ProdutoDisponivel}} \text{Funcionario}$$

Deve-se notar que o sistema AOR não tem conhecimento sobre as atividades de alto nível do processo. Ao invés disso, os agentes têm regras para implementar as trocas de mensagens e são reativos ao executar tais tarefas. Além disso, supõe-se que:

- Para a solicitação de compra ser aprovada, o Compras pode perguntar ao Funcionario um número arbitrário de vezes para obter mais informações sobre o pedido de compra. Isto significa que as trocas de mensagens SolicitaInfo e RespostaInfo podem ocorrer várias vezes (ou até mesmo não ocorrer).
- O departamento de compras pode não estar satisfeito com as condições de pagamento de um Fornecedor específico e pode optar por negociar os termos ou entrar

em contato com outro Fornecedor. Isso significa que PedidoCompra e TermosPagamento podem ocorrer várias vezes (mas eles devem executar pelo menos uma vez).

A Tabela 4.1 apresenta um exemplo do log de eventos registrado pelos agentes. Aqui, cada mensagem é representada como um determinado tipo ou significado. Por exemplo, SolicitacaoEstoque é uma mensagem enviada a partir de Funcionario para o Deposito. Enquanto esta mensagem parece estar relacionada com a atividade "Requisicao" para outras mensagens o relacionamento com a atividade de alto nível pode não ser tão claro.

instância	*remetente*	*mensagem*	*destinatário*	*data e hora*
1	Funcionario	SolicitacaoEstoque	Deposito	2013-02-02 11:26
1	Deposito	RespostaEstoque	Funcionario	2013-02-04 16:07
1	Funcionario	BuscarProduto	Deposito	2013-02-05 08:54
1	Deposito	ProdutoPronto	Funcionario	2013-02-06 10:23
1	Funcionario	ProdutoRecebido	Deposito	2013-02-07 15:47
2	Funcionario	SolicitacaoEstoque	Deposito	2013-02-12 09:31
2	Deposito	RespostaEstoque	Funcionario	2013-02-14 14:10
2	Funcionario	SolicitacaoCompra	Compras	2013-02-15 16:35
2	Compras	SolicitaInfo	Funcionario	2013-02-18 17:21
2	Funcionario	RespostaInfo	Compras	2013-02-19 10:52
2	Compras	ResultadoAprovacao	Funcionario	2013-02-20 12:05
2	Compras	PedidoCompra	Fornecedor	2013-02-21 15:19
...	...	...	...	...

Tabela 4.1: Extrato de um log de eventos

4.1.2 Mineração de processo

A partir de um log de eventos na forma de Tabela 4.1 é possível obter diferentes tipos de micro-sequências, dependendo de qual coluna é escolhida para análise. Aqui há três opções possíveis: a coluna *remetente*, *mensagem* e *destinatário*. A coluna *mensagem* pode ser usada para estudar a sequência de mensagens, enquanto que as colunas *remetente* e *destinatario* podem ser utilizadas para derivar os modelos de interação. Em qualquer caso, os eventos podem ser agrupados pela instância e classificados pela data e hora. Por exemplo, para a instância 1 temos a sequência de mensagens,

SolicitacaoEstoque→RespostaEstoque→BuscarProduto→ProdutoPronto→ProdutoRecebido

e sequência de *remetentes*,

Funcionario → Deposito → Funcionario → Deposito → Funcionario

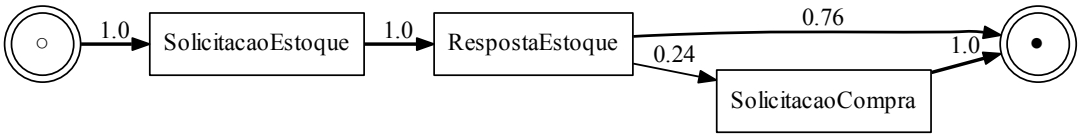
e sequência de *destinatários*,

Deposito → Funcionario → Deposito → Funcionario → Deposito.

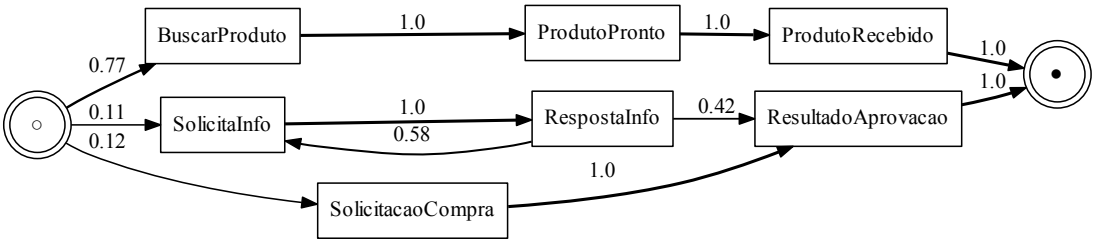
Estes diferentes tipos de micro-sequências nos permitem estudar tanto a perspectiva de controle de fluxo (a partir da sequência de mensagens) e a perspectiva organizacional (a partir de sequência quem enviou ou recebeu a mensagem). Ambas as perspectivas são bem conhecidas na literatura de mineração de processos (Mans et al. 2008, Bozkaya et al. 2009). Enquanto a perspectiva de controle de fluxo aborda a sequência de atividades no processo, a perspectiva organizacional permite estudar outros aspectos, tais como a entrega do trabalho (*handover of work*) entre os agentes, conforme descrito em maiores detalhes na Seção 2.3.

Perspectiva de controle de fluxo

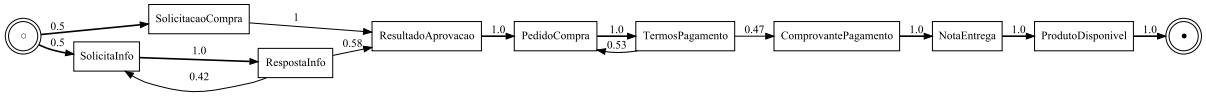
As micro-sequências juntamente com o macro-modelo da Figura 4.5, são fornecidos como entrada para o Algoritmo 4, que produz o micro-modelos mostrados na Figura 4.3. Nota-se que na Figura 4.3 existe comportamento duplicado nas atividades "Despachar Produto" e "Comprar Produto", resultando em micro-modelos mais complexos e maiores. Estes micro-modelos foram avaliados utilizando as métricas definidas na Seção 3.4 e os resultados são apresentados na Tabela 4.2.



(a) Requisicao



(b) Despachar produto



(c) Comprar produto

Figura 4.3: Micro-modelos para o processo de compra (versão 1)

Os resultados na Tabela 4.2 refletem o fato que o macro modelo não representa todas as informações existentes na operacionalização do processo de compras, o que dificulta o

Modelo	Métricas						
	NAN	RD	NP	PL	CC	FIO	SUB
Macro modelo	1.00	0.33	2.00	2.50	2.00	3.13	–
Micro Modelo (Requisicao)	1.00	0.33	2.00	2.50	2.00	1.77	–
Micro Modelo (Despachar Produto)	1.22	0.17	4.00	3.25	4.00	2.16	–
Micro Modelo (Comprar Produto)	1.18	0.13	6.00	9.33	4.00	2.63	–
Modelo Completo [Eq. (3.1)]	1.07	0.27	3.00	3.76	2.67	2.66	3

Tabela 4.2: Métricas aplicadas no processo de compra (versão 1)

entendimento e desenvolvimento dos sistemas correspondentes, i.e. os micro-modelos na Figura 4.3 são relativamente mais complexos quando comparados com o macro-modelo da Figura 4.2. Em particular, as métricas NP (No. de caminhos) e PL (tamanho do caminho) são significativamente mais elevados para os micro-modelos, quando comparados com os mesmos parâmetros para o macro-modelo. Isto significa que o modelo hierárquico é um pouco "desequilibrado", no sentido em que o macro-modelo é significativamente menos complexo do que as micro-modelos e portanto ele é provavelmente uma representação mais simplificada do processo de negócio. Parece haver bem mais comportamento nos micro-modelos do que no macro-modelo.

Portanto, especialistas em negócios são encorajados a descrever o processo em maiores detalhes. Tal descrição pode ser como segue:

> Em uma empresa, um funcionário precisa de um certo produto (e.g. um cartucho de impressora) e envia uma solicitação do produto para o depósito. Se o produto está disponível no depósito, o produto é enviado para o funcionário. Caso contrário, o produto deve ser comprado de um fornecedor externo. Todas as compras devem ser aprovadas pelo departamento de compras. Se a compra não for aprovada, o processo termina nesse ponto. Por outro lado, se a compra for aprovada, o departamento encomenda e paga pelo produto para o fornecedor. O fornecedor entrega o produto ao depósito e o depósito despacha o produto para o funcionário.

Esta nova versão do processo é ilustrada em BPMN na Figura 4.4 e é expressa como uma cadeia de Markov (i.e. macro-modelo) na Figura 4.5. Por outro lado, para minerar o modelo hierárquico, vamos precisar de um log de eventos com o comportamento em tempo de execução para este processo. Ao invés de implantar o modelo na organização e esperar pelo log de eventos ser registrado, usamos o simulador AOR para construir um cenário de simulação com base na versão anterior do processo (i.e. o modelo hierárquico das Figuras 4.2 e 4.3.

Se o comportamento do processo de alto nível havia sido alterado nesta nova versão (e.g. se duas atividades de alto nível haviam trocado a sua ordem de execução), então seria preciso adaptar o cenário de simulação, a fim de refletir essas alterações. No entanto, aqui o novo macro-modelo é apenas um refinamento do anterior, por isso podemos usar o modelo hierárquico anterior como base para a simulação, sem a necessidade de novas alterações. Mesmo que o modelo hierárquico nas Figuras 4.2 e 4.3 não seja um

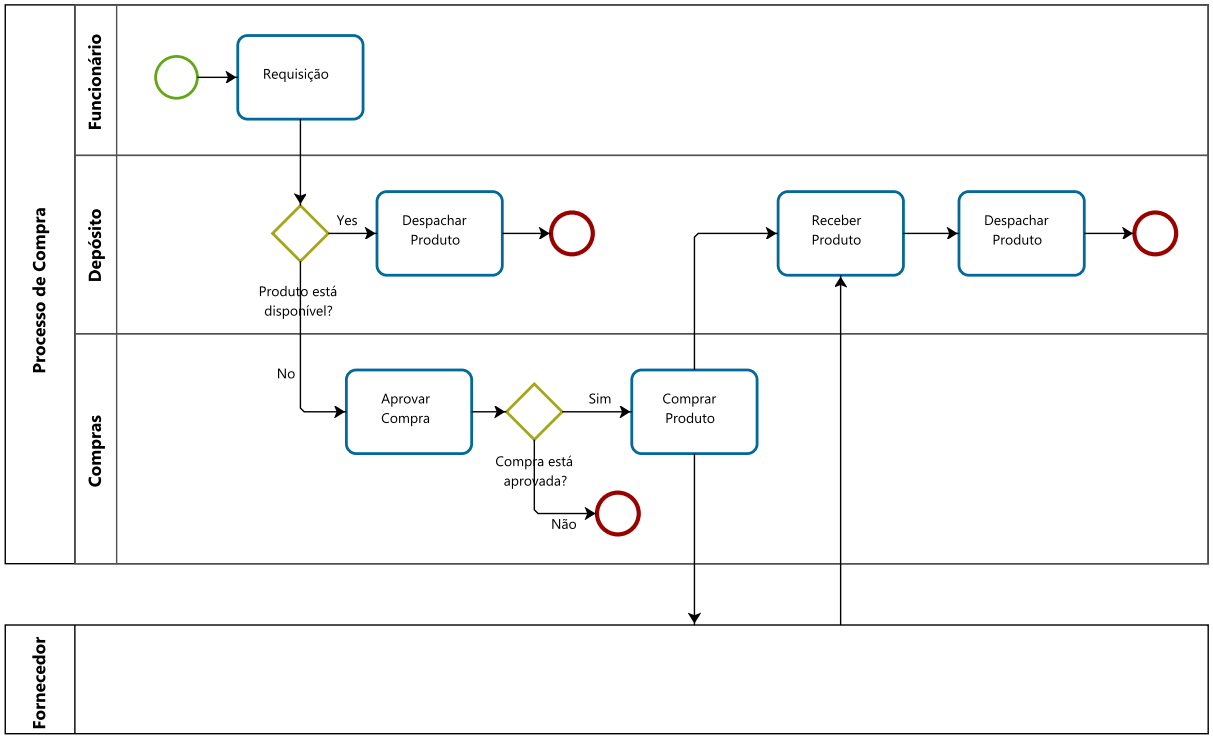


Figura 4.4: Descrição de alto nível do processo de compra (versão 2)

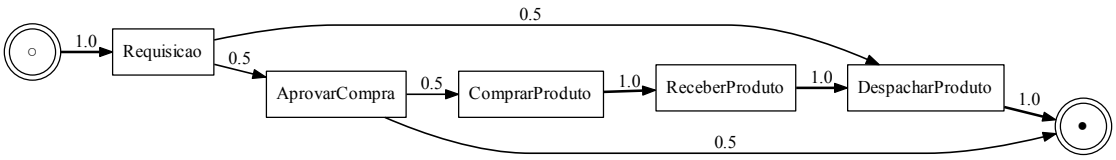
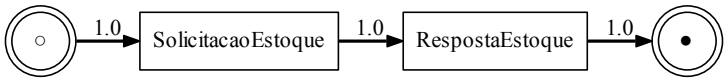


Figura 4.5: Representação do macro-modelo para o processo de compra (versão 2)

modelo equilibrado, para fins de simulação ainda é um modelo perfeitamente válido para a reprodução do comportamento do processo.

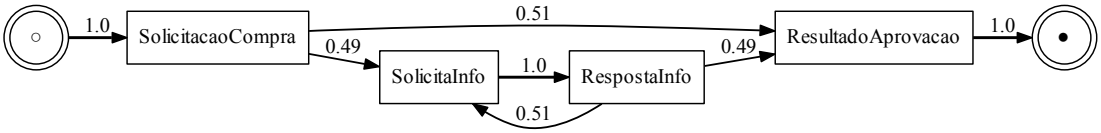
Ao executar o cenário de simulação no AOR, é possível coletar um novo e possivelmente maior log de eventos para minerar a próxima versão do modelo hierárquico. Os logs de eventos que são gerados pela estrutura AOR são em formato XML, mas eles podem ser facilmente convertidos para o formato descrito na Tabela 4.1, conforme informado na Seção 3.2.2. Neste experimento, fizemos uma simulação com 10.000 passos, que produziu um log de eventos com 140 instâncias do processo e um total de 1.136 eventos. O log de eventos, juntamente com o macro-modelo da Figura 4.5, foram fornecidos como entrada para o Algoritmo 4, que produziu os micro-modelos mostrados na Figura 4.6.

Este novo modelo hierárquico (i.e. Figuras 4.5 e 4.6) foi avaliado usando as mesmas métricas. Conforme apresentado na Tabela 4.3, note que este modelo se apresenta mais

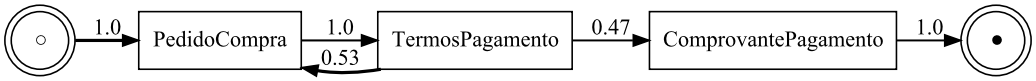


(a) Requisicao

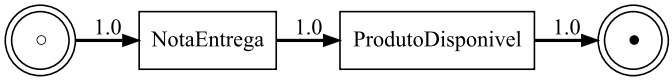
(b) Despachar produto



(c) Aprovar compra



(d) Comprar produto



(e) Receber produto

Figura 4.6: Micro-modelos para o processo de compra (versão 2)

equilibrado, porque os micro-modelos são, em geral, significativamente de menor complexidade, enquanto que o macro-modelo é apenas um pouco mais complexo, i.e. apresenta duas atividades a mais. Comparando a última linha da Tabela 4.3 com a última linha da Tabela 4.2 em geral, o novo modelo é menos complexo do que o anterior. Em particular, NP (No. de caminhos), PL (comprimento médio do caminho) e CC (complexidade ciclomática) são significativamente menores. Isto significa que o modelo na Figura 4.4 não é apenas uma descrição mais exata do processo, mas também o comportamento de baixo nível associado com cada atividade de alto nível é mais simples e mais fácil de entender.

Perspectiva organizacional

Em relação à perspectiva organizacional, com as micro-sequências para os remetentes e para os destinatários obtivemos os resultados ilustrados na Figura 4.7 e na Figura 4.8, respectivamente. Os resultados na Figura 4.7 apresenta algumas incompatibilidades, em

Modelo	Métricas						
	NAN	RD	NP	PL	CC	FIO	SUB
Macro modelo	1.14	0.23	3.00	3.00	3.00	2.82	–
Micro Modelo (Requisicao)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro Modelo (Despachar Produto)	0.80	0.27	1.00	3.00	1.00	1.00	–
Micro Modelo (Aprovar Compra)	1.17	0.29	3.00	4.00	3.00	5.06	–
Micro Modelo (Comprar Produto)	1.00	0.33	2.00	4.00	2.00	3.13	–
Micro Modelo (Receber Produto)	0.75	0.38	1.00	2.00	1.00	1.00	–
Modelo Completo [Eq. (3.1)]	1.02	0.28	2.30	3.00	2.30	2.53	5

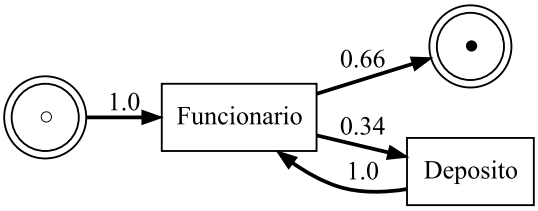
Tabela 4.3: Métricas aplicadas no processo de compra (versão 2)

comparação com a sequência real de troca de mensagens associadas a cada atividade de nível macro. Por exemplo, o ciclo entre Funcionario e Deposito que aparece na Figura 4.7(a) ("Requisicao") deveria ter aparecido na Figura 4.7 (b) ("Despachar produto"). Novamente, isto está relacionado com a forma com que as macro-sequências estão sendo inicializadas e a solução que o Algoritmo 4 converge. Além disso, o Deposito foi capturado na Figura 4.7(c) ("Aprovar compra"), mas não deveria ter ocorrido desta forma. A interação entre Compras e Fornecedores na Figura 4.7(d) ("Comprar produto") está correto, mas esses mesmos agentes aparecem na Figura 4.7(e) ("Receber produto"), onde apenas Deposito e Funcionario devem aparecer.

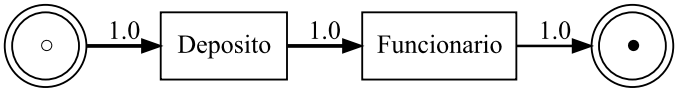
Os resultados na Figura 4.8 são mais precisos. Os micro-modelos na Figura 4.8(a), (b) e (c) são absolutamente corretos. O micro-modelo da Figura 4.8(d) ("Comprar produto") também está correto, a não ser que, na realidade, a atividade termina com Fornecedor recebendo uma mensagem (ComprovantePagamento), em vez de Compras. A troca de mensagens que está faltando em "Comprar produto"' acabou sendo capturada em "Receber produto". Na verdade, o micro-modelo na Figura 4.8(e) ("Receber produto") tem um extra Fornecedor pertencente a atividade "Comprar produto".

4.1.3 Comparação com o ProM

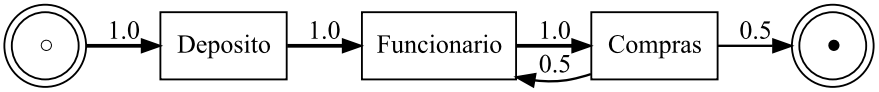
Como descrito na Seção 2.3.4, na área de mineração de processos, o *framework* ProM (van Dongen et al. 2005) é uma ferramenta de referência que inclui a implementação de muitas técnicas de mineração de processos disponíveis atualmente. Nesta seção, o nosso objetivo é comparar algumas técnicas de mineração disponíveis no ProM com a abordagem desenvolvida neste trabalho, a fim de destacar as diferenças e vantagens do modelo hierárquico proposto na abstração e entendimento de modelos de processo. Embora o ProM inclui vários plug-ins com recursos avançados, aqui fazemos uso de apenas os recursos mais básicos que estão relacionados com as perspectivas analisadas anteriormente, ou seja, a perspectiva de controle de fluxo e a perspectiva organizacional. Para isso, usamos o *heuristics miner* (Weijters et al. 2006) e *socialnetwork miner* (Song and van der Aalst 2008), respectivamente.



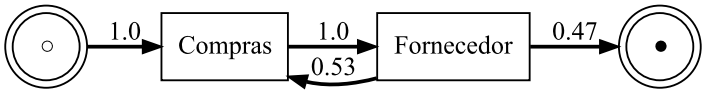
(a) Requisicao



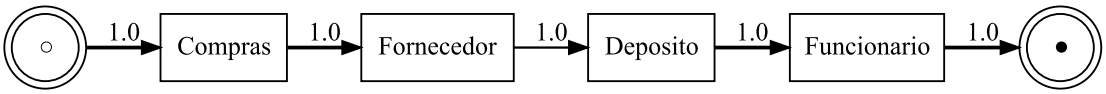
(b) Despachar produto



(c) Aprovar compra



(d) Comprar produto

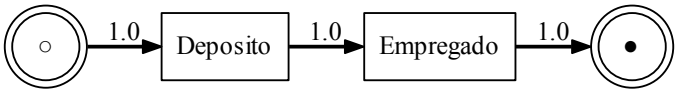


(e) Receber produto

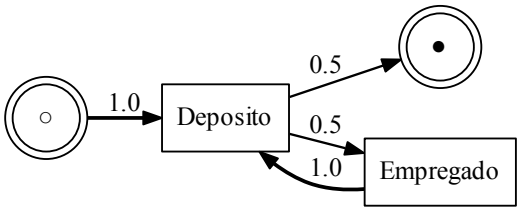
Figura 4.7: Interação entre agentes a partir da perspectiva dos agentes que enviaram as mensagens (remetentes)

Em primeiro lugar, convertemos o log de eventos do AOR no formato MXML usado pelo ProM (van Dongen and van der Aalst 2005) . Em seguida, abrimos o arquivo no ProM e executamos o plug-in *heuristics miner*, que produziu o modelo mostrado na Figura 4.9. Note que na Figura 4.9 todo o comportamento de baixo nível está em um único modelo. É possível reconhecer algumas características do processo de negócio: por exemplo, o ciclo entre SolicitaInfo e RespostaInfo é imediatamente reconhecível, bem como o ciclo entre PedidoCompra e TermosPagamento. Seguindo o fluxo também é possível reconhecer um *OR-split* em RespostaEstoque e um *OR-join* em BuscarProduto. No entanto, a partir deste modelo, é difícil identificar a relação entre esses eventos de baixo nível e as atividades de alto nível da Figura 4.4

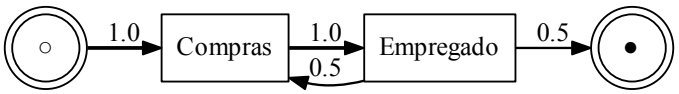
Em contraste, a nossa abordagem produz os resultados mostrados na Figura 4.6, onde cada grafo descreve o comportamento de baixo nível que ocorre dentro de cada uma das atividades de alto nível. Essa vantagem hierárquica de relacionamento entre o baixo e alto nível que queremos destacar em comparação com as técnicas tradicionais de mineração de processos disponíveis no ProM. Na prática, a análise do comportamento baseado exclusivamente em eventos nível micro registrados em um log de eventos (como na Figura 4.9) muitas vezes leva a modelos muito grandes e complexos que são difíceis de interpretar e que são conhecidos como *modelos spaghetti*. Uma vantagem da nossa abordagem é que



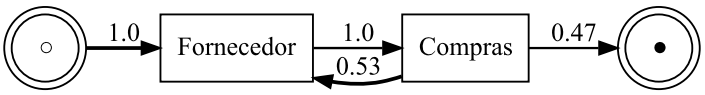
(a) Requisicao



(b) Despachar produto



(c) Aprovar compra



(d) Comprar produto

(e) Receber produto

Figura 4.8: Interação entre agentes a partir da perspectiva dos agentes que receberam as mensagens (destinatários)

esse comportamento é dividido em um conjunto de atividades de nível macro e, portanto, torna-se mais fácil de entender e interpretar. Visando facilitar o entendimento, esta abstração de macro-modelo e micro-modelos está representada através da linha tracejada na Figura 4.9.

Conforme observado na Figura 4.9, o modelo sem nenhum tipo de hierarquização dificulta o entendimento dada a complexidade dos modelos gerados que são modelos muito grandes e difíceis de interpretar. A complexidade do modelo parece ser muito maior do

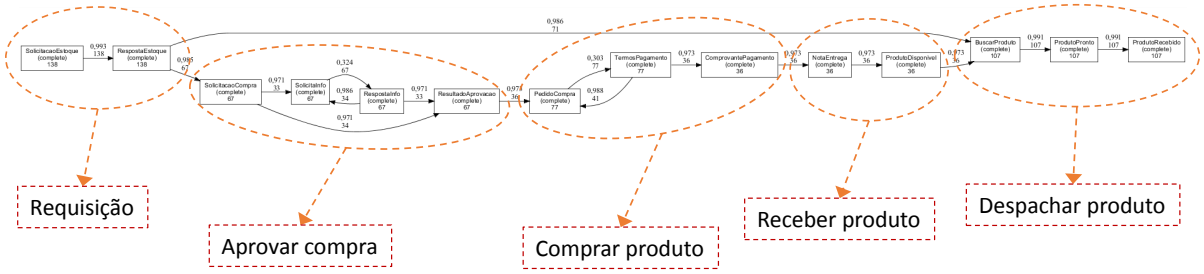
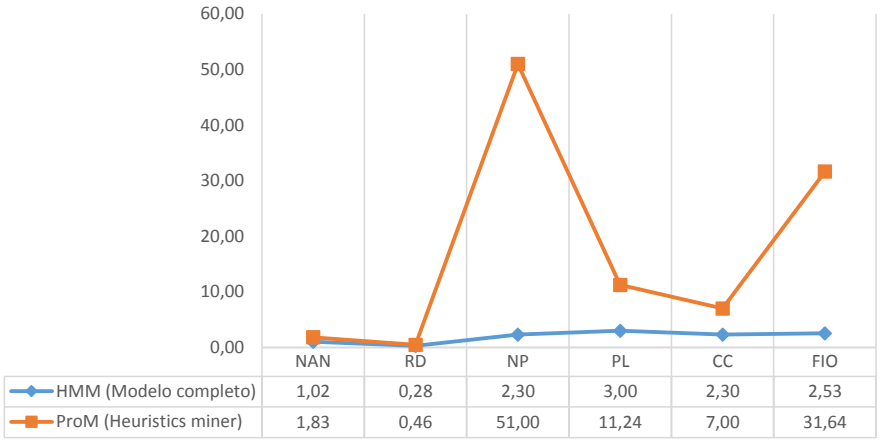


Figura 4.9: Resultados obtidos utilizando a abordagem *heuristics miner* no ProM

que a complexidade encontrada nos modelos hierárquicos para o mesmo processo descrito na Seção 4.1 (versão 2), conforme visto no comparativo ilustrado na Figura 4.10.



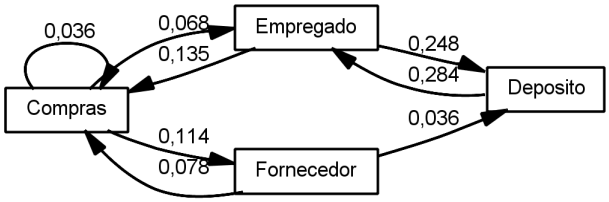
	NAN	RD	NP	PL	CC	FIO
HMM (Modelo completo)	1,02	0,28	2,30	3,00	2,30	2,53
ProM (Heuristics miner)	1,83	0,46	51,00	11,24	7,00	31,64

Figura 4.10: Comparativo das métricas para o processo de compra

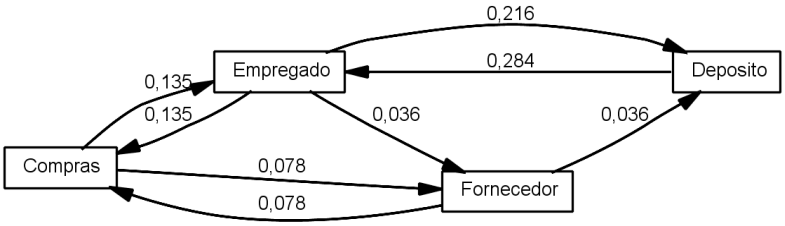
Observando o gráfico ilustrado na Figura 4.10, podemos destacar algumas questões importantes que são recomendações a gestores de TI/analistas de negócio no sentido de melhorar a compreensão do modelo de processo. A métrica NP (No. de caminhos) apresenta um pico elevado no modelo minerado através da técnica *heuristics miner*. Esta situação indica a possibilidade de caminhos redundantes no interior do processo, que pode levar falta de clareza no processamento do processo, i.e. o fato de ocorrer a possibilidade ir de qualquer atividade para qualquer outra atividade na execução do processo aponta para alta complexidade do modelo que leva a dificuldade de entendimento do modelo. Outra métrica que deve ser destacada é FIO (Fan-in/Fan-out) que também apresenta um pico elevado no modelo minerado através da técnica *heuristics miner*. Esta ocorrência indica que modelos com valor elevado para esta métrica são difíceis de utilizar e com grande probabilidade de erros de projeto, i.e. o modelo está sobrecarregado indicando a necessidade de decomposição do modelo em sub-modelos (hierarquização).

Uma análise do mesmo log de eventos com o *social network miner* disponível no ProM é apresentado na Figura 4.11. Estes resultados foram obtidos usando a métrica *handover of work* (van der Aalst et al. 2005) tanto para as colunas *remetente* e *destinatário* no log

de eventos da Tabela 4.1. Em ambos os grafos da Figura 4.11 é claro que o Funcionario tem uma interação com o Depósito e uma interação em separado com o Compras. Por outro lado, o Depósito e o Compras não interagem diretamente. De um modo semelhante, o Funcionario não interage diretamente com o Fornecedor; é o Compras, que interage com o Fornecedor. Por isso, algumas conclusões úteis podem ser tiradas sobre o processo, mesmo a partir de modelos que são obtidos a partir de eventos únicos de baixo nível. No entanto, os micro-modelos na Figura 4.7 e 4.8 tem a vantagem de mostrar as interações que ocorrem dentro de cada atividade de alto nível. Novamente, como no caso da perspectiva de controle de fluxo, dividindo o comportamento observado em um conjunto de atividades de alto nível facilita a análise do que é realizado no tempo de execução.



(a) perspecitva do remetente



(b) perspectiva do destinatário

Figura 4.11: Resultados obtidos utilizando a abordagem *social network miner* no ProM

4.2 Processo de indenização de seguro

Este cenário de aplicação é baseado em um processo de indenização de seguro para veículos com planos de cobertura total. O processo pode ser descrito como se segue:

> Um cliente com um plano de cobertura total do seguro danificou seu veículo e chama a companhia de seguros, a fim de apresentar um pedido de indenização. Depois do pedido ter sido preenchido com todos os detalhes, a seguradora atribui um perito para avaliar os danos. Quando o especialista retorna a avaliação de danos, a empresa decide se aprova ou não o pedido. Se a empresa não aprova, o pedido é indeferido e o processo termina aqui. Caso contrário, se o pedido for aprovado, o cliente terá seu carro consertado na oficina. Depois disso, a seguradora paga o conserto e espera por um período de 30 dias. Se o cliente não se pronuncia em 30 dias, assume-se que o reparo foi bem sucedido e o pedido está fechado. Caso contrário, se o cliente reclama no prazo de 30 dias, o processo volta para a etapa de avaliação de danos para uma re-avaliação e, possivelmente, um novo reparo.

Um modelo BPMN deste processo é mostrado na Figura 4.12. Em particular, ele mostra que existem três agentes neste processo: o Cliente, o Escritório de seguros, bem

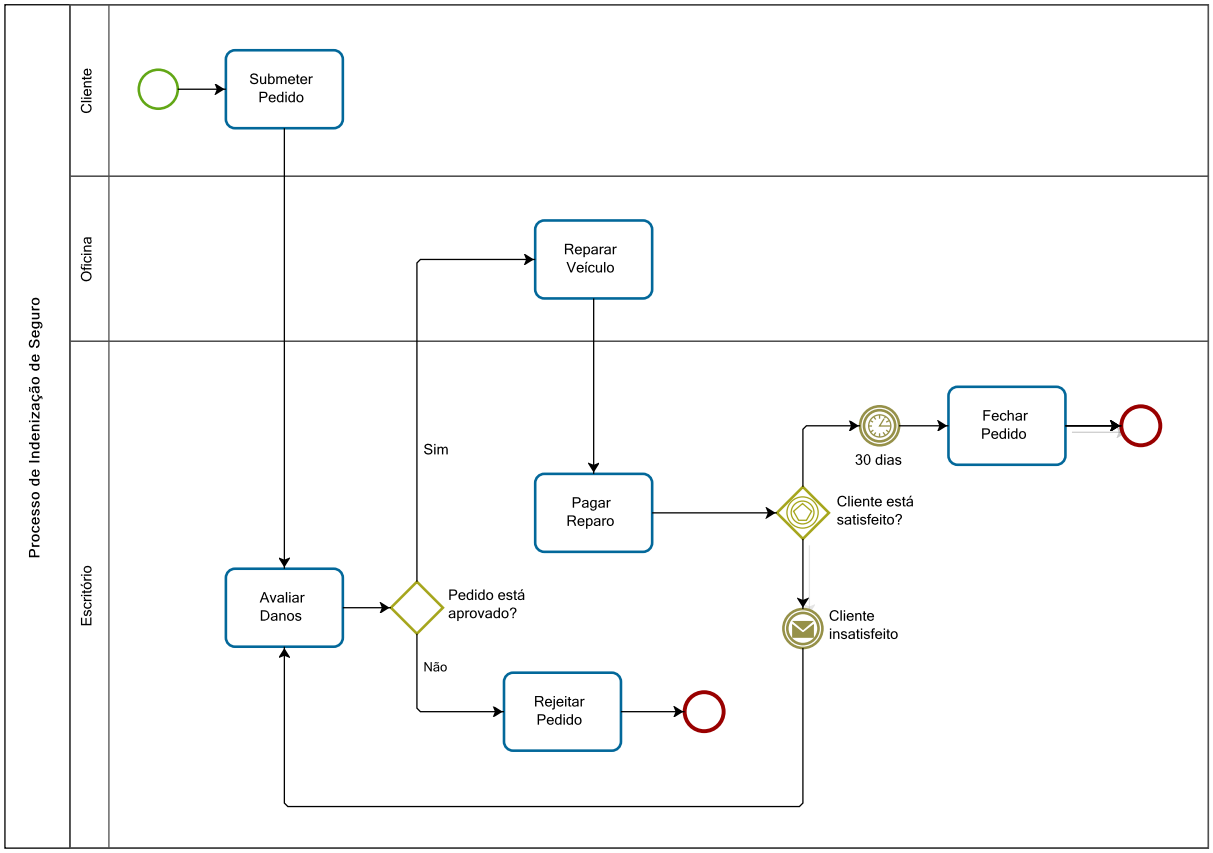


Figura 4.12: Modelo BPMN para o processo de indenização de seguro

como a Oficina. Cada um desses participantes desempenha um papel neste processo ao realizar algumas das atividades descritas acima. No entanto, ao realizar estas atividades, estes agentes interagem uns com os outros de uma forma que não é totalmente captada neste modelo. A Figura 4.13 mostra as trocas de mensagens que ocorrem entre os agentes em cada atividade do processo. Como característica deste modelo de processo, ressaltamos a utilização dos padrões básicos de *workflow* *XOR-split*, *XOR-join* e *loops*.

No final do processo na Figura 4.12, existe um *gateway*[1] baseado em eventos para representar a possibilidade do Cliente não ficar satisfeito com o reparo. No modelo da Figura 4.12, existem dois possíveis eventos: um temporizador que expira após 30 dias e um evento de mensagem que pode ser acionado pelo cliente, se ele não está satisfeito com o reparo. Isto significa que se o Cliente não enviar uma mensagem dentro de 30 dias, expira o temporizador e o processo segue para "Pedido encerrado"

Este evento de mensagem que segue o *gateway* baseado no evento da Figura 4.12 representa uma troca de mensagens entre o Cliente e o Escritorio. Esta troca de mensagens também está representada na Figura 4.13.

[1]Um *gateway* baseado em eventos é uma construção especial BPMN que representa uma decisão que é impulsionada por eventos, i.e. o primeiro evento a ocorrer decide qual ramo será seguido.

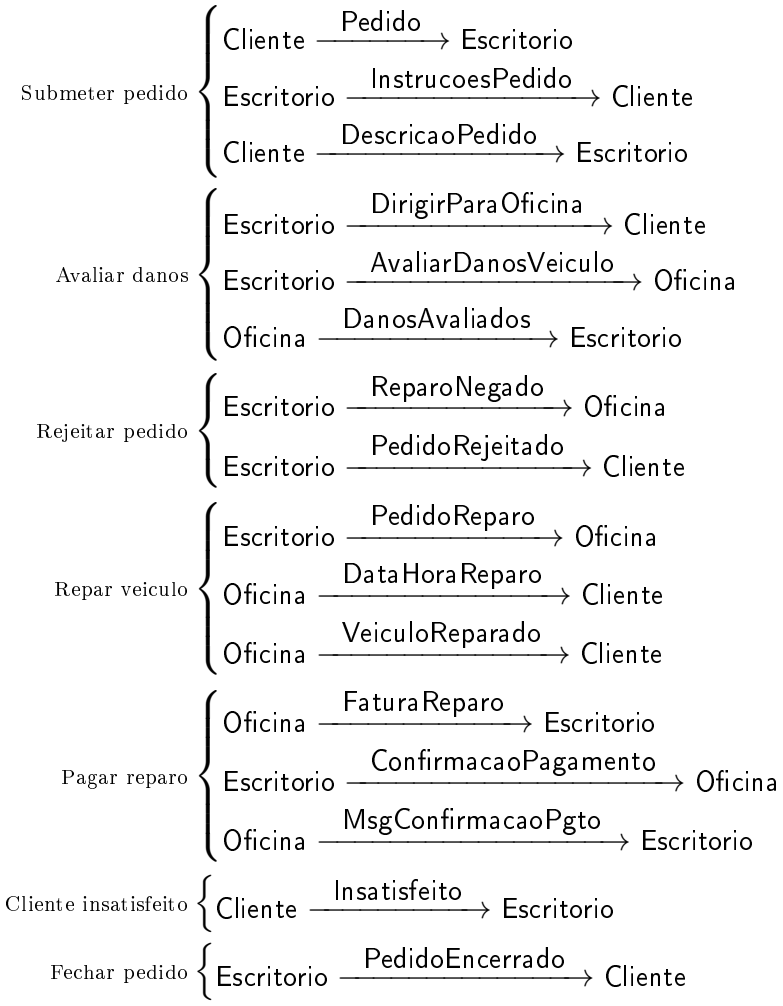


Figura 4.13: Mensagens trocadas no processo de indenização de seguro

4.2.1 Simulação baseada em agentes

Implementamos as trocas de mensagens da Figura 4.13 como um cenário de simulação no AOR. O cenário completo de simulação está disponibilizado no Apêndice C.2. Para efeitos de simulação, assumimos que 50% dos pedidos são rejeitados e, para os processos aprovados onde um reparo ocorre, há uma probabilidade de 20% do cliente não estar satisfeito com a reparo . Deve-se notar que esta informação refere-se às atividades no processo e, em um caso prático, tal informação pode ser fornecida por especialistas em negócios, pelo menos sob a forma de uma estimativa grosseira. Fornecido o modelo da Figura 4.12 e as probabilidades de transição entre as atividades daquele modelo, podemos dizer que o processo de indenização de seguro é descrito pelo macro-modelo ilustrado na Figura 4.14.

Através de simulação no AOR, geramos um log de eventos com 100 instâncias do processo (i.e. 100 micro-sequências). A Tabela 4.4 descreve um trecho do início do log de eventos, que inclui todos os eventos para a primeira instância do processo. Como explicado

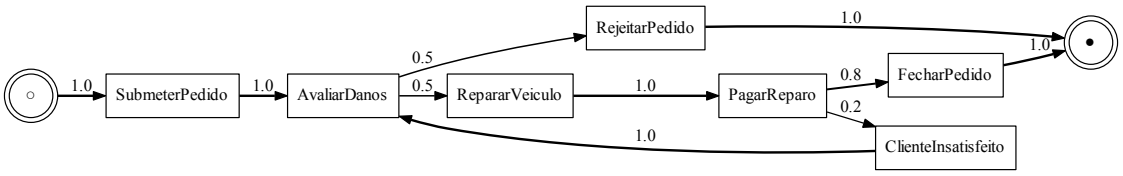


Figura 4.14: Macro-modelo para o processo de indenização de seguro

na Seção 1, este log de eventos capta as trocas de mensagens entre agentes, mas não as atividades que pertencem essas trocas de mensagens. Além disso, na Figura 3.3 ilustramos um exemplo simples onde X, Y e Z vieram da coluna *remetente* na Tabela 1.1, mas aqui vamos concentrar a nossa análise na coluna *mensagem* da Tabela 4.4. Isto significa que iremos analisar a sequência de mensagens, em oposição à sequência de *remetentes* ou a sequência de *destinatários* que poderia também ser analisada.

instância	*remetente*	*mensagem*	*destinatário*	*data e hora*
1	Cliente	Pedido	Escritorio	2
1	Escritorio	InstrucoesPedido	Cliente	3
1	Cliente	DescricaoPedido	Escritorio	5
1	Escritorio	DirigirParaOficina	Cliente	8
1	Escritorio	AvaliarDanosVeiculo	Oficina	9
1	Oficina	DanosAvaliados	Escritorio	12
1	Escritorio	PedidoReparo	Oficina	13
1	Oficina	DataHoraReparo	Cliente	14
1	Oficina	VeiculoReparado	Cliente	17
1	Oficina	FaturaReparo	Escritorio	18
1	Escritorio	ConfirmacaoPagamento	Oficina	20
1	Oficina	MsgConfirmacaoPgto	Escritorio	21
1	Escritorio	PedidoEncerrado	Cliente	32
2	Cliente	Pedido	Escritorio	39
2	Escritorio	InstrucoesPedido	Cliente	41
2	Cliente	DescricaoPedido	Escritorio	42
...	...	...	...	...

Tabela 4.4: Log de eventos a partir da simulação do processo de indenização de seguro

A partir da primeira instância do processo no log de eventos da Tabela 4.4, obtemos a seguinte micro-sequência:

Pedido → InstrucoesPedido → DescricaoPedido → DirigirParaOficina → AvaliarDanos-Veiculo → DanosAvaliados → PedidoReparo → DataHoraReparo → VeiculoReparado → FaturaReparo → ConfirmacaoPagamento → MsgConfirmacaoPgto → PedidoEncerrado

4.2.2 Mineração de processo

Nesta seção vamos analisar o log de eventos sob a perspectiva de controle de fluxo. Nesta direção, ao coletar a micro-sequência para cada instância de processo no log de eventos e, juntamente com o macro-modelo da Figura 4.14, estamos agora em uma posição onde podemos aplicar o Algoritmo 4 descrito na Seção 3.3, a fim de descobrir as micro-modelos associados a cada atividade do macro-modelo.

4.2.3 Análise e avaliação

A Figura 4.15 ilustra os resultados de mineração obtidos a partir da aplicação do Algoritmo 4. Estes resultados devem ser comparados com as trocas de mensagens que foram usadas para implementar o cenário de simulação e que são mostradas na Figura 4.13. Ao comparar a Figura 4.15 com Figura 4.13, verifica-se que:

(a) O micro-modelo para a atividade "Submeter pedido" está correto.

(b) O micro-modelo para a atividade "Avaliar danos" está faltando a mensgagem DanosAvaliados. Esta mensagem acabou sendo capturada nos micro-modelos de "Rejeitar pedido" and "Reparar veiculo", que são as duas atividades que podem seguir "Avaliar danos".

(c) Além da mensagem DanosAvaliados, o micro-modelo para a atividade "Rejeitar pedido" está correto.

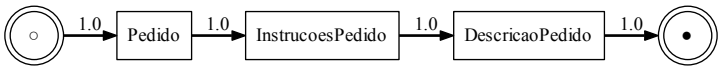
(d) Além da mensagem DanosAvaliados, o micro-modelo para a atividade "Reparar veiculo" está correto.

(e) No micro-modelo para a atividade "Pagar reparo" está faltando a mensagem MsgConfirmacaoPgto. esta mensagem acabou sendo capturada nos micro-modelos "Cliente insatisfeito" e "Fechar pedido", que são as duas atividade que podem seguir "Pagar reparo".

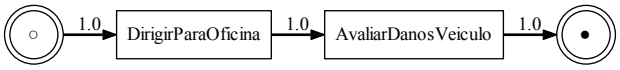
(f) Além da mensagem MsgConfirmacaoPgto, os micro-modelos "Cliente insatisfeito" e "Fechar pedido" estão corretos.

No geral, com exceção de alguns leves extravios de mensagens nestes micro-modelos (i.e. o fato de que algumas mensagens acabarem sendo capturadas na atividade que vem em seguida), a mineração de processos foi capaz de encontrar um modelo hierárquico que, em essência, captura o comportamento correto para cada atividade no processo. A precisão desta técnica de mineração foi demonstrada também no cenário de aplicação da Seção 4.1, que apresenta um cenário de aplicação diferente (um processo de compra).

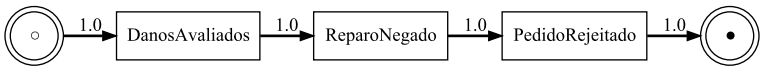
A Tabela 4.5 mostra os resultados da aplicação do conjunto de métricas da Seção 2.4 para o modelo hierárquico das Figuras 4.14 e 4.15.



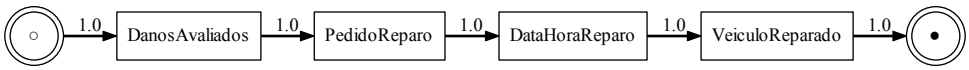
(a) Submeter pedido



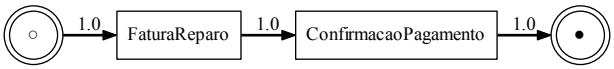
(b) Avaliar danos



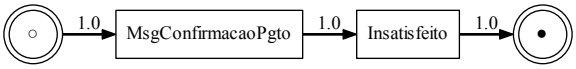
(c) Rejeitar pedido



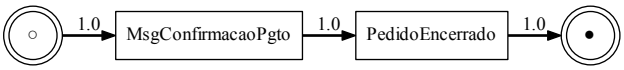
(d) Reparar veiculo



(e) Pagar reparo



(f) Cliente insatisfeito



(g) Fechar pedido

Figura 4.15: Micro-modelos para o processo de indenização de seguro

A partir dos resultados da Tabela 4.5 verifica-se que o macro-modelo da Figura 4.14 é realmente o componente com maior complexidade no modelo hierárquico. Em comparação com o macro-modelo, os micro-modelos são mais simples em todos os aspectos. Além

Modelo	Métricas						
	NAN	RD	NP	PL	CC	FIO	SUB
Macro-modelo	1.11	0.16	4.00	6.00	3.00	2.16	–
Micro-modelo (Submeter pedido)	0.80	0.27	1.00	3.00	1.00	1.00	–
Micro-modelo (Avaliar danos)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro-modelo (Rejeitar pedido)	0.80	0.27	1.00	3.00	1.00	1.00	–
Micro-modelo (Reparar veiculo)	0.83	0.21	1.00	4.00	1.00	1.00	–
Micro-modelo (Pagar reparo)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro-modelo (Cliente insatisfeito)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro-modeol (Fechar pedido)	0.75	0.38	1.00	2.00	1.00	1.00	–
Modelo Completo [Eq. (3.1)]	0.94	0.24	2.50	4.29	2.00	1.58	7

Tabela 4.5: Métricas aplicadas no processo de indenização de seguro

disso, podemos dizer que este modelo hierárquico é equilibrado, no sentido de que os micro-modelos são todos os modelos de complexidade similar. Em outras palavras, o comportamento de baixo nível parece estar distribuído uniformemente entre as atividades de alto nível neste processo.

4.3 Processo de empréstimo bancário

No cenário de aplicação do processo de indenização de seguro, os micro-modelos eram relativamente simples, no sentido em que eles consistiam em sequências lineares de eventos. Passamos agora para outro cenário de aplicação, onde iremos encontrar micro-modelos que são um pouco mais complexos. Este cenário de aplicação envolve um processo de aprovação para empréstimos bancários, e pode ser descrito da seguinte forma:

> O processo começa quando um cliente prepara e envia um pedido de empréstimo ao banco. Como um passo preliminar, um funcionário do banco vai analisar o pedido e verificar se ele está em conformidade com certas regras. Em particular, o funcionário irá verificar a situação financeira do cliente e determinar se o cliente é elegível para um empréstimo. Se o requerente ou o pedido de empréstimo não está de acordo com as regras, o processo termina imediatamente. Caso contrário, ele passa para a fase de análise de crédito, onde o banco entra em contato com uma agência de crédito externo para determinar se o cliente tem algum pagamento pendente em outros lugares. Dependendo das informações recebidas da agência de crédito, o banco atribui um fator de risco para o pedido de empréstimo. Aplicações com um fator de risco acima de um certo limite deve ser aprovada pelo gerente do banco, que vai decidir se os recursos devem ser liberados ou não. Para aplicações com um baixo fator de risco, o funcionário pode liberar os recursos imediatamente. Em qualquer caso, o cliente deve ser notificado do resultado.

A Figura 4.16 exibe um modelo BPMN para este processo. Há quatro participantes neste processo – o Cliente, o Funcionario do banco , o Gerente do banco, a Agencia de crédito – e estes participantes serão representados por agentes no cenário de simulação do

AOR. Além disso, quando o funcionário do banco executa suas funções, há uma interação com um sistema bancário. Uma vez que esta interação é uma característica relevante do comportamento de baixo nível deste processo, o sistema bancário é representado como um quinto agente chamado Sistema. Como característica deste modelo de processo, ressaltamos a utilização de padrões básico de *workflow XOR-split*, *XOR-join*, *AND-splits* e *AND-joins*.

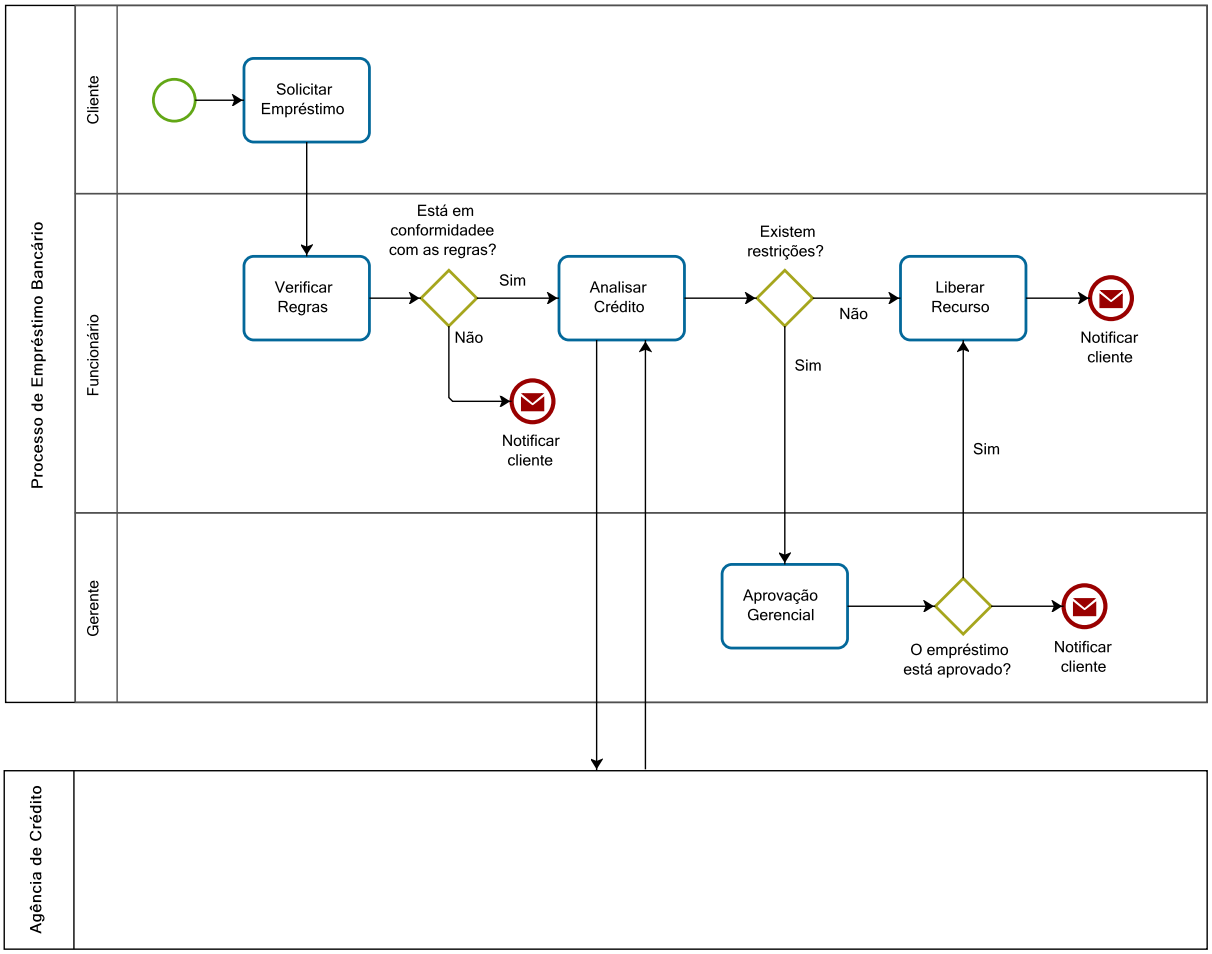


Figura 4.16: Modelo BPMN para o processo de empréstimo bancário

4.3.1 Simulação baseada em agentes

A Figura 4.17 exibe o comportamento de baixo nível do processo, como ele foi implementado no AOR. O cenário completo de simulação está disponibilizado no Apêndice C.3. Em particular, durante a execução da atividade "Solicitar emprestimo" existe uma mensagem DetalhesAdicionais que é trocada entre o Cliente e o Funcionario e essa troca pode ocorrer várias vezes, pois o Cliente pode fornecer os detalhes para o pedido de empréstimo de uma só vez ou em mensagens separadas. Além disso, existe um comando AtualizarSolEmprestimo que o Funcionario envia para o Sistema em várias atividades. Esta interação é opcional no

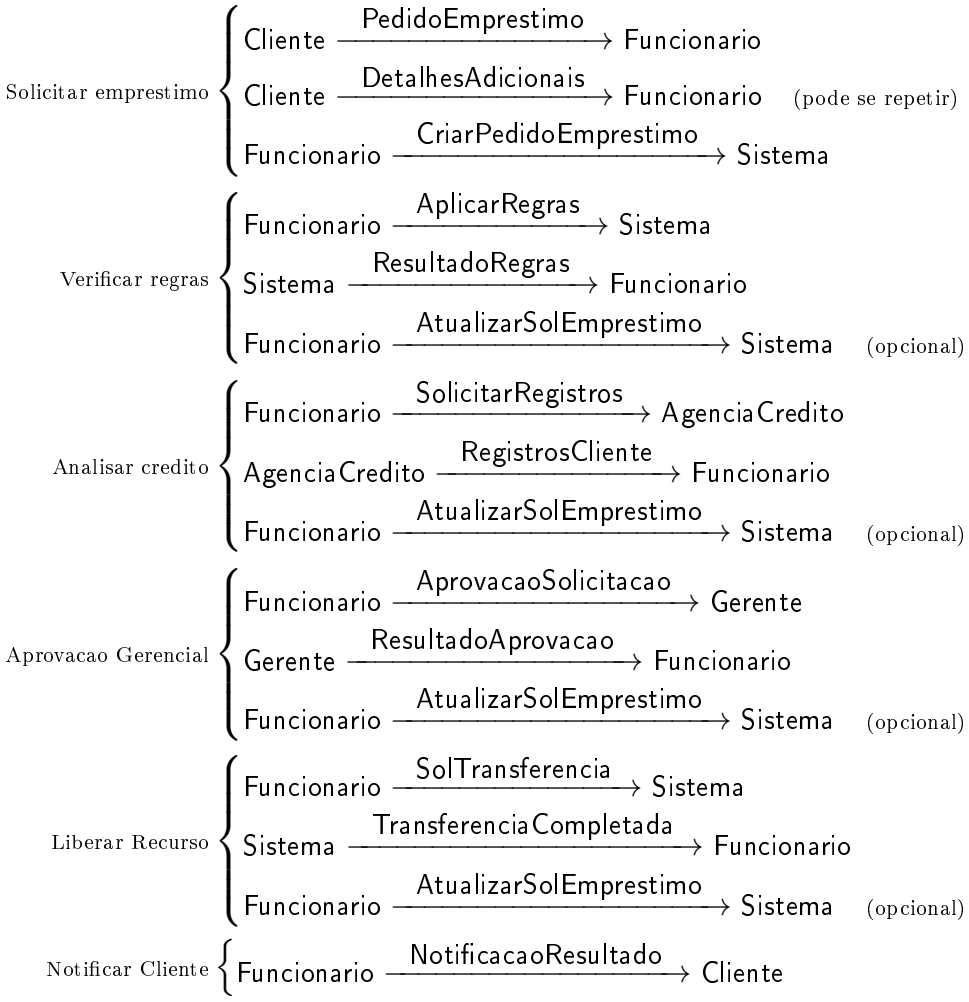


Figura 4.17: Troca de mensagens no processo de empréstimo bancário

sentido em que pode ou não ocorrer, que depende no caso de haver alguma mudança com o pedido de empréstimo.

Através da simulação no AOR, geramos um log de eventos com 500 instâncias do processo. Assim como no cenário de aplicação do processo de indenização de seguro (Seção 4.2), que extraiu os micro-sequências da coluna *mensagem*. Quanto ao macro-modelo, assumimos uma probabilidade de 20% do empréstimo negado após a atividade "Verificar regras", uma probabilidade de 70% do empréstimo precisando "Aprovacao gerencial", e probabilidade de 40% do empréstimo ser aprovado pelo Gerente. O macro-modelo resultante é ilustrado na Figura 4.18.

4.3.2 Mineração de processo

Da mesma forma como no exemplo da Seção 4.2.2, nesta seção vamos analisar o log de eventos sob a perspectiva de controle de fluxo. Nesta direção, fornecendo este macro-

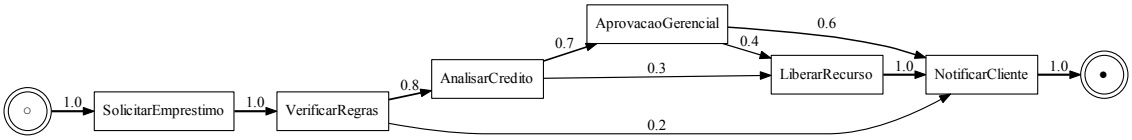
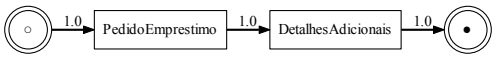
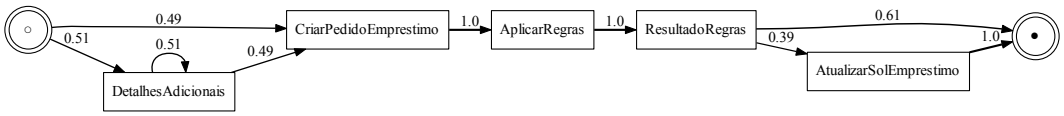


Figura 4.18: Macro-modelo para o processo de empréstimo bancário

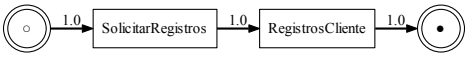
modelo juntamente com as micro-sequências para o Algoritmo 4 que implementa a técnica de mineração descrita na Seção 3.3, obtivemos os micro-modelos mostrados na Figura 4.19.



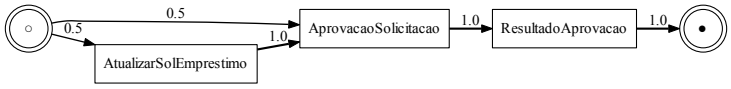
(a) Solicitar emprestimo



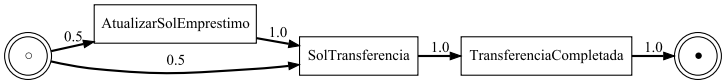
(b) Verificar regras



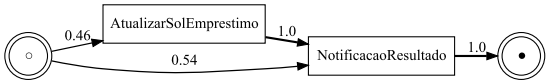
(c) Analisar Credito



(d) Aprovacao gerencial



(e) Liberar recurso



(f) Notificar cliente

Figura 4.19: Micro-modelos para o processo de empréstimo bancário

4.3.3 Análise e avaliação

Comparando a Figura 4.19 com a Figura 4.17, existem duas principais diferenças. Primeiro, o *loop* das mensagens DetalhesAdicionais e CriarPedidoEmprestimo acabou na atividade "Verificar regras", em vez da atividade "Solicitar emprestimo". Isso ocorre porque a atividade "'Solicitar emprestimo" é sempre seguida pela atividade "Verificar regras", por isso é impossível saber exatamente quais eventos pertencem a uma atividade e os eventos que pertencem a outra. A segunda diferença é que, em diversas atividades, a mensagem AtualizarSolEmprestimo acabou sendo capturada na atividade que vem depois. Por exemplo, a mensagem AtualizarSolEmprestimo está faltando no final da atividade "Analisar credito", mas ela aparece no início de "Aprovacao gerencial" e "Liberar recurso", bem como no início da atividade "Notificar cliente".

Além destas pequenas diferenças, o procedimento de mineração foi capaz de capturar o comportamento de cada atividade desenvolvida, embora os micro-modelos resultantes não parecem tão equilibrados como no caso do processo de indenização de seguro, i.e. existem atividades lineares pequenas e outras com *loop* e maiores. Uma avaliação deste modelo hierárquico com base nas métricas da Seção 2.4 deve fornecer uma indicação deste fato. Realmente, os resultados da Tabela 4.6 sugerem que existe um micro-modelo que tem maior complexidade do que o restante ("Verificar regras" com $NP = 4.00$). De acordo com os números, a complexidade deste micro-modelo é comparável ao modelo macro na Figura 4.18. Por outro lado, há dois micro-modelos que parecem ser um pouco mais simples ("Solicitar emprestimo" e "Analisar credito"). A complexidade destes dois micro-modelos parece ser muito semelhante a alguns dos micro-modelos que foram encontrados no processo de indenização de seguro (Tabela 4.5).

	Métricas						
Modelo	NAN	RD	NP	PL	CC	FIO	SUB
Macro-modelo	1.25	0.21	4.00	4.75	4.00	5.06	–
Micro-modelo (Solicitar emprestimo)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro-modelo (Verificar regras)	1.29	0.26	4.00	4.00	4.00	3.84	–
Micro-modelo (Analisar credito)	0.75	0.38	1.00	2.00	1.00	1.00	–
Micro-modelo (Aprovacao gerencial)	1.00	0.33	2.00	2.50	2.00	1.77	–
Micro-modelo (Liberar recurso)	1.00	0.33	2.00	2.50	2.00	1.77	–
Micro-modelo (Notificar cliente)	1.00	0.50	2.00	1.50	2.00	2.25	–
Modelo completo [Eq. (3.1)]	1.11	0.29	3.00	3.58	3.00	3.50	6

Tabela 4.6: Métricas aplicadas ao processo de empréstimo bancário

No geral, comparando a última linha da Tabela 4.6 com a última linha da Tabela 4.5, podemos dizer que o modelo hierárquico para o processo de empréstimo bancário parece ser mais complexo do que o modelo hierárquico para o processo de indenização de seguro. A única exceção é a métrica PL (i.e. tamanho do caminho), que parece ser menor para o processo de empréstimo bancário. Isso ocorre porque o macro-modelo para o processo

de pedido de indenização tem um *loop*, o que acaba tendo um peso notável no resultado para essa métrica em particular.

4.4 Comparação com as técnicas tradicionais

As técnicas tradicionais de mineração normalmente não fazem o relacionamento dos eventos de baixo nível registrado no log de eventos com as atividades de alto nível descritas por analistas de negócio. Nesta seção, vamos destacar vantagem da abordagem proposta neste trabalho na abstração que é gerada em comparação a outras técnicas, que geram modelos sem nenhum tipo de hierarquia.

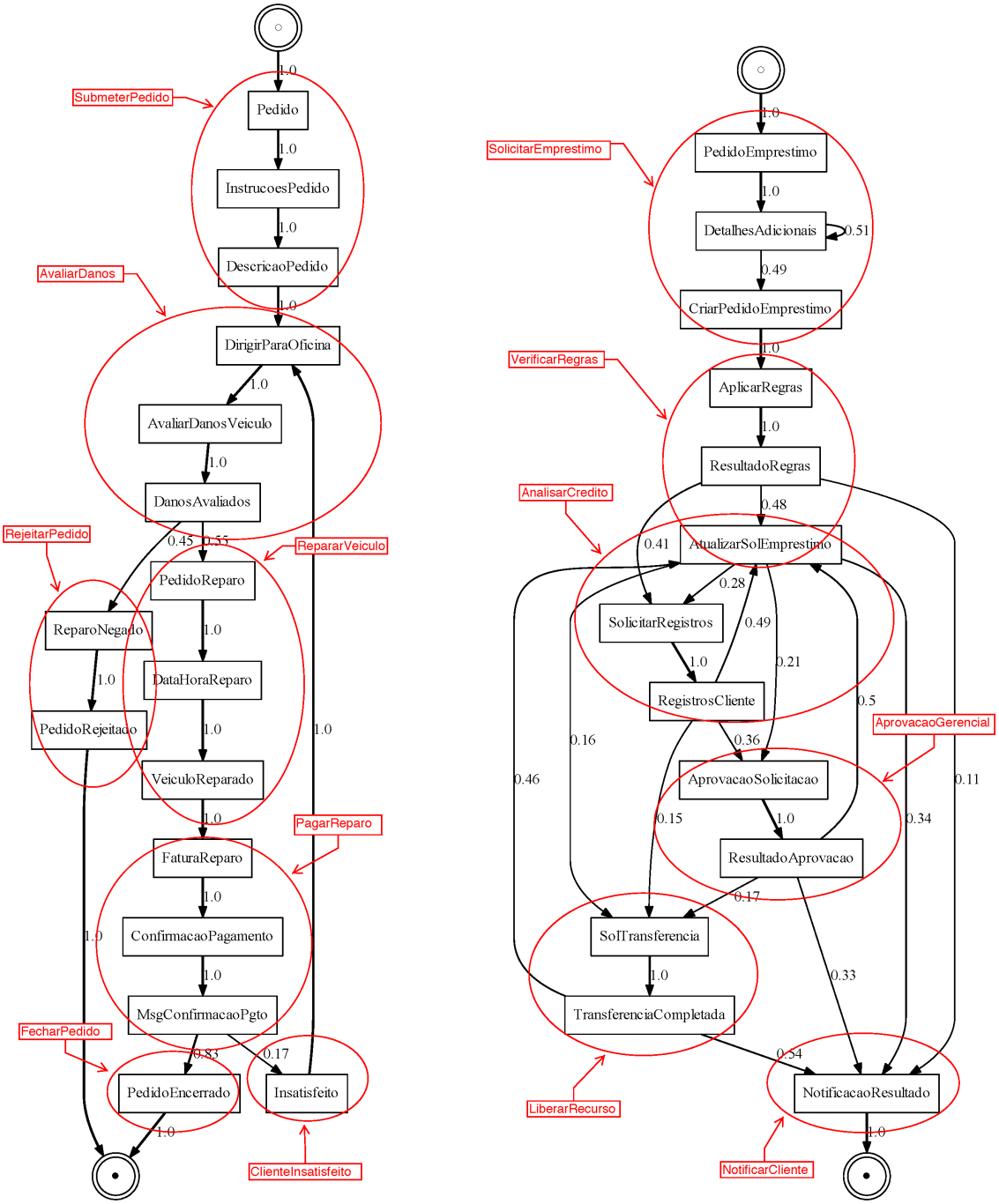
Embora no cenário de aplicação do processo de compra (Seção 4.1) foi realizada uma comparação com outras técnicas de mineração, como o *heuristics miner* (Weijters et al. 2006) e *social network miner* (Song and van der Aalst 2008), nesta seção geramos modelos sem o mapeamento hierárquico e avaliamos a complexidade destes modelos através das métricas definidas na Seção 3.4. Esta avaliação foi realizada para processos de indenização de seguro (Seção 4.2) e empréstimo bancário (Seção 4.3) que foram minerados sob a perspectiva de controle de fluxo sem a hierarquização proposta, a partir dos mesmos log de eventos gerados via simulação pelo AOR para os processos das Seções 4.2 e 4.3, respectivamente. A Figura 4.20 ilustra os modelos que foram minerados a partir destes logs de eventos.

Conforme observado nas Figuras 4.20(a) (indenização de seguro) e 4.20(b) (empréstimo bancário), os modelos sem nenhum tipo de hierarquização dificultam o entendimento dada a complexidade dos modelos gerados que são modelos muito grandes e difíceis de interpretar. A complexidade destes dois modelos parece ser muito maior do que a complexidade encontrada nos modelos hierárquicos para os mesmos processos descritos nas Seções 4.2.2 e 4.3.2, conforme visto no comparativo ilustrado na Figura 4.21.

Especificamente, no comparativo ilustrado na Figura 4.21(a) (indenização de seguro), podemos verificar que a métrica PL (Tamanho do caminho) apresenta um pico elevado no modelo minerado através da técnica *heuristics miner*. Esta situação indica a dificuldade para chegar a outro nó designado no modelo, i.e. o tamanho do modelo está muito elevado se compararmos ao modelo hierárquico.

No comparativo ilustrado na Figura 4.21(b) (empréstimo bancário) podemos observar diversas questões relacionadas aos resultados obtidos. A métrica NP (No. de caminhos) apresenta um pico elevado no modelo minerado através da técnica *heuristics miner*, que indica problemas como a dificuldade no entendimento do modelo, caminhos redundantes no interior do processo e falta de clareza no processamento do processo. Da mesma forma como na Figura 4.21(a), podemos verificar na Figura 4.21(b) que a métrica PL (Tamanho do caminho) apresenta um valor elevado no modelo minerado através da técnica *heuristics miner*, indicando a dificuldade para chegar a outro nó designado no modelo. Outra questão importante a destacar na Figura 4.21(b) é a métrica PL (Complexidade ciclomática), pois a diferença encontrada indica que o modelo minerado através da técnica *heuristics miner* apresenta alta ciclicidade. Isto significa que está ocorrendo muito retrabalho durante a execução do processo, onde recursos podem estar sendo desperdiçados, i.e. há uma grande incerteza neste modelo de processo.

Portanto, os resultados ilustrados na Figura 4.21 demonstra que utilizando técnicas de mineração tradicionais, os modelos encontrados apresentam relativamente maior com-

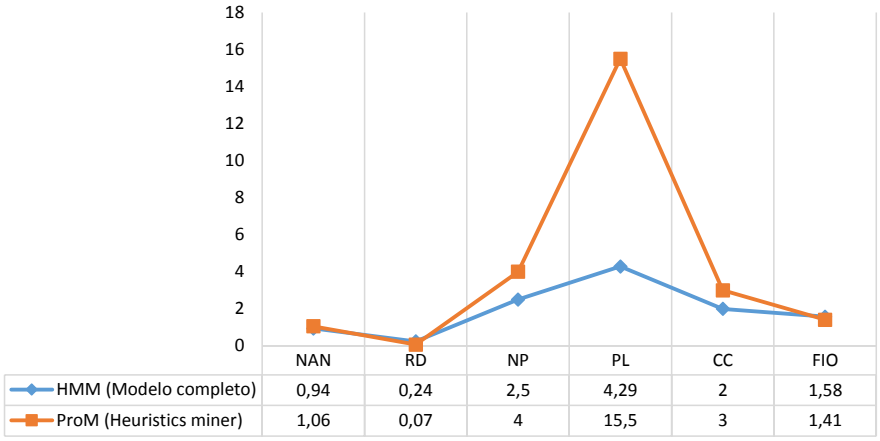


(a) Processo de indenização de seguro

(b) Processo de empréstimo bancário

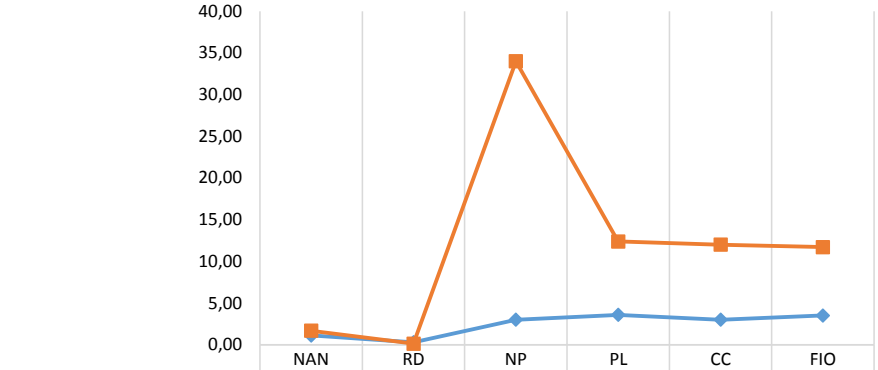
Figura 4.20: Modelos minerados sem o mapeamento hierárquico

plexidade se comparados aos resultados encontrados para os modelos hierárquicos. Isto confirma que a modularidade em um modelo de processo parece ter uma conexão positiva com a sua compreensão, em particular, os modelos que apresentam este tipo de decomposição parecem apoiar a compreensão de modelos onde é necessária uma visão em partes



	NAN	RD	NP	PL	CC	FIO
HMM (Modelo completo)	0,94	0,24	2,5	4,29	2	1,58
ProM (Heuristics miner)	1,06	0,07	4	15,5	3	1,41

(a) Processo de indenização de seguro



	NAN	RD	NP	PL	CC	FIO
HMM (Modelo completo)	1,11	0,29	3,00	3,58	3,00	3,50
ProM (Heuristics miner)	1,67	0,13	34,00	12,38	12,00	11,71

(b) Processo de empréstimo bancário

Figura 4.21: Comparativo das métricas

específicas.

4.5 Estudo de caso

Visando analisar o abordagem que foi descrita no Capítulo 3 em ambientes reais, buscamos realizar um estudo de caso utilizando um log de eventos real. Verificamos que a *3TU.Datacentrum*[2] oferece o conhecimento, a experiência e as ferramentas para arquivar dados de pesquisa de forma padronizada, segura e bem documentada. Neste repositório, há um diretório onde estão disponíveis log de eventos reais[3] contemplando os seguintes casos: *IEEE Task Force on Process Mining - Event Logs*; *hospital logs*; *BPI Challenge 2012*; *BPI Challenge 2013*.

[2] `http://datacentrum.3tu.nl/`
[3] `http://data.3tu.nl/repository/collection:event_logs_real`

No contexto deste trabalho, selecionamos o log de eventos real coletado a partir de um sistema de gerenciamento de incidentes da Volvo TI Bélgica. Originalmente, este log de eventos foi disponibilizado publicamente pelo *BPI Challenge* 2013[4]. Neste desafio, o objetivo foi analisar o log de eventos a fim de responder a uma série de questões específicas colocadas pelo proprietário do processo. Aqui vamos nos concentrar em encontrar um modelo de processo de alto nível que captura de forma mais equilibrada o comportamento de baixo nível registrado no log de eventos.

Para o propósito do *BPI Challenge* 2013, havia na verdade três logs de eventos disponíveis para dois processos diferentes: (1) gerenciamento de incidentes; e (2) gerenciamento de problemas. Aqui, ilustramos a aplicação da nossa abordagem de mineração para o processo de gestão de incidentes (*incident handling*), que está descrito detalhadamente na documentação existente.

4.5.1 O log de eventos

No log de eventos[5], encontramos os eventos que correspondem a todas as mudanças no status de cada incidente (um incidente também é referido como uma solicitação de serviço (*service request*). Cada evento tem:

- *SR number*, que é um identificador único para a solicitação de serviço;
- *Status*, que pode ser *Accepted*, *Queued* ou *Completed*;
- *Sub-status*, o qual é um de *In Progress*, *Awaiting Assignment*, *Assigned*, *Resolved*, *Closed*, etc.;
- *Involved ST*, equipe de suporte que que esta trabalhando no incidente e que mudou seu *status* ou *sub-status*;
- *timestamp* para o evento, na forma de data e hora;
- outros campos, como o impacto do incidente (alto, médio, baixo), o produto que o incidente se refere e o país que a equipe de suporte pertence.

Um trecho do log de eventos com os campos *SR number*, *Status*, *Sub-status*, *Involved ST* e *timestamp* é apresentado na Tabela 4.7. Aqui é possível ver que o primeiro evento no log é de Março de 2010 e o último evento é de Maio de 2012. Existem 7554 *traces* (i.e. incidentes) registrados no log de eventos e um total de 65533 eventos. A partir destes 7554 *traces*, existem 2278 *traces* diferentes (i.e. micro-sequências).

O campo *Involved ST* contém o identificador da equipe de suporte, algumas vezes com o sufixo "2nd" ou "3rd". Isto significa que a equipe de suporte pertencente à segunda linha e/ou terceira linha, em vez da primeira linha. A primeira linha é o serviço de *helpdesk*, que serve como uma interface para o usuário ou cliente que enviou a solicitação de serviço. A segunda e terceira linhas referem-se ao apoio das equipes que cuidam destes incidentes que não podem ser resolvidos pela primeira linha. Em particular, a terceira

[4] `http://www.win.tue.nl/bpi2013/doku.php?id=challenge`

[5] O log de eventos tem a seguinte referência DOI: `http://dx.doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee`

SR number	Status	Sub-status	Involved ST	timestamp
1-364285768	Accepted	In Progress	V30	2010-03-31 15:59:42
1-364285768	Accepted	In Progress	V30	2010-03-31 16:00:56
1-364285768	Queued	Awaiting Assignment	V5 3rd	2010-03-31 16:45:48
1-364285768	Accepted	In Progress	V5 3rd	2010-04-06 15:44:07
1-364285768	Queued	Awaiting Assignment	V30	2010-04-06 15:44:38
1-364285768	Accepted	In Progress	V13 2nd 3rd	2010-04-06 15:44:47
1-364285768	Completed	Resolved	V13 2nd 3rd	2010-04-06 15:44:51
...	...	...	...	...
1-467153946	Completed	Closed	S42	2012-05-23 00:22:25

Tabela 4.7: Um trecho do log de eventos da Volvo TI Bélgica

linha é constituída por especialistas no desenvolvimento de produtos que só devem ser envolvidos se o incidente não pode ser resolvido na primeira ou segunda linha.

O ato de passar o trabalho da primeira para a segunda linha ou mesmo para a terceira linha é popularmente conhecido no ITIL framework (Bon 2005) como escalada (*escalation*). Uma das questões a serem analisadas no *BPI Challenge 2013* era se a escalada das solicitações de serviço ocorre com muita ou pouca frequência. Neste estudo de caso vamos nos concentrar nas colunas *Status* e *Sub-status*, a fim de mapear estes estados a um conjunto de atividades de alto nível.

4.5.2 O modelo do processo

Uma característica interessante do *BPI Challenge* 2013 é que o log de eventos foi disponibilizado em conjunto com alguma documentação sobre o processo de gestão de incidentes. Um modelo BPMN para este processo é apresentado na Figura 4.22.

O modelo mostra como um incidente que será "investigado" dentro da primeira linha pode escalar para a segunda linha e, possivelmente, para a terceira linha. Isto acontece de forma semelhante em todas as linhas: se nenhuma solução pode ser encontrada dentro da linha atual, o incidente aumenta para a próxima linha, onde será comparado com um banco de dados de problemas conhecidos, se o incidente é reconhecido como um problema conhecido, em seguida, uma solução existente é aplicada para resolvê-lo, caso contrário, a equipe de suporte irá tentar encontrar uma solução para o problema. Eventualmente, se o problema é rastreado até um determinado componente, a equipe de suporte pode entrar em contato com o fornecedor correspondente.

O objetivo do presente estudo de caso é mapear os eventos de baixo nível da Tabela 4.7 para as atividades de alto nível da Figura 4.22. Para este propósito, definimos um evento de baixo nível como sendo a concatenação dos campos *Status* e *Sub-status* da Tabela 4.7. Por exemplo, se o *Status* é *Accepted* e o *Sub-status* é *In Progress*, concatenamos estes campos na string AcceptedInProgress. Então, o mapeamento que estamos procurando conterá pares de eventos e atividades, tais como AcceptedInProgress $\mapsto$ Investigate.

De fato, no log de eventos descobrimos que cerca de $84,4\%$ dos casos começam com AcceptedInProgress, $15,3\%$ começam com QueuedAwaitingAssignment e os $0,3\%$ restantes co-

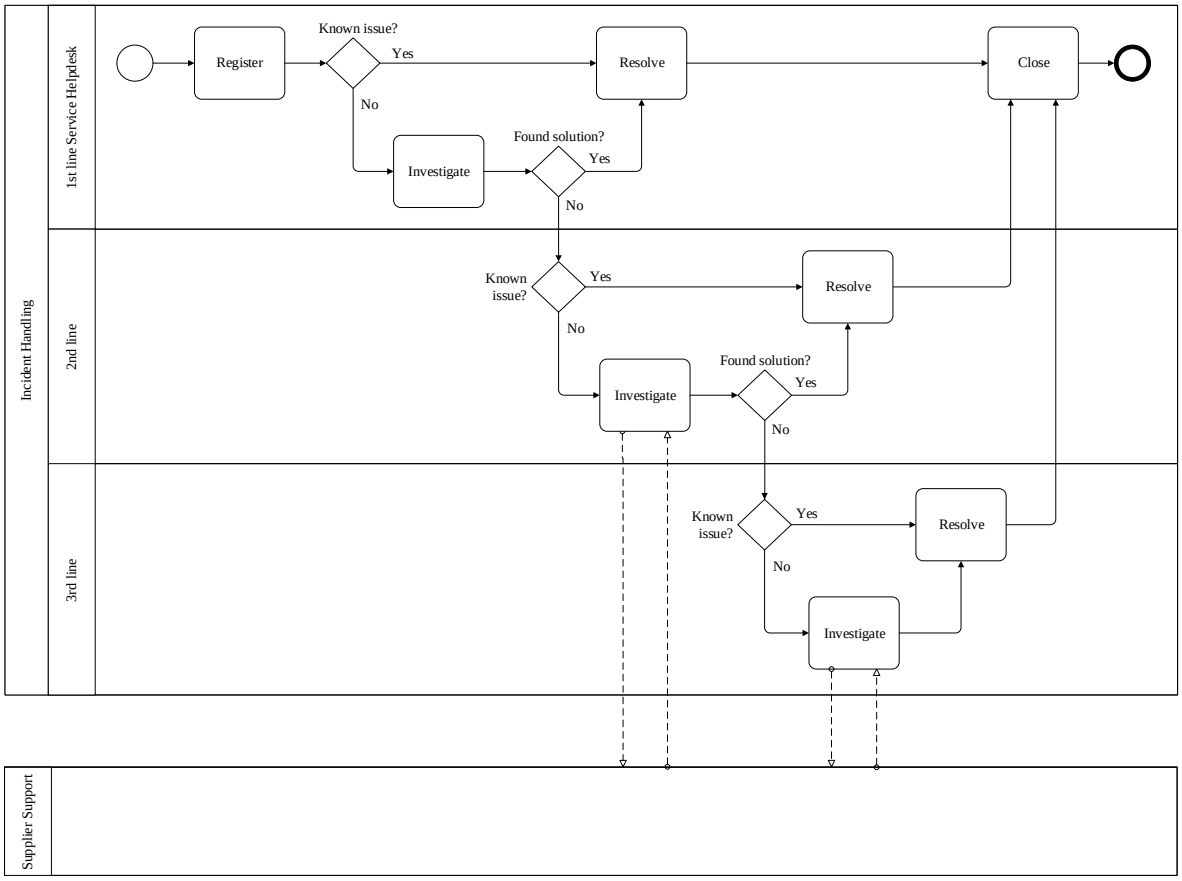


Figura 4.22: Um modelo BPMN do processo de alto nível associado a gestão de incidentes

meçam com outros eventos. O fato de que uma grande porcentagem de casos começam com AcceptedInProgress parece sugerir que este evento deve ser mapeado para a primeira atividade no processo, i.e. Register. No entanto, a Tabela 4.7 mostra que o evento AcceptedInProgress pode aparecer várias vezes em cada *trace*, onde não pode ser possível pertencer a atividade Register.

Veja por exemplo a escalada da primeira linha (V30) diretamente para a terceira linha (V5 3rd), que ocorre a partir do segundo para o terceiro evento da Tabela 4.7. De acordo com a Figura 4.22, não há atividade Register na terceira linha, por isso AcceptedInProgress não pode ser mapeado para Register. Portanto, podemos concluir que o comportamento observável registrado no log de eventos começa com a atividade Investigate

Talvez seja o caso de que o primeiro AcceptedInProgress na Tabela 4.7 pertence a atividade Register e as próximas ocorrências desse evento dentro do mesmo *trace* pertencem a uma outra atividade, tal como Investigate. No entanto, porque AcceptedInProgress provavelmente será mapeado para uma atividade diferente de Register, vamos simplesmente ignorar a atividade inicial Register e assumir que o processo começa com Investigate. Isso não é um problema, já que a atividade Register não deve representar mais do que a primeira ocorrência do evento AcceptedInProgress. O macro-modelo refinado que resultou desta análise é ilustrado na Figura 4.23.

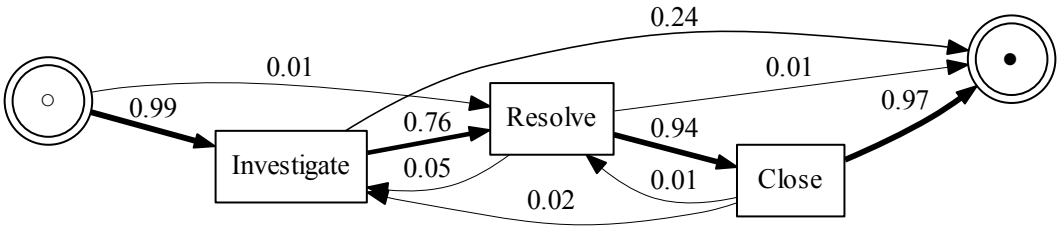


Figura 4.23: Representação do macro-modelo para o processo de gestão de incidentes (versão 1)

Antes de prosseguir com a mineração do processo, devemos mencionar que foi realizado o pré-processamento no log de eventos, i.e. foram filtrados eventos com um limite (*threshold*) de ocorrência abaixo de 1%. Isto é importante mencionar, porque na prática é muito difícil analisar os logs de eventos do mundo real, sem algum tipo de pré-processamento (Mans et al. 2009). Desta forma, com esta atividade somos capazes de analisar o log de eventos sem se preocupar com os *ruídos* (van der Aalst and Weijters 2004).

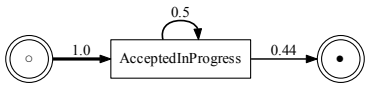
4.5.3 Mineração de processos

Nesta seção, vamos proceder com a mineração do log de eventos sob a perspectiva de controle de fluxo. Inicialmente, através de um *script* de conversão implementado em Python, as instâncias registradas no log de eventos da Tabela 4.7 foram convertidas para o formato de entrada aceito pelo modelo hierárquico de Markov. Posteriormente, as micro-sequências convertidas juntamente com o macro-modelo da Figura 4.23, foram fornecidos como entrada para o Algoritmo 4, que produziu os micro-modelos ilustrados na Figura 4.24.

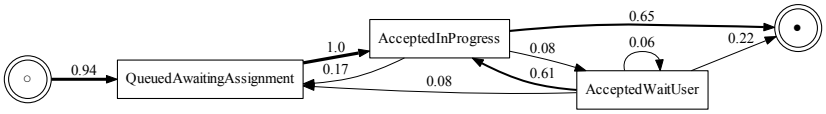
Estes modelos resultantes foram avaliados utilizando as métricas definidas na Seção 3.4 e os resultados são apresentados na Tabela 4.8.

	Métricas						
Modelo	NAN	RD	NP	PL	CC	FIO	SUB
Macro-modelo	2.00	0.67	4.00	3.25	7.00	38.70	–
Micro-modelo (Investigate)	1.88	0.31	6.00	2.17	9.00	15.77	–
Micro-modelo (Resolve)	2.83	0.28	22.00	4.55	24.00	67.96	–
Micro-modelo (Close)	2.75	0.28	29.00	5.55	23.00	58.98	–
Modelo completo [Eq. (3.1)]	2.24	0.48	11.50	3.67	12.83	43.14	3

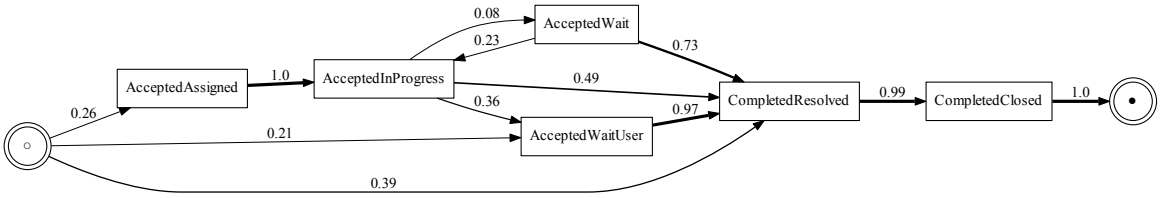
Tabela 4.8: Métricas aplicadas ao processo de gestão de incidentes



(a) Investigate



(b) Resolve



(c) Close

Figura 4.24: Micro-modelos para o processo de gestão de incidentes

No presente estudo de caso envolvendo um log de eventos do mundo real do *BPI Challenge* 2013, demonstramos como a abordagem de mineração de processos proposta contribuiu para chegar a um modelo que captura o comportamento de tempo de execução do processo em termos de atividades de alto nível usadas para documentar o processo, sendo capaz de lidar com ruídos. Também destacou o fato de que as métricas utilizadas auxiliam na avaliação da complexidade do modelos hierárquicos.

4.5.4 Comparação com as técnicas tradicionais

Da mesma forma como foi realização na Seção4.4, vamos fazer uma comparação com as técnicas tradicionais de mineração, que normalmente não fazem o relacionamento dos eventos de baixo nível registrado no log de eventos com as atividades de alto nível descritas por analistas de negócio. Nesta seção, vamos destacar vantagem da abordagem proposta neste trabalho na abstração que é gerada em comparação com a mineração realizada sem nenhum tipo de hierarquia. A Figura 4.25 ilustra o modelo que foi minerado a partir do log de eventos da Tabela 4.7.

Conforme observado na Figuras 4.25, os modelos sem nenhum tipo de hierarquização dificultam o entendimento, dado que os modelos gerados são modelos muito grandes e difíceis de interpretar. A complexidade deste modelo parece ser muito maior do que a complexidade encontrada no modelo hierárquico para o mesmo processo, conforme ilustrado na Figura 4.26.

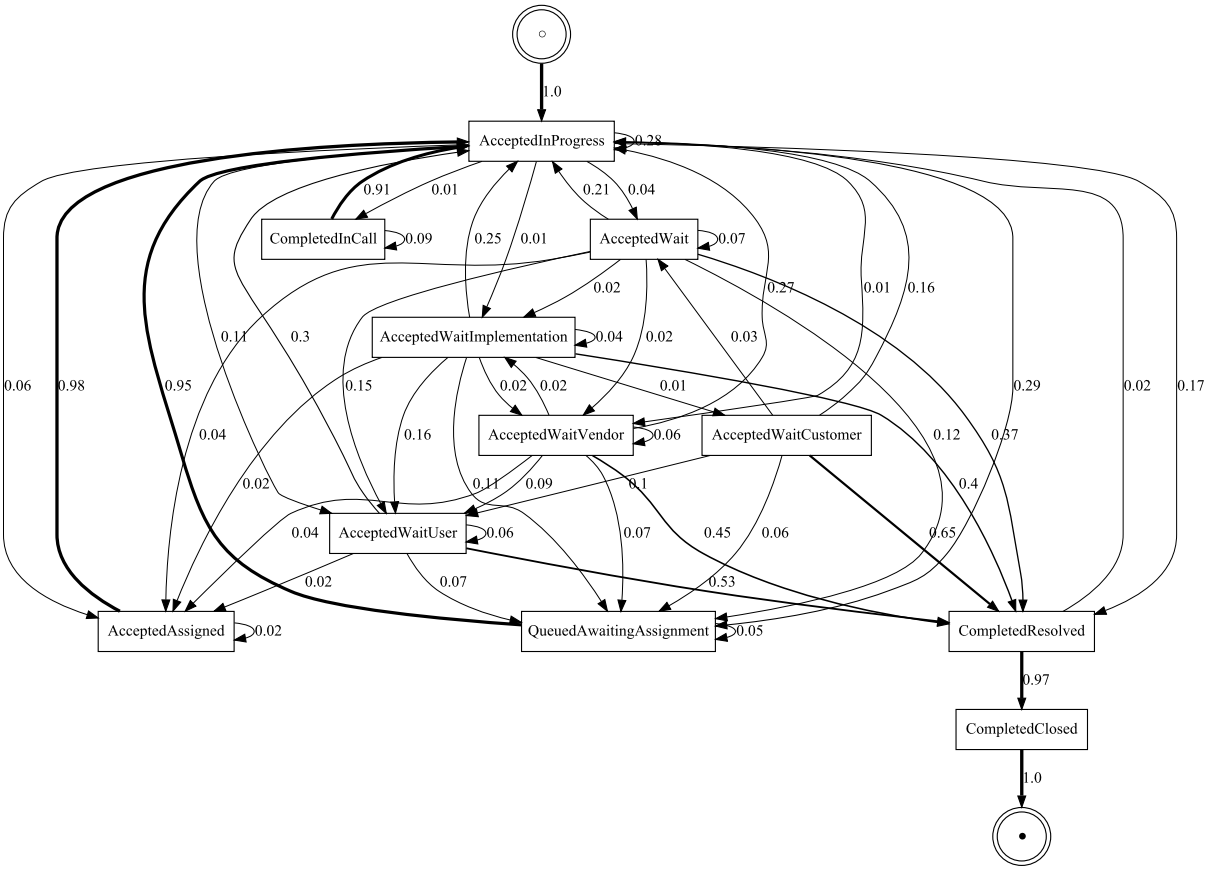
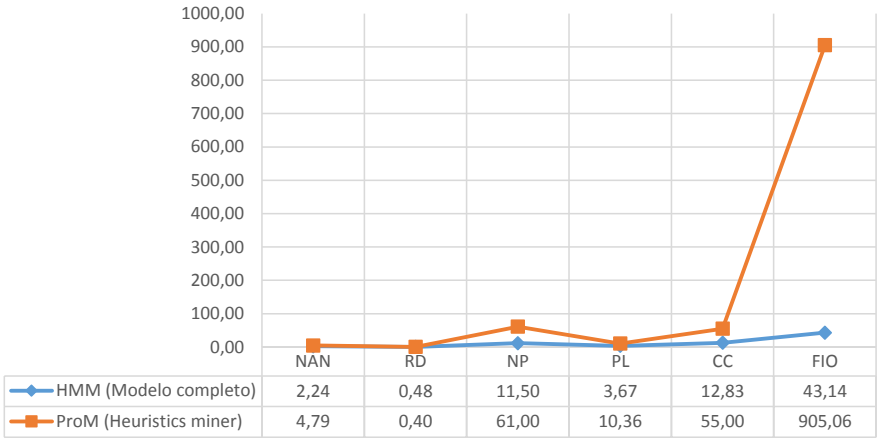


Figura 4.25: Modelo minerado sem o mapeamento hierárquico



	NAN	RD	NP	PL	CC	FIO
HMM (Modelo completo)	2,24	0,48	11,50	3,67	12,83	43,14
ProM (Heuristics miner)	4,79	0,40	61,00	10,36	55,00	905,06

Figura 4.26: Comparativo das métricas para de gestão de incidentes

Os resultados descritos na Figura 4.26 variam de acordo com cada métrica. Conforme observado na métrica NAN, o modelo sem hierarquia apresenta um valor maior (213%). Isto significa que modelo minerado pelo modelo hierárquico de Markov apresenta menor densidade no interior do processo, i.e. menor dimensão em relação ao outro modelo com

baixo *cross-linking* dentro do modelo. Diferentemente das demais métricas, a métrica RD no modelo sem hierarquia apresenta um valor um pouco menor (83%) em relação ao modelo hierárquico, que está relacionada com a intensidade do modelo de processo. Pois neste caso há possibilidade de mais arcos no modelo hierárquico de Markov. A métrica NP, que avalia a clareza do processo indica que o modelo sem hierarquia apresenta um número bem mais elevado de caminhos possíveis (530%), i.e. apresenta maior complexidade. Da mesma forma para a métrica PL, que é uma métrica importante para avaliar o tamanho dos caminhos, o modelo sem hierarquia apresenta um valor maior (282%). Isto demonstra a dificuldade em se chegar a outro nó designado no modelo. Na avaliação da métrica CC, que avalia características importantes, como as iterações do processo, o modelo sem hierarquia é bem superior ao outro modelo com hierarquia (423%), apontando para o nível de incerteza do processo, i.e. determina maior grau de possível re-trabalho durante a execução do processo. Finalmente, a questão mais importante a destacar neste comparativo é a métrica FIO, que foi a extremamente superior no modelo sem hierarquia. Esta ocorrência indica que modelos com valor elevado para esta métrica são difíceis de utilizar e com grande probabilidade de erros de projeto, i.e. o modelo está sobrecarregado indicando a necessidade de decomposição do modelo em sub-modelos (hierarquização).

Portanto, através destes experimentos e resultados comparativos concluímos que o modelo sem hierarquia apresenta complexidade superior ao modelo hierárquico. Isto confirma a hipótese que a modularidade em um modelo de processo auxilia na melhoria do modelo através da melhoria da compreensão e entendimento do mesmo.

Capítulo 5

Conclusões

Neste trabalho apresentamos uma abordagem iterativa para o melhoria de modelos de processos de negócio com base na mineração de processos e simulação baseada em agentes. A mineração de processos oferece meios ágeis para elaborar, monitorar, analisar e compreender diferentes pontos de vista dos processos reais e/ou simulados através da extração do conhecimento dos eventos. A necessidade desta abordagem é justificada pelo fato de facilitar a análise dos modelos de processo, possibilitando assim a introdução de melhorias, pois o comportamento de baixo nível com grande complexidade sugere que sejam inseridas outras atividades de alto nível no macro-modelo.

Para atingir os objetivos deste trabalho, foi desenvolvida uma técnica de mineração de processos, o *modelo hierárquico de Markov*, que é capaz de capturar a relação entre as atividades de nível macro em um modelo de processos de negócio e os eventos de nível micro que estão registrados em um log de eventos. A técnica captura esta relação na forma de um modelo hierárquico de Markov. Também desenvolvemos um algoritmo baseado em técnicas de EM para estimar os parâmetros de tal modelo. Esta abordagem pode ser usada como uma técnica de mineração de processo em cenários onde um log de eventos está disponível, juntamente com uma descrição de alto nível do processo de negócio. Portanto, de fato existe uma diferença entre o alto nível de abstração em que os processos são geralmente modelados por analistas de negócio e a natureza de baixo nível de eventos que são gerados durante a execução do processo.

Demonstramos que a técnica proposta (Algoritmo 4) é capaz de descobrir os micro-modelos para cada macro-atividade. Primeiramente, isso foi demonstrado em experimentos com um conjunto de padrões básicos de *workflow*. Depois, através da aplicação de estudos de caso em três diferentes cenários (e.g. processo de compra, processo de pedido de seguro e processo de solicitação de empréstimo), demostramos que o método proposto é capaz de realizar uma descoberta correta, pois o algoritmo foi capaz de associar os micro-eventos com as macro-atividades corretas e ao mesmo tempo proporcionar um micro-modelo para o comportamento dos agentes em cada uma dessas atividades. Os resultados também foram comparados com outras abordagens existentes de mineração de processos para demonstrar a compreensibilidade dos modelos hierárquicos.

Por outro lado, uma plataforma (*state-of-the-art*) de simulação baseada em agentes foi utilizada como um meio para gerar o comportamento de baixo nível (i.e. log de eventos) de versões dos processos que foram especificadas nos estudos de caso. Tal como são ilustradas nestes cenários de aplicação, a abordagem pode ser utilizada para analisar

tanto a perspectiva de controle de fluxo como a perspectiva de organizacional.

A grande maioria dos esforços neste trabalho foram na análise da perspectiva de controle de fluxo, mas o mesmo método também demonstrou que pode ser aplicado para a análise do ponto de vista organizacional, que inclui a entrega de trabalho entre os agentes e a colaboração entre os agentes dentro de cada *case*. Isso pode ser feito selecionando outras colunas para análise, i.e. a coluna *remetente* ou a coluna *destinatário* no log de eventos, conforme descrito na Tabela 4.1.

Através de um conjunto de métricas que servem como orientação, o analista de processo pode inserir alterações no modelo de processo, reconfigurar a plataforma de simulação, gerar um novo log de eventos e minerar um novo modelo hierárquico que refine o modelo inicial, captando a melhor relação entre as atividades de alto nível e o comportamento de baixo nível. Fazendo isso de forma iterativa levará a um modelo mais preciso, mais equilibrado, menos complexo e mais fácil de entender.

A abordagem que foi desenvolvida neste trabalho aponta para várias opções de trabalhos futuros. Agrupamos tais possibilidades de acordo com suas respectivas sub-áreas:

(a) Desenvolvimento algoritmico e ferramental:

Uma opção está no estabelecimento de diretrizes para a conversão de modelos BPMN para modelos de Markov, apoiando a geração automática de um cenário de simulação AOR a partir de um determinado modelo hierárquico. Em outras palavras, podemos pensar em automatizar todas as fases do ciclo iterativo de melhoria de modelos de processo de negócio. Outra possibilidade também é implementar a técnica de mineração de processos desenvolvida no âmbito deste trabalho (i.e. modelo hierárquico de Markov) no ProM.

Desenvolver de forma mais aprofundada a integração entre a mineração de processos e a simulação baseada em agentes para a descoberta e análise de processos de negócios, i.e, prover a utilização dos agentes cognitivos, possibilitando a execução de processos de negócio mais próximos da realidade.

Outra opção de trabalhos futuros é verificar a possibilidade de usar heurísticas adicionais na etapa randômica de inicialização do Algoritmo 4. Ordenar as micro- e macro-sequências pelo tamanho, como explicado na Seção 3.3.3, fornece uma grande melhoria em termos da probabilidade do log nas soluções encontradas pelo algoritmo. É possível que as heurísticas adicionais, com base no fato de que o macro-modelo é conhecido, possam contribuir para encontrar a melhor solução possível no menor número de execuções.

(b) Expansão do conjunto de métricas:

Outra possibilidade é expandir o conjunto de métricas que foram usadas na fase de análise e avaliação. Vamos observar como essas métricas de complexidade se relacionam com a compreensibilidade desses modelos, e se possível, fazer uso dessas métricas nos algoritmos de mineração de modo que no caso de várias soluções possíveis, onde uma que parecer ser menos complexa (ou mais compreensível) possa ser selecionada.

(c) Realização de outros experimentos:

Também podemos utilizar outros logs de eventos reais ao invés de simular com agentes, que por sua natureza pode apresentar ruídos e variação de comportamento. A partir

deste log de eventos, podemos descobrir o comportamento registrado neste log e, através do ciclo iterativo de melhoria, podemos inserir melhorias no modelo e analisar estas novas propostas com o foco na compreensibilidade.

Referências

R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '98, pages 469–483, London, UK, 1998. Springer-Verlag. ISBN 3-540-64264-1. 22, 27

E. R. Aguilar, F. Ruiz, F. García, and M. Piattini. Towards a suite of metrics for business process models in bpmn. In Yannis Manolopoulos, Joaquim Filipe, Panos Constantopoulos, and José Cordeiro, editors, *ICEIS (3)*, pages 440–443, 2006a. 36, 43

E.R. Aguilar, F. Ruiz, F. García, and M. Piattini. Applying software metrics to evaluate business process models. *CLEI Electronic Journal*, 9(1), June 2006b. 47

R.S. Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129 – 149, 2004. 11, 49

R.S. Aguilar-Savén and J. Olhager. Integration of product, process and functional orientations: Principles and a case study. In Harinder Jagdev, Johan C. Wortmann, and Henk Jan Pels, editors, *APMS*, volume 257 of *IFIP Conference Proceedings*, pages 375–389. Kluwer, 2002. 11

I.M. Ailenei. Process mining tools: A comparative analysis. Master's thesis, Eindhoven University of Technology. Department of Mathematics and Computer Science, 2011. 31, 50

K. Anders, B. Michael, M. I.Saira, S. Kimmen, and H. David. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501 – 1531, 1994. ISSN 0022-2836. 33

R. Axelrod. Agent-based modeling as a bridge between disciplines. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 33, pages 1565–1584. Elsevier, June 2006. 17

M. Azuma and D. Mole. Software management practice and metrics in the european community and japan: Some results of a survey. *Journal of Systems and Software*, 26: 5–18, 1994. 38

T. Baier and J. Mendling. Bridging abstraction layers in process mining: Event to activity mapping. In Selmin Nurcan, HenderikA. Proper, Pnina Soffer, John Krogstie, Rainer Schmidt, Terry Halpin, and Ilia Bider, editors, *Enterprise, Business-Process*

and Information Systems Modeling, volume 147 of *Lecture Notes in Business Information Processing*, pages 109–123. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38483-7. doi: 10.1007/978-3-642-38484-4_9. URL `http://dx.doi.org/10.1007/978-3-642-38484-4_9`. 46, 48

S. Balasubramanian and M. Gupta. Structural metrics for goal based business process design and evaluation. *Business Process Management Journal*, 2005. 42, 43, 47

V.R. Basili, G. Caldiera, and D.H. Rombach. *The Goal Question Metric Approach*, volume I. John Wiley & Sons, 1994. 38

K. Beck and W. Cunningham. Using pattern languages for object oriented programs. Technical report, OOPSLA - Conference on Object-Oriented Programming, Systems, Languages, and Applications, 1987. 46

B.W. Boehm. *Software engineering economics*. Prentice-Hall, 1981. 38

D.C. Boger and N.R. Lyons. The organization of the software quality assurance process. *DATA BASE*, 16(2):11–15, 1985. 37

J.B. Bon. *Foundations of IT Service Management: based on ITIL*. Van Haren Publishing, 2005. 113

E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *PNAS*, 99(Suppl 3):7280–7287, 2002. 17

R.P.J.C. Bose and W.M.P. van der Aalst. Analysis of Patient Treatment Procedures. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2011)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 165–166. Springer-Verlag, Berlin, 2012. 28

R.P.J.C. Bose, E.H.M.W. Verbeek, and W.M.P. van der Aalst. Discovering hierarchical process models using ProM. In *CAiSE Forum 2011*, volume 107 of *LNBIP*, pages 33–48. Springer, 2012. 46, 48, 50

M. Bozkaya, J. Gabriels, and J.M. van der Werf. Process diagnostics: A method based on process mining. In *International Conference on Information, Process, and Knowledge Management (eKNOW '09)*, pages 22–27, 2009. 90

U. Brandes and T. Erlebach. *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540249796. 37

J. Cardoso. Process control-flow complexity metric: An empirical validation. In *IEEE SCC*, pages 167–173. IEEE Computer Society, 2006. ISBN 0-7695-2670-5. 41, 47, 80

J. Cardoso. Control-flow complexity measurement of processes and Weyuker's properties. In *6th International Enformatika Conference*, volume 8 of *Transactions on Enformatika, Systems Sciences and Engineering*, pages 213–218, October 2005. 42, 43, 45, 46, 47

J. Cardoso. Business process quality metrics: Log-based complexity of workflow patterns. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *LNCS*, pages 427–434. Springer, 2007. 44, 47

J. Cardoso, J. Mendling, G. Neumann, and H.A. Reijers. A discourse on complexity of process models. In *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 117–128. Springer, 2006. 36, 43

C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: image segmentation using expectation-maximization and its application to image querying. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(8):1026–1038, 2002. ISSN 0162-8828. 33

S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994. 39

J. Claes and G. Poels. Process mining and the prom framework: An exploratory survey. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 187–198. Springer, 2012. 31, 50

J.E. Cook. *Process Discovery and Validation through Event-Data Analysis.* Phd thesis, University of Colorado, 1996. 22

J.E. Cook and A.L. Wolf. Automating process discovery through event-data analysis. In *Proceedings of the 17th international conference on Software engineering*, ICSE '95, pages 73–82, New York, NY, USA, 1995. ACM. 22, 27, 34

J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7:215–249, July 1998a. 22

J.E. Cook and A.L. Wolf. Event-based detection of concurrency. In *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, SIGSOFT '98/FSE-6, pages 35–45, New York, NY, USA, 1998b. ACM. 22

B. Curtis, M.I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35(9):75–90, September 1992. 13

M. Daneva, R. Heib, and A.W Scheer. *Benchmarking Business Process Models.* Veröffentlichungen des Instituts für Wirtschaftsinformatik im Deutschen Forschungszentrum für Künstliche Intelligenz. Iwi, 1996. 42, 47

T. H. Davenport. *Process innovation: reengineering work through information technology.* Harvard Business School Press, Boston, MA, USA, 1993. 11

P. Davidsson, J. Holmgren, H. Kyhlbäck, D. Mengistu, and M. Persson. Applications of agent based simulation. In *7th International Workshop on Multi-Agent-Based Simulation*, volume 4442 of *LNCS*. Springer, 2007. 17

A.R. De Pierro. A modified expectation maximization algorithm for penalized likelihood estimation in emission tomography. *Medical Imaging, IEEE Transactions on*, 14(1): 132–137, 1995. 33

A. De Toni and S. Tonchia. Performance measurement systems: models, characteristics and measures. *International Journal of Operations & Production Management*, 21(1/2): 46–70, 2001. 37

T. DeMarco. *Controlling software projects : management, measurement & estimation*. Yourdon Press, New York, NY, 1982. 36

A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977. 33, 63

R. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498–516, 2011. 36, 53

S.R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, June 1994. ISSN 0305-1048. doi: 10.1093/nar/22.11.2079. 33

N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1998. ISBN 0534954251. 36

D.R. Ferreira and D. Gillblad. Discovering process models from unlabelled event logs. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 2009. Proceedings*, volume 5701 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2009. 22, 33

D.R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 360–374. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-75182-3. 33, 34

S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, 1997. Springer-Verlag. ISBN 3-540-62507-0. 16

L. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1 (3):215–239, 1979. 37

R. Galliers. *Information Systems Research: Issues, Methods and Practical Guidelines*. Information Systems Series. Blackwell Scientific Publications, 1992. 8

A. Gemino and Y. Wand. Evaluating modeling techniques based on models of learning. *Commun. ACM*, 46(10):79–84, 2003. 41

A. Gemino and Y. Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requirements Engineering*, V9(4):248–260, November 2004. 41

D. Georgakopoulos, M.F. Hornick, and A.P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995. 1

A.A.A. Ghani, K.T. Wei, G.M. Muketha, and W.P. Wen. Complexity metrics for measuring the understandability and maintainability of business process models using goal-question-metric (GQM). *International Journal of Computer Science and Network Security*, 8(5):219–225, May 2008. 44, 47

C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2002. ISBN 0133056996. 36

G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management, 3rd International Conference, BPM 2005. Proceedings*, volume 3649, pages 32–47. Springer, 2005. 28, 46, 47, 50

V. Gruhn and R. Laue. Approaches for business process model complexity metrics. In *Technologies for Business Information Systems*, pages 13–24. Springer, 2007. 36, 44, 47, 53, 80

C.W. Günther and W.M.P. van der Aalst. A generic import framework for process event logs. Technical report, Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006), volume 4103 of Lecture Notes in Computer Science, 2006. 22

C.W. Günther and W.M.P. van der Aalst. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In *5th International Conference on Business Process Management*, volume 4714 of *LNCS*, pages 328–343. Springer, 2007. 5, 44, 46, 47, 50

C.W. Günther, A. Rozinat, and W.M.P. van der Aalst. Activity mining by global trace segmentation. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 128–139. Springer, 2010. 46, 48, 50

M.H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA, 1977. ISBN 0444002057. 38

M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness, 1993. 11, 12, 49

H.J. Harrington. *Business process improvement*. McGraw-Hill, New York [u.a.], 1991. ISBN 0070267685. 12

S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, SE-7(5):510–518, September 1981. 38

J. Herbst and D. Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *9th International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998. 27

A.O.I. Hoffmann, S.A. Delre, J.H. von Eije, and W. Jager. Artificial multi-agent stock markets: Simple strategies, complex outcomes. In Charlotte Bruun, editor, *Advances in Artificial Economics*, volume 584 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 2006. 17

Z. Irani, J. Ezingeard, R.J. Grieve, and P. Race. A case study strategy as part of an information systems research methodology: a critique. *Int. J. Comput. Appl. Technol.*, 12(2-5):190–198, July 1999. 8

E. Kindler. Model-based software engineering and process-aware information systems. *T. Petri Nets and Other Models of Concurrency*, 2:27–45, 2009. 1

B. Korherr and B. List. Extending the uml 2 activity diagram with business process goals and performance measures and the mapping to bpel. In *Advances in Conceptual Modeling - Theory and Practice*, pages 7–18, 2006. 13

M. Kreimeyer and U. Lindemann. The foundations of complexity metrics. In *Complexity Metrics in Engineering Design*, pages 33–92. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-20962-8. 36, 38, 41, 80

M. Laguna and J.P. Marklund. *Business Process Modeling, Simulation and Design*. Prentice Hall, February 2004. 1

L.M. Laird and M.C. Brennan. *Software Measurement and Estimation: A Practical Approach (Quantitative Software Engineering Series)*. Wiley-IEEE Computer Society Pr, 2006. 36

Y. Lashkari, M. Maes, and P. Maes. Collaborative interface agents. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 111–116. Morgan Kaufmann, 1998. 2

K.B. Lassen and W.M.P. van der Aalst. Complexity metrics for workflow nets. *Information & Software Technology*, 51(3):610–626, March 2009. 36, 45, 46, 47

A. M. Latva-Koivisto. Finding a complexity measure for business process models. Technical report, Helsinki University of Technology, February 2001. 43, 47

R. Laue and V. Gruhn. Complexity metrics for business process models. In Witold Abramowicz and Heinrich C. Mayr, editors, *BIS*, volume 85 of *LNI*, pages 1–12. GI, 2006. ISBN 3-88579-179-X. 36, 41, 80

B. List and B. KorList. An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1532–1539, New York, NY, USA, 2006. ACM. 13, 49

J. Luftman, R. Papp, and T. Brier. Enablers and inhibitors of business-it alignment. *Commun. AIS*, 1, March 1999. 1

C.M. Macal and M.J. North. Tutorial on agent-based modeling and simulation. In *Winter Simulation Conference*, pages 2–15, 2005. 16

Y. Malhotra. Integrating knowledge management technologies in organizational business processes: getting real time enterprises to deliver real business performance. *Journal of Knowledge Management*, 9(1):7–28, 2005. 11

R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, and P.J.M. Bakker. Process mining in healthcare: A case study. In *Proceedings of the International Conference on Health Informatics (HEALTHINF'08)*, pages 118–125. INSTICC, 2008. 90

R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, and P.J.M. Bakker. Application of process mining in healthcare — a case study in a dutch hospital. In Ana Fred, Joaquim Filipe, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies*, volume 25 of *Communications in Computer and Information Science*, pages 425–438. Springer, 2009. 115

A.A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. In R. Howard, editor, *Dynamic Probabilistic Systems (Volume I: Markov Models)*, chapter Appendix B, pages 552–577. John Wiley & Sons, Inc., New York City, 1971. 34

M.T. Martinez, P. Fouletier, K.H. Park, and J. Favrel. Virtual enterprise - organisation, evolution and control. *International Journal of Production Economics*, 74(1-3):225–238, 2001. 11

L. Maruster, A.J.M.M.T Weijters, W.M.P.Wil Aalst, and A. Bosch. Process mining: Discovering direct successors in process logs. In S. Lange, K. Satoh, and C. Smith, editors, *Discovery Science*, volume 2534 of *Lecture Notes in Computer Science*, pages 364–373. Springer Berlin Heidelberg, 2002. 34

A. Mcafee and E. Brynjolfsson. Investing in the it that makes a competitive difference. *Harvard Business Review*, 86(7/8):98–107, 2008. 11, 49

T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4): 308–320, 1976. 36, 38, 45, 46, 80

G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley Series in Probability and Statistics. Wiley-Interscience, 2008. 33, 63

A.K.A. Medeiros. *Genetic Process Mining*. Phd thesis, Eindhoven: Technische Universiteit Eindhoven, 2006. 27, 28, 50

A.K.A. Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M Weijters. Process mining: Extending the α-algorithm to mine short loops. BETA Working Paper Series WP 113, Eindhoven University of Technology, 2004. 78

A.K.A. Medeiros, A.J.M.M Weijters, A.J., and W.M.P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2): 245–304, 2007. 22, 48, 50

A.K.A. Medeiros, A. Guzzo, G. Greco, W.M.P. van der Aalst, A. J. M. M Weijters, B. F. Van Dongen, and D. Saccà. Process mining based on clustering: a quest for precision. In *Proceedings of the 2007 international conference on Business process management*, BPM'07, pages 17–29, Berlin, Heidelberg, 2008. Springer-Verlag. 50

J. Mendling. *Detection and prediction of errors in EPC business process models.* PhD thesis, Vienna University of Economics and Business Administration, Vienna, Austria, 2007. 36, 37, 38, 39, 41, 80

J. Mendling. *Metrics for Process Models*, volume 6 of *LNBIP*. Springer, 2009. 36, 43, 45, 47, 53, 80

J. Mendling, M. Moser, G. Neumann, E.H.M.W. Verbeek, and B.F. van Dongen. Faulty epcs in the sap reference model. In *International Conference on Business Process Management (BPM 2006*, pages 451–457. Springer-Verlag, 2006. 39

J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. *Lecture Notes in Computer Science*, LNCS 4803:113–130, 2007a. 39

J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 48–63. Springer Berlin / Heidelberg, 2007b. 41

J. Mendling, H.A. Reijers, and W.M.P. van der Aalst. Seven process modeling guidelines (7pmg). *Inf. Softw. Technol.*, 52(2):127–136, February 2010. 36

D.L. Moody. Metrics for evaluating the quality of entity relationship models. In Tok Wang Ling, Sudha Ram, and Mong-Li Lee, editors, *ER*, volume 1507 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 1998. ISBN 3-540-65189-6. 35

T.K. Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996. 33, 63

S. Morasca. Measuring attributes of concurrent software specifications in petri nets. In *Proceedings of the 6th International Symposium on Software Metrics*, METRICS '99, pages 100–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0403-5. 42, 47

F. Neri. A comparative study of a financial agent based simulator across learning scenarios. In Longbing Cao, Ana Bazzan, Andreas Symeonidis, Vladimir Gorodetsky, Gerhard Weiss, and Philip Yu, editors, *Agents and Data Mining Interaction*, volume 7103 of *LNCS*, pages 86–97. Springer, 2012. 17

A.I. Nesvizhskii, A. Keller, E. Kolker, and R. Aebersold. A statistical model for identifying proteins by tandem mass spectrometry. *Analytical Chemistry*, 75(17):4646–4658, 2003. 33

M.E.J. Newman. The structure and function of complex networks. *SIAM review*, 45(2): 167–256, 2003. 37, 80

O. Nicolae, G. Wagner, and J. Werner. Towards an executable semantics for activities using discrete event simulation. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 369–380. Springer, 2010. 20

M.E. Nissen. Redesigning reengineering through measurement-driven inference. *MIS Quarterly*, 22(4):509–534, December 1998. 42, 47

OMG. *Business Process Model and Notation (BPMN), Version 2.0*, 2011. 1, 13

OMG. Business process definition metamodel volume ii: Process definitions. Technical report, OMG, November 2008. 13

E.I. Oviedo. Software engineering metrics i. chapter Control flow, data flow and program complexity, pages 52–65. McGraw-Hill, Inc., New York, NY, USA, 1993. ISBN 0-07-707410-6. 38

S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623, 2006. 19

Á. Rebuge and D.R. Ferreira. Business process analysis in healthcare environments: A methodology based on process mining. *Information Systems*, 37(2):99 – 116, 2012. ISSN 0306-4379. 33

H.A. Reijers and J. Mendling. A study into the factors that influence the understandability of business process models. *Trans. Sys. Man Cyber. Part A*, 41(3):449–462, May 2011. 36

H.A. Reijers and J. Mendling. Modularity in process models: Review and effects. In *Business Process Management*, volume 5240 of *LNCS*, pages 20–35. Springer, 2008. 48, 50, 80

H.A. Reijers, J. Mendling, and R.M. Dijkman. On the usefulness of subprocesses in business process models. BPM center report BPM-10-03, Eindhoven University of Technology, 2010. 48, 50

J. Reungrungsee, S. Intarasema, P. Porouhan, and W. Premchaiswadi. Prom: Analysis of social network in students registration system. In *ICT and Knowledge Engineering (ICT Knowledge Engineering), 2012 10th International Conference on*, pages 236–243, 2012. 29, 50

A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008. 23, 45, 47

S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. 16, 17

T. Salamon. *Design of Agent-Based Models : Developing Computer Simulations for a Better Understanding of Social Processes*. Academic series. Bruckner Publishing, Repin, Czech Republic, 9 2011. 17

L. SáNchez-GonzáLez, F. GarcíA, F. Ruiz, and J. Mendling. Quality indicators for business process models from a gateway complexity perspective. *Inf. Softw. Technol.*, 54 (11):1159–1174, November 2012. 36

A.W. Scheer. *ARIS: Business Process Modeling*. Springer, 3rd edition, 2000. 2, 13

A. Schroeder. Integrated program measurement and documentation tools. In *Proceedings of the 7th international conference on Software engineering*, ICSE'84, pages 304–313, 1984. 42, 47

E. Segal, R. Yelensky, and D. Koller. Genome-wide discovery of transcriptional modules from dna sequence and gene expression. In *ISMB (Supplement of Bioinformatics)*, pages 273–282, 2003. 33

H.A. Simon. *The Sciences of the Artificial, 3rd Edition*. MIT Press, 1996. 40

I. Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321313798. 36

M. Song and W.M.P. van der Aalst. Towards comprehensive support for organizational mining. *Decis. Support Syst.*, 46(1):300–317, December 2008. 29, 50, 94, 109

M. Song and W.M.P. van der Aalst. Supporting process mining by showing events at a glance. In *Proceedings of 17th Annual Workshop on Information Technologies and Systems*, pages 139–145, 2007. 30, 50

L.G. Soo and Y. Jung-Mo. An empirical study on the complexity metrics of petri nets. *Microelectronics Reliability*, 32(3):323 – 329, 1992. 41

J.Y.L. Thong, C.S. Yap, and K.L. Seah. A consolidated methodology for business process reengineering. *IJCAT*, 17(1):1–15, 2003. 12

A. Tiwari, C.J. Turner, and B. Majeed. A review of business process mining: state-of-the-art and future trends. *Business Process Management Journal*, 14(1):5–22, 2008. 22

G.S. Tjaden, S. Narasimhan, and S. Mitra. Structural effectiveness metrics for business processes. In *Proceedings of the INFORMS Conference on Information Systems and Technology*, pages 396–400, May 1996. 42, 43, 47

W.M.P. van der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, August 2012. ISSN 0001-0782. 22

W.M.P. van der Aalst. Business Process Management: Models, Techniques, and Empirical Studies. volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000. 14, 49

W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. x, 2, 3, 22, 23

W.M.P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. 2, 13, 26

W.M.P. van der Aalst and C.W. Günther. Finding structure in unstructured processes: The case for process mining. In *Proceedings the 7th International Conference on Applications of Concurrency to System Design (ACSD 2007)*, pages 3–12. IEEE Computer Society Press, 2007. 28

W.M.P. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 2004. ISBN 3-540-22235-9. 29, 30, 50

W.M.P. van der Aalst and A.H.M ter Hostede. Yawl: Yet another workflow language. *Information Systems*, 2004. 13

W.M.P. van der Aalst and B.F. van Dongen. Discovering workflow performance models from timed logs. In Yanbo Han, Stefan Tai, and Dietmar Wikarski, editors, *Engineering and Deployment of Cooperative Information Systems*, volume 2480 of *Lecture Notes in Computer Science*, pages 107–110. Springer Berlin / Heidelberg, 2002. 30

W.M.P. van der Aalst and A.J.M.M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, 2004. 115

W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003a. 46, 74, 75

W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business process management: A survey. In W.M.P. van der Aalst and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2003b. 12

W.M.P. van der Aalst, A.J.M.M Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:1128–1142, 2004. 22, 26, 48, 50

W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593, 2005. 29, 50, 97

W.M.P. van der Aalst, V. Rubin, E.H.M.W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010. 76

W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. 23

B.F. van Dongen and W.M.P. van der Aalst. Multi-phase process mining: Building instance graphs. In Paolo Atzeni, Wesley W. Chu, Hongjun Lu, Shuigeng Zhou, and Tok Wang Ling, editors, *ER*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 2004. 28

B.F. van Dongen and W.M.P. van der Aalst. A meta model for process mining data. In *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, volume 2, pages 309–320, 2005. 95

B.F. van Dongen, A.K.A. Medeiros, E.H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005. ISBN 3-540-26301-2. 22, 30, 32, 50, 94

I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, and W.M.P. van der Aalst. *2007 BPM & Workflow Handbook*, chapter Quality Metrics for Business Process Models, pages 179–190. Future Strategies Inc., 2007. 35, 36, 40, 53

I. Vanderfeesten, H.A. Reijers, J. Mendling, W.M.P. van der Aalst, and J. Cardoso. On a quest for good process models: The cross-connectivity metric. In *Advanced Information Systems Engineering*, volume 5074 of *LNCS*, pages 480–494, 2008. 44, 47, 80

G.M. Veiga and D.R. Ferreira. Understanding spaghetti models with sequence clustering for ProM. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 92–103. Springer, 2010. 60

E.H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W. M. P. van der Aalst. Xes, xesame, and prom 6. In Pnina Soffer and Erik Proper, editors, *CAiSE Forum*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010. 32

H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. Computing Science Report 99/02, Eindhoven University of Technology, Eindhoven, 1999. 2

G. Wagner. A uml profile for agent-oriented modeling. In *Proceedings of the 3rd International Workshop on AgentOriented Software Engineering*, 2002. 21

G. Wagner. AOR modelling and simulation: Towards a general architecture for agent-based discrete event simulation. In *Agent-Oriented Information Systems*, volume 3030 of *LNCS*, pages 174–188. Springer, 2004. x, 6, 19, 20, 21, 86

G. Wagner. The agent-object-relationship metamodel: Towards a unified view of state and behavior. *Information Systems*, 28(5), 2003. 17, 21

G. Wagner and M. Diaconescu. Aor-simulation.org: cognitive agent simulation. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1405–1406, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-7-8. xii, 19, 20

G. Wagner, O. Nicolae, and J. Werner. Extending discrete event simulation by adding an activity concept for business process modeling and simulation. In *Proceedings of the 2009 Winter Simulation Conference*, pages 2951–2962, 2009. 19, 86

M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering*, 37 (3):410–429, 2011. 46, 47

A.J.M.M Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 10:151–162, April 2003. 30

A.J.M.M Weijters, W.M.P. van der Aalst, and A.K.A. Medeiros. Process mining with the HeuristicsMiner algorithm. BETA Working Paper Series WP 166, Eindhoven University of Technology, 2006. 22, 27, 48, 50, 94, 109

G. Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999. 17

M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. 1

P.Y.H. Wong and J. Gibbons. A Process Semantics for BPMN. In S. Liu, T. Maibaum, and K. Arki, editors, *International Conference on Formal Engineering Methods (ICFEM 2008)*, volume 5256 of *Lecture Notes in Computer Science*, pages 27–31. Springer-Verlag, Berlin, 2008. 2

M. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., Chichester, UK, 3 edition, 2009. 16, 17

M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995. 2, 18

R.K. Yin. *Applications of case study research*. Sage, 2003. 8

E. Yourdon. *Modern structured analysis*. Prentice-Hall international series. Prentice-Hall Internat., Englewood Cliffs, NJ [u.a.], 1989. 1

S. Zugal, J. Pinggera, B. Weber, J. Mendling, and H.A. Reijers. Assessing the impact of hierarchy on model understandability – a cognitive perspective. In *Models in Software Engineering*, volume 7167 of *LNCS*, pages 123–133. Springer, 2012. 48, 49, 50

H. Zuse. *Software complexity: measures and methods*. Programming complex systems. W. de Gruyter, 1991. 39

H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter, 1998. 37

Apêndice A

Publicações

Este apêndice apresenta todas as publicações científicas deste projeto de pesquisa de doutorado relacionado ao Programa de Pós-graduação em Informática (PPGInf) da Universidade de Brasília (UnB) em ordem cronológica descendente.

(1) Diogo R. Ferreira, Fernando Szimanski e Célia Ghedini Ralha. Mining the low-level behaviour of agents in high-level business processes. *International Journal of Business Process Integration and Management (IJBPIM)*, volume 6, número 2, p. 146-166, 2013. Estrato B4 Qualis Capes na Ciência da Computação.

(2) Fernando Szimanski, Célia Ghedini Ralha, Gerd Wagner e Diogo R. Ferreira. Improving Business Process Models with Agent-based Simulation and Process Mining. In: *14th Business Process Modeling, Development, and Support Working Conference (BPMDS)*, 2013, Valência, Espanha. Proceedings of BPMDS'13 - Lecture Notes in Business Information Processing (LNBIP). Berlin Heidelberg: Springer-Verlag. v. 147. p. 124-138, 2013. Estrato B3 Qualis Capes na Ciência da Computação. Selecionado entre os melhores artigos.

(3) Diogo R. Ferreira, Fernando Szimanski e Célia Ghedini Ralha. A Hierarchical Markov Model to Understand the Behaviour of Agents in Business Processes. In: *8th International Workshop on Business Process Intelligence (BPI)*, 2012, Tallin, Estonia. Lecture Notes in Business Information Processing (LNBIP). Berlin Heidelberg: Springer-Verlag, 2012. v. 132. p. 150-161. Em conjunto com o *10th International Conference in Business Process Management (BPM)*. Sendo o BPM estrato A2 Qualis Capes na Ciência da Computação. Selecionado entre os melhores artigos.

(4) Fernando Szimanski, Célia Ghedini Ralha e Ricardo Pezzuol Jacobi. Uma Abordagem de Redescoberta e Melhoria de Processos de Negócio Através da Integração de Simulação Baseada em Agentes e Mineração de Processos. In: *Conferência IADIS Ibero-Americana WWW/Internet (CIAWI)*, Rio de Janeiro. 2011, p. 1-8. Estrato B2 Qualis Capes na Ciência da Computação em 2011.

(5) Fernando Szimanski, Célia Ghedini Ralha e Ricardo Pezzuol Jacobi. Uma Abordagem de Integração de Simulação Baseada em Agentes e Mineração de Processos. In: *Simpósio Brasileiro de Sistemas de Informação (SBSI)*, Salvador. 2011. Estrato B4 Qualis Capes na Ciência da Computação.

Apêndice B

Código Fonte do Modelo Hierárquico de Markov

B.1 Mining.py

```
import sys

from model import *
from metrics import Initialize_Metrics
import time

start = time.clock()

macrofile = 'macro-model.txt'
print 'Reading:', macrofile
fmacro = open(macrofile, 'r')
macro = readmarkov(fmacro)

markov2dot(macro, 'macro-model')

logfile = 'aorlog-eventlog-purchase.txt'
#logfile = 'aorlog-eventlog.txt'
print 'Reading:', logfile
flog = open(logfile, 'r')

log = dict()

for line in flog:
    line = line.strip()
    if len(line) == 0:
        continue
    parts = line.split()

    caseid = parts[0]
    sender = parts[2]
```

```
    message = parts[1]
    receiver = parts[3]
    #timestamp = parts[4]

    if caseid not in log:
        log[caseid] = []
    log[caseid].append((sender, message, receiver))

microseqs = []
for caseid in log:
    microseq = []
    for (sender, message, receiver) in log[caseid]:
        #microseq.append(message)
        microseq.append(message)
    microseqs.append(microseq)

nruns = 10
if len(sys.argv) > 1:
    nruns = int(sys.argv[1])
print 'No. runs:', nruns

bestmodel = None
bestlogprob = None
for i in range(0, nruns):
    print '.',
    model = estimate(microseqs, macro)
    logprob = 0.0
    for i in range(0, len(microseqs)):
        result = []
        findmacroseq(microseqs[i], model, result)
        logprob += result[0]
    if (bestlogprob == None) or (logprob > bestlogprob):
        bestmodel = model
        bestlogprob = logprob
print
print 'LogProb:', bestlogprob

for activity in bestmodel[0][0][1:-1]:
    if activity in bestmodel[1]:
        markov2dot(bestmodel[1][activity], 'fig-' + activity.lower
            ())

end = time.clock()
print 'Code time %.6f seconds' % (end - start)

#compute metrics
Initialize_Metrics(model, bestmodel, microseqs)
```

B.2 Model.py

```
import os, sys
import math
import random
from subprocess import check_call
import pydot


BEGIN = None
END = None

# Return non-empty line from text file
def readline(fin):
    line = fin.readline()
    while len(line.strip()) == 0:
        line = fin.readline()
    return line

# Randomize macro sequence based on macro model
def randmacroseq(microseq, macro):
    stateseq = list(runmarkov(macro))
    quantity = len(stateseq) * [1]
    i = 0
    while sum(quantity) < len(microseq):
        quantity[i] += 1
        i += 1
        if i == len(stateseq):
            i = 0
    macroseq = ''
    for i in range(0, len(stateseq)):
        macroseq += quantity[i] * stateseq[i]
    return macroseq

# Read a markov model from text file
def readmarkov(fin):
    global BEGIN, END
    line = readline(fin)
    states = line.split()
    BEGIN = states[0]
    END = states[-1]
    matrix = []
    for state in states:
        line = readline(fin)
        matrix.append([float(prob) for prob in line.split()])
    markov = (states, matrix)
    return markov
```

```
# Read a hierarchical markov model from text file
def readmodel(fin):
    macro = readmarkov(fin)
    micro = dict()
    for activity in macro[0][1:-1]:
        micro[activity] = readmarkov(fin)
    model = (macro, micro)
    return model

# Random draw from list of probabilities
def draw(probs):
    rand = random.random()
    sumprob = 0.0
    for i in range(0, len(probs)):
        sumprob += probs[i]
        if rand < sumprob:
            return i

# Generate a sequence from a markov chain
def runmarkov(markov):
    (states, matrix) = markov
    sequence = []
    i = draw(matrix[0])
    while states[i] != states[-1]:
        sequence.append(states[i])
        i = draw(matrix[i])
    return sequence

# Generate a sequence from a hierarchical markov model
def runmodel(model):
    (macro, micro) = model
    macroseq = runmarkov(macro)
    microseq = []
    for activity in macroseq:
        microseq += runmarkov(micro[activity])
    return microseq

# Get transition probabilities from sequences
def seqs2markov(seqs):
    # get list of states
    states = set()
    for seq in seqs:
        for i in range(0, len(seq)):
            states.add(seq[i])
    states = [BEGIN] + sorted(list(states)) + [END]
    # count transitions between states
    matrix = []
    for i in range(0, len(states)):
        matrix.append(len(states)*[0.0])
```

```
    for seq in seqs:
        for i in range(0, len(seq)):
            a = states.index(seq[i])
            if i == 0:
                matrix[0][a] += 1.0
            if i < len(seq)-1:
                b = states.index(seq[i+1])
                matrix[a][b] += 1.0
            if i == len(seq)-1:
                matrix[a][-1] += 1.0
    # normalize matrix
    for i in range(0, len(matrix)):
        rowsum = 0.0
        for j in range(0, len(matrix[i])):
            rowsum += matrix[i][j]
        if rowsum > 0.0:
            for j in range(0, len(matrix[i])):
                matrix[i][j] /= rowsum
    markov = (states, matrix)
    return markov

# Partition a list of micro sequences according to a list of macro
   sequences
def partseqs(microseqs, macroseqs):
    parts = dict()
    for i in range(0, len(microseqs)):
        microseq = microseqs[i]
        macroseq = macroseqs[i]
        for j in range(0, len(microseq)):
            if macroseq[j] not in parts:
                parts[macroseq[j]] = []
            if (j == 0) or (macroseq[j] != macroseq[j-1]):
                parts[macroseq[j]].append([])
            parts[macroseq[j]][-1].append(microseq[j])
    return parts

# Recursive method to find macro sequence with highest probability
# the third parameter is passed by reference and contains the
   result
# in the form: result = [logprob, macroseq]
def findmacroseq(microseq, model, result=[], i=0, logprob=0.0,
   macroseq=[]):
    # prune if probability is less than best macro sequence found
       so far
    if (len(result) > 0) and (logprob < result[0]):
        return
    # if sequence is over, handle last element in sequence and
       check if result is best
    if i == len(microseq):
```

```
        # probability of micro sequence ending here
        (microstates, micromatrix) = model[1][macroseq[-1]]
        pmicro = micromatrix[microstates.index(microseq[i-1])][-1]
        if pmicro > 0.0:
            # probability of macro sequence ending here
            (macrostates, macromatrix) = model[0]
            pmacro = macromatrix[macrostates.index(macroseq[-1])
               ][-1]
            if pmacro > 0.0:
                # total probability for macro sequence
                logprob += math.log(pmicro) + math.log(pmacro)
                # check if result is best so far
                if (len(result) == 0):
                    # if there is no previous result, then present
                       result is best
                    result.append(logprob)
                    result.append(macroseq)
                else:
                    # if there is a previous result, then compare
                    if logprob > result[0]:
                        result[0] = logprob
                        result[1] = macroseq
        # sequence is over, so return in any case
        return
    # handle first element in sequence
    if i == 0:
        (macrostates, macromatrix) = model[0]
        for macrostate in macrostates[1:-1]:
            # probability of macro sequence beginning here
            pmacro = macromatrix[0][macrostates.index(macrostate)]
            if (pmacro > 0.0) and (macrostate in model[1]):
                (microstates, micromatrix) = model[1][macrostate]
                if microseq[i] in microstates:
                    # probability of micro sequence beginning here
                    pmicro = micromatrix[0][microstates.index(
                       microseq[i])]
                    if pmicro > 0.0:
                        findmacroseq(microseq, model, result, i+1,
                           logprob+math.log(pmacro)+math.log(pmicro
                           ), macroseq+[macrostate])
    # handle other elements in sequence
    else:
        # suppose that macrostate is the same as previous one
        (microstates, micromatrix) = model[1][macroseq[-1]]
        if microseq[i] in microstates:
            # probability of micro sequence changing to new state
            pmicro = micromatrix[microstates.index(microseq[i-1])][
               microstates.index(microseq[i])]
            if pmicro > 0.0:
```

```
                findmacroseq(microseq, model, result, i+1, logprob+
                   math.log(pmicro), macroseq+[macroseq[-1]])
        # suppose that macrostate changes to new one
        # probability of micro sequence ending in previous state
        pmicro1 = micromatrix[microstates.index(microseq[i-1])][-1]
        if pmicro1 > 0.0:
            (macrostates, macromatrix) = model[0]
            for macrostate in macrostates[1:-1]:
                # probability of macro sequence changing to new
                   state
                pmacro = macromatrix[macrostates.index(macroseq
                   [-1])][macrostates.index(macrostate)]
                if (pmacro > 0.0) and (macrostate in model[1]):
                    (microstates, micromatrix) = model[1][
                       macrostate]
                    if microseq[i] in microstates:
                        # probability of micro sequence beginning
                           here
                        pmicro2 = micromatrix[0][microstates.index(
                           microseq[i])]
                        if pmicro2 > 0.0:
                            findmacroseq(microseq, model, result, i
                               +1, logprob+math.log(pmicro1)+math.
                               log(pmacro)+math.log(pmicro2),
                               macroseq+[macrostate])

# Print markov
def printmarkov(markov):
    tab = 5
    (states, matrix) = markov
    print ' ',
    for state in states:
        print state.rjust(tab),
    print
    for i in range(0, len(states)):
        print states[i],
        for j in range(0, len(states)):
            if matrix[i][j] == 0.0:
                print '-'.rjust(tab),
            elif matrix[i][j] == 1.0:
                print '1'.rjust(tab),
            else:
                print str(round(matrix[i][j],2)).rjust(tab),
        print

# Print model
def printmodel(model):
    (macro, micro) = model
    printmarkov(macro)
```

```
    macrostates = macro[0]
    for state in macrostates[1:-1]:
        print '['+state+']'
        if state in micro:
            printmarkov(micro[state])

# Extend macro sequence to the length of a given micro sequence
def extend(macroseq, microseq):
    n = len(macroseq)
    quantity = n * [1]
    while sum(quantity) < len(microseq):
        i = random.randint(0,n-1)
        quantity[i] += 1
    macroext = []
    for i in range(0, len(macroseq)):
        macroext += quantity[i] * [macroseq[i]]
    return macroext

# Iterative procedure for estimating complete model from given
   micro sequences and macro model
def estimate(microseqs, macro):
    model = None
    # initialize procedure by generating a set of random macroseqs
    macroseqs = []
    for i in range(0, len(microseqs)):
        macroseq = runmarkov(macro)
        macroseqs.append(macroseq)
    # sort macroseqs and microseqs by length
    microseqs = [(len(seq), seq) for seq in microseqs]
    microseqs.sort()
    microseqs = [seq for (lenseq, seq) in microseqs]
    macroseqs = [(len(seq), seq) for seq in macroseqs]
    macroseqs.sort()
    macroseqs = [seq for (lenseq, seq) in macroseqs]
    # extend macroseqs to the length of microseqs
    for i in range(0, len(macroseqs)):
        macroseqs[i] = extend(macroseqs[i], microseqs[i])
    # iteration begins here
    it = 0
    while True:
        it += 1
        ##print 'Iteration', it
        # separate micro sequences
        parts = partseqs(microseqs, macroseqs)
        # estimate micro model
        micro = dict()
        for state in parts.keys():
            micro[state] = seqs2markov(parts[state])
        # build complete model
```

```
        model = (macro, micro)
        # find best macro sequences according to model
        lastmacroseqs = macroseqs
        macroseqs = []
        for microseq in microseqs:
            result = []
            findmacroseq(microseq, model, result)
            # what if there is no possible macro sequence?
            # (i.e. no macro sequence with probability > 0)
            if result == []:
                macroseq = randmacroseq(microseq, macro)
                macroseqs.append(macroseq)
            else:
                [logprob, macroseq] = result
                macroseqs.append(macroseq)
        # check convergence
        if macroseqs == lastmacroseqs:
            break
    # return complete model
    return model

# Compute the difference between two markov models
def diffmarkov(markov1, markov2):
    diff = 0.0
    (states1, matrix1) = markov1
    (states2, matrix2) = markov2
    states = set()
    states.update(set(states1))
    states.update(set(states2))
    states = list(states)
    for i in range(0, len(states)):
        for j in range(0, len(states)):
            p1 = 0.0
            if (states[i] in states1) and (states[j] in states1):
                p1 = matrix1[states1.index(states[i])][states1.
                   index(states[j])]
            p2 = 0.0
            if (states[i] in states2) and (states[j] in states2):
                p2 = matrix2[states2.index(states[i])][states2.
                   index(states[j])]
            diff += abs(p1-p2)
    diff /= 2.0 * float(len(states)-1)
    return diff

# Compute the difference between two micro models
def diffmicro(micro1, micro2):
    diff = 0.0
    states = set()
    states.update(set(micro1.keys()))
```

```
    states.update(set(micro2.keys()))
    states = sorted(list(states))
    for state in states:
        if (state in micro1) and (state in micro2):
            diff += diffmarkov(micro1[state], micro2[state])
        if (state in micro1) and (state not in micro2):
            diff += diffmarkov(micro1[state], ([], None))
        if (state not in micro1) and (state in micro2):
            diff += diffmarkov(([], None), micro2[state])
    diff /= float(len(states))
    return diff

def markov2dot(markov, name):
    name = '.\\Temp\\' + name
    fout = open(name+'.dot', 'w')
    (states, matrix) = markov
    fout.write('digraph model { \n')
    fout.write('\t rankdir = "LR"; \n')
    fout.write('\t ranksep = 0.3; \n')
    fout.write('\t node [shape = doublecircle]; ')
    if BEGIN[0:2] == '&#':
        fout.write('<'+BEGIN+'> ')
    else:
        fout.write('"'+BEGIN+'" ')
    if END[0:2] == '&#':
        fout.write('<'+END+'>\n')
    else:
        fout.write('"'+END+'"\n')
    fout.write('\t node [shape = rectangle]; \n')
    for i in range(0,len(states)):
        for j in range(0,len(states)):
            p = matrix[i][j]
            if p > 0.0:
                fout.write('\t ')
                if states[i] == BEGIN:
                    if BEGIN[0:2] == '&#':
                        fout.write('<'+BEGIN+'> ')
                    else:
                        fout.write('"'+BEGIN+'" ')
                else:
                    fout.write('"'+states[i]+'" ')
                fout.write('-> ')
                if states[j] == END:
                    if END[0:2] == '&#':
                        fout.write('<'+END+'> ')
                    else:
                        fout.write('"'+END+'" ')
                else:
                    fout.write('"'+states[j]+'" ')
```

```
                fout.write('[penwidth=' + str(1.0+p**2))
                fout.write(', label = "' + str(round(p,2)) + '"]; \
                   n')
    fout.write('} \n')
    fout.flush()
    fout.close()
    cmd = 'dot -Tsvg ' + name + '.dot > ' + name + '.svg'
    print 'Running:', cmd
    os.system(cmd)

    cmd = 'java -jar \\batik-1.7\\batik-rasterizer.jar -m
       application/pdf ' + name + '.svg'
    print 'Running:', cmd
    os.system(cmd)

    cmd = 'dot -Tpdf ' + name + '.dot > ' + name + '.pdf'
    print 'Running:', cmd
    os.system(cmd)
```

B.3 Metrics.py

```
from __future__ import division
import random
import networkx as nx
import os
from model import *

BEGIN = '&#9675;'
END = '&#9679;'

# Return metrics in a model
def Compute_Metrics(model, bestmodel):

    #=======================================
    # Compute metric type: NUMBER OF NODES
    # Count number of tasks in a process model (macro and micro),
       including start and end node
    #=======================================

    # initialization
    metrics = []
    md = dict()
    metrics.append(('MODEL', 'NN', 'NA', 'NAN', 'RD', 'NP', 'AVG-PL
       ', 'NC', 'CC', 'FANIN', 'FANOUT', 'FAN-IO'))
    metrics.append(12*[0])
    metrics[1][0] = 1
    md[1] = 'Macro model'
```

```
act = 2
for activity in bestmodel[0][0]:
    if activity in bestmodel[1]:
        metrics.append(12*[0])
        metrics[act][0] = act
        md[act] = activity
        act += 1
############ End of initialization #################

nodes = []
for macroactivity in model[0][0]:
    if (not macroactivity in nodes):
        nodes.append(macroactivity)
metrics[1][1] = len(nodes)
nodes = []
for activity in bestmodel[0][0]:
    if activity in bestmodel[1]:
        metrics[md.keys()[md.values().index(activity)]][1] =
           len(bestmodel[1][activity][0])
        for activity in bestmodel[1][activity][0]:
            if activity not in nodes:
                if (activity != '&#9675;') and (activity != '
                   &#9679;'):
                    nodes.append(activity)

#=======================================
# Compute metric type: NUMBER OF ARCS
# Count number of arcs in a process model (macro and micro),
   include arcs from start and end points
#=======================================
#estimate arcs from macro model
arcs = 0
FANOUT = 0
BeginEnd = 0
for macroactivity in model[0][1]:
    outgoing = 0
    for transition in macroactivity:
        if transition > 0:
            arcs += 1
            if (BeginEnd != 0) and (BeginEnd != len(model
               [0][1])):
                outgoing +=1
    FANOUT += outgoing
    BeginEnd += 1
FANOUT /= (metrics[1][1] - 2)
metrics[1][2] = arcs
metrics[1][10] = FANOUT

#estimate arcs from micro models
```

```
for activity in bestmodel[0][0][1:-1]:
    FANOUT = 0
    arcs = 0
    line = 0
    for Mmodel in bestmodel[1][activity][1]:
        for transition in Mmodel:
            if transition > 0:
                arcs += 1
                if (line != 0) and (line != len(bestmodel[1][
                   activity][1])):
                    FANOUT += 1
        line += 1
    FANOUT /= (len(bestmodel[1][activity][1]) - 2)

    for line in range(1,len(metrics)):
        if metrics[line][0] == md.keys()[md.values().index(
           activity)]:
            metrics[line][2] =  arcs
            metrics[line][10] =  FANOUT

    NAN = metrics[line][2] / metrics[line][1]
    CC = metrics[line][2] - metrics[line][1] + 2
    metrics[line][3] = NAN

#=======================================
# Compute metric type: (FAN-IN)
# Number of other modules that are called from the module under
    investigation
#=======================================

#estimate FAN-IN from macro model
IncomingArcs = 0
for line in range(len(model[0][1])):
   for col in model[0][1]:
       if col[line] > 0:
           if (line != (len(model[0][1])-1)):
               IncomingArcs += 1
IncomingArcs /= (metrics[1][1] - 2)
metrics[1][9] = IncomingArcs

for activity in bestmodel[0][0][1:-1]:
    arcs = 0
    line = 0
    IncomingArcs = 0
    for line in range(len(bestmodel[1][activity][1])):
        for col in bestmodel[1][activity][1]:
            if col[line] > 0:
                if (line != (len(col)-1)):
                    IncomingArcs += 1
```

```
    IncomingArcs /= (len(bestmodel[1][activity][1])- 2)

    for line in range(1,len(metrics)):
        if metrics[line][0] == md.keys()[md.values().index(
           activity)]:
            metrics[line][9] =  IncomingArcs


#=======================================
# Compute metric type: NUMBER OF ARCS PER NODE
# Quotient of the number of arcs and the number of nodes
#
# Compute metric type: CYCLOMATIC COMPLEXITY
# measures the relation between the amount of arcs against the
   amount of nodes and arcs
#=======================================
for line in range(1,len(metrics)):
    NAN = metrics[line][2] / metrics[line][1]
    CC = metrics[line][2] - metrics[line][1] + 2
    metrics[line][3] = NAN
    metrics[line][8] = CC

#=======================================
# Compute metric type: RELATIONAL DENSITY
# Quotient of the number of arcs and the number of possible
   arcs
# Obs: Without begin and end arcs
#=======================================

for line in range(1,len(metrics)):
    states = metrics[line][1] - 2
    RD =metrics[line][2] / ((states * 2) + (states * states))
    metrics[line][4] = RD

states = (metrics[1][1] - 2)
metrics[1][4] = metrics[1][2] / ((states * 2) + (states *
   states))

#=======================================
# Compute metric type: NUMBER OF PATHS
# Number of all possible distinct paths per pair of start and
   end-nodes
#
# Compute metric type: PATH LENGTH
# Number of nodes between start and end-node (for a path)
#
# Compute metric type: NUMBER OF CYCLES
# Are evaluated by counting the number of cycles, involved
   nodes, and edges
```

```
#=======================================

# MACRO MODEL
model = bestmodel[0]
traces = []
for steps in range(0, 100):
    simmacro = runmodel(model)
    if simmacro not in traces:
        traces.append(simmacro)
NP = len(traces)


# METRIC: PATH LENGTH
G=nx.DiGraph()
count = avglength = 0
for path in traces:
    count += 1
    avglength += len(path)
    # compute cycles in a path
    # add nodes
    for task in path:
        if task not in G:
            G.add_node(task)
    # add edges
    for edgetimes in range(0,len(path)-1):
        G.add_edge(path[edgetimes],path[edgetimes+1])
PL = avglength / count
NC = len(nx.simple_cycles(G))

metrics[1][5] = NP
metrics[1][6] = PL
metrics[1][7] = NC
metrics[1][11] = (float(format(metrics[1][9], '.2f')) *  float(
   format(metrics[1][10], '.2f'))) * (float(format(metrics
   [1][9], '.2f')) *  float(format(metrics[1][10], '.2f')))

# MICRO MODELS
for micromodel in bestmodel[1]:
    model = bestmodel[1][micromodel]
    traces = []
    for steps in range(0, 100):
        simmacro = runmodel(model)
        if simmacro not in traces:
            traces.append(simmacro)
    NP = len(traces)

    # METRIC: PATH LENGTH
    G=nx.DiGraph()
    count = avglength = 0
```

```
    for path in traces:
        count += 1
        avglength += len(path)
        # compute cycles in a path
        # add nodes
        for task in path:
            if task not in G:
                G.add_node(task)
        # add edges
        for edgetimes in range(0,len(path)-1):
            G.add_edge(path[edgetimes],path[edgetimes+1])
    PL = avglength / count
    NC = len(nx.simple_cycles(G))

    for line in range(1,len(metrics)):
        if metrics[line][0] == md.keys()[md.values().index(
           micromodel)]:
            if (metrics[line][10] > 0) and (metrics[line][9] >
               0):
                FANINOUT = ((metrics[line][9]*metrics[line
                   ][10]) * metrics[line][9])
            else:
                FANINOUT = 0
            metrics[md.keys()[md.values().index(micromodel)
               ]][5] = NP
            metrics[md.keys()[md.values().index(micromodel)
               ]][6] = PL
            metrics[md.keys()[md.values().index(micromodel)
               ]][7] = NC
            fi = float(format(metrics[md.keys()[md.values().
               index(micromodel)]][9], '.2f'))
            fo = float(format(metrics[md.keys()[md.values().
               index(micromodel)]][10], '.2f'))
            metrics[md.keys()[md.values().index(micromodel)
               ]][11] = (fi * fo) * (fi * fo)



#=======================================
# Print result - summary of metrics
#=======================================
print '## 1: (NN) Number of nodes - Size '
print '## 2: (NA) Number of arcs - Size '
print '## 3: (NAN) Number of arcs per node - Density '
print '## 4: (RD) Relational Density - Density '
print '## 5: (NP) Number of paths - Paths'
print '## 6: (NP) Path Length - Paths'
print '## 7: (NC) Number of Cycles - Cycles'
print '## 8: (CC) Cyclomatic Complexity - Cycles'
```

```
    print '## 9: (FAN-IN) - Modularity'
    print '## 10: (FAN-OUT) - Modularity '
    print '## 11: (FANINOUT) - Modularity '

    print '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - -'

    print "%15s" %'', ["%6s" % v for v in metrics[0]]
    metrics.pop(0)
    for line in range(len(metrics)):
            print "%15s" % md[metrics[line][0]], ["%6.2f" % v for v
                in metrics[line]]
    print '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - -'

    computeM =  []
    for tam in range(len(metrics[0])):
        computeM.append(sum(a[tam] for a in metrics))
    print "%15s" % 'SUM all models', ["%6.2f" % v for v in computeM
       ]

#'- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - -'

    computeM =  []
    for tam in range(len(metrics[0])):
        computeM.append(sum(a[tam] / len(metrics) for a in metrics)
           )
    print "%15s" % 'AVG all models', ["%6.2f" % v for v in computeM
       ]

#'- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - -'

    computeM =  []
    for tam in range(len(metrics[0])):
        computeM.append(sum(a[tam] for a in metrics[1:len(metrics)
           ]))
    print "%15s" % 'SUM MicroModels', ["%6.2f" % v for v in
       computeM]

#'- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    - - -'
```

```
    computeM =  []
    for tam in range(len(metrics[0])):
        computeM.append(sum(a[tam] / (len(metrics) -1) for a in
           metrics[1:len(metrics)]))
    print "%15s" % 'AVG MicroModels', ["%6.2f" % v for v in
       computeM]
    print '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       - - - - - - - - - -'

# FUNCTIONS

def Initialize_Metrics(model, bestmodel, microseqs):
   Compute_Metrics(model, bestmodel)


# Random draw from list of probabilities
def draw(probs):
   rand = random.random()
   sumprob = 0.0
   for i in range(0, len(probs)):
       sumprob += probs[i]
       if rand < sumprob:
           return i

# Generate a sequence from a markov chain
def runmarkov(markov):
   (states, matrix) = markov
   sequence = []
   i = draw(matrix[0])
   while states[i] != states[-1]:
       sequence.append(states[i])
       i = draw(matrix[i])
   return sequence

# Run a hierarchical markov model
def runmodel(model):
   macroseq = runmarkov(model)
   return macroseq

 # Get transition probabilities from sequences
def seqs2markov(seqs):
    # get list of states
    states = set()
    for seq in seqs:
        for i in range(0, len(seq)):
            states.add(seq[i])
    states = [BEGIN] + sorted(list(states)) + [END]
    # count transitions between states
```

```
    matrix = []
    for i in range(0, len(states)):
        matrix.append(len(states) * [0.0])
    for seq in seqs:
        for i in range(0, len(seq)):
            a = states.index(seq[i])
            if i == 0:
                matrix[0][a] += 1.0
            if i < len(seq) - 1:
                b = states.index(seq[i + 1])
                matrix[a][b] += 1.0
            if i == len(seq) - 1:
                matrix[a][-1] += 1.0
    # normalize matrix
    for i in range(0, len(matrix)):
        rowsum = 0.0
        for j in range(0, len(matrix[i])):
            rowsum += matrix[i][j]
        if rowsum > 0.0:
            for j in range(0, len(matrix[i])):
                matrix[i][j] /= rowsum
    markov = (states, matrix)
    return markov


def markov2dot(markov, name):
    name = '.\\Temp\\' + name
    fout = open(name+'.dot', 'w')
    (states, matrix) = markov
    fout.write('digraph model { \n')
    fout.write('\t rankdir = "LR"; \n')
    fout.write('\t ranksep = 0.3; \n')
    fout.write('\t node [shape = doublecircle]; ')
    if BEGIN[0:2] == '&#':
        fout.write('<'+BEGIN+'> ')
    else:
        fout.write('"'+BEGIN+'" ')
    if END[0:2] == '&#':
        fout.write('<'+END+'>\n')
    else:
        fout.write('"'+END+'"\n')
    fout.write('\t node [shape = rectangle]; \n')
    for i in range(0,len(states)):
        for j in range(0,len(states)):
            p = matrix[i][j]
            if p > 0.0:
                fout.write('\t ')
                if states[i] == BEGIN:
                    if BEGIN[0:2] == '&#':
```

```
                    fout.write('<'+BEGIN+'> ')
                else:
                    fout.write('"'+BEGIN+'" ')
            else:
                fout.write('"'+states[i]+'" ')
            fout.write('-> ')
            if states[j] == END:
                if END[0:2] == '&#':
                    fout.write('<'+END+'> ')
                else:
                    fout.write('"'+END+'" ')
            else:
                fout.write('"'+states[j]+'" ')
            fout.write('[penwidth=' + str(1.0+p**2))
            fout.write(', label = "' + str(round(p,2)) + '"]; \
               n')
fout.write('} \n')
fout.flush()
fout.close()
cmd = 'dot -Tsvg ' + name + '.dot > ' + name + '.svg'
print 'Running:', cmd
os.system(cmd)
cmd = 'java -jar ..\\batik-1.7\\batik-rasterizer.jar -m
   application/pdf ' + name + '.svg'
print 'Running:', cmd
os.system(cmd)
```

Apêndice C

Cenário de Simulação do AOR

C.1 Processo de compra

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="prettyprint.xsl"?>
3
4 <SimulationScenario version="0.8.3" xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance"
5      xsi:schemaLocation="http://aor-simulation.org ../../../ext/aorsl/
          AORSL-0-8-3.xsd"
6      xmlns="http://aor-simulation.org" xmlns:aors="http://aor-simulation.
          org"
7      xmlns:dc="http://purl.org/dc/elements/1.1/"
8      scenarioName="PurchaseProcess_AgentBasedVersion"
9      scenarioTitle="A purchasing scenario with a four types of agent:
          Employee, Warehouse, Purchasing, and Supplier"
10     simulationManagerDirectory="../../..">
11
12     <SimulationParameters simulationSteps="10000" stepDuration="1"
          timeUnit="s" randomSeed="100" />
13     <SimulationModel modelName="PurchaseProcess_AgentBasedVersion"
          modelTitle="A purchase process model">
14
15         <documentation>
16             <description>In a company, an employee needs a certain
                   commodity (e.g. a printer cartridge). If the product is
                   available at the warehouse, then the warehouse dispatches
                    the product to the employee. Other- wise, the product
                   must be purchased from an external supplier. All
                   purchases must be previously approved by the purchasing
                   department. If the purchase is not approved, the process
                   ends immediately. If the purchase is approved, the
                   process proceeds with the purchasing department ordering
                   and paying for the product from the supplier. The
                   supplier delivers the product to the warehouse, and the
                   warehouse dispatches the product to the employee.</
                   description>
17             <dc:created>20120530</dc:created>
18             <dc:modified>20120530</dc:modified>
19             <dc:creator>Fernando Szimanski</dc:creator>
```

```
20              <dc:contributor>Diogo R. Ferreira</dc:contributor>
21              <dc:modified>20120530</dc:modified>
22              <dc:source>"A Hierarchical Markov Model to Understand the
                    Behaviour of Agents in Business Processes. In: BPI
                    2012.</dc:source>
23              <processModelDiagram>case-study.png</processModelDiagram>
24          </documentation>
25
26
27          <Globals>
28
29              <GlobalFunction name="twoDice" resultType="Integer">
30
31                  <Parameter name="factor" type="Integer" />
32                  <Body language="Java"><![CDATA[
33 int r1 = Random.uniformInt(1,6);
34 int r2 = Random.uniformInt(1,6);
35     if (r1+r2 < 3)  return (2*factor);            // with probability
           0.028
36     if (r1+r2 < 4)  return (3*factor);            // with probability
           0.056
37     if (r1+r2 < 5)  return (4*factor);            // with probability
           0.083
38     if (r1+r2 < 6)  return (5*factor);            // with probability
           0.111
39     if (r1+r2 < 7)  return (6*factor);            // with probability
           0.139
40     if (r1+r2 < 8)  return (7*factor);            // with probability
           0.166
41     if (r1+r2 < 9)  return (8*factor);            // with probability
           0.139
42     if (r1+r2 < 10) return (9*factor);            // with probability
           0.111
43     if (r1+r2 < 11) return (10*factor);           // with probability
           0.083
44     if (r1+r2 < 12) return (11*factor);           // with probability
           0.056
45                     return (12*factor);          // with probability
                           0.028
46              ]]></Body>
47              </GlobalFunction>
48          </Globals>
49
50          <EntityTypes>
51
52              <MessageType name="StockRequest" >
53
54                  <Attribute type="Integer" name="CaseId" />
55              </MessageType>
56
57              <MessageType name="StockResponse" >
58
59                  <Attribute type="Integer" name="CaseId" />
60              </MessageType>
61
62              <MessageType name="FetchProduct" >
```

```
63
64              <Attribute type="Integer" name="CaseId" />
65          </MessageType>
66
67          <MessageType name="ProductReady" >
68
69              <Attribute type="Integer" name="CaseId" />
70          </MessageType>
71
72          <MessageType name="ProductReceived" >
73
74              <Attribute type="Integer" name="CaseId" />
75          </MessageType>
76
77          <MessageType name="PurchaseRequest" >
78
79              <Attribute type="Integer" name="CaseId" />
80          </MessageType>
81
82          <MessageType name="InfoRequest" >
83
84              <Attribute type="Integer" name="CaseId" />
85          </MessageType>
86
87          <MessageType name="InfoResponse" >
88
89              <Attribute type="Integer" name="CaseId" />
90          </MessageType>
91
92          <MessageType name="ApprovalResult" >
93
94              <Attribute type="Integer" name="CaseId" />
95          </MessageType>
96
97          <MessageType name="PurchaseOrder" >
98
99              <Attribute type="Integer" name="CaseId" />
100         </MessageType>
101
102         <MessageType name="PaymentTerms" >
103
104             <Attribute type="Integer" name="CaseId" />
105         </MessageType>
106
107         <MessageType name="PaymentVoucher" >
108
109             <Attribute type="Integer" name="CaseId" />
110         </MessageType>
111
112         <MessageType name="DeliveryNote" >
113
114             <Attribute type="Integer" name="CaseId" />
115         </MessageType>
116
117         <MessageType name="ProductAvailable" >
118
```

```
119             <Attribute type="Integer" name="CaseId" />
120         </MessageType>
121
122
123         <ExogenousEventType name="EmployeeRequest">
124
125         <Periodicity>
126           <ValueExpr language="Java">Global.twoDice(10)</ValueExpr>
127         </Periodicity>
128       </ExogenousEventType>
129
130         <CausedEventType name="StartPurchaseService">
131
132             <Attribute type="Integer" name="StartPurchaseServiceProp
                    " />
133         </CausedEventType>
134
135         <CausedEventType name="EndPurchaseService">
136
137             <Attribute type="Integer" name="EndPurchaseServiceProp"
                    />
138         </CausedEventType>
139
140         <CausedEventType name="PurchaseNotApproved">
141
142             <Attribute type="Integer" name="PurchaseNotApprovedProp"
                    />
143         </CausedEventType>
144
145         <CausedEventType name="PurchaseApproved">
146
147             <Attribute type="Integer" name="PurchaseApprovedProp" />
148         </CausedEventType>
149
150         <AgentType name="Employee" >
151
152             <CommunicationRule name="StockResponse_Rule"
                    agentVariable="emp" >
153
154                 <documentation>
155                     <description>Employee receives StockResponse and
                            sends either FetchProduct or PurchaseRequest
                           </description>
156                 </documentation>
157                 <WHEN eventType="InMessageEvent" messageType="
                        StockResponse" eventVariable="evt"
                        messageVariable="mes"/>
158                 <IF language="Java"><![CDATA[ Random.uniformInt
                        (1,10) <= 5 ]]></IF>
159                 <THEN>
160                     <SCHEDULE-EVT>
161                         <OutMessageEventExpr messageType="
                                FetchProduct" >
162                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
```

```
163                         <Slot property="CaseId">
164                             <ValueExpr language="Java"><![CDATA[
                                    emp.getId() ]]></ValueExpr>
165                         </Slot>
166                     </OutMessageEventExpr>
167                 </SCHEDULE-EVT>
168             </THEN>
169             <ELSE>
170                 <SCHEDULE-EVT>
171                     <OutMessageEventExpr messageType="
                            PurchaseRequest">
172                         <ReceiverIdRef language="Java"><![CDATA[
                                2 ]]></ReceiverIdRef>
173                         <Slot property="CaseId">
174                             <ValueExpr language="Java"><![CDATA[
                                    mes.getCaseId()  ]]></ValueExpr>
175                         </Slot>
176                     </OutMessageEventExpr>
177                 </SCHEDULE-EVT>
178             </ELSE>
179         </CommunicationRule>
180
181         <CommunicationRule name="ProductReady_Rule" >
182
183             <documentation>
184                 <description>Employee receives ProductReady and
                        sends ProductReceived</description>
185             </documentation>
186             <WHEN eventType="InMessageEvent" messageType="
                    ProductReady" eventVariable="evt"/>
187             <DO>
188                 <SCHEDULE-EVT>
189                     <OutMessageEventExpr messageType="
                            ProductReceived" >
190                         <ReceiverIdRef language="Java"><![CDATA[
                                evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
191                     </OutMessageEventExpr>
192                 </SCHEDULE-EVT>
193             </DO>
194         </CommunicationRule>
195
196         <CommunicationRule name="InfoRequest_Rule"  >
197
198             <documentation>
199                 <description>Employee receives InfoRequest and
                        sends InfoResponse</description>
200             </documentation>
201             <WHEN eventType="InMessageEvent" messageType="
                    InfoRequest" eventVariable="evt" messageVariable=
                    "mes"/>
202             <DO>
203                 <SCHEDULE-EVT>
204                     <OutMessageEventExpr messageType="
                            InfoResponse">
```

```
205                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
206                             <Slot property="CaseId">
207                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
208                             </Slot>
209                         </OutMessageEventExpr>
210                     </SCHEDULE_EVT>
211                 </DO>
212             </CommunicationRule>
213
214             <CommunicationRule name="ProductAvailable_Rule"  >
215
216                 <documentation>
217                     <description>Employee receives ProductAvailable
                            and sends FetchProduct</description>
218                 </documentation>
219                 <WHEN eventType="InMessageEvent" messageType="
                        ProductAvailable" eventVariable="evt"
                        messageVariable="mes" />
220                 <DO>
221                     <SCHEDULE_EVT>
222                         <OutMessageEventExpr messageType="
                                FetchProduct">
223                             <ReceiverIdRef language="Java"><![CDATA[
                                    1 ]]></ReceiverIdRef>
224                             <Slot property="CaseId">
225                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
226                             </Slot>
227                         </OutMessageEventExpr>
228                     </SCHEDULE_EVT>
229                 </DO>
230             </CommunicationRule>
231         </AgentType>
232
233         <AgentType name="Warehouse">
234
235             <CommunicationRule name="StockRequest_Rule"
                    agentVariable="wha" >
236
237                 <documentation>
238                     <description>Warehouse receives StockRequest and
                            sends StockResponse</description>
239                 </documentation>
240                 <WHEN eventType="InMessageEvent" messageType="
                        StockRequest" eventVariable="evt" />
241                 <DO>
242                     <SCHEDULE_EVT>
243                         <OutMessageEventExpr messageType="
                                StockResponse">
244                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
245                             <Slot property="CaseId">
```

```
246                                 <ValueExpr language="Java"><![CDATA[
                                        evt.getSenderIdRef() ]]></
                                    ValueExpr>
247                             </Slot>
248                         </OutMessageEventExpr>
249
250                     </SCHEDULE_EVT>
251                 </DO>
252             </CommunicationRule>
253
254             <CommunicationRule name="FetchProduct_Rule" >
255
256                 <documentation>
257                     <description>Warehouse receives FetchProduct and
                            sends ProductReady</description>
258                 </documentation>
259                 <WHEN eventType="InMessageEvent" messageType="
                        FetchProduct" eventVariable="evt" messageVariable
                        ="mes"/>
260                 <DO>
261                     <SCHEDULE_EVT>
262                         <OutMessageEventExpr messageType="
                                ProductReady" >
263                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
264                             <Slot property="CaseId">
265                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
266                             </Slot>
267                         </OutMessageEventExpr>
268                     </SCHEDULE_EVT>
269                 </DO>
270             </CommunicationRule>
271
272             <CommunicationRule name="DeliveryNote_Rule" >
273
274                 <documentation>
275                     <description>Warehouse receives DeliveryNote and
                            sends ProductAvailable</description>
276                 </documentation>
277                 <WHEN eventType="InMessageEvent" messageType="
                        DeliveryNote" eventVariable="evt" messageVariable
                        ="mes" />
278                 <DO>
279                     <SCHEDULE_EVT>
280                         <OutMessageEventExpr messageType="
                                ProductAvailable" >
281                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
282                             <Slot property="CaseId">
283                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
284                             </Slot>
285                         </OutMessageEventExpr>
286                     </SCHEDULE_EVT>
```

```
287                     </DO>
288                 </CommunicationRule>
289             </AgentType>
290
291             <AgentType name="Purchasing">
292
293                 <CommunicationRule name="PurchaseRequest_Rule"
                        agentVariable="pur" >
294
295                     <documentation>
296                         <description>Purchasing receives PurchaseRequest
                                and sends either InfoRequest or
                                ApprovalResult</description>
297                     </documentation>
298                     <WHEN eventType="InMessageEvent" messageType="
                            PurchaseRequest" eventVariable="evt"
                            messageVariable="mes" />
299                     <IF language="Java"><![CDATA[ Random.uniformInt
                            (1,10) <= 5 ]]></IF>
300                     <THEN>
301                         <SCHEDULE-EVT>
302                             <OutMessageEventExpr messageType="
                                    InfoRequest">
303                                 <ReceiverIdRef language="Java"><![CDATA[
                                        evt.getSenderIdRef() ]]></
                                        ReceiverIdRef>
304                                 <Slot property="CaseId">
305                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId()  ]]></ValueExpr>
306                                 </Slot>
307                             </OutMessageEventExpr>
308                         </SCHEDULE-EVT>
309                     </THEN>
310                     <ELSE>
311                         <SCHEDULE-EVT>
312                             <OutMessageEventExpr messageType="
                                    ApprovalResult">
313                                 <ReceiverIdRef language="Java"><![CDATA[
                                        evt.getSenderIdRef() ]]></
                                        ReceiverIdRef>
314                                 <Slot property="CaseId">
315                                     <ValueExpr language="Java"><![CDATA[
                                            evt.getSenderIdRef() ]]></
                                            ValueExpr>
316                                 </Slot>
317                             </OutMessageEventExpr>
318                         </SCHEDULE-EVT>
319                     </ELSE>
320                 </CommunicationRule>
321
322                 <CommunicationRule name="InfoResponse_Rule"
                        agentVariable="pur" >
323
324                     <documentation>
```

```
325                     <description>Purchasing receives InfoResponse
                            and sends either ApprovalResult or
                            InfoRequest </description>
326                 </documentation>
327                 <WHEN eventType="InMessageEvent" messageType="
                        InfoResponse" eventVariable="evt" />
328                 <IF language="Java"><![CDATA[ Random.uniformInt
                        (1,10) <= 5 ]]></IF>
329                 <THEN>
330                     <SCHEDULE-EVT>
331                         <OutMessageEventExpr messageType="
                                ApprovalResult">
332                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                    ReceiverIdRef>
333                             <Slot property="CaseId">
334                                 <ValueExpr language="Java"><![CDATA[
                                        evt.getSenderIdRef() ]]></
                                        ValueExpr>
335                             </Slot>
336                         </OutMessageEventExpr>
337                     </SCHEDULE-EVT>
338                 </THEN>
339                 <ELSE>
340                     <SCHEDULE-EVT>
341                         <OutMessageEventExpr messageType="
                                InfoRequest">
342                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                    ReceiverIdRef>
343                             <Slot property="CaseId">
344                                 <ValueExpr language="Java"><![CDATA[
                                        evt.getSenderIdRef() ]]></
                                        ValueExpr>
345                             </Slot>
346                         </OutMessageEventExpr>
347                     </SCHEDULE-EVT>
348                 </ELSE>
349             </CommunicationRule>
350
351
352             <CommunicationRule name="PaymentTerms_Rule"
                    agentVariable="pur" >
353
354                 <documentation>
355                     <description> Purchasing receives PaymentTerms
                            and sends either PaymentVoucher or
                            PurchaseOrder </description>
356                 </documentation>
357                 <WHEN eventType="InMessageEvent" messageType="
                        PaymentTerms" eventVariable="evt" messageVariable
                        ="mes" />
358                 <IF language="Java"><![CDATA[ Random.uniformInt
                        (1,10) <= 5 ]]></IF>
359                 <THEN>
360                     <SCHEDULE-EVT>
```

```
361                         <OutMessageEventExpr messageType="
                                PaymentVoucher">
362                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
363                             <Slot property="CaseId">
364                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
365                             </Slot>
366                         </OutMessageEventExpr>
367                     </SCHEDULE-EVT>
368                 </THEN>
369                 <ELSE>
370                     <SCHEDULE-EVT>
371                         <OutMessageEventExpr messageType="
                                PurchaseOrder">
372                             <ReceiverIdRef language="Java"><![CDATA[
                                    evt.getSenderIdRef() ]]></
                                ReceiverIdRef>
373                             <Slot property="CaseId">
374                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
375                             </Slot>
376                         </OutMessageEventExpr>
377                     </SCHEDULE-EVT>
378                 </ELSE>
379             </CommunicationRule>
380         </AgentType>
381         <AgentType name="Supplier">
382
383             <CommunicationRule name="PurchaseOrder_Rule"
                    agentVariable="sup" >
384
385                 <documentation>
386                     <description> Supplier receives PurchaseOrder
                            and sends PaymentTerms </description>
387                 </documentation>
388                 <WHEN eventType="InMessageEvent" messageType="
                        PurchaseOrder" eventVariable="evt"
                        messageVariable="mes" />
389                 <DO>
390                     <SCHEDULE-EVT>
391                         <OutMessageEventExpr messageType="
                                PaymentTerms">
392                             <ReceiverIdRef language="Java"><![CDATA[
                                    2 ]]></ReceiverIdRef>
393                             <Slot property="CaseId">
394                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
395                             </Slot>
396                         </OutMessageEventExpr>
397                     </SCHEDULE-EVT>
398                 </DO>
399             </CommunicationRule>
400
```

```
401                 <CommunicationRule name="PaymentVoucher_Rule"
                        agentVariable="sup" >
402
403                     <documentation>
404                         <description> Supplier receives PaymentVoucher
                                and sends DeliveryNote </description>
405                     </documentation>
406                     <WHEN eventType="InMessageEvent" messageType="
                            PaymentVoucher" eventVariable="evt"
                            messageVariable="mes" />
407                     <DO>
408                         <SCHEDULE-EVT>
409                             <OutMessageEventExpr messageType="
                                    DeliveryNote">
410                                 <ReceiverIdRef language="Java"><![CDATA[
                                        1 ]]></ReceiverIdRef>
411                                 <Slot property="CaseId">
412                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
413                                 </Slot>
414                             </OutMessageEventExpr>
415                         </SCHEDULE-EVT>
416                     </DO>
417                 </CommunicationRule>
418             </AgentType>
419         </EntityTypes>
420
421         <EnvironmentRules>
422
423             <EnvironmentRule name="EmployeeRequest_Rule">
424
425                 <documentation>
426                     <description> Environment receives EmployeeRequest
                            sends EmployeeRequest </description>
427                 </documentation>
428                 <WHEN eventType="EmployeeRequest" eventVariable="evt" />
429                 <DO>
430                     <UPDATE-ENV>
431                         <Create>
432                             <Agent type="Employee" objectVariable="emp">
433                                 <Slot property="Timestamp">
434                                     <ValueExpr language="Java"><![CDATA[
                                            evt.getOccurrenceTime() ]]></
                                            ValueExpr>
435                                 </Slot>
436                             </Agent>
437                         </Create>
438                     </UPDATE-ENV>
439                     <SCHEDULE-EVT>
440                         <CausedEventExpr eventType="StartPurchaseService
                                ">
441                             <Slot property="StartPurchaseServiceProp">
442                                 <ValueExpr language="Java">emp.getId()</
                                        ValueExpr>
443                             </Slot>
444                         </CausedEventExpr>
```

```
445                 </SCHEDULE-EVT>
446             </DO>
447         </EnvironmentRule>
448
449         <EnvironmentRule name="StartPurchaseService_Rule">
450
451             <documentation>
452                 <description> Environment receives
                        StartPurchaseService and sends StockRequest </
                        description>
453             </documentation>
454             <WHEN eventType="StartPurchaseService" />
455             <FOR objectVariable="emp" objectType="Employee" />
456             <FOR objectVariable="wh" objectType="Warehouse"
                    objectIdRef="1" />
457             <DO>
458                 <SCHEDULE-EVT>
459                     <InMessageEventExpr messageType="StockRequest">
460                         <SenderIdRef language="Java"><![CDATA[ emp.
                                getId() ]]></SenderIdRef>
461                         <ReceiverIdRef language="Java"><![CDATA[ wh.
                                getId()  ]]></ReceiverIdRef>
462                         <Slot property="CaseId">
463                             <ValueExpr language="Java"><![CDATA[ emp
                                    .getId() ]]></ValueExpr>
464                         </Slot>
465                     </InMessageEventExpr>
466
467                 </SCHEDULE-EVT>
468             </DO>
469         </EnvironmentRule>
470
471         <EnvironmentRule name="EndPurchaseService_Rule">
472
473             <documentation>
474                 <description>Destroys Employee when ProductReceived
                        </description>
475             </documentation>
476             <WHEN eventType="OutMessageEvent" messageType="
                    ProductReceived" eventVariable="evt" />
477             <FOR objectVariable="emp" objectType="Employee" />
478             <DO>
479                 <UPDATE-ENV>
480                     <DestroyObject  objectType="Employee"
                            objectVariable="emp" ></DestroyObject>
481                 </UPDATE-ENV>
482             </DO>
483         </EnvironmentRule>
484
485         <EnvironmentRule name="ApprovalResult_Rule" >
486
487             <documentation>
488                 <description>Environment receives ApprovalResult and
                        sends either PurchaseApproved or
                        PurchaseNotApproved</description>
489             </documentation>
```

```
490             <WHEN eventType="OutMessageEvent" messageType="
                    ApprovalResult" eventVariable="evt"  messageVariable=
                    "mes"/>
491             <FOR objectVariable="emp" objectType="Employee" />
492             <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <=
                     5 ]]></IF>
493             <THEN>
494                 <SCHEDULE-EVT>
495                     <CausedEventExpr eventType="PurchaseNotApproved"
                            >
496                         <Slot property="PurchaseNotApprovedProp">
497                             <ValueExpr language="Java"><![CDATA[ emp
                                    .getId() ]]></ValueExpr>
498                         </Slot>
499                     </CausedEventExpr>
500                 </SCHEDULE-EVT>
501             </THEN>
502             <ELSE>
503                 <SCHEDULE-EVT>
504                     <CausedEventExpr eventType="PurchaseApproved">
505                         <Slot property="PurchaseApprovedProp">
506                             <ValueExpr language="Java"><![CDATA[ emp
                                    .getId()  ]]></ValueExpr>
507                         </Slot>
508                     </CausedEventExpr>
509                 </SCHEDULE-EVT>
510             </ELSE>
511         </EnvironmentRule>
512
513         <EnvironmentRule name="PurchaseApproved_Rule">
514
515             <documentation>
516                 <description> Environment receives PurchaseApproved
                        and sends PurchaseOrder </description>
517             </documentation>
518             <WHEN eventType="PurchaseApproved" eventVariable="evt"
                    messageVariable="mes"/>
519             <FOR objectVariable="emp" objectType="Employee" />
520             <DO>
521                 <SCHEDULE-EVT>
522                     <InMessageEventExpr messageType="PurchaseOrder">
523                         <SenderIdRef language="Java"><![CDATA[ 2
                                ]]></SenderIdRef>
524                         <ReceiverIdRef language="Java"><![CDATA[ 3
                                ]]></ReceiverIdRef>
525                         <Slot property="CaseId">
526                             <ValueExpr language="Java"><![CDATA[ emp
                                    .getId() ]]></ValueExpr>
527                         </Slot>
528                     </InMessageEventExpr>
529                 </SCHEDULE-EVT>
530             </DO>
531         </EnvironmentRule>
532
533         <EnvironmentRule name="PurchaseNotApproved_Rule">
534
```

```
535                 <documentation>
536                     <description> Destroys Employee when
                            PurchaseNotApproved </description>
537                 </documentation>
538                 <WHEN eventType="PurchaseNotApproved"  eventVariable="
                        evt" messageVariable="mes"/>
539                 <FOR objectVariable="emp" objectType="Employee" />
540                 <DO>
541                     <UPDATE-ENV>
542                         <DestroyObject  objectType="Employee"
                                objectVariable="emp" ></DestroyObject>
543                     </UPDATE-ENV>
544                 </DO>
545             </EnvironmentRule>
546
547         </EnvironmentRules>
548
549
550     </SimulationModel>
551
552
553     <InitialState>
554
555         <Agent type="Warehouse" name="wh" id="1" />
556
557         <Agent type="Purchasing" name="pc" id="2" />
558
559         <Agent type="Supplier" name="sp" id="3" />
560
561         <ExogenousEvent type="EmployeeRequest" occurrenceTime="1" />
562     </InitialState>
563
564 </SimulationScenario>
```

C.2 Processo de indenização de seguro

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="prettyprint.xsl"?>
3
4 <SimulationScenario version="0.8.3" xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance"
5     xsi:schemaLocation="http://aor-simulation.org ../../../ext/aorsl/
          AORSL-0-8-3.xsd"
6     xmlns="http://aor-simulation.org" xmlns:aors="http://aor-simulation.
          org"
7     xmlns:dc="http://purl.org/dc/elements/1.1/"
8     scenarioName="InsuranceClaim_AgentBasedVersion"
9     scenarioTitle="An Insurance Claim scenario with a three types of
          agent: Customer, Office, and Workshop"
10    simulationManagerDirectory="../../..">
11
12    <SimulationParameters simulationSteps="10000" stepDuration="1"
          timeUnit="s" randomSeed="100" />
```

```
13      <SimulationModel modelName="InsuranceClaim_AgentBasedVersion"
            modelTitle="An insurance claim model">
14
15          <documentation>
16              <description>A customer with an all-inclusive insurance plan
                     has damaged her vehicle and calls the insurance company
                    in order to submit a claim. After the claim has been
                    filled with all the details, the insurance company
                    assigns an expert to assess the damage. When the expert
                    returns the damage assessment, the company decides
                    whether to approve the claim or not. If the company does
                    not approve, the claim is rejected and the process ends
                    here. Otherwise, if the claim is approved, the customer
                    will have her car repaired at the workshop. After that,
                    the insurance company pays for the repair and waits for a
                     period of 30 days. If the customer does not say anything
                     within 30 days, the repair is assumed to have been
                    successful and the claim is closed. Otherwise, if the
                    customer complains within 30 days, the process goes back
                    to the damage assessment stage for a re-assessment and
                    possibly a new repair.</description>
17              <dc:created>20130415</dc:created>
18              <dc:modified> 20130415 </dc:modified>
19              <dc:creator>Fernando Szimanski</dc:creator>
20              <dc:contributor>Diogo R. Ferreira</dc:contributor>
21          <dc:contributor>Celia G. Ralha</dc:contributor>
22              <dc:modified> 20130415 </dc:modified>
23              <dc:source>"A Hierarchical Markov Model to Understand the
                    Behaviour of Agents in Business Processes. In: ISeb
                    2013.</dc:source>
24              <processModelDiagram>Insurance-Claim-v3.png</
                    processModelDiagram>
25          </documentation>
26
27
28          <Globals>
29
30              <GlobalFunction name="twoDice" resultType="Integer">
31
32                  <Parameter name="factor" type="Integer" />
33                  <Body language="Java"><![CDATA[
34  int r1 = Random.uniformInt(1,6);
35  int r2 = Random.uniformInt(1,6);
36      if (r1+r2 < 3)  return (2*factor);          // with probability
            0.028
37      if (r1+r2 < 4)  return (3*factor);          // with probability
            0.056
38      if (r1+r2 < 5)  return (4*factor);          // with probability
            0.083
39      if (r1+r2 < 6)  return (5*factor);          // with probability
            0.111
40      if (r1+r2 < 7)  return (6*factor);          // with probability
            0.139
41      if (r1+r2 < 8)  return (7*factor);          // with probability
            0.166
```

```
42    if (r1+r2 < 9)  return (8*factor);            // with probability
          0.139
43    if (r1+r2 < 10) return (9*factor);            // with probability
          0.111
44    if (r1+r2 < 11) return (10*factor);           // with probability
          0.083
45    if (r1+r2 < 12) return (11*factor);           // with probability
          0.056
46                    return (12*factor);          // with probability
                          0.028
47              ]]></Body>
48               </GlobalFunction>
49          </Globals>
50
51          <EntityTypes>
52
53              <MessageType name="ClaimEvent" >
54
55                  <Attribute type="Integer" name="CaseId" />
56              </MessageType>
57
58              <MessageType name="ClaimInstructions" >
59
60                  <Attribute type="Integer" name="CaseId" />
61              </MessageType>
62
63              <MessageType name="ClaimDescription" >
64
65                  <Attribute type="Integer" name="CaseId" />
66              </MessageType>
67
68              <MessageType name="DriveToWorkshop" >
69
70                  <Attribute type="Integer" name="CaseId" />
71              </MessageType>
72
73              <MessageType name="AssessVehicleDamage" >
74
75                  <Attribute type="Integer" name="CaseId" />
76              </MessageType>
77
78              <MessageType name="DamageAssessment" >
79
80                  <Attribute type="Integer" name="CaseId" />
81              </MessageType>
82
83              <MessageType name="RepairDenied" >
84
85                  <Attribute type="Integer" name="CaseId" />
86              </MessageType>
87
88              <MessageType name="ClaimRejected" >
89
90                  <Attribute type="Integer" name="CaseId" />
91              </MessageType>
92
```

```
93            <MessageType name="RepairOrder" >
94
95                <Attribute type="Integer" name="CaseId" />
96            </MessageType>
97
98            <MessageType name="RepairDateTime" >
99
100               <Attribute type="Integer" name="CaseId" />
101           </MessageType>
102
103           <MessageType name="VehicleRepaired" >
104
105               <Attribute type="Integer" name="CaseId" />
106           </MessageType>
107
108           <MessageType name="RepairInvoice" >
109
110               <Attribute type="Integer" name="CaseId" />
111           </MessageType>
112
113           <MessageType name="PaymentConfirmation" >
114
115               <Attribute type="Integer" name="CaseId" />
116           </MessageType>
117
118           <MessageType name="PaymentAck" >
119
120               <Attribute type="Integer" name="CaseId" />
121           </MessageType>
122
123           <MessageType name="ClaimClosed" >
124
125               <Attribute type="Integer" name="CaseId" />
126           </MessageType>
127
128           <MessageType name="NotSatisfied" >
129
130               <Attribute type="Integer" name="CaseId" />
131       </MessageType>
132
133           <MessageType name="Reclamation" >
134
135               <Attribute type="Integer" name="CaseId" />
136       </MessageType>
137
138           <ExogenousEventType name="CustomerRequest">
139
140       <Periodicity>
141          <ValueExpr language="Java">100</ValueExpr>
142       </Periodicity>
143        </ExogenousEventType>
144
145           <CausedEventType name="StartClaimEventService">
146
147               <Attribute type="Integer" name="
                      StartClaimEventServiceProp" />
```

```
148             </CausedEventType>
149
150             <CausedEventType name="EndClaimEventService">
151
152                 <Attribute type="Integer" name="EndClaimEventServiceProp
                        " />
153             </CausedEventType>
154
155             <CausedEventType name="EndClaimClosed">
156
157                 <Attribute type="Integer" name="EndClaimClosedProp" />
158             </CausedEventType>
159
160             <CausedEventType name="EndClaimRejected">
161
162                 <Attribute type="Integer" name="EndClaimRejectedProp" />
163             </CausedEventType>
164
165             <AgentType name="Customer" >
166
167                 <CommunicationRule name="ClaimInstructions_Rule"
                        agentVariable="cust" >
168
169                     <documentation>
170                         <description>Customer receives ClaimInstructions
                                and sends ClaimDescription</description>
171                     </documentation>
172                     <WHEN eventType="InMessageEvent" messageType="
                            ClaimInstructions" eventVariable="evt"
                            messageVariable="mes"/>
173                     <DO>
174                         <SCHEDULE-EVT>
175                             <OutMessageEventExpr messageType="
                                    ClaimDescription" >
176                                 <ReceiverIdRef language="Java">1</
                                        ReceiverIdRef>
177                                 <Slot property="CaseId">
178                                     <ValueExpr language="Java"><![CDATA[
                                            cust.getId() ]]></ValueExpr>
179                                 </Slot>
180                             </OutMessageEventExpr>
181                         </SCHEDULE-EVT>
182                     </DO>
183                 </CommunicationRule>
184
185                 <CommunicationRule name="Reclamation_Rule" agentVariable
                        ="cust" >
186
187                     <documentation>
188                         <description>Customer receives Reclamation and
                                sends NotSatisfied</description>
189                     </documentation>
190                     <WHEN eventType="InMessageEvent" messageType="
                            Reclamation" eventVariable="evt"  messageVariable
                            ="mes"/>
191                     <DO>
```

```
192                         <SCHEDULE-EVT>
193                             <OutMessageEventExpr messageType="
                                    NotSatisfied" >
194                                 <ReceiverIdRef language="Java">1</
                                        ReceiverIdRef>
195                                 <Slot property="CaseId">
196                                     <ValueExpr language="Java"><![CDATA[
                                            cust.getId() ]]></ValueExpr>
197                                 </Slot>
198                             </OutMessageEventExpr>
199                         </SCHEDULE-EVT>
200                     </DO>
201                 </CommunicationRule>
202             </AgentType>
203
204             <AgentType name="Office">
205
206                 <CommunicationRule name="ClaimEvent_Rule"  agentVariable
                        ="off" >
207
208                     <documentation>
209                         <description>Office receives ClaimEvent and
                                sends StockResponse</description>
210                     </documentation>
211                     <WHEN eventType="InMessageEvent" messageType="
                            ClaimEvent" eventVariable="evt" messageVariable="
                            mes"/>
212                     <DO>
213                         <SCHEDULE-EVT>
214                             <OutMessageEventExpr messageType="
                                    ClaimInstructions">
215                                 <ReceiverIdRef language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ReceiverIdRef>
216                                 <Slot property="CaseId">
217                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId()  ]]></ValueExpr>
218                                 </Slot>
219                             </OutMessageEventExpr>
220                         </SCHEDULE-EVT>
221                     </DO>
222                 </CommunicationRule>
223
224                 <CommunicationRule name="ClaimDescription_Rule" >
225
226                     <documentation>
227                         <description>Office receives ClaimDescription
                                and sends either DriveToWorkshop and
                                AssessVehicleDamage</description>
228                     </documentation>
229                     <WHEN eventType="InMessageEvent" messageType="
                            ClaimDescription" eventVariable="evt"
                            messageVariable="mes"/>
230                     <DO>
231                         <SCHEDULE-EVT>
232                             <OutMessageEventExpr messageType="
                                    DriveToWorkshop">
```

```
233                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId()  ]]></ReceiverIdRef>
234                             <Slot property="CaseId">
235                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ValueExpr>
236                             </Slot>
237                         </OutMessageEventExpr>
238                         <OutMessageEventExpr messageType="
                                AssessVehicleDamage">
239                             <ReceiverIdRef language="Java">2</
                                ReceiverIdRef>
240                             <Slot property="CaseId">
241                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ValueExpr>
242                             </Slot>
243                         </OutMessageEventExpr>
244                     </SCHEDULE-EVT>
245                 </DO>
246             </CommunicationRule>
247
248             <CommunicationRule name="DamageAssessment_Rule" >
249
250                 <documentation>
251                     <description>Office receives DamageAssessment
                            and sends either RepairDenied and
                            ClaimRejected, or RepairOrder</description>
252                 </documentation>
253                 <WHEN eventType="InMessageEvent" messageType="
                        DamageAssessment" eventVariable="evt"
                        messageVariable="mes" />
254 <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 5
    ]]></IF>
255                 <THEN>
256                     <SCHEDULE-EVT>
257                         <OutMessageEventExpr messageType="
                                RepairDenied">
258                             <ReceiverIdRef language="Java">2</
                                ReceiverIdRef>
259                             <Slot property="CaseId">
260                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
261                             </Slot>
262                         </OutMessageEventExpr>
263
264                         <OutMessageEventExpr messageType="
                                ClaimRejected">
265                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
266                             <Slot property="CaseId">
267                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
268                             </Slot>
269                         </OutMessageEventExpr>
270
271                     </SCHEDULE-EVT>
272                 </THEN>
```

```
273                     <ELSE>
274                         <SCHEDULE-EVT>
275                             <OutMessageEventExpr messageType="
                                    RepairOrder">
276                                 <ReceiverIdRef language="Java">2</
                                        ReceiverIdRef>
277                                 <Slot property="CaseId">
278                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
279                                 </Slot>
280                             </OutMessageEventExpr>
281                         </SCHEDULE-EVT>
282                     </ELSE>
283                 </CommunicationRule>
284 
285                 <CommunicationRule name="RepairInvoice_Rule"
                        agentVariable="off" >
286 
287                     <documentation>
288                         <description>Office receives RepairInvoice and
                                sends PaymentConfirmation </description>
289                     </documentation>
290                     <WHEN eventType="InMessageEvent" messageType="
                            RepairInvoice" eventVariable="evt"
                            messageVariable="mes"/>
291                     <DO>
292                         <SCHEDULE-EVT>
293                             <OutMessageEventExpr messageType="
                                    PaymentConfirmation">
294                                 <ReceiverIdRef language="Java">2</
                                        ReceiverIdRef>
295                                 <Slot property="CaseId">
296                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
297                                 </Slot>
298                             </OutMessageEventExpr>
299 
300                         </SCHEDULE-EVT>
301                     </DO>
302                 </CommunicationRule>
303 
304                 <CommunicationRule name="PaymentAck_Rule"  agentVariable
                        ="off" >
305 
306                     <documentation>
307                         <description>Office receives PaymentAck and
                                sends either ClaimClosed or sometimes a
                                Reclamation</description>
308                     </documentation>
309                     <WHEN eventType="InMessageEvent" messageType="
                            PaymentAck" eventVariable="evt" messageVariable="
                            mes"/>
310             <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 8
                    ]]></IF>
311                     <THEN>
312                         <SCHEDULE-EVT>
```

```
313                         <OutMessageEventExpr messageType="
                                ClaimClosed">
314                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
315                             <Slot property="CaseId">
316                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
317                             </Slot>
318                         </OutMessageEventExpr>
319                     </SCHEDULE-EVT>
320                 </THEN>
321                 <ELSE>
322                     <SCHEDULE-EVT>
323                         <OutMessageEventExpr messageType="
                                Reclamation">
324                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
325                             <Slot property="CaseId">
326                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
327                             </Slot>
328                         </OutMessageEventExpr>
329                     </SCHEDULE-EVT>
330                 </ELSE>
331             </CommunicationRule>
332
333             <CommunicationRule name="NotSatisfied_Rule"
                    agentVariable="off" >
334
335                 <documentation>
336                     <description>Office receives NotSatisfied and
                            sends either DriveToWorkshop and
                            AssessVehicleDamage </description>
337                 </documentation>
338                 <WHEN eventType="InMessageEvent" messageType="
                        NotSatisfied" eventVariable="evt" messageVariable
                        ="mes"/>
339                 <DO>
340                     <SCHEDULE-EVT>
341                         <OutMessageEventExpr messageType="
                                DriveToWorkshop">
342                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
343                             <Slot property="CaseId">
344                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
345                             </Slot>
346                         </OutMessageEventExpr>
347                         <OutMessageEventExpr messageType="
                                AssessVehicleDamage">
348                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
349                             <Slot property="CaseId">
350                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
351                             </Slot>
```

```
352                         </OutMessageEventExpr>
353                     </SCHEDULE-EVT>
354                 </DO>
355             </CommunicationRule>
356         </AgentType>
357
358         <AgentType name="Workshop">
359
360             <CommunicationRule name="AssessVehicleDamage_Rule"
                    agentVariable="work" >
361
362                 <documentation>
363                     <description>Workshop receives
                            AssessVehicleDamage and sends
                            DamageAssessment</description>
364                 </documentation>
365                 <WHEN eventType="InMessageEvent" messageType="
                        AssessVehicleDamage" eventVariable="evt"
                        messageVariable="mes" />
366                 <DO>
367                     <SCHEDULE-EVT>
368                         <OutMessageEventExpr messageType="
                                DamageAssessment">
369                             <ReceiverIdRef language="Java">1</
                                    ReceiverIdRef>
370                             <Slot property="CaseId">
371                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ValueExpr>
372                             </Slot>
373                         </OutMessageEventExpr>
374                     </SCHEDULE-EVT>
375                 </DO>
376             </CommunicationRule>
377
378             <CommunicationRule name="RepairOrder_Rule"
                    agentVariable="work" >
379
380                 <documentation>
381                     <description>Workshop receives RepairOrder and
                            sends either RepairDateTime, VehicleRepaired
                            and, RepairInvoice </description>
382                 </documentation>
383                 <WHEN eventType="InMessageEvent" messageType="
                        RepairOrder" eventVariable="evt" messageVariable=
                        "mes"/>
384                 <DO>
385                     <SCHEDULE-EVT>
386                         <OutMessageEventExpr messageType="
                                RepairDateTime">
387                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId()  ]]></ReceiverIdRef>
388                             <Slot property="CaseId">
389                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ValueExpr>
390                             </Slot>
391                         </OutMessageEventExpr>
```

```
392
393                         <OutMessageEventExpr messageType="
                                VehicleRepaired">
394                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
395                             <Slot property="CaseId">
396                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
397                             </Slot>
398                         </OutMessageEventExpr>
399
400                         <OutMessageEventExpr messageType="
                                RepairInvoice">
401                             <ReceiverIdRef language="Java">1</
                                ReceiverIdRef>
402                             <Slot property="CaseId">
403                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
404                             </Slot>
405                         </OutMessageEventExpr>
406                     </SCHEDULE-EVT>
407             </DO>
408         </CommunicationRule>
409
410         <CommunicationRule name="PaymentConfirmation_Rule"
              agentVariable="off" >
411
412             <documentation>
413                 <description> Office receives
                        PaymentConfirmation and sends PaymentAck</
                        description>
414             </documentation>
415             <WHEN eventType="InMessageEvent" messageType="
                    PaymentConfirmation" eventVariable="evt"
                    messageVariable="mes" />
416             <DO>
417                 <SCHEDULE-EVT>
418                     <OutMessageEventExpr messageType="PaymentAck
                            ">
419                         <ReceiverIdRef language="Java">1</
                                ReceiverIdRef>
420                         <Slot property="CaseId">
421                             <ValueExpr language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ValueExpr>
422                         </Slot>
423                     </OutMessageEventExpr>
424                 </SCHEDULE-EVT>
425             </DO>
426         </CommunicationRule>
427     </AgentType>
428 </EntityTypes>
429
430 <EnvironmentRules>
431
432     <EnvironmentRule name="CustomerRequest_Rule">
433
```

```
434                 <documentation>
435                     <description> Environment receives CustomerRequest
                            and sends ClaimEvent </description>
436                 </documentation>
437                 <WHEN eventType="CustomerRequest" />
438                 <DO>
439                     <UPDATE-ENV>
440                         <Create>
441                             <Agent type="Customer" objectVariable="cust"
                                    >
442                                 <Slot property="Timestamp">
443                                     <ValueExpr language="Java"><![CDATA[
                                            evt.getOccurrenceTime() ]]></
                                        ValueExpr>
444                                 </Slot>
445                             </Agent>
446                         </Create>
447                     </UPDATE-ENV>
448                     <SCHEDULE-EVT>
449                         <CausedEventExpr eventType="
                                StartClaimEventService">
450                             <Slot property="StartClaimEventServiceProp">
451                                 <ValueExpr language="Java">cust.getId()
                                        </ValueExpr>
452                             </Slot>
453                         </CausedEventExpr>
454                     </SCHEDULE-EVT>
455                 </DO>
456             </EnvironmentRule>
457
458             <EnvironmentRule name="StartClaimEventService_Rule">
459
460                 <documentation>
461                     <description> Environment receives
                            StartClaimEventService and sends ClaimEvent </
                            description>
462                 </documentation>
463                 <WHEN eventType="StartClaimEventService" />
464                 <FOR objectVariable="cust" objectType="Customer" />
465                 <FOR objectVariable="off" objectType="Office"
                        objectIdRef="1" />
466                 <DO>
467                     <SCHEDULE-EVT>
468                         <InMessageEventExpr messageType="ClaimEvent">
469                             <SenderIdRef language="Java"><![CDATA[ cust.
                                    getId() ]]></SenderIdRef>
470                             <ReceiverIdRef language="Java"><![CDATA[ off
                                    .getId() ]]></ReceiverIdRef>
471                             <Slot property="CaseId">
472                                 <ValueExpr language="Java"><![CDATA[
                                        cust.getId() ]]></ValueExpr>
473                             </Slot>
474                         </InMessageEventExpr>
475                     </SCHEDULE-EVT>
476                 </DO>
477             </EnvironmentRule>
```

```
478
479
480             <EnvironmentRule name="EndClaimEventService_Rule">
481
482                 <documentation>
483                     <description>Destroys Customer when ClaimRejected </
                            description>
484                 </documentation>
485                 <WHEN eventType="OutMessageEvent" messageType="
                        ClaimRejected" eventVariable="evt" />
486       <FOR objectVariable="cust" objectType="Customer">
487           <ObjectIdRef language="Java"><![CDATA[ evt.getReceiverIdRef()
                ]]></ObjectIdRef>
488       </FOR>
489                 <DO>
490                     <UPDATE-ENV>
491                         <DestroyObject objectVariable="cust"> </
                                DestroyObject>
492                     </UPDATE-ENV>
493                 </DO>
494             </EnvironmentRule>
495
496             <EnvironmentRule name="EndClaimClosed_Rule">
497
498                 <documentation>
499                     <description>Destroys Customer when ClaimClosed </
                            description>
500                 </documentation>
501                 <WHEN eventType="OutMessageEvent" messageType="
                        ClaimClosed" eventVariable="evt" />
502       <FOR objectVariable="cust" objectType="Customer">
503           <ObjectIdRef language="Java"><![CDATA[ evt.getReceiverIdRef()
                ]]></ObjectIdRef>
504       </FOR>
505                 <DO>
506                     <UPDATE-ENV>
507                         <DestroyObject objectVariable="cust"> </
                                DestroyObject>
508                     </UPDATE-ENV>
509                 </DO>
510             </EnvironmentRule>
511         </EnvironmentRules>
512
513
514     </SimulationModel>
515
516
517     <InitialState>
518
519         <Agent type="Office" name="off" id="1" />
520
521         <Agent type="Workshop" name="work" id="2" />
522
523         <ExogenousEvent type="CustomerRequest" occurrenceTime="1" />
524     </InitialState>
525
```

```
526 </SimulationScenario>
```

C.3 Processo de empréstimo bancário

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="prettyprint.xsl"?>
3
4 <SimulationScenario version="0.8.3" xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance"
5      xsi:schemaLocation="http://aor-simulation.org ../../../ext/aorsl/
          AORSL-0-8-3.xsd"
6      xmlns="http://aor-simulation.org" xmlns:aors="http://aor-simulation.
          org"
7      xmlns:dc="http://purl.org/dc/elements/1.1/"
8      scenarioName="BankLoan_AgentBasedVersion"
9      scenarioTitle="A Bank Loan scenario with a four types of agent:
          Customer, Clerk, CreditAgency, LoanSystem and, Manager"
10     simulationManagerDirectory="../../..">
11
12     <SimulationParameters simulationSteps="10000" stepDuration="1"
          timeUnit="s" randomSeed="100" />
13     <SimulationModel modelName="InsuranceClaim_AgentBasedVersion"
          modelTitle="A Bank Loan model">
14
15         <documentation>
16             <description>The process begins when a customer prepares and
                   submits a loan application to the bank. As a preliminary
                   step, a bank clerk will look at the application and
                  check if it complies with certain rules. In particular,
                  the clerk will check the financial situation of the
                  customer and determine whether the customer is eligible
                  for a loan. If the applicant or the loan application does
                   not comply with the rules, the process ends immediately
                  at this stage. Otherwise, it proceeds to a credit
                  analysis stage, where the bank gets in contact with an
                  external credit agency to determine whether the customer
                  has any outstanding payments elsewhere. Depending on the
                  information received from the credit agency, the bank
                  assigns a risk factor to the loan application.
                  Applications with a risk factor above a certain threshold
                   must be approved by a bank manager, who will decide
                  whether the funds should be released or not. For
                  applications with a low risk factor, the clerk may
                  release the funds immediately. In any case, the customer
                  must be notified of the outcome.</description>
17             <dc:created>20130416</dc:created>
18             <dc:modified> 20130416 </dc:modified>
19             <dc:creator>Fernando Szimanski</dc:creator>
20             <dc:contributor>Diogo R. Ferreira</dc:contributor>
21         <dc:contributor>Celia G. Ralha</dc:contributor>
22             <dc:modified> 20130416 </dc:modified>
```

```
23            <dc:source>"A Hierarchical Markov Model to Understand the
                  Behaviour of Agents in Business Processes. In: ISeb
                  2013.</dc:source>
24            <processModelDiagram>Bank-Loan-v2.png</processModelDiagram>
25        </documentation>
26
27
28        <Globals>
29
30            <GlobalFunction name="twoDice" resultType="Integer">
31
32                <Parameter name="factor" type="Integer" />
33                <Body language="Java"><![CDATA[
34 int r1 = Random.uniformInt(1,6);
35 int r2 = Random.uniformInt(1,6);
36    if (r1+r2 < 3)  return (2*factor);            // with probability
          0.028
37    if (r1+r2 < 4)  return (3*factor);            // with probability
          0.056
38    if (r1+r2 < 5)  return (4*factor);            // with probability
          0.083
39    if (r1+r2 < 6)  return (5*factor);            // with probability
          0.111
40    if (r1+r2 < 7)  return (6*factor);            // with probability
          0.139
41    if (r1+r2 < 8)  return (7*factor);            // with probability
          0.166
42    if (r1+r2 < 9)  return (8*factor);            // with probability
          0.139
43    if (r1+r2 < 10) return (9*factor);            // with probability
          0.111
44    if (r1+r2 < 11) return (10*factor);           // with probability
          0.083
45    if (r1+r2 < 12) return (11*factor);           // with probability
          0.056
46                    return (100);           // with probability  0.028
47            ]]></Body>
48            </GlobalFunction>
49        </Globals>
50
51        <EntityTypes>
52
53            <MessageType name="LoanRequest" >
54
55                <Attribute type="Integer" name="CaseId" />
56            </MessageType>
57
58            <MessageType name="AdditionalDetails" >
59
60                <Attribute type="Integer" name="CaseId" />
61            </MessageType>
62
63            <MessageType name="CreateLoanApplication" >
64
65                <Attribute type="Integer" name="CaseId" />
66            </MessageType>
```

```
67
68              <MessageType name="ApplyRules" >
69
70                  <Attribute type="Integer" name="CaseId" />
71              </MessageType>
72
73              <MessageType name="RulesResult" >
74
75                  <Attribute type="Integer" name="CaseId" />
76              </MessageType>
77
78              <MessageType name="UpdateLoanApplication" >
79
80                  <Attribute type="Integer" name="CaseId" />
81          <Attribute type="String" name="Result" />
82              </MessageType>
83
84              <MessageType name="RequestRecords" >
85
86                  <Attribute type="Integer" name="CaseId" />
87              </MessageType>
88
89              <MessageType name="CustomerRecords" >
90
91                  <Attribute type="Integer" name="CaseId" />
92              </MessageType>
93
94              <MessageType name="RequestApproval" >
95
96                  <Attribute type="Integer" name="CaseId" />
97              </MessageType>
98
99              <MessageType name="ApprovalResult" >
100
101                 <Attribute type="Integer" name="CaseId" />
102             </MessageType>
103
104             <MessageType name="RequestTransfer" >
105
106                 <Attribute type="Integer" name="CaseId" />
107             </MessageType>
108
109             <MessageType name="TransferCompleted" >
110
111                 <Attribute type="Integer" name="CaseId" />
112             </MessageType>
113
114             <MessageType name="ResultNotification" >
115
116                 <Attribute type="Integer" name="CaseId" />
117             </MessageType>
118
119             <ExogenousEventType name="CustomerRequest">
120
121         <Periodicity>
122            <ValueExpr language="Java">100</ValueExpr>
```

```
123         </Periodicity>
124          </ExogenousEventType>
125
126               <CausedEventType name="StartLoanRequestService">
127
128                    <Attribute type="Integer" name="
                          StartLoanRequestServiceProp" />
129               </CausedEventType>
130
131               <CausedEventType name="EndResultNotification">
132
133                    <Attribute type="Integer" name="
                          EndResultNotificationProp" />
134               </CausedEventType>
135
136
137               <AgentType name="Customer" >
138
139               </AgentType>
140
141               <AgentType name="Clerk">
142
143                    <CommunicationRule name="RulesResult_Rule" >
144
145                         <documentation>
146                              <description>Clerk receives RulesResult and
                                    sends either UpdateLoanApplication and
                                    RequestRecords</description>
147                         </documentation>
148                         <WHEN eventType="InMessageEvent" messageType="
                              RulesResult" eventVariable="evt" messageVariable=
                              "mes"/>
149           <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 8
                  ]]></IF>
150                         <THEN>
151           <SCHEDULE-EVT>
152                              <OutMessageEventExpr messageType="
                                    UpdateLoanApplication">
153              <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                       <= 5 ]]></Condition>
154                                   <ReceiverIdRef language="Java"><![CDATA[
                                         mes.getCaseId()  ]]></ReceiverIdRef>
155                                   <Slot property="CaseId">
156                                        <ValueExpr language="Java"><![CDATA[
                                              mes.getCaseId()  ]]></ValueExpr>
157                                   </Slot>
158              <Slot property="Result" value="True"></Slot>
159                              </OutMessageEventExpr>
160                              <OutMessageEventExpr messageType="
                                    RequestRecords">
161                                   <ReceiverIdRef language="Java">3</
                                         ReceiverIdRef>
162                                   <Slot property="CaseId">
163                                        <ValueExpr language="Java"><![CDATA[
                                              mes.getCaseId()  ]]></ValueExpr>
164                                   </Slot>
```

```
165                         </OutMessageEventExpr>
166                     </SCHEDULE-EVT>
167                 </THEN>
168                 <ELSE>
169                     <SCHEDULE-EVT>
170                         <OutMessageEventExpr messageType="
                                UpdateLoanApplication">
171           <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                <= 5 ]]></Condition>
172                         <ReceiverIdRef language="Java">2</
                                ReceiverIdRef>
173                         <Slot property="CaseId">
174                             <ValueExpr language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ValueExpr>
175                         </Slot>
176           <Slot property="Result" value="False"></Slot>
177                         </OutMessageEventExpr>
178
179                         <OutMessageEventExpr messageType="
                                ResultNotification">
180                         <ReceiverIdRef language="Java"><![CDATA[
                                mes.getCaseId() ]]></ReceiverIdRef>
181                         <Slot property="CaseId">
182                             <ValueExpr language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ValueExpr>
183                         </Slot>
184                         </OutMessageEventExpr>
185                     </SCHEDULE-EVT>
186                 </ELSE>
187             </CommunicationRule>
188
189             <CommunicationRule name="CustomerRecords_Rule" >
190
191                 <documentation>
192                     <description>Clerk receives CustomerRecords and
                            sends either UpdateLoanApplication and
                            RequestApproval</description>
193                 </documentation>
194                 <WHEN eventType="InMessageEvent" messageType="
                        CustomerRecords" eventVariable="evt"
                        messageVariable="mes" />
195           <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 7
                ]]></IF>
196                 <THEN>
197                     <SCHEDULE-EVT>
198                         <OutMessageEventExpr messageType="
                                UpdateLoanApplication">
199           <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                <= 5 ]]></Condition>
200                         <ReceiverIdRef language="Java">2</
                                ReceiverIdRef>
201                         <Slot property="CaseId">
202                             <ValueExpr language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ValueExpr>
203                         </Slot>
204           <Slot property="Result" value="True"></Slot>
```

```
205                         </OutMessageEventExpr>
206
207                         <OutMessageEventExpr messageType="
                                RequestApproval">
208                             <ReceiverIdRef language="Java">4</
                                    ReceiverIdRef>
209                             <Slot property="CaseId">
210                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
211                             </Slot>
212                         </OutMessageEventExpr>
213
214                     </SCHEDULE-EVT>
215                 </THEN>
216                 <ELSE>
217                     <SCHEDULE-EVT>
218                         <OutMessageEventExpr messageType="
                                UpdateLoanApplication">
219             <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                    <= 5 ]]></Condition>
220                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
221                             <Slot property="CaseId">
222                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
223                             </Slot>
224                             <Slot property="Result" value="False"></
                                    Slot>
225                         </OutMessageEventExpr>
226
227                         <OutMessageEventExpr messageType="
                                RequestTransfer">
228                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
229                             <Slot property="CaseId">
230                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
231                             </Slot>
232                         </OutMessageEventExpr>
233                     </SCHEDULE-EVT>
234                 </ELSE>
235             </CommunicationRule>
236
237             <CommunicationRule name="ApprovalResult_Rule"
                    agentVariable="cle" >
238
239                 <documentation>
240                     <description>Clerk receives ApprovalResult and
                            sends either UpdateLoanApplication and
                            RequestTransfer</description>
241                 </documentation>
242                 <WHEN eventType="InMessageEvent" messageType="
                        ApprovalResult" eventVariable="evt"
                        messageVariable="mes"/>
243             <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 4
                    ]]></IF>
```

```
244                     <THEN>
245                         <SCHEDULE-EVT>
246                             <OutMessageEventExpr messageType="
                                    UpdateLoanApplication">
247                 <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                        <= 5 ]]></Condition>
248                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
249                             <Slot property="CaseId">
250                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
251                             </Slot>
252                 <Slot property="Result" value="True"></Slot>
253                         </OutMessageEventExpr>
254                         <OutMessageEventExpr messageType="
                                RequestTransfer">
255                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
256                             <Slot property="CaseId">
257                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
258                             </Slot>
259                         </OutMessageEventExpr>
260                     </SCHEDULE-EVT>
261             </THEN>
262                     <ELSE>
263                         <SCHEDULE-EVT>
264                             <OutMessageEventExpr messageType="
                                    UpdateLoanApplication">
265                 <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                        <= 5 ]]></Condition>
266                             <ReceiverIdRef language="Java">2</
                                    ReceiverIdRef>
267                             <Slot property="CaseId">
268                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
269                             </Slot>
270                 <Slot property="Result" value="False"></Slot>
271                         </OutMessageEventExpr>
272
273                         <OutMessageEventExpr messageType="
                                ResultNotification">
274                             <ReceiverIdRef language="Java"><![CDATA[
                                    mes.getCaseId() ]]></ReceiverIdRef>
275                             <Slot property="CaseId">
276                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
277                             </Slot>
278                         </OutMessageEventExpr>
279                     </SCHEDULE-EVT>
280                 </ELSE>
281             </CommunicationRule>
282
283             <CommunicationRule name="TransferCompleted_Rule"
                    agentVariable="off" >
284
```

```
285                     <documentation>
286                         <description>Clerk receives TransferCompleted
                                and sends UpdateLoanApplication </description
                                >
287                     </documentation>
288                     <WHEN eventType="InMessageEvent" messageType="
                            TransferCompleted" eventVariable="evt"
                            messageVariable="mes"/>
289                     <DO>
290                         <SCHEDULE_EVT>
291                             <OutMessageEventExpr messageType="
                                    UpdateLoanApplication">
292                 <Condition language="Java"><![CDATA[ Random.uniformInt(1,10)
                        <= 5 ]]></Condition>
293                                 <ReceiverIdRef language="Java">2</
                                        ReceiverIdRef>
294                                 <Slot property="CaseId">
295                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
296                                 </Slot>
297                 <Slot property="Result" value="True"></Slot>
298                             </OutMessageEventExpr>
299                             <OutMessageEventExpr messageType="
                                    ResultNotification">
300                                 <ReceiverIdRef language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ReceiverIdRef>
301                                 <Slot property="CaseId">
302                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
303                                 </Slot>
304                             </OutMessageEventExpr>
305                         </SCHEDULE_EVT>
306                     </DO>
307                 </CommunicationRule>
308
309                 <CommunicationRule name="AdditionalDetails_Rule" >
310
311                     <documentation>
312                         <description>Clerk receives AdditionalDetails
                                and sends AdditionalDetails or terminate the
                                loop</description>
313                     </documentation>
314                     <WHEN eventType="InMessageEvent" messageType="
                            AdditionalDetails" eventVariable="evt"
                            messageVariable="mes" />
315                 <IF language="Java"><![CDATA[ Random.uniformInt(1,10) <= 5
                        ]]></IF>
316                     <THEN>
317                         <SCHEDULE_EVT>
318                             <OutMessageEventExpr messageType="
                                    AdditionalDetails">
319                                 <ReceiverIdRef language="Java">1</
                                        ReceiverIdRef>
320                                 <Slot property="CaseId">
321                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
```

```
322                                 </Slot>
323                             </OutMessageEventExpr>
324                         </SCHEDULE-EVT>
325                     </THEN>
326                     <ELSE>
327                         <SCHEDULE-EVT>
328                             <OutMessageEventExpr messageType="
                                    CreateLoanApplication">
329                                 <ReceiverIdRef language="Java">2</
                                        ReceiverIdRef>
330                                 <Slot property="CaseId">
331                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId()  ]]></ValueExpr>
332                                 </Slot>
333                             </OutMessageEventExpr>
334                             <OutMessageEventExpr messageType="ApplyRules
                                    ">
335                                 <ReceiverIdRef language="Java">2</
                                        ReceiverIdRef>
336                                 <Slot property="CaseId">
337                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId()  ]]></ValueExpr>
338                                 </Slot>
339                             </OutMessageEventExpr>
340                         </SCHEDULE-EVT>
341                     </ELSE>
342                 </CommunicationRule>
343
344             </AgentType>
345
346             <AgentType name="LoanSystem">
347
348                 <CommunicationRule name="ApplyRules_Rule"  agentVariable
                        ="sys" >
349
350                     <documentation>
351                         <description>LoanSystem receives ApplyRules and
                                sends RulesResult</description>
352                     </documentation>
353                     <WHEN eventType="InMessageEvent" messageType="
                            ApplyRules" eventVariable="evt" messageVariable="
                            mes" />
354                     <DO>
355                         <SCHEDULE-EVT>
356                             <OutMessageEventExpr messageType="
                                    RulesResult">
357                                 <ReceiverIdRef language="Java">1</
                                        ReceiverIdRef>
358                                 <Slot property="CaseId">
359                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId()  ]]></ValueExpr>
360                                 </Slot>
361                             </OutMessageEventExpr>
362
363                         </SCHEDULE-EVT>
364                     </DO>
```

```
365             </CommunicationRule>
366
367             <CommunicationRule name="RequestTransfer_Rule"
                    agentVariable="sys" >
368
369                 <documentation>
370                     <description>LoanSystem receives RequestTransfer
                            and sends TransferCompleted</description>
371                 </documentation>
372                 <WHEN eventType="InMessageEvent" messageType="
                        RequestTransfer" eventVariable="evt"
                        messageVariable="mes"/>
373                 <DO>
374                     <SCHEDULE-EVT>
375                         <OutMessageEventExpr messageType="
                                TransferCompleted">
376                             <ReceiverIdRef language="Java">1</
                                    ReceiverIdRef>
377                             <Slot property="CaseId">
378                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId()  ]]></ValueExpr>
379                             </Slot>
380                         </OutMessageEventExpr>
381                     </SCHEDULE-EVT>
382             </DO>
383             </CommunicationRule>
384
385         </AgentType>
386
387         <AgentType name="CreditAgency">
388
389             <CommunicationRule name="RequestRecords_Rule"
                    agentVariable="cred" >
390
391                 <documentation>
392                     <description>CreditAgency receives
                            RequestRecords and sends CustomerRecords</
                            description>
393                 </documentation>
394                 <WHEN eventType="InMessageEvent" messageType="
                        RequestRecords" eventVariable="evt"
                        messageVariable="mes"/>
395                 <DO>
396                     <SCHEDULE-EVT>
397                         <OutMessageEventExpr messageType="
                                CustomerRecords">
398                             <ReceiverIdRef language="Java">1</
                                    ReceiverIdRef>
399                             <Slot property="CaseId">
400                                 <ValueExpr language="Java"><![CDATA[
                                        mes.getCaseId() ]]></ValueExpr>
401                             </Slot>
402                         </OutMessageEventExpr>
403                     </SCHEDULE-EVT>
404             </DO>
405             </CommunicationRule>
```

```
406             </AgentType>
407
408             <AgentType name="Manager">
409
410                 <CommunicationRule name="RequestApproval_Rule"
                        agentVariable="man" >
411
412                     <documentation>
413                         <description>Manager receives RequestApproval
                                and sends ApprovalResult</description>
414                     </documentation>
415                     <WHEN eventType="InMessageEvent" messageType="
                            RequestApproval" eventVariable="evt"
                            messageVariable="mes"/>
416                     <DO>
417                         <SCHEDULE-EVT>
418                             <OutMessageEventExpr messageType="
                                    ApprovalResult">
419                                 <ReceiverIdRef language="Java">1</
                                        ReceiverIdRef>
420                                 <Slot property="CaseId">
421                                     <ValueExpr language="Java"><![CDATA[
                                            mes.getCaseId() ]]></ValueExpr>
422                                 </Slot>
423                             </OutMessageEventExpr>
424                         </SCHEDULE-EVT>
425     </DO>
426                 </CommunicationRule>
427             </AgentType>
428         </EntityTypes>
429
430         <EnvironmentRules>
431
432             <EnvironmentRule name="CustomerRequest_Rule">
433
434                 <documentation>
435                     <description> Environment receives CustomerRequest
                            and sends LoanRequest </description>
436                 </documentation>
437                 <WHEN eventType="CustomerRequest" />
438                 <DO>
439                     <UPDATE-ENV>
440                         <Create>
441                             <Agent type="Customer" objectVariable="cust"
                                    >
442                                 <Slot property="Timestamp">
443                                     <ValueExpr language="Java"><![CDATA[
                                            evt.getOccurrenceTime() ]]></
                                            ValueExpr>
444                                 </Slot>
445                             </Agent>
446                         </Create>
447                     </UPDATE-ENV>
448                     <SCHEDULE-EVT>
449                         <CausedEventExpr eventType="
                                StartLoanRequestService">
```

```
450                         <Slot property="StartLoanRequestServiceProp"
                                >
451                             <ValueExpr language="Java">cust.getId()
                                    </ValueExpr>
452                         </Slot>
453                     </CausedEventExpr>
454                 </SCHEDULE-EVT>
455             </DO>
456         </EnvironmentRule>
457
458         <EnvironmentRule name="StartLoanRequestService_Rule">
459
460             <documentation>
461                 <description> Environment receives
                        StartLoanRequestService and sends LoanRequest </
                        description>
462             </documentation>
463             <WHEN eventType="StartLoanRequestService" />
464             <FOR objectVariable="cust" objectType="Customer" />
465             <FOR objectVariable="cle" objectType="Clerk" objectIdRef
                    ="1" />
466             <DO>
467                 <SCHEDULE-EVT>
468                     <InMessageEventExpr messageType="LoanRequest">
469                         <SenderIdRef language="Java"><![CDATA[ cust.
                                getId() ]]></SenderIdRef>
470                         <ReceiverIdRef language="Java"><![CDATA[ cle
                                .getId() ]]></ReceiverIdRef>
471                         <Slot property="CaseId">
472                             <ValueExpr language="Java"><![CDATA[
                                    cust.getId() ]]></ValueExpr>
473                         </Slot>
474                     </InMessageEventExpr>
475                     <InMessageEventExpr messageType="
                            AdditionalDetails">
476                         <SenderIdRef language="Java"><![CDATA[ cust.
                                getId() ]]></SenderIdRef>
477                         <ReceiverIdRef language="Java"><![CDATA[ cle
                                .getId() ]]></ReceiverIdRef>
478                         <Slot property="CaseId">
479                             <ValueExpr language="Java"><![CDATA[
                                    cust.getId() ]]></ValueExpr>
480                         </Slot>
481                     </InMessageEventExpr>
482                 </SCHEDULE-EVT>
483             </DO>
484         </EnvironmentRule>
485
486
487         <EnvironmentRule name="EndResultNotificationService_Rule">
488
489             <documentation>
490                 <description>Destroys Customer when
                        ResultNotification </description>
491             </documentation>
```

```
492                     <WHEN eventType="OutMessageEvent" messageType="
                            ResultNotification" eventVariable="evt" />
493         <FOR objectVariable="cust" objectType="Customer">
494             <ObjectIdRef language="Java"><![CDATA[ evt.getReceiverIdRef()
                    ]]></ObjectIdRef>
495         </FOR>
496                     <DO>
497                         <UPDATE-ENV>
498                             <DestroyObject objectVariable="cust"> </
                                    DestroyObject>
499                         </UPDATE-ENV>
500                     </DO>
501                 </EnvironmentRule>
502         </EnvironmentRules>
503     </SimulationModel>
504
505
506     <InitialState>
507
508         <Agent type="Clerk" name="cle" id="1" />
509
510         <Agent type="LoanSystem" name="sys" id="2" />
511
512         <Agent type="CreditAgency" name="cred" id="3" />
513
514         <Agent type="Manager" name="man" id="4" />
515
516         <ExogenousEvent type="CustomerRequest" occurrenceTime="1" />
517     </InitialState>
518
519 </SimulationScenario>
```