UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOSE RAFAEL XAVIER DOS SANTOS

Quantificação da Complexidade de Processos de TI Interdomínios

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Lisandro Z. Granville
Orientador

Porto Alegre, Fevereiro de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Santos, Jose Rafael Xavier dos

Quantificação da Complexidade de Processos de TI Interdomínios / Jose Rafael Xavier dos Santos. – Porto Alegre: PPGC da UFRGS, 2012.

85 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Lisandro Z. Granville.

1. Procedimentos de configuração. 2. Processos de TI. 3. Complexidade de processos. 4. Processos interdomínios. I. Granville, Lisandro Z.. II. Título.

“It’s so simple to be wise. Just think of something stupid to say and then don’t say it.”
— SAM LEVENSON

AGRADECIMENTOS

Agradecer a todos os que merecem quando se tem tão pouco espaço é sempre complicado. Por isso peço que – já neste início – sintam-se carinhosamente incluídos todos aqueles que conviveram comigo neste percurso, participando dos bons e maus momentos, ou mesmo aqueles que participaram de minha formação prévia, seja acadêmica, profissional ou como indivíduo. Vocês definiram o que sou, e portanto saibam que a culpa é toda de vocês.

A justiça almejada pela generalização traz injustiça àqueles que mais perto de mim estiveram até hoje, e neste aspecto peço perdão se a partir daqui eu for seletivo demais e injusto de alguma forma.

Bom, meu principal agradecimento dirijo a minha esposa Thais, a meus pais, irmãos, avós, familiares e aos meus amigos sempre próximos. Apoio, amor, suporte, conselhos, tudo isso formou os princípios que me compõem como indivíduo, algo que vai muito além desta dissertação. Espero que minha resposta a este carinho tenha em algo servido de retribuição.

No âmbito acadêmico, agradeço especialmente à UFRGS e ao PPGC pela valiosa oportunidade e por entender os motivos de meu afastamento temporário, e ao professor Lisandro Z. Granville – meu orientador na graduação e na atual empreitada – pelo apoio, paciência, compreensão, conselhos, cobranças, boas ideias e pela liberdade de criação. Também cito com muita importância o professor Luciano P. Gaspary, pelos conselhos e sugestões sempre muito valiosos, além dos colegas das disciplinas cursadas e de tantas tardes em laboratório, entre eles Giovane Moura, Carlos Raniery e Ewerton Salvador. Espero ter contribuído como esperado e que possamos repetir a dose muitas outras vezes.

No âmbito profissional, reservo espaço especial ao sempre chefe e amigo Alexandre Goldmeier, um marco em minha formação, além de todos os colegas de NET. Incluo também colegas e gestores do Grupo RBS, pela oportunidade de almejar horizontes ainda mais desafiadores, e todos os colegas com quem convivo diariamente. Estes conseguem melhorar dias agradáveis de trabalho e divertir aqueles nem tão bons. Tanto na NET quanto no Grupo RBS, a compreensão e a generosidade de colegas foi crucial em momentos de grande dedicação a esta dissertação.

Obrigado a todos e sempre contem comigo.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 9

LISTA DE FIGURAS 11

LISTA DE TABELAS 13

RESUMO 15

ABSTRACT 17

1 INTRODUÇÃO 19

2 QUANTIFICAÇÃO DA COMPLEXIDADE 23
2.1 *Benchmarks* dirigidos por processos 23
2.2 Complexidade de procedimentos de configuração 25
2.3 Complexidade de processos de TI 27
2.3.1 Modelo de processos de TI 28
2.3.2 Modelo de complexidade 29
2.3.3 Metodologia 31
2.4 Complexidade em cenários mais distribuídos 32

3 COMPLEXIDADE E PROCESSOS INTERDOMíNIOS 33
3.1 Modelo de processo de TI 33
3.2 Processos interdomínios 34
3.3 Quantificação da complexidade de processos de TI interdomínios 35
3.3.1 Processos e subprocessos 35
3.3.2 Fator de ajuste de complexidade (IDX) 36
3.3.3 Algoritmo de troca de subprocessos 37
3.3.4 Cálculo ajustado das complexidades 38

4 PROTÓTIPO 41
4.1 Arquitetura 42
4.2 Representação do processo em XML 44
4.3 Implementação 45
4.3.1 Módulo *XML parser* 45
4.3.2 Módulo *metric application* 47
4.3.3 Módulo *diagram plotter* 47
4.3.4 Módulo *histogram plotter* 48
4.3.5 Módulo *XML builder* 49

4.4 Ferramenta de visualização 50

5 AVALIAÇÃO DA METODOLOGIA 53
5.1 Processo avaliado 53
5.2 Aplicação da metodologia 54
5.2.1 Passo (i): pré-requisitos para aplicação da metodologia 54
5.2.2 Passo (ii): definição de subprocessos 55
5.2.3 Passo (iii): complexidade do processo 58
5.3 Avaliação dos resultados 60

6 CONTRIBUIÇÕES 63
6.1 Realizações e conclusões 63
6.2 Trabalhos Futuros 63

REFERÊNCIAS 65

ANEXO I - CÓDIGO FONTE DO PROTÓTIPO 69

ANEXO II - XML SCHEMA 83

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecnologia da informação
XML	Extensible markup language
COBIT	Control objectives for information and related technology
ITIL	The information technology infrastructure library
PSTN	Public switched telephone network
LIFO	Last in, first out
FIFO	First in, first out
LPR	Local private process
LPU	Local public process
FPR	Full private process
FP	Full process
FAP	Full adjusted process
EMF	Eclipse modelling framework
SVG	Scalar vector framework
HTML	Hypertext markup language
HFC	Hybrid fiber coaxial
LM	Last mile
PP	Phone provider
SD	Service desk

LISTA DE FIGURAS

Figura 2.1: Tipos de *benchmark* (BROWN; HELLERSTEIN, 2004) 23
Figura 2.2: Restrições de ciclo de vida dos sistemas (BROWN; HELLERSTEIN, 2004) 25
Figura 2.3: Diagrama de contêineres (KELLER; BROWN; HELLERSTEIN, 2007) 25
Figura 2.4: Diagrama de atividades (KELLER; BROWN; HELLERSTEIN, 2007) 26
Figura 2.5: Exemplo de processo de TI 28
Figura 2.6: Ciclos de medida para processos de TI (DIAO; KELLER, 2006) . . . 31

Figura 3.1: Configuração de reserva de recursos 35
Figura 3.2: Troca de subprocessos 37

Figura 4.1: Arquitetura do protótipo do *Complexity Analyzer* 42
Figura 4.2: Diagrama de classes 43
Figura 4.3: Código XML de um processo de TI 46
Figura 4.4: Uso da classe *SAXBuilder* para ler arquivos XML de entrada 46
Figura 4.5: Exemplo de instanciação de entidades do processo utilizando as classes projetadas 46
Figura 4.6: Chamada de métodos de cálculo de complexidade 47
Figura 4.7: Configuração do Graphviz para gerar um diagrama de processo com duas tarefas e pontuação da complexidade 47
Figura 4.8: Exemplo de processo desenhado pelo Graphviz 48
Figura 4.9: Exemplo de definição de gráfico do Gnuplot 48
Figura 4.10: Exemplo de arquivo de dados do Gnuplot 49
Figura 4.11: Exemplo de histograma desenhado pelo Gnuplot 49
Figura 4.12: Trecho de código Java do módulo *XML builder* 50
Figura 4.13: Tela de exibição de resultados, com diagrama dos processos 51
Figura 4.14: Tela de exibição de resultados, com histograma de complexidade . . . 51

Figura 5.1: Suporte em múltiplos níveis (ITIL, 2009) 56
Figura 5.2: Subprocesso LPR do domínio SD 57
Figura 5.3: Subprocesso LPU do domínio SD 57
Figura 5.4: Subprocesso LPR do domínio LM 57
Figura 5.5: Subprocesso LPU do domínio LM 58
Figura 5.6: Subprocesso LPR do domínio PP 58
Figura 5.7: Subprocesso LPU do domínio PP 58
Figura 5.8: Processo FP do domínio SD 59
Figura 5.9: Processo FPR 61

LISTA DE TABELAS

Tabela 4.1: Valores possíveis para a métrica *sourceScore* 45

Tabela 5.1: Valores de complexidade e IDX calculados para os subprocessos LPU e LPR dos domínios SD, LM e PP 59
Tabela 5.2: Valores de complexidade para os domínios SD, LM e PP 60
Tabela 5.3: Valores de complexidade – para os domínios SD, LM e PP – comparados com a complexidade do processo ideal (FPR) 60

RESUMO

Serviços distribuídos exigem esforços cooperativos entre provedores de serviços distintos, com comunicação e troca de informações. Isto, entretanto, aumenta a complexidade associada aos processos de TI, e a entrega de serviços competitivos e com custos predizíveis depende do conhecimento e do controle desta complexidade. No presente trabalho, é proposta uma metodologia capaz de endereçar este cenário, no qual provedores de serviços distintos trabalham em conjunto, trocam informações, entretanto, por vezes omitem informações confidenciais entre si. Para isto, aplica um conjunto de métricas, quantifica a complexidade dos processos, define como os provedores trocarão entre si informações de seus procedimentos internos e como estas informações serão ajustadas para garantir a todos o mesmo resultado apesar de informações confidenciais serem, porventura, omitidas. A avaliação – cíclica – começa com o processo sendo projetado por todos os provedores, chamados de domínios de autoridade. Após finalizados os processos locais de cada um, estes são enviados a todos os domínios integrantes do processo, seguida da aplicação de ajustes de complexidade, o que garante que cada um terá uma visão completa e de complexidade coerente do processo. A aplicação das métricas é efetuada pela ferramenta *complexity analyzer*, que analisa gramaticalmente processos representados em XML até alcançar valores aceitáveis de complexidade. Um protótipo da ferramenta foi implementado e utilizado para avaliar um processo de TI mapeado de um caso real, validando a metodologia proposta.

Palavras-chave: Procedimentos de configuração, processos de TI, complexidade de processos, processos interdomínios.

Quantifying the Completity of Inter-Domain IT Processes

ABSTRACT

Distributed services requires cooperative efforts among partner service providers, like communication and information exchange. This, however, increases the complexity associated to IT processes, and the delivery of services with predictable and competitive costs also depends on the knowledge and control of process complexity along service provider federations. Our proposed methodology is capable to address this scenario, that different service providers work together, exchanging information, however, sometimes hidding confidential information from one another. To reach these goals its applied a set of complexity metrics that quantifies the process complexity. It is also defined how all service providers will exchange and adjust process informations aiming to give to each one the same results even when confidential informations are ommited. The cyclic evaluation starts with the process design by each autonomous provider, called authority domain. When all parts of the processes are done, they are sent to all domains, followed by complexity adjustments, what guarantees that each domain will have a complete and trustable version of the entire process. The complexity analyzer is the tool used to collect the metrics. This tool analyzes grammaticaly XML represented processes until reached an aceptable complexity value. To validate the proposed methodology, a prototype of the complexity analyzer was developed and applied against a real case IT process.

Keywords: configuration procedures, inter-domain IT processes, process complexity, IT processes.

1 INTRODUÇÃO

A evolução da comunicação possibilita negócios e processos distribuídos entre provedores de serviço distintos, aumentando a dependência de sistemas de informação cada dia mais complexos. Esta complexidade é inerente à execução e à manutenção dos processos, e sua quantificação é o tema do presente trabalho. Portanto, busca-se conhecer tal complexidade como um passo importante no objetivo de fornecer serviços qualificado com redução de erros e menos custo operacional (DIAO; KELLER, 2006).

Quando os processos envolvem estruturas e tecnologias de crescente complexidade, estes tornam-se mais suscetíveis a falhas, exigindo maior tempo de execução e qualificação pessoal para alcançar resultados satisfatórios. Se a qualificação dos executores dos processos não for suficiente para reduzir tais falhas, a um baixo custo operacional, maiores devem ser os esforços para o controle da complexidade dos processos e na detecção de pontos alvo de melhorias.

Diversas são as iniciativas que visam a representar e a controlar a complexidade de processos. A padronização de serviços e processos de TI, por meio de recomendações práticas de administração, como o COBIT (*Control Objectives for Information and related Technology*) (ISACA, 2005) e o ITIL (*The Information Technology Infrastructure Library*) (ITIL, 2009), constitui uma maneira eficiente e bem difundida de controle da eficiência dos processos de TI.

O ITIL promove a gestão com foco no cliente e na qualidade por meio de definições para suporte e entrega de serviços (*Service Support* e *Service Delivery*). Entre suas especificações, a definição de *Service Transition* garante eficiência e subsídios necessários ao negócio de forma otimizada e agregando confidencialidade. Como a gestão de mudanças é um fator chave para o *Service Transition*, diversos trabalhos visam a otimizar processos de negócio com esta ferramenta, utilizando ou estendendo suas definições (HAGEN; KEMPER, 2010) (COSTA CORDEIRO et al., 2008) (SANTOS et al., 2011).

Apesar de abranger necessidades do negócio e ter boa absorção no mercado, padrões para processos de TI buscam controlar a complexidade sem quantificá-la. Mas o uso de uma metodologia neste sentido mostra-se essencial, pois para ajudar na identificação e na otimização de pontos críticos é importante que os projetistas saibam mensurar a complexidade de cada etapa (DIAO; KELLER, 2006).

Em busca de quantificar indicadores de complexidade, Brown et al. (BROWN; HELLERSTEIN, 2004) tratam procedimentos de configuração aplicando *benchmarks* dirigidos por processos. Ao contrário de um *benchmark* comum – em que uma carga é aplicada até atingir um nível desejado de consumo de recursos –, na abordagem orientada a processos, os procedimentos são executados por agentes humanos que registram dados de cada passo. Os registros são avaliados a cada ciclo até serem atingidos patamares aceitáveis de acordo com critérios definidos pelos responsáveis pela análise, o que depende

do cenário, das métricas utilizadas e do perfil do analista. A metodologia utilizada nesta avaliação consiste em aplicar um conjunto pré-definido de métricas e medir a capacidade dos analistas e a qualidade do processo em cada passo. Esta abordagem é empregada pelos autores em um modelo de análise da complexidade de procedimentos de configuração (KELLER; BROWN; HELLERSTEIN, 2007), validado posteriormente por meio de sua aplicação nos procedimentos de um sistema de gerenciamento de mudanças, o CHAMPS (KELLER et al., 2004).

A mesma abordagem utilizada para a complexidade de procedimentos de configuração pode ser aplicada na análise da complexidade de processos de TI (DIAO; KELLER, 2006). Neste caso, com a inclusão de métricas e ciclos de avaliação nos moldes de trabalhos anteriores, mas específicos para o distinto cenário. Pode-se considerar que em tal modelo, processos são formados por tarefas compostas em seu interior por procedimentos de configuração, e como a metodologia anterior foca em procedimentos (KELLER; BROWN; HELLERSTEIN, 2007), que o foco em processos aumenta o nível de abstração e aproxima a metodologia da gestão da TI. Os autores citam o cálculo da complexidade de procedimentos completos como um possível método de cálculo de complexidade interna de uma única tarefa.

A avaliação da metodologia é baseada em sua aplicação a um processo de gestão de mudanças definido no ITIL, composto por diversos níveis de análise em processo de suporte, e amplamente aplicado nas empresas. Assim, torna-se capaz de quantificar a complexidade de processos de TI. Entretanto, o processo é avaliado considerando uma autoridade única para a execução de tarefas, para a tomada de decisões e para o desenho do processo, enquanto processos de TI podem cruzar provedores de serviço distintos, com objetivos e metas distintos, entre outras características capazes de agregar complexidade ao processo ou inviabilizar ciclos de concepção do próprio processo.

A execução de processos através de provedores de serviço distintos fica mais evidente se descritos casos conhecidos, como a interconexão entre fornecedores de *backbone* Internet, assim como entre operadoras de telefonia, além de processos de portabilidade de números telefônicos etc. Nestes casos, tecnologias permitem a qualidade do serviço, e requisitos legais ou de mercado exigem um processo eficiente. Porém, cada um tem suas peculiaridades, como objetivos estratégicos e informações confidenciais, o que pode comprometer a cooperação entre os provedores, bem como a própria metodologia.

Como cada provedor pode manter um processo funcional e eficiente com outras instituições sem comprometer seus segredos? O objetivo desta dissertação é estender os trabalhos existentes e aqui citados com foco na análise de complexidade de procedimentos de configuração e processos de TI, e assim definir novas funcionalidades capazes de endereçar a complexidade de processos que transpassem diferentes provedores de serviços e mantenham-na sob controle apesar das diferenças entre seus integrantes.

A base da metodologia proposta é formada por dois tipos de processos, um público e outro privado, cujos conteúdos são diferenciados pela existência de dados confidenciais ou inelegíveis à divulgação. Além disso, é definido um fluxo de troca de informações capaz de conceber uma visão local do processo completo para cada provedor, mantendo íntegro o valor de complexidade calculado para a visão de cada participante no processo, sempre com níveis de erros no cálculo acordados entre as partes. Como as tarefas e as decisões cruzam os diferentes provedores de serviços distintos, denominados aqui domínios de autoridade, as visões locais do processo completo dão a cada um visibilidade do funcionamento total, dentro do que cada um tem permissão para ver. Como todos obtêm valores similares e aproximados para a complexidade, é possível que cada um conheça,

fiscalize e melhore os pontos de maior dificuldade e prejuízo.

Na metodologia proposta, dando seguimento aos trabalhos anteriores, dois cenários possíveis são mapeados: um de desenho do projeto, utilizando ferramentas de mapeamento de processos e incluindo as comunicações e as negociações entre as partes envolvidas, e outro de execução, no qual o processo é colocado em uso e avaliado na prática. No cenário de projeto, cada domínio desenha sua parte do processo, com todos os detalhes necessários para seu controle local, chamado de processo privado. A partir deste, são definidas as versões públicas, omitindo informações não divulgáveis aos parceiros, e estas são trocadas ciclicamente até que se obtenha uma versão de complexidade razoável para todos. Por não ser o foco deste trabalho, o cenário de projeto considera bem definidas as funções de cada domínio dentro das definições iniciais do processo: é reconhecido que complexas tarefas de negociação estão presentes neste cenário e são estudadas com mais profundidade em outros trabalhos (DUNNE; WOOLDRIDGE; LAURENCE, 2005) (DUNG; THANG; TONI, 2008) (GRIFFEL et al., 1997). Prosseguindo, o segundo cenário previsto é o de execução, no qual o processo é posto em prática e também avaliado ciclicamente, entre execuções, iniciando procedimentos de ajustes ou redesenho do projeto sempre que necessário.

Para viabilizar os ciclos de avaliação, é utilizado como centro da metodologia um conjunto de métricas relacionado à complexidade de processos de TI e definido em trabalhos anteriores (DIAO; KELLER, 2006). Estas são implementadas em uma ferramenta capaz de receber processos como entrada, calcular a sua complexidade através da aplicação de tais métricas, e originar, como saída, o mesmo processo com pontuações de complexidade por tarefa. Além disso, são gerados gráficos e histogramas de auxílio para os agentes envolvidos nos fluxos da metodologia.

Como as métricas são projetadas para avaliar características de desenho e execução dos processos, existe a possibilidade de domínios esconderem informações e enviarem processos públicos diferentes para cada domínio parceiro. Isso pode trazer inconsistências no número final obtido por cada domínio, quebrando a premissa de manter o mesmo valor de complexidade para cada participante do processo. Para equalizar estes valores dentro de uma margem de erro satisfatória, é proposto um índice de ajuste baseado nas diferenças entre as partições pública e privada do processo local. Este ajuste é enviado aos parceiros junto com o processo, e estes, no momento de compor o processo completo, devem usar os índices recebidos de cada domínio parceiro como entrada da ferramenta de cálculo, equalizando o resultado entre todos.

A ferramenta de análise de complexidade, base para a aplicação da metodologia na prática, é chamada de *Complexity Analyzer*. Para validar a metodologia proposta foi completamente modelado e implementado um protótipo, escrito em Java, capaz de interpretar as entradas em formato XML, entender as *tags* de modelagem de processos, as métricas e os índices de cálculo e fornecer os objetos de saída necessários. Prosseguindo, foi mapeado e avaliado um processo real de negócio, o provisionamento de serviços de telefonia em uma rede de cabos coaxiais e fibras óticas. O processo envolve comunicação entre três domínios de autoridade distintos: o de fornecimento de conexão com as redes públicas de telefonia (PSTN), o de fornecimento de última milha – chegando na casa do cliente – e o de fornecimento de suporte, de atendimento e de comercialização dos serviços ofertados. Como resultado, são apresentados o processo, os índices calculados e os valores das complexidades de cada versão do processo.

Deve-se ressaltar que, para fomentar a gestão, é necessário representar tal complexidade em unidades que o negócio compreenda. Para tanto, é necessário um modelo de

custos capaz de traduzir a complexidade em valores aplicáveis na prática. Patterson et al. propôs um modelo de custo para medir o impacto financeiro de períodos de indisponibilidade de serviço (A SIMPLE WAY TO ESTIMATE THE COST OF DOWNTIME, 2002). Com objetivo semelhante, Susanto et al. (COUCH; WU; SUSANTO, 2005) propõem um modelo de custos mais abrangente, envolvendo todo o ciclo de vida da administração de sistemas. Ambos partem da definição de métricas, cuja metodologia pode ser aplicada aos trabalhos aqui citados para definir modelos de custos, o que compõe tarefa futura essencial, porém fora do escopo desta dissertação.

Assim, este documento foca estudo na complexidade de processos de TI que transpassam domínios de autoridade distintos. Para tanto, está organizado como segue: o Capítulo 2 contextualiza o problema investigado e a metodologia existente, estendida neste trabalho. O Capítulo 3 apresenta a solução proposta descrevendo as extensões necessárias e como aplicá-las. Já os Capítulos 4 e 5 abordam o protótipo implementado e sua validação em um processo real. Finalizando, o Capítulo 6 desenvolve as considerações finais e os trabalhos futuros mapeados a partir deste.

2 QUANTIFICAÇÃO DA COMPLEXIDADE

Processos e procedimentos ineficientes comprometem a entrega rápida de serviços qualificados. Por isso mesmo, diversos são os esforços em busca do controle de sua complexidade e, especialmente, iniciativas que busquem quantificá-la. Esta seção descreve estes esforços, citados como base e referência para o desenvolvimento dos objetivos desta dissertação. Portanto, os conceitos e modelos aqui descritos foram previamente definidos pelos autores dos referidos trabalhos.

Inicialmente é descrito o início da linha de análise de complexidade, focada na quantificação de complexidade de procedimentos de configuração. Prosseguindo, é descrito como os autores a estendem para quantificar tal complexidade de processos de TI, e ao final é descrita uma breve introdução do problema a ser abordado no capítulo seguinte.

2.1 *Benchmarks* dirigidos por processos

Brown et al. (BROWN; HELLERSTEIN, 2004) propõem um modelo quantitativo para a complexidade de procedimentos de configuração, sedimentando a base para a construção de um *benchmark* com esta finalidade. Para isso, mapeiam e parametrizam ações de configuração executadas por operadores, depois aplicadas a um *benchmark* dirigido por processos. Este, diferentemente dos *benchmarks* tradicionais, nos quais são aplicadas cargas de trabalho até o esgotamento da capacidade do sistema, aplica um processo a fim de mensurar suas propriedades de execução. Estas duas abordagens podem ser observadas na Figura 2.1.

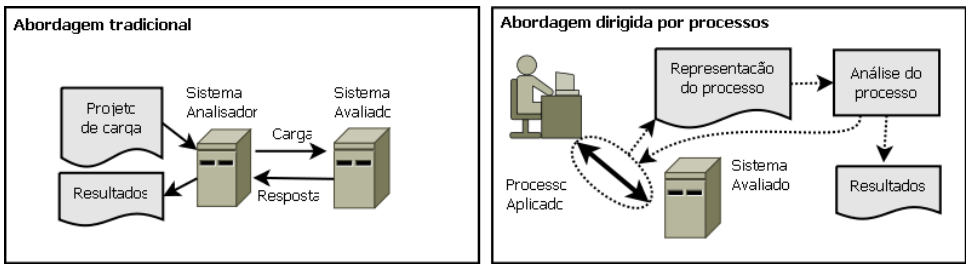

Figura 2.1: Tipos de *benchmark* (BROWN; HELLERSTEIN, 2004)

A quantificação da complexidade de processos de configuração é o núcleo da metodologia de *benchmark* proposta pelos autores. Esta envolve interações humanas, medindo tempo de execução das ações e a probabilidade do processo ser completado sem erro, e parametrizando os cálculos pelo nível de conhecimento do operador envolvido. São utilizadas aproximações baseadas na avaliação do melhor caso – no qual um operador do mais

alto nível configura o sistema – e cujo desempenho é utilizado para balizar o desempenho de outros indivíduos de menor nível de conhecimento. A metodologia consiste em quatro fases: coleta, decomposição e análise, pontuação, e validação.

Na **coleta**, um operador de conhecimento avançado executa o procedimento de configuração, e o sistema de *benchmark* armazena cada passos em sua estrutura interna de dados. Esta etapa é repetida até que o resultado seja aceito como o melhor caso possível de acordo com o *know-how* de operadores experientes. Já na **decomposição e análise**, os dados armazenados são mapeados, analisados e convertidos manualmente pelos analistas em ações de configuração. Com as ações mapeadas, o avaliador inicia a pontuação, estimando valores numéricos de complexidade por ação também com base em seu *know-how*. Estes valores são calibrados por meio da aplicação e da comparação com dados reais de outros operadores, processo que termina quando o intervalo de confiança entre o valor medido e o estimado atinge um nível aceitável, sendo o valor aceitável balizado pelo processo de calibragem. A fase de **validação** analisa restrições mínimas de qualidade relativas a procedimentos viciados, ou seja, que sejam modelados a partir do próprio *benchmark*.

Utilizando as respostas armazenadas, cada tipo de ação pode ser parametrizada. Por exemplo: em um procedimento, uma ação *selecionar funcionalidades* pode ser parametrizada pelo número de funcionalidades desejadas, pelo número total de funcionalidades existentes, por quantas delas são obrigatórias ou foram modificadas dos valores iniciais etc. Assim, o conjunto de parâmetros – chamados também de métricas – deriva das variações de produtividade dos analistas registrada no sistema, e constitui o modelo de complexidade. Diferentes versões do modelo de complexidade devem ser definidas para diferentes grupos de nível de conhecimento, como iniciante, certificado e experiente. Além disso, dentro de um mesmo grupo os níveis de conhecimento podem se distanciar, tornando essencial o controle desta variância, que está ligada aos intervalos de confiança avaliados durante o processo de calibragem do modelo.

Com as ações e os modelos definidos, o sistema de *benchmark* deve se precaver de projetistas maliciosos. Como são modelados parâmetros baseados no comportamento humano, avaliadores maliciosos tendem a redesenhar seus processos de forma a conseguir resultados otimizados com base no *benchmark*, ou seja, recebendo alta pontuação sem de fato focar a eficiência na execução real das ações. Para evitar estes problemas, são definidas restrições de ciclo de vida dos sistemas. São previstos três ciclos distintos: **inicial**, que representa a configuração inicial do sistema, **reconfiguração**, contemplando adaptações em tempo de execução, e **recuperação**, para quando o sistema está inoperante devido a falhas.

Visto que diferentes ciclos de vida proporcionam diferentes níveis de complexidade – já que configurar um sistema para implantação geralmente é mais simples do que corrigir erros desconhecidos ou tratá-los em tempo de execução – cada ciclo possui um conjunto de restrições próprias. As áreas de desempenho são dispostas em um sistema de múltiplas dimensões: *throughput*, latência, integridade etc. A Figura 2.2 exibe um exemplo em duas dimensões: se o funcionamento se mantém dentro de limites pré-configurados, os objetivos estão sendo alcançados. Se não, é necessário reconfigurar o sistema para corrigir sua trajetória.

Para validar a metodologia, esta é aplicada à configuração do sistema SPECjAppServer. Já o SPECjAppServer2002 *Java Enterprise application benchmark* é utilizado para garantir que o desempenho dos operadores se mantenha próximo de um pré-determinado nível de desempenho, ou seja, impõe restrições de funcionamento ao ciclo de vida do

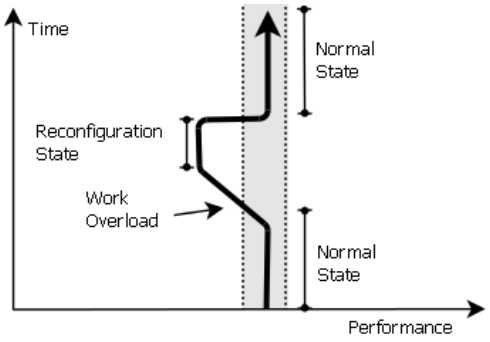

Figura 2.2: Restrições de ciclo de vida dos sistemas (BROWN; HELLERSTEIN, 2004)

sistema. Por fim, o analista obtém uma pontuação para a complexidade da configuração executada.

As técnicas apresentadas constituem o início de um trabalho de análise de complexidade de procedimentos de configuração. Os autores apontam o trabalho por eles desenvolvido como base para outros estudos de quantificação destas complexidades, como visto a seguir.

2.2 Complexidade de procedimentos de configuração

Estendendo o trabalho descrito na seção anterior, Keller et al. (KELLER; BROWN; HELLERSTEIN, 2007) apresentam problemas de procedimentos de configuração e esforços que visam a minimizá-los. Um destes esforços é o CHAMPS (KELLER et al., 2004), cuja abordagem baseia-se em automação e caracteriza um ambiente de gerenciamento de mudança, incluindo provisionamento, instalação, desenvolvimento e configuração de serviços. Os autores ainda evidenciam a necessidade de quantificar os resultados das simplificações fornecidas pelo sistema e, para isso, propõem métricas que modelam a complexidade dos procedimentos de configuração, além de modelos para descrição destes.

Para caracterizar um procedimento, são definidos dois modelos, o de sistema e o de atividade. O modelo de sistema é composto por contêineres, representando recursos utilizados e disponibilizando funcionalidades. O modelo de sistema do experimento realizado pelos autores pode ser visto na figura 2.3.

Contêineres podem ser dinamicamente criados e destruídos e devem englobar tudo o que for característico para o funcionamento do sistema. Um exemplo de funcionalidade do sistema operacional aborda o gerenciamento de processos: este deve fornecer mecanismos para criar e destruir processos, representando-os como novos contêineres.

Já o modelo de atividade consiste em três conceitos básicos: objetivos, procedimentos e ações. Objetivos são descrições claras do estado que se pretende alcançar, como configurar o sistema SPECjAppServer e todo ambiente de suporte. Estes são alcançados através de procedimentos, que por sua vez são formados por ações, cada uma executada em um contêiner do modelo de sistema. A Figura 2.4 demonstra uma pequena parte do modelo de atividades referente à Figura 2.3. Nela são modeladas ações de criação de um usuário, instalação de banco de dados DB2, instalação da máquina virtual Java, instalação do SPECjAppServer e criação de suas tabelas relacionais. As linhas pontilhadas evidenciam a reutilização de variáveis.

Com os modelos de sistema e de atividades já definidos, o modelo de complexidade é

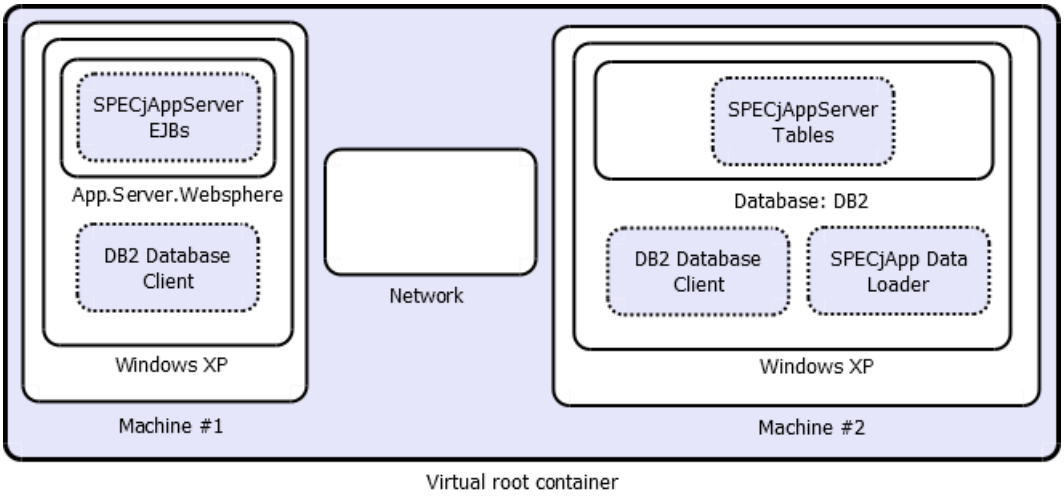

Figura 2.3: Diagrama de contêineres (KELLER; BROWN; HELLERSTEIN, 2007)

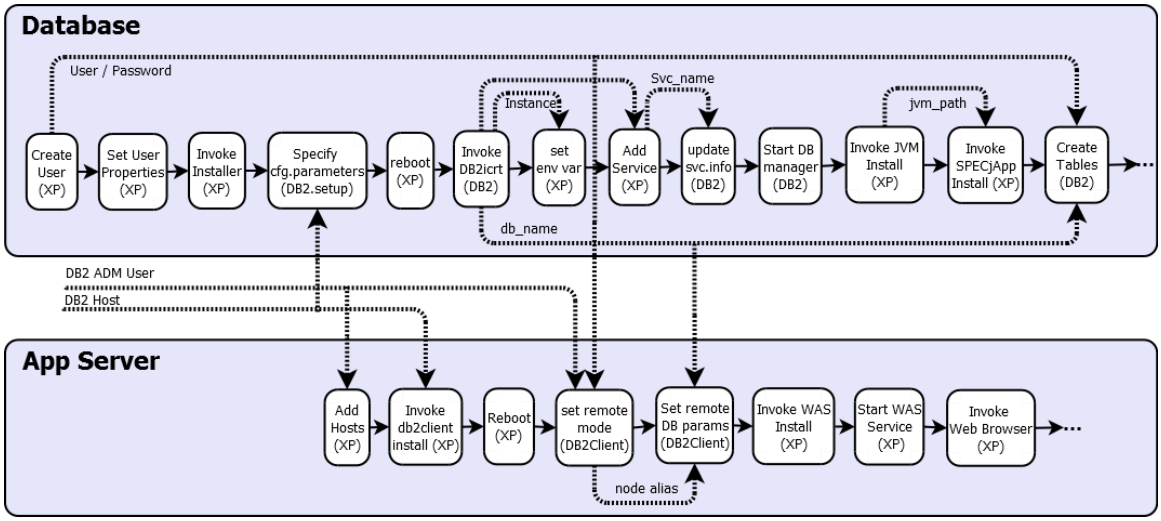

Figura 2.4: Diagrama de atividades (KELLER; BROWN; HELLERSTEIN, 2007)

dividido em três componentes: **complexidade de execução**, **complexidade de memória** e **complexidade de parâmetros**, cada um com seu conjunto de métricas.

A complexidade de execução envolve duas métricas: *numActions*, o total de ações do procedimento, e *contextSwitchSum*, o total de trocas entre contextos diferentes do modelo de sistema, ou seja, migrações entre um contêiner e outro. Já a complexidade de parâmetros define cinco métricas: *paramCount* representa o total de parâmetros, *paramUseCount* enumera as vezes em que parâmetros são fornecidos, *paramCrossContext* totaliza o número de vezes em que parâmetros são fornecidos a mais de um contexto, *paramAdaptCount* armazena o número de reutilizações de parâmetros em formas sintáticas diferentes e *paramSourceScore* agrupa parâmetros pela dificuldade de extração de seus dados, pontuando-os entre zero, para automático, e seis, para dificuldade máxima. Os autores defendem que os parâmetros fornecidos a mais de um contexto predominam nos valores finais de complexidade obtidos.

Ademais, a complexidade de memória modela a memória utilizada pelo operador. A preocupação principal é a necessidade de retenção de informação de um contexto para o uso em outro, que pode ser representada pela utilização de uma pilha LIFO (*last in, first*

out). Sobre ela, três métricas são previstas: *memSize* considera a profundidade máxima utilizada na pilha, *memDepth* armazena a média de profundidade de todas as movimentações efetuadas, e *memLatency* soma o número de passos executados dentro da pilha até chegar ao parâmetro desejado. São consideradas mais complexas ações nas quais o operador necessita movimentar muitos valores antes de chegar ao parâmetro desejado.

O modelo de complexidade é aplicado ao modelo de atividades, reconhecendo a complexidade de cada etapa. Portanto, o conjunto de valores calculados para cada ação do procedimento indica onde esforços de otimização são mais importantes.

Os autores utilizam o sistema CHAMPS para validar sua metodologia. Este sistema possibilita otimizar o procedimento utilizado como exemplo em seu estudo, e a metodologia proposta é atestada ao quantificar de forma numérica a eficiência do CHAMPS.

2.3 Complexidade de processos de TI

Estudados os *benchmarks* dirigidos por processos e a complexidade de procedimentos de configuração, uma abordagem similar é utilizada para mensurar a complexidade de processos de TI (DIAO; KELLER, 2006).

Considerando a necessidade de padronização e otimização de processos de TI, o ITIL (ITIL, 2009) é um recurso popular. Como a complexidade de processos de TI pode representar um empecilho para a entrega de serviços eficientes, de qualidade, livre de erros e de baixo custo, o estudo e o mapeamento dos processos internos, de acordo com recomendações práticas propostas pelo ITIL e já de conhecimento amplo no campo de aplicação, traz benefícios. No entanto, quão significativos eles serão? Seus resultados não podem ser agrupados numericamente. Além do mais, para identificar e otimizar pontos críticos na fase de projeto dos processos é importante saber mensurar a complexidade de cada etapa (DIAO; KELLER, 2006).

Keller et al. (DIAO; KELLER, 2006) propõem uma metodologia para representar numericamente a complexidade de processos de TI. Nela, quando um processo é modelado, a metodologia é aplicada a ele para identificar pontos ineficientes e de maior complexidade. Depois de conhecer os pontos críticos, técnicas de otimização permitem obter um processo mais eficiente. Assim, com um novo processo modelado, aplica-se novamente a metodologia, reportando os resultados que, comparados com a análise anterior, resultam na quantificação do desempenho das melhorias realizadas.

Depois de definir como os processos e um modelo de complexidade capaz de quantificar suas complexidades, cujos detalhes serão vistos nas próximas seções, os autores descrevem uma ferramenta desenvolvida para suportá-los, chamada *complexity analyzer*. Seu objetivo é calcular a complexidade no período de desenvolvimento do processo de TI, utilizando para isso o *IBM WebSphere Business Modeller* como ferramenta de modelagem de processos de TI, ou seja, uma interface interativa entre o *complexity analyzer* e o projetista. Os resultados do *complexity analyzer* são estudados – em formato gráfico – por um arquiteto de processos, e o resultado é delegado a um operador, que executa o processo e insere informações desta execução na interface gráfica. As informações coletadas realimentam o *complexity analyzer*, e novamente os resultados são enviados ao arquiteto, que pode então redesenhar o processo, reconhecendo pontos críticos elegíveis para melhorias e mensurando seus ganhos.

Para embasar a metodologia proposta, os autores definem modelos de processos de TI e da respectiva complexidade, vistos a seguir.

2.3.1 Modelo de processos de TI

Os processos de TI estudados são constituídos por quatro tipos de entidades: papéis, tarefas, decisões e itens de negócio (DIAO; KELLER, 2006).

Papéis (*roles*) representam entidades que executam as tarefas do processo. Podem ser pessoas, setores, sistemas etc. Enfim, entidades da instituição que possuam delegações de função diferentes. Nos diagramas, papéis são representados por linhas horizontais ou raiadas, como na notação BPMN (WHITE, 2004). Estas raias são utilizadas para separar as tarefas realizadas por cada um. Exemplos desta entidade e das outras descritas a seguir podem ser vistos na Figura 2.5.

Tarefas (*tasks*) são formadas por um ou mais procedimentos, e nelas são executadas as ações necessárias para alcançar os objetivos dos processos. Para sua representação, são utilizados retângulos, colocados dentro do fluxo de informações, contendo em seu interior uma breve descrição de suas funcionalidades.

Decisões (*decisions*) são tarefas que envolvem desvios condicionais, representadas por losangos. Seu interior apresenta uma condição, e as possibilidades de desvio são colocadas como braços, descrevendo em cada um a condição que o levou por aquele caminho.

Itens de negócio (*business itens*) modelam os dados produzidos, transportados ou consumidos pelas tarefas, e podem representar um amplo domínio de valores, como documentação para configuração, sinalização de status etc. Em suma, qualquer dado que seja relevante e necessário à execução de uma tarefa.

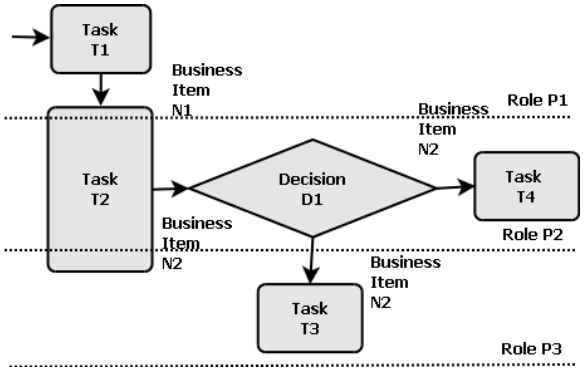

Figura 2.5: Exemplo de processo de TI

A Figura 2.5 mostra um exemplo de processo de TI modelado com o uso destas entidades. Nela percebe-se a existência de três papéis diferentes: *P1*, *P2* e *P3*. Inicialmente, o papel *P1* executa a tarefa *T1*, enviando seus resultados através do item de negócio *N1*. De posse destes dados, os três papéis modelados executam a tarefa *T2*, consumindo o item de negócio *N1* e produzindo o item de negócio *N2*. Com base no conteúdo de *N2*, o papel *P2* executa a tarefa *D1*, uma decisão, e desvia *N2* para as tarefas *T3* ou *T4*, executadas pelos papéis *P1* e *P3*, respectivamente, e encerrando o processo.

Depois de definir as partes que compõem um modelo de processo de TI, é necessário definir como estes serão considerados no cálculo da complexidade.

2.3.2 Modelo de complexidade

Para representar as propriedades dos processos que motivam a metodologia, são definidos três tipos de complexidade: a de execução, a de itens de negócio e a de coordenação . Cada tipo é composto por métricas focadas na representação de características que podem influenciar na complexidade das tarefas, cada métrica com um domínio de valores possíveis, representados no formato *possibilidade(valor)*. Os valores possíveis parametrizam as métricas e são aplicados às fórmulas de cada tipo de complexidade, resultando na complexidade de todo o processo, conforme visto a seguir (DIAO; KELLER, 2006).

Complexidade de execução

A complexidade de execução representa o custo imposto pelos procedimentos internos das tarefas nos cálculos totalizados. Considerando tarefas e decisões, este tipo de complexidade é dividida em dois subtipos, a complexidade básica de execução e a complexidade de decisão.

A complexidade básica de execução classifica a tarefa de acordo com as dificuldades de execução, utilizando a métrica *execType*. Seus valores podem ser *automatic(0)*, quando a tarefa é executada de forma totalmente automatizada, *toolAssisted(1)*, quando ferramentas auxiliam na automação de partes da tarefa, e *manual(2)*, quando esta é executada sem nenhuma automação. Assim, para uma tarefa que envolva n papéis, sendo i o papel avaliado, a complexidade básica de execução de uma tarefa é dada pela Equação 2.1.

$$C_{exec-base} = \sum_{i=1}^{n} execType(i) \tag{2.1}$$

Já a complexidade de decisão contabiliza o custo de decisões condicionais, e para isto utiliza quatro métricas. A métrica *nBranches* representa o número de caminhos possíveis a seguir. Já *gFactor* avalia a influência externa necessária para a tomada da decisão, podendo assumir os valores *recomendation(1)*, quando uma das possibilidades é recomendada explicitamente, *information(2)*, quando informações recebidas explicitam a escolha a ser tomada, e *internal(3)*, quando a decisão é totalmente baseada no conteúdo da ação, sem influências externas. Por sua vez, a métrica *cFactor* trata o impacto de um possível erro de escolha: *negligible(1)*, quando as consequências são toleráveis, *moderate(2)*, quando são moderadas, e *severe(3)*, quando inviabilizam a eficiência do processo. Já a métrica *vFactor* classifica o tempo de propagação da decisão no processo, e assume os valores *imediate(1)*, quando papéis recebem o resultado imediatamente, *shortTerm(2)*, quando o tempo de propagação é curto, e *longTerm(3)*, quando o tempo de propagação é excessivo. Assim, considerando i o total de papéis envolvidos na decisão, esta complexidade é representada pela Equação 2.2.

$$C_{decision} = i \times (nBranches - 1) \times gFactor \times cFactor \times vFactor \tag{2.2}$$

Complexidade de coordenação

A complexidade de coordenação avalia a interação entre os papéis envolvidos nas tarefas e a influência destas no tratamento dado aos itens de negócio. São definidos dois subtipos de complexidades: o de coordenação de links e o de tarefas compartilhadas.

A complexidade de coordenação de links investiga quão complexa é a comunicação entre papéis distintos quando estão envolvidos em tarefas em comum. Para tanto, con-

sidera se são transferidos itens de negócio e se estes são alterados durante a execução. Neste contexto, a métrica associada é a *linkType* e seus valores possíveis são *autoLink(0)*, quando a tarefa envolvida é totalmente automatizada, *controlLink(1)*, se não são transferidos itens de negócio, *dataTransfered(2)*, se são transferidos itens de negócio, e *dataAdapted(3)*, se existe a necessidade de adaptar o conteúdo do item de negócio. Considerando n o número de links presentes na tarefa, a complexidade de coordenação de links será a soma da complexidade de todos os links, de um até n, ponderada pelo número de papéis envolvidos. Considerando m o número de papéis executores da tarefa, o resultado é representado pela Equação 2.3.

$$C_{coord-link} = m \times \sum_{i=1}^{n} linkType(i) \tag{2.3}$$

Já a complexidade de tarefas compartilhadas classifica tarefas quanto a sua execução por mais de um papel. Para isto, são definidas duas métricas: *taskType* indica o tipo de compartilhamento como *notShared(0)*, quando um único papel está envolvido, *shared(1)*, quando a tarefa é executada por mais de um papel, *BIConsumed(2)*, quando a tarefa consome itens de negócio, e *BIProduced(3)*, quando a tarefa produz itens de negócio. A métrica *meetingindicator*, porém, assume valores zero ou um, indicando se existe ou não a necessidade de conhecimento entre os papéis, se interagem regularmente ou há sempre necessidade de se apresentarem. Assim, supondo n o número de papéis envolvidos na tarefa, o cálculo da complexidade de tarefas compartilhadas é formalizado pela Equação 2.4.

$$C_{coord-shared} = n \times taskType \times (meetingIndicator + 1) \tag{2.4}$$

Complexidade de itens de negócio

Para abordar a complexidade envolvida no tratamento dado pelas tarefas aos itens de negócio, a metodologia prevê dois subtipos de complexidade, o de base de itens de negócio e o de origem de itens de negócio.

A complexidade base de itens de negócio equivale ao número de itens de negócio presentes na tarefa ponderado pelo número de papéis envolvidos. Considerando n o número de papéis envolvidos e m o número de itens de negócio relacionados à tarefa investigada, a complexidade base de itens de negócio é calculada pela Equação 2.5.

$$C_{bi-base} = n \times m \tag{2.5}$$

Além disso, a complexidade de origem de itens de negócio avalia as dificuldades de tratamento dos mesmos, e para isso os classifica conforme a origem de seus dados internos (métrica *sourceScore*). Para contemplar este objetivo, cada item de negócio é modelado como um conjunto de campos, que denominam variáveis e possuem valores distintos. Cada campo pode apresentar um nível distinto de dificuldade de aquisição do valor da variável, cujo domínio de níveis compreende *internal(0)*, se os dados são resultados de ações automatizadas, *freeChoice(1)*, se a escolha do conteúdo é livre, *documentationDirect(2)*, se a decisão do conteúdo é explicitamente indicada em documentações de ferramentas ou processos, *documentationAdapted(3)*, se a documentação indicada necessita de adaptações, *bestPratice(4)*, se o conteúdo é resultado de recomendações de administradores mais experientes, *environmentFixed(5)*, se o valor é fornecido pelo ambiente de execução, e *environmentConstrained(6)*, se o valor é vinculado a um conjunto de possibilidades pré-definido e fornecido pelo ambiente de execução.

Assim, considerando n o número de papéis envolvidos na tarefa, m o número de itens de negócio e $Campos_i$ o número de campos de um item de negócio i, a complexidade de origem de itens de negócio pode ser calculada com a Equação 2.6.

$$C_{bi-source} = n \times \sum_{i=1}^{m} \sum_{j=1}^{Campos_i} sourceScore(i,j) \tag{2.6}$$

Cálculo da complexidade final

Após calcular os tipos de complexidade, é necessário aglutiná-los em um único resultado numérico. O cálculo é feito sobre cada tarefa do processo, e tanto uma única tarefa quanto um grupo de tarefas ou o processo inteiro podem ter sua complexidade total calculada pela Equação 2.7, considerando *C()* a função de cálculo de complexidade e i o processo avaliado.

$$C(i) = C_{exec-base} + C_{decision} + C_{coord-link} + C_{coord-shared} + C_{bi-base} + C_{bi-source} \tag{2.7}$$

O resultado é um número sem unidade de medida definido, mas suficiente para comparar processos ou entidades entre si. Isto é suficiente para algumas necessidades de otimização dos processos, mas para fomentar a gestão é necessário converter o resultado em unidades de medida que o negócio entenda, como tempo e dinheiro. Para atingir este objetivo, é necessário um modelo de custos, foco de diversos estudos conhecidos (A SIMPLE WAY TO ESTIMATE THE COST OF DOWNTIME, 2002) (COUCH; WU; SUSANTO, 2005), contudo fora do escopo do presente trabalho.

2.3.3 Metodologia

Com os modelos de processo e complexidade definidos, os autores descrevem sua metodologia de medição, conforme visto na Figura 2.6.

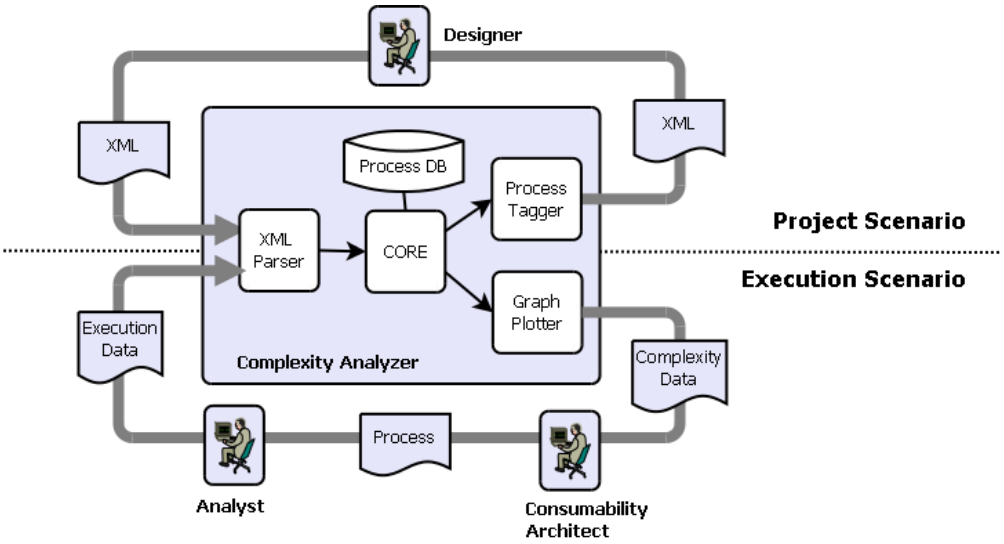

Figura 2.6: Ciclos de medida para processos de TI (DIAO; KELLER, 2006)

São dois os cenários de uso previstos. O primeiro, presente acima da linha pontilhada da Figura 2.6, usa o *complexity analyzer* – cujo conteúdo engloba todas as métricas

descritas até este ponto – em tempo de desenho do processo: um projetista o desenha conforme necessidades do negócio em uma ferramenta de modelagem de mercado, gerando um arquivo XML, e o coloca sob avaliação da ferramenta de cálculo. Como resultado, recebe um arquivo XML similar, porém pontuado com as complexidades calculadas para cada tarefa, o que permite realizar novos ajustes e repetir o ciclo até ser atingido um nível de complexidade aceitável. No cenário complementar, o de execução (presente abaixo da linha pontilhada da Figura 2.6), um analista recebe o processo para execução, grava seu desempenho em uma ferramenta específica para este fim, gera um arquivo XML com o resultado das execuções e envia o mesmo para o *complexity analyzer*. A saída, também pontuada pela ferramenta de cálculo, é enviada a um terceiro personagem, o arquiteto, cuja análise possibilita encaminhar novas execuções do processo ou esforços de redesenho e otimização. Estes ciclos se repetem até o final da vida útil do processo.

2.4 Complexidade em cenários mais distribuídos

Com os modelos, as métricas e a metodologia propostos pelos autores, é possível modelar um processo de TI, associar a cada tarefa uma complexidade numérica e, ao executá-lo, conhecer a complexidade de cada etapa. Entretanto, as métricas representam um ambiente de autoridade ideal, onde a hierarquia entre os papéis compõe um único domínio de autoridade.

O presente trabalho usará os conceitos descritos nesta seção como base para o estudo e para a proposição de uma extensão de tal metodologia, sendo esta o objetivo deste trabalho, ou seja, estender para contemplar ambientes cujas hierarquias de autoridade são independentes e para quantificar a complexidade da existência de múltiplos domínios de autoridade em um único processo de TI. Na próxima Seção, o problema a ser resolvido será melhor detalhado e exemplificado, bem como a solução proposta, incluindo definições de termos, modelos e metodologia estendidos.

3 COMPLEXIDADE E PROCESSOS INTERDOMÍNIOS

A interação entre entidades autônomas é uma característica de ambientes corporativos, seja por crescimento mútuo, por obrigações legais etc. Diversos são os motivos para o uso de inúmeras tecnologias de comunicação disponíveis. Contudo, isto acarreta a necessidade de troca de informações de forma eficiente, com a preservação da confidencialidade e da autonomia de cada instituição envolvida. Uma forma de garantir a eficiência, mesmo em ambientes mais distribuídos, é conhecer e controlar a complexidade das atividades exercidas.

Na busca por conhecer a complexidade de processos e procedimentos, os trabalhos relacionados no capítulo anterior mostram-se complementares. Em um deles, Keller et al. (DIAO; KELLER, 2006) focam a complexidade de processos de TI, propõem métricas que abordam a complexidade de suas etapas e quantificam sua complexidade. No entanto, quando um processo cruza diferentes instituições em seus fluxos, surgem fatores que extrapolam as possibilidades das metodologias propostas, entre eles a cooperação entre as partes envolvidas e a confidencialidade do conteúdo das tarefas que estas compartilham.

Esta é a problemática analisada no presente trabalho, que se propõe a estender as metodologias atuais para contemplar cenários que envolvam múltiplas instituições autônomas. Com este intuito, são definidos os modelos de processo e complexidade utilizados na metodologia, vistos a seguir.

3.1 Modelo de processo de TI

A modelagem de processos de TI utiliza como base o modelo apresentado na seção 2.3.1, composto por papéis, decisões, tarefas e itens de negócio, conjunto suficiente para representar todos os processos analisados, e os itens a seguir, que sustentam a metodologia:

Domínios de autoridade consistem em conjuntos de papéis, ou seja, instituições capazes de executar por completo a metodologia descrita na seção 2.3 (DIAO; KELLER, 2006). São independentes, com estratégias e objetivos próprios.

Federações de provisionamento de serviços são aqui sinônimo de domínio de autoridade. O termo é citado com a finalidade de compatibilizar este estudo com outros importantes esforços que visam a cooperação e a confidencialidade nas relações entre federações distintas ou hierarquizadas (CAI; TURNER; GAN, 2001) (CHEN et al., 2010).

Processos de TI interdomínios são caracterizados pela presença de tarefas executadas por diferentes domínios de autoridade, constituindo o nicho estudado no presente

estudo.

Com base nestas definições, a próxima seção detalha o problema central motivador desta dissertação.

3.2 Processos interdomínios

Por definição, processos interdomínios devem ser executados por dois ou mais papéis pertencentes a domínios de autoridade diferentes. Neste contexto, um exemplo de processo interdomínio é a interoperabilidade entre as empresas de telecomunicação no Brasil, ou mesmo seus serviços de portabilidade numérica de telefonia. Obedecendo à regulação governamental e às necessidades de seu nicho de mercado, empresas concorrentes cooperam e executam processos operacionais entre si, fornecendo serviços em conformidade com a lei e fiscalizados quanto a sua eficiência (ANATEL, 2012). Entretanto, o envio de informações confidenciais entre empresas concorrentes pode atribuir risco aos seus negócios e as suas estratégias. Sem contar que a qualidade destas relações pode distorcer as informações trocadas entre os domínios de autoridade envolvidos, prejudicando o conhecimento do processo por completo e as iniciativas de otimização e controle sobre o mesmo.

Outro exemplo é a restrição legal – no Brasil – de provedores de conectividade de última milha, cujos serviços chegam às casas dos clientes, interligarem os usuários finais diretamente à Internet ou às redes públicas de telefonia (PSTN) (BRASIL, 2012). Esta atividade envolve diferentes instituições, como provedores de última milha, de Internet e de acesso às redes de telefonia, que em conjunto executam processos interdomínios para fornecimento de serviços e suporte. Neste exemplo, os domínios não são necessariamente concorrentes, o que não os impede de proteger informações estratégicas, pois alguns objetivos e algumas estratégias podem apresentar conflitos de interesse.

Em um cenário fictício, a Figura 3.1 demonstra a passagem de um mesmo processo de TI por domínios administrativos diferentes. Considerando quatro domínios distintos, denominados por A, B, C e D, a figura modela um ambiente no qual deve ser configurada uma reserva de recursos para a comunicação entre os domínios A e C, neste caso banda em links de dados entre roteadores. Para cumprir este objetivo, dois são os caminhos possíveis, transpondo os domínios B ou D, e os operadores de cada domínio devem configurar seus roteadores adequadamente.

Objetivos de negócio diferentes podem levar domínios diferentes a dar níveis de importância distintos a tarefas iguais. Isto pode influenciar no nível de conhecimento dos operadores e, consequentemente, no desempenho frente à tarefa executada. Imaginando uma situação em que A e B denominam instituições concorrentes, enquanto D possui melhor relação com A, é possível que as dificuldades de interação ou comunicação aumentem a complexidade do processo ao passar por B. Ainda no mesmo exemplo, mas considerando que B foca seus esforços em um nicho diferente de negócio, como a telefonia, enquanto o domínio D tem maior *know-how* em tráfego de dados, é possível que a complexidade do provisionamento de um serviço de dados efetuado pelo domínio D seja menor do que de B, fazendo deste caminho o de menor esforço de execução.

Além disso, domínios não necessariamente caracterizam instituições distintas. Se os domínios denominarem instituições de boa relação, ou departamentos de uma mesma instituição, a interação pode tender a maior simplicidade e transparência.

As hipóteses citadas nesta seção motivam uma metodologia capaz de quantificar a complexidade destes processos mesmo quando um domínio de autoridade omite infor-

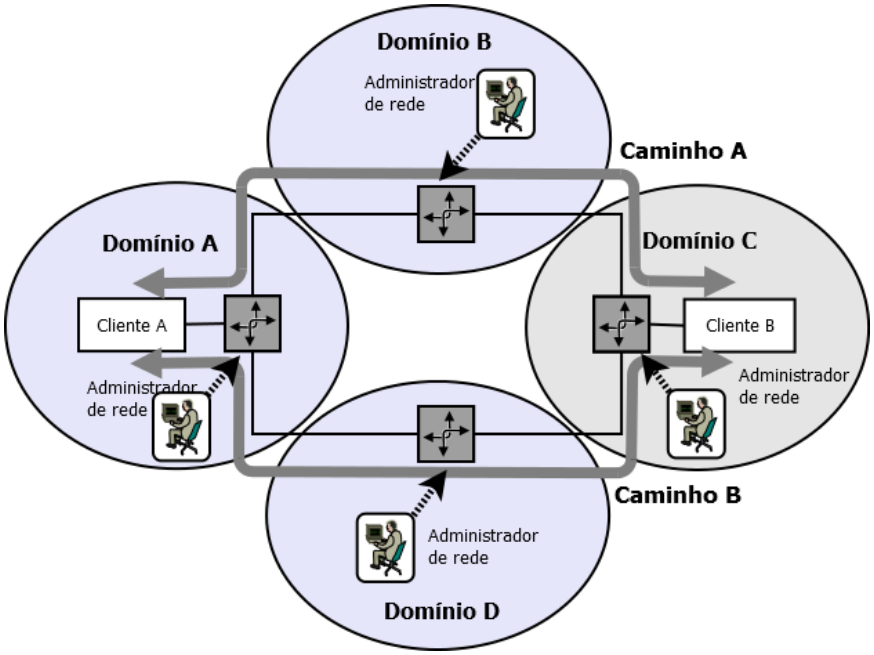

Figura 3.1: Configuração de reserva de recursos

mações. Os resultados podem fomentar a gestão da eficiência dos processos, baseando decisões em dados confiáveis.

3.3 Quantificação da complexidade de processos de TI interdomínios

As seções anteriores reforçam que, quando envolvidos em um mesmo processo, domínios de autoridade distintos necessitam de conhecimento sobre as partes do processo executadas externamente e, ao mesmo tempo, precisam manter seus riscos minimizados omitindo informações sigilosas. Assim, a qualidade da informação depende de quais itens são considerados confidenciais e divulgáveis por cada domínio envolvido.

Com o objetivo de possibilitar que diversos projetistas interajam em busca de um processo de complexidade aproximada e conhecida por todos, é necessário padronizar a comunicação entre os envolvidos. Com este objetivo, são definidos fluxos de troca de informações baseada em subprocessos e em ajustes de complexidade.

3.3.1 Processos e subprocessos

Para possibilitar a troca de informações, a omissão de informações sigilosas e os ajustes de complexidade que garantam os mesmos resultados para todos os domínios envolvidos, são definidos os seguintes subprocessos:

***Local Private Process* (LPR)** é definido por cada domínio e contém todas as tarefas executadas pelos papéis locais, com todos os detalhes necessários e conhecidos pelo projetista local. Assim, esta versão contém dados confidenciais a serem omitidos, mas apenas das tarefas do domínio referente.

***Local Public Process* (LPU)** é a versão possível de divulgar do LPR. Omite informações, e seu conteúdo varia conforme a relação entre o domínio de origem e o de destino. Neste contexto, um domínio pode gerar um LPU diferente para cada domí-

nio parceiro do processo, omitindo informações conforme necessário e conveniente em cada caso.

***Full Private Process* (FPR)** define a versão completa do processo, com todas as informações incluídas, ou seja, unindo todos os LPRs de todos os domínios. Embora improvável na realidade – já que nenhum domínio deverá receber todas as informações confidenciais de todos os outros domínios –, este item é conceitualmente importante para a avaliação da própria metodologia, ou seja, para comparar os resultados ajustados de cada domínio com o processo ideal. No entanto, nunca poderá ser realizada em um caso real.

***Full Process* (FP)** define a versão completa do processo na visão de um domínio, unindo seu LPR com todos os LPUs recebidos. Esta é a versão que dá a visibilidade necessária do todo a cada um, porém ainda sem os ajustes necessários, ou seja, ainda com as diferenças impostas pela omissão de informações.

***Full Adjusted Process* (FAD)** representa o FP após realizados os ajustes de complexidade, ou seja, após compensadas as diferenças de cálculo entre o LPR e o LPU de cada domínio. Para isto, é utilizado o índice de ajuste de complexidade IDX, descrito a seguir.

3.3.2 Fator de ajuste de complexidade (IDX)

Conforme descrito nas definições de processos e subprocessos, a estrutura do LPU difere da real, descrita no LPR. Ao serem analisados pelo *complexity analyzer*, considerando que o LPU omite informações, resultarão em diferentes valores de complexidade. Assim, um domínio enxergaria a complexidade total de forma diferente de outros, quebrando a premissa de ter a mesma complexidade resultante para cada domínio de autoridade. Visando equalizar as visões de todos, é proposto um índice de ajuste de complexidade, o IDX.

Considerando *C()* a função executada pelo *complexity analyzer* e i o domínio de autoridade executor do cálculo, o objetivo do IDX é garantir que, ao final de todos os fluxos de desenho do processo, a Equação 3.1 seja respeitada.

$$C(FAD_i) \equiv C(FPR_i) \tag{3.1}$$

Dando prosseguimento à aplicação da metodologia, de forma a obedecer à aproximação representada pela Equação 3.1, o FAD será a melhor aproximação possível do FPR. Para chegar ao FAD, é necessário que todos os domínios calculem e enviem aos parceiros, junto com sua parte do processo, o respectivo índice de ajuste de complexidade IDX. Considerando que um domínio i pode ter n domínios parceiros d – e portanto n versões de seu LPU – e que por definição seu LPR sempre será o mesmo, para cada um dos n domínios parceiros o $IDX_{i,d}$ deve ser calculado de acordo com a Equação 3.2.

$$IDX_{i,d} = C(LPR_i)/C(LPU_{i,d}) \tag{3.2}$$

Considerando bem definidos os tipos de processos e o método de cálculo do índice de ajuste de complexidade, pode-se definir como estas informações transitarão entre os domínios.

3.3.3 Algoritmo de troca de subprocessos

O objetivo deste algoritmo é garantir que todos os domínios desenhem seus processos locais, calculem seus índices de ajuste e, ao final da comunicação entre os domínios, todos obtenham uma visão confiável da complexidade do processo como um todo. Assim, por definição, este estudo considera a premissa de que todos os domínios executam completamente a metodologia descrita na seção 2.3 e que a extensão proposta é baseada na comunicação entre os projetistas de todos os domínios envolvidos, conforme visto na Figura 3.2.

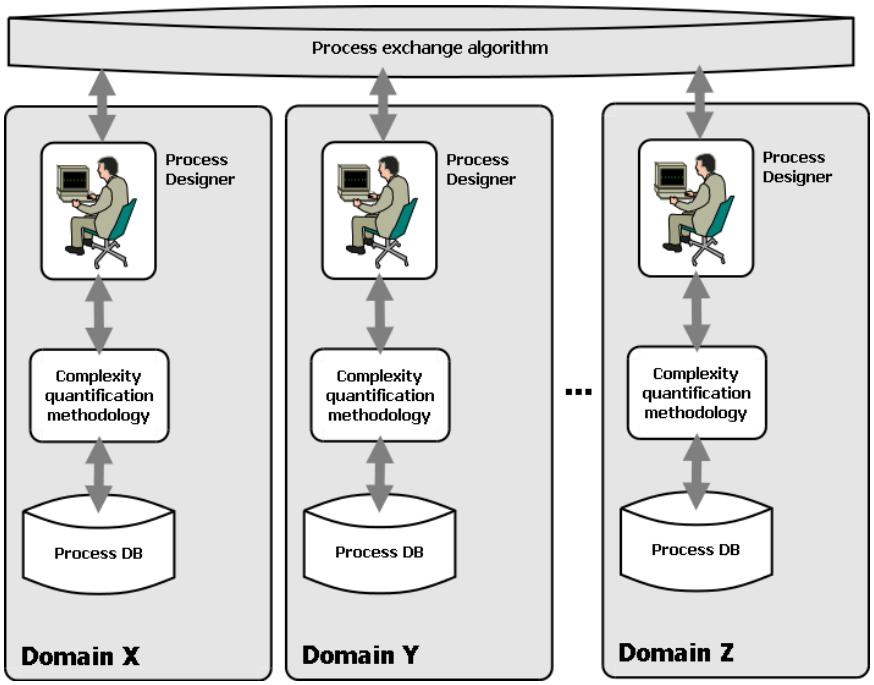

Figura 3.2: Troca de subprocessos

Com base nas definições de processos, subprocessos e IDX e na troca de informações entre projetistas, cada domínio i adquire seu FAD_i conforme os passos a seguir:

(i) **Pré-requisitos.** Antes da fase de desenho do processo, na qual o processo será modelado visualmente, todos os domínios interagem até definir funções e interfaces de cada um. Ou seja, definem o que será executado por cada domínio e como cada um terá seu subprocesso conectado aos subprocessos de domínios parceiros. Conforme citado anteriormente, outros trabalhos estudam a complexidade deste tipo de negociação, e por isto estas complicações serão consideradas bem resolvidas, mantendo o foco na análise do processo após as negociações serem concluídas.

(ii) **Definição de subprocessos.** Após definidas funções e interfaces, cada domínio i modela seu LPR_i. Com base nele, considerando a confidencialidade de seus dados e a qualidade de sua relação com cada domínio parceiro d, para cada combinação *(i,d)* o domínio i modela seus subprocessos passíveis de divulgação, os $LPU_{i,d}$. Depois de definidas todas as versões de LPU, cada domínio i envia para cada domínio parceiro d seu respectivo $LPU_{i,d}$.

(iii) **Complexidade do Processo.** Neste ponto, como cada domínio i possui um $LPU_{d,i}$ para cada domínio parceiro d, é possível aplicar a Equação 3.2 e calcular todos os

índices $IDX_{i,d}$ necessários. Com todos os *IDX* definidos, cada domínio *i* envia a cada parceiro *d* seu $IDX_{i,d}$, completando o envio de informações necessárias para que cada domínio tenha seus FP_i e FAD_i bem formados. Possuindo todas as informações necessárias, cada domínio *i* calcula a complexidade do processo FAD_i – e de suas tarefas – utilizando as Equações 3.4 e 3.3.

3.3.4 Cálculo ajustado das complexidades

Complexidade de uma tarefa

Sendo *C()* a função de aplicação das métricas de complexidade, *AC()* a função de complexidade ajustada pelo *IDX*, *i* o domínio de autoridade local e *t* uma tarefa contida no processo FP_i e executada por um domínio parceiro *d*, o cálculo da complexidade ajustada de *t* pode ser efetuado conforme a Equação 3.3.

$$AC(t) = C(t) \times IDX_{d,i} \tag{3.3}$$

Complexidade de um processo

Sendo *i* o domínio local, *n* o total de domínios contidos no processo avaliado FAD_i, TC_m o total de tarefas executadas por um domínio de autoridade *m* no FAD_i e $T_{d,j}$ uma tarefa *j* executada pelo domínio *d*, a complexidade $AC(FAD_i)$ é calculada conforme abaixo:

$$AC(FAD_i) = \sum_{k=1}^{n} \sum_{l=1}^{TC_k} (C(T_{k,l}) \times IDX_{k,i}) \tag{3.4}$$

O cálculo da complexidade é efetuado automaticamente pela ferramenta *complexity analyzer* com base na aplicação das métricas descritas na seção 2.3. Conforme definido anteriormente, cada domínio deve ter sua própria estrutura de análise da complexidade e ser capaz de executar toda a parte da metodologia que lhe compete.

Além do citado até agora, o IDX pode ser utilizado como um indicador do nível de omissão de informações, ou seja, da qualidade da relação entre os domínios. Um valor próximo de zero pode indicar uma relação mais aberta, enquanto um alto valor pode indicar maior atrito ou desconfiança entre os domínios, e possível maior omissão de informações. A avaliação do uso do IDX como métrica consiste em um relevante trabalho futuro.

Também é importante ressaltar que é provável que ocorram casos nos quais valores de complexidade diferentes são calculados para tarefas de mesmo nome, porém alocadas nas visões locais de domínio diferentes. Entretanto, isto não inviabiliza o uso da metodologia, pois sempre o domínio local terá a noção da complexidade imposta por cada domínio parceiro e pelo processo completo. Assim, todos poderão identificar as tarefas de maior custo e que exigem ações locais ou cobranças externas para tratamento da complexidade, e por fim aplicar um modelo de custo – após sua definição em um trabalho futuro – para obter resultados em unidades de medida que o negócio e os gestores entendam, como tempo, dinheiro etc.

Neste capítulo foram apresentadas as extensões propostas para metodologias existentes. Para avaliá-las foi implementado um protótipo da ferramenta *complexity analyzer*,

alterada para possibilitar os cálculos aqui citados. Este protótipo será apresentado no próximo Capítulo.

4 PROTÓTIPO

O objetivo deste trabalho é estender uma metodologia já existente, descrita no Capítulo 2, para possibilitar a quantificação da complexidade de processos de TI que cruzem diferentes domínios de autoridade (Capítulo 3). Assim, após propor a extensão do modelo, é necessário avaliar sua eficácia e, para isto, foi desenvolvido um protótipo do centro da metodologia, o *Complexity Analyzer*.

Por definição, esta ferramenta recebe os processos em formato de arquivo XML, interpreta seu conteúdo, aplica as métricas para cálculo de complexidade de processos de TI descritas na Seção 2.3.2, calcula a complexidade resultante para cada tarefa e fornece como saída o mesmo processo, incluindo os valores de complexidade calculados e outras informações úteis a projetistas e analistas, como diagramas e histogramas. Como resultado, os dados de saída – também em formato XML – devem indicar que o processo está dentro dos objetivos desejados ou embasar decisões sobre que etapas do processo devem ser o foco de esforços de automação ou redesenho do processo.

O *Complexity Analyzer* é o centro da metodologia estendida no presente trabalho e cuja dinâmica foi apresentada na Seção 2.3. Esta prevê uma série de ferramentas e tecnologias utilizadas para compor a solução, como XML (BRAY et al., 2003), para representar processos, *Eclipse Modelling Framework* (STEINBERG et al., 2008) e *IBM Websphere Modeling Tool* (IBM, 2011), na modelagem dos mesmos, e *Scalar Vector Graphics* (SVG) (QUINT, 2003), para representá-los visualmente. Mantendo o foco central da metodologia e as mesmas funcionalidades, o protótipo aqui descrito utiliza ferramentas e tecnologias similares: o conjunto de métricas foi implementado utilizando classes Java e biblioteca JDOM (MCLAUGHLIN, 2001), cuja composição é capaz de ler arquivos XML, computar a complexidade de processos de TI nele representados e gerar como saída um novo XML pontuado com os valores de complexidade resultantes, além dos arquivos de dados necessários para gerar diagramas e histogramas. Assim, depois de lidos os arquivos e pontuada a complexidade, os diagramas são gerados com a biblioteca de automação de visualização de gráficos Graphviz (ELLSON et al., 2002), mesmo procedimento executado para histogramas pelo Gnuplot (RACINE, 2006), um difundido e eficiente gerador de gráficos científicos. Finalizando, um sistema Web é disponibilizado para agregar todos os resultados dos processos analisados e oferecer a projetistas e analistas as informações necessárias.

Esta estrutura torna o protótipo suficiente para representar os resultados da metodologia e possibilita a análise de processos com foco na avaliação de sua eficácia para processos interdomínios. Para tornar isto possível, é necessário definir a arquitetura do sistema, as suas interfaces e os seus componentes internos, o que será visto a seguir.

4.1 Arquitetura

A arquitetura do protótipo consiste em módulos responsáveis pela leitura do processo XML, gerado pela ferramenta de modelagem visual de processos de TI, pela aplicação de métricas e fórmulas de cálculo de complexidade nos dados inseridos e pela geração de arquivos de saída, cujo formato deve ser compatível como a ferramenta utilizada para gerar as entradas inseridas. Vista na Figura 4.1, a arquitetura é composta pelos itens listados abaixo:

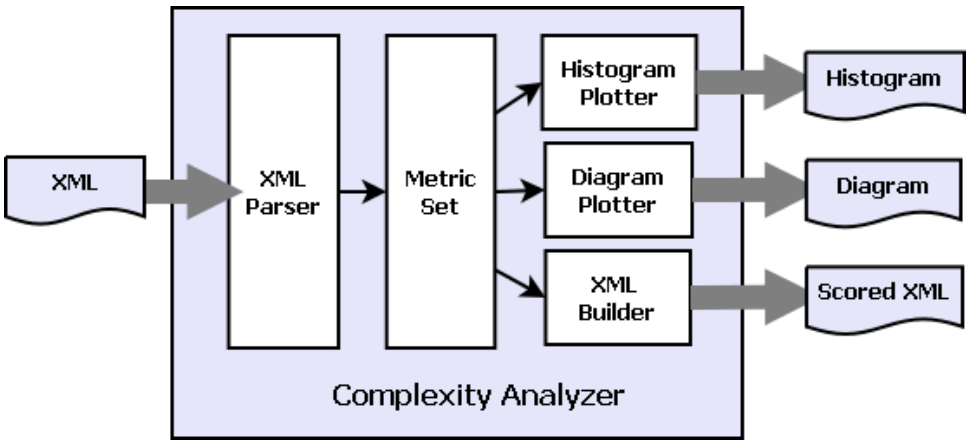

Figura 4.1: Arquitetura do protótipo do *Complexity Analyzer*

XML parser. Este módulo é responsável por ler e entender o arquivo XML de entrada, além de carregar em estruturas de dados – neste caso, classes Java – todas as entidades e todos os atributos de formação do processo analisado.

Metric application. Este módulo é responsável por aplicar as métricas do modelo de complexidade na estrutura gerada pelo módulo *XML Parser* e por calcular os valores de complexidade para cada tarefa do processo. Entre suas atribuições está aplicar o ajuste de complexidade proposto pelo fator IDX, descrito na Seção 3.3.2.

Diagram plotter. Este módulo é responsável por ler os dados de complexidade calculados e a estrutura do processo e por redesenhar graficamente o mesmo, incluindo no diagrama gerado as complexidades calculadas. O objetivo é embasar visualmente a análise de projetistas e analistas, que poderão identificar os pontos onde é necessário tratar tais complexidades até alcançar valores aceitáveis.

Histogram plotter. Realiza tarefa similar à do módulo anterior, lendo as complexidades calculadas e a estrutura do processo. Porém, diferencia-se por gerar um histograma em formato de gráfico de barras, representando a complexidade de cada tarefa do processo.

XML builder. Após conhecer o processo e sua complexidade, este módulo reescreve o processo em um novo arquivo XML de saída, contendo as complexidades mensuradas e possibilitando um novo ciclo da metodologia. Tanto o XML de entrada quanto este de saída podem ser importados ou adaptados para ferramentas de modelagem de processos de mercado.

Para implementar esta arquitetura, o protótipo foi modelado em um diagrama de classes, conforme as funcionalidades acima descritas e o modelo de processo utilizado. Segundo o modelo, processos são formados por conjuntos de tarefas que constituem um fluxo de trabalho e que podem ser consideradas decisões quando se comunicam com mais de uma tarefa do mesmo conjunto. Neste contexto, tarefas podem ser executadas por um ou mais papéis, e comunicações entre tarefas podem envolver zero ou mais dados transmitidos, estes chamados de itens de negócio, formados por um ou mais campos de dados. Destas definições, foi extraído o diagrama de classes do sistema, cujos principais componentes podem ser vistos no diagrama da Figura 4.2 e são descritos a seguir.

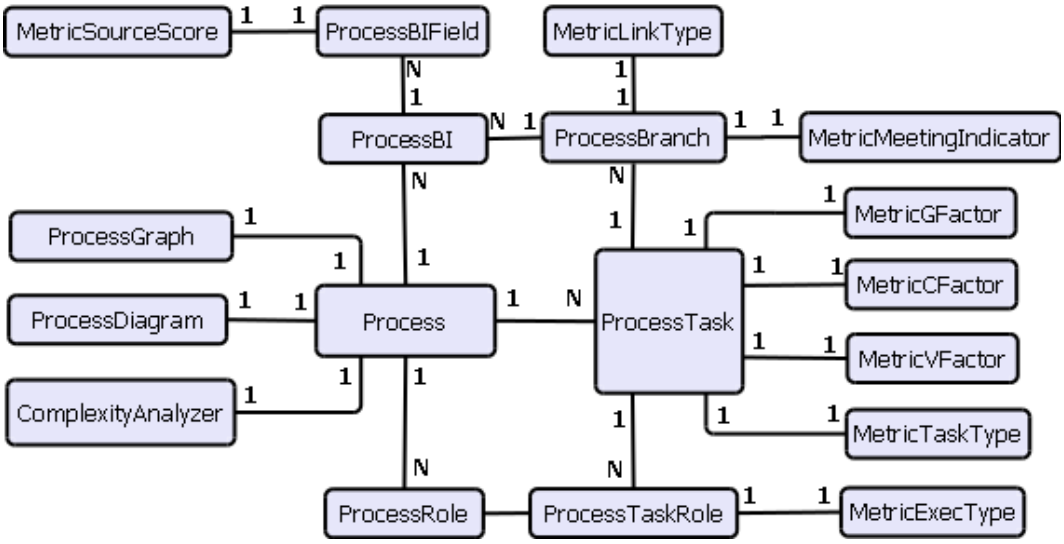

Figura 4.2: Diagrama de classes

Classe *MetricTaskType.* Representa a métrica *taskType*, indicando o grau de compartilhamento da tarefa entre papéis distintos e como esta trata informações geradas ou consumidas por ela (itens de negócio).

Classe *MetricSourceScore.* Esta classe representa a métrica *sourceScore*, que classifica itens de negócio conforme a dificuldade de captação de seus valores internos.

Classe *MetricExecType.* Representa a métrica *execType*, classificando tarefas quanto ao seu nível de automação.

Classe *MetricLinkType.* Classifica a complexidade da comunicação entre tarefas, função da métrica *linkType*.

Classe *MetricMeetingIndicator.* Representa a métrica *meetingIndicator* para cada fluxo entre duas tarefas, ou seja, se a relação já existe ou não.

Classe *MetricGFactor.* Esta classe implementa a métrica *gFactor*, relacionada a decisões, indicando o grau de necessidade de fatores externos na escolha do caminho a seguir.

Classe *MetricCFactor.* Representa a métrica *cFactor*, relacionada a decisões, indicando o impacto de uma decisão errada.

Classe *MetricVFactor.* Também relacionada a decisões, representa a métrica *vFactor*, indicando o tempo de propagação da decisão no processo.

Classe *Complexity Analyzer.* É a classe raiz do protótipo, responsável por instanciar a classe *Process* e chamar suas rotinas de leitura de arquivos de entrada e escrita dos arquivos de saída.

Classe *Process.* Esta classe modela a raiz dos processos avaliados, que por definição possuem listas de tarefas, papéis e itens de negócio, todos relacionados entre si conforme o modelo de processo de TI utilizado. Além de instanciar estas entidades, possui relação com as classes de geração de diagramas e histogramas.

Classe *ProcessDiagram.* Faz interface com a biblioteca Graphviz, responsável pelo desenho dinâmico do diagrama do processo.

Classe *ProcessGraph.* Faz interface com a biblioteca Gnuplot, responsável pelo desenho dos histogramas fornecidos como saída pelo protótipo.

Classe *ProcessBI.* Esta classe instancia itens de negócio. Cada processo possui uma lista fixa deles e, quando necessário, tais itens podem ser referenciados pela classe *ProcessBranch*, que representa as comunicações entre tarefas.

Classe *ProcessBIField.* Cada item de negócio é formado por uma lista de campos, cada um com características próprias. Esta classe representa um campo de um item relacionado à classe *MetricSourceScore*.

Classe *ProcessTask.* Instancia as tarefas do processo, definindo em conjunto instâncias das classes *MetricGFactor, MetricCFactor e MetricVFactor*, as três relacionadas a métricas de análise de tarefas. Além destas três, implementa a métrica *nBranches*, cujo valor indica o número de possíveis caminhos: um, para tarefas, ou a partir de dois, caso a tarefa constitua uma decisão.

Classe *ProcessRole.* Instancia os papéis do processo.

Classe *ProcessTaskRole.* Representa um papel quando este é executor de uma tarefa. Cada tarefa pode ter uma lista de papéis executores, ou seja, pode ser compartilhada entre eles.

Classe *ProcessBranch.* Representa a conexão entre duas tarefas distintas, em que se aplica a métrica *meetingIndicator* e, por isto, instancia a classe *MetricMeetingIndicator*.

Estas são as principais classes modeladas. Nelas estão representadas todas as métricas previstas pelo modelo de complexidade utilizado. Além destas, as classes auxiliares são responsáveis por tarefas internas do protótipo, como a leitura e a gravação dos processos em formato XML, entre outras funções.

4.2 Representação do processo em XML

Os dados são inseridos no protótipo através de arquivos XML modelados a partir dos processos de TI analisados. A Figura 4.3 representa o código XML de um processo criado para exemplo. Este conta com duas tarefas, dois papéis e um item de negócio. Neste

Tabela 4.1: Valores possíveis para a métrica *sourceScore*

Conteúdo	Peso no cálculo
INTERNAL	0
FREE_CHOICE	1
DOCUMENTATION_DIRECT	2
DOCUMENTATION_ADAPTED	3
BEST_PRATICE	4
ENVIRONMENT_FIXED	5
ENVIRONMENT_CONSTRAINED	6

exemplo, a raiz do processo é representada pela *tag* <*process*>. Já as *tags* <*bi*>, <*role*> e <*task*> representam as entidades item de negócio, papel e tarefa, respectivamente. A *tag* <*branch*>, por sua vez, indica comunicação com outra tarefa do processo, e atributos como *sourcescore* e *exectype* identificam métricas do modelo de complexidade.

Os atributos responsáveis pelas métricas do modelo têm seu conteúdo limitado às possibilidades de valores previstos para cada uma. Por exemplo, o item de negócio *BI1*, definido na Figura 4.3, tem dois campos em sua composição, o *BIF1* e o *BIF2*. Como *SourceScore* é uma métrica que incide sobre campos de itens de negócio, o atributo *sourcescore* está presente na *tag* <*field*>, cujos valores possíveis estão dispostos na Tabela 4.1, conforme definido na Seção 2.3. A primeira coluna (Conteúdo) lista os nomes dos possíveis valores, como *FREE_CHOICE*, para escolha livre do valor, ou *BEST_PRATICE*, para valores provenientes de códigos de boas práticas, entre outras possibilidades. Por outro lado, a segunda coluna (Peso no cálculo) indica qual será a influência destas informações ao se aplicar a fórmula responsável pelo cálculo da métrica avaliada (*sourceScore*).

O mesmo ocorre para outras entidades, outros atributos e outras métricas: todos estes componentes são definidos em um XML *Schema* (HAROLD; MEANS, 2004) utilizado como padrão para composição e avaliação dos processos modelados e gerados, cujo conteúdo está integralmente disponível no Anexo II. Independentes do *Schema*, os pesos de cálculo de cada possibilidade de parametrização de cada métrica estão definidos no modelo de complexidade descrito na Seção 2.3 e implementados no código do protótipo.

O mesmo XML *Schema* é utilizado para o arquivo de saída, após o processamento das métricas e das fórmulas pelo *Complexity Analyzer*, possibilitando a metodologia cíclica de análise.

4.3 Implementação

Com a arquitetura definida, a linguagem escolhida para implementação do protótipo foi o Java, em função das facilidades de modelagem em classes, o que fornece um código fiel à arquitetura planejada. Além disso, a partir do *core* do protótipo são integradas algumas ferramentas para geração de gráficos e histogramas, conforme descrito a seguir.

4.3.1 Módulo *XML parser*

Baseado na biblioteca Java JDOM, este módulo utiliza a classe *SAXBuilder* para ler os arquivos XML modelados a partir do processo real e para carregar todos os seus dados em classes Java, incluindo todas as informações necessárias para a aplicação da metodologia.

```
<?xml version="1.0" encoding="UTF-8"?>
<process>
  <desc>Process A</desc>
  <bi name="BI1">
    <field name="BIF1" sourcescore="FREE_CHOICE" />
    <field name="BIF2" sourcescore="ENVIRONMENT_CONSTRAINED" />
  </bi>
  <role name="Role1" />
  <role name="Role2" />
  <task name="Task1" cfactor="SEVERE" vfactor="LONG_TERM"
                gfactor="INTERNAL" tasktype="BI_PRODUCED"
                meetingindicator="MEETING_NEEDED">
    <role name="Role1" exectype="MANUAL" />
    <branch name="Task1" linktype="DATA_ADAPTED">
      <bi name="BI1" />
    </branch>
  </task>
  <task name="Task2" cfactor="SEVERE" vfactor="LONG_TERM"
                gfactor="INTERNAL" tasktype="BI_PRODUCED"
                meetingindicator="MEETING_NEEDED">
    <role name="Role2" exectype="MANUAL" />
  </task>
</process>
```

Figura 4.3: Código XML de um processo de TI

O trecho de código da Figura 4.4 mostra a carga de um arquivo, partindo da *tag <process>*, raiz do arquivo, e varrendo todas as suas *tags*, encadeadas de forma concisa com base no XML *Schema*. Ao detectar cada *tag*, são instanciados novos objetos para cada tipo de entidade do processo, o que é exemplificado na Figura 4.5 para a carga de novas entidades do tipo papel.

```
SAXBuilder builder = new SAXBuilder(false);
Document xmlDoc = builder.build(input_xml);
Element root = xmlDoc.getRootElement();
```

Figura 4.4: Uso da classe *SAXBuilder* para ler arquivos XML de entrada

```
i = proc_roles.iterator();
while (i.hasNext()) {
        Element proc_role = (Element) i.next();
        P.roleList.add(new ProcessRole(proc_role.getAttributeValue("name")));
}
```

Figura 4.5: Exemplo de instanciação de entidades do processo utilizando as classes projetadas

Ciclos como o da Figura 4.5 repetem-se até que todo o arquivo tenha sido lido com sucesso e todas as entidades tenham sido replicadas através das classes definidas na modelagem do sistema. Assim, com o processo totalmente carregado, pode-se aplicar sobre ele as métricas de cálculo de complexidade.

4.3.2 Módulo *metric application*

Este módulo do sistema analisa todo o processo de TI, aplicando as fórmulas de cálculo de complexidade e registrando para cada tarefa a complexidade calculada. A Figura 4.6 exemplifica o cálculo de cada tipo de complexidade por tarefa do processo, listando os métodos definidos para a aplicação das fórmulas descritas no Capítulo 2.3.

```
1 for (i = this.process.taskList.iterator(); i.hasNext(); ) {
2         t = (ProcessTask) i.next();
3         int comp_exec = t.execComplexity();
4         int comp_coord = t.coordComplexity();
5         int comp_bi = t.biComplexity();
6         ...
7 }
```

Figura 4.6: Chamada de métodos de cálculo de complexidade

Na Figura 4.6, o objeto *t* representa uma tarefa do processo, e os métodos *t.execComplexity(), t.coordComplexity()* e *t.biComplexity()* retornam o resultados das aplicações das métricas para os três tipos de complexidade, de execução, coordenação e itens de negócio, respectivamente. Portanto, a carga do processo em formato XML e o processamento das complexidades permitem gerar os arquivos de saída previstos neste protótipo, tarefas dos três módulos a seguir.

4.3.3 Módulo *diagram plotter*

Os diagramas são gerados utilizando uma ferramenta de automação de geração de gráficos, o Graphviz (ELLSON et al., 2002), que possui uma linguagem própria, exemplificada na Figura 4.7 para um processo com duas tarefas.

```
1  digraph G {
2   color="lightgrey";
3   rankdir="TD";
4   nodesep="2";
5   Task01 [
6    shape=plaintext,
7    label=<
8     <TABLE BORDER="2" CELLBORDER="1" CELLSPACING="0" CELLPADDING="1">
9     <TR><TD BGCOLOR="#dedede">Task01</TD><TD COLSPAN="3">Role01</TD>
10    </TR><TR><TD ROWSPAN="2">25</TD><TD>Exec</TD><TD>Coord</TD><TD>BI</TD>
11    </TR><TR><TD>2</TD><TD>3</TD><TD>20</TD></TR></TABLE>
12   >
13  ];
14  Task02 [
15   shape=plaintext,
16   label=<
17    <TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="1">
18    <TR><TD BGCOLOR="#dedede">Task01</TD><TD COLSPAN="3">Role01</TD>
19    </TR><TR><TD ROWSPAN="2">66</TD><TD>Exec</TD><TD>Coord</TD><TD>BI</TD>
20    </TR><TR><TD>2</TD><TD>6</TD><TD>58</TD></TR></TABLE>
21   >
22  ];
23  Task01 -> Task02;
24 }
```

Figura 4.7: Configuração do Graphviz para gerar um diagrama de processo com duas tarefas e pontuação da complexidade

Na Figura 4.7, a estrutura *digraph* indica o uso de um grafo direcional na formação do diagrama, seguida das diretivas de formatação *color*, *rankdir* e *nodesep*, e das definições das entidades dos processos e de suas relações. As diretivas *Task01* e *Task02* definem tarefas do processo de TI. Na diretiva *label*, é incluído código HTML de uma tabela com os valores de complexidade calculados. Finalizando, a diretiva de conectividade *Task01->Task02* define o fluxo do processo. Outras tarefas ou decisões seguem o mesmo princípio, necessitando de diretivas de conectividade para cada comunicação entre tarefas. Assim, esta ferramenta é capaz de desenhar automaticamente o processo visto descrito na Figura 4.7, gerando o diagrama da Figura 4.8, com todas as entidades e todos os valores de complexidade calculados.

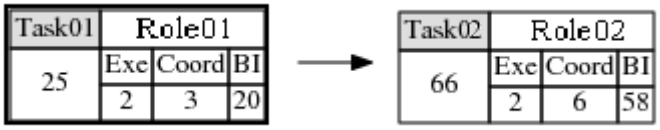

Figura 4.8: Exemplo de processo desenhado pelo Graphviz

Este mecanismo foi utilizado como ferramenta de avaliação. Para a aplicação prática, é necessário utilizar uma interface de modelagem de processos que dê maior flexibilidade a projetistas e executores. Uma ferramenta interativa permitiria modelar diretamente no diagrama, o que o Graphviz não possibilita, embora não inviabilize a avaliação da metodologia proposta.

4.3.4 Módulo *histogram plotter*

Os histogramas são gerados com o objetivo de destacar as diferenças de complexidade entre as tarefas do processo, permitindo a projetistas e analistas identificarem pontos de maior complexidade ou que necessitem de melhorias ou automação. Com este intuito, a ferramenta escolhida para gerar histogramas no formato de gráficos de barras foi o Gnuplot (RACINE, 2006), que permite automatizar o processo. A Figura 4.9 exemplifica resumidamente a linguagem desta ferramenta.

```
1  set terminal postscript eps color lw 15 "Helvetica" 36
2  set output 'histogram.eps'
3  set size '1.7,1.7'
4  set border 3 front linetype -1 linewidth 1.000
5  set boxwidth 0.8 absolute
6  set style fill solid 1.00
7  set grid nopolar
8  set title "Exemplo"
9  set xlabel ""
10 set xlabel offset character 0, -2, 0 font "" textcolor lt -1 norotate
11 plot 'complexity.dat' using 2:xtic(1) t 2
```

Figura 4.9: Exemplo de definição de gráfico do Gnuplot

As diretivas *set* definem propriedades de formatação e de tratamento dos dados e do gráfico gerado. Já a diretiva *plot* indica de qual arquivo serão coletados os dados a serem desenhados – neste caso, o arquivo *complexity.dat*, gerado pelo protótipo. Assim, definido como o gráfico será formatado, o padrão de formatação é cruzado com os dados gerados

pela ferramenta, desenhando os histogramas que mostram a complexidade de cada tarefa do processo.

A Figura 4.10 detalha um arquivo de dados com resultados calculados: o arquivo é tabulado por colunas, em que a primeira linha cita os rótulos do eixo horizontal do gráfico e as linhas subsequentes indicam as complexidades para cada tarefa. Por exemplo, a tarefa *Task02* apresenta complexidades de execução, coordenação, itens de negócio e total, respectivamente: 2, 6, 58 e 66.

```
1 - Execution Coordination Business_Itens Total
2 Task01 2 3 20 25
3 Task02 2 6 58 66
```

Figura 4.10: Exemplo de arquivo de dados do Gnuplot

Ao utilizar como entrada no Gnuplot as configurações detalhadas na Figura 4.9 e os dados detalhados na Figura 4.10, é gerado histograma semelhante ao presente na Figura 4.11.

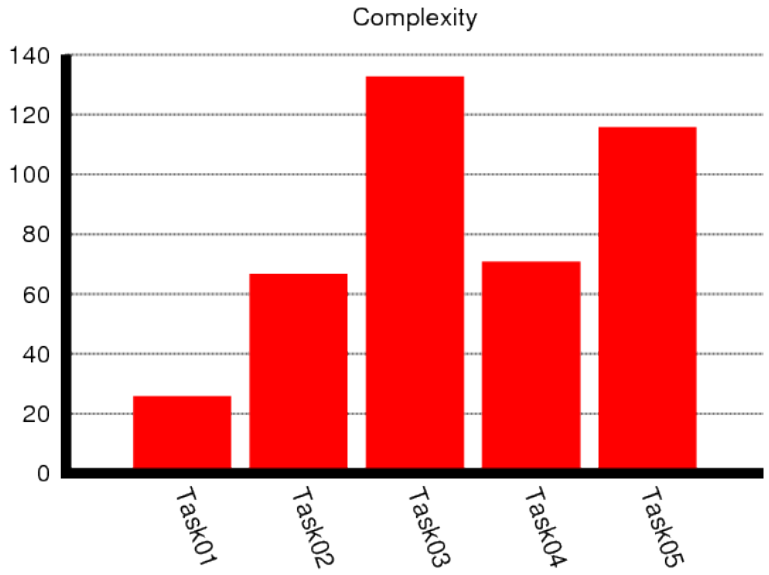

Figura 4.11: Exemplo de histograma desenhado pelo Gnuplot

4.3.5 Módulo *XML builder*

Semelhante ao *XML parser*, este módulo utiliza a biblioteca Java JDOM para gerar o arquivo XML de saída, contendo os atributos *execComplexity*, *coordComplexity* e *biComplexity* em cada *tag* <*task*> do processo de entrada e o atributo *IDX* para cada papel do processo a fim de possibilitar a aplicação da metodologia proposta. A Figura 4.12 exemplifica resumidamente trechos de código de criação das entidades dentro do arquivo XML e da gravação do arquivo de saída.

O código busca todos os elementos do processo a partir de *P* – instância da classe *Process* que representa a raiz do mesmo – e carrega os dados em um arquivo XML utilizando métodos e classes fornecidos pela biblioteca Java JDOM. Para isso, é instanciada a classe *Element* para cada entidade de processo detectada e que necessite de uma nova

```
1  Element process = new Element("process");
2  Document procDoc = new Document(process);
3  process.addContent(new Element("desc").setText(P.getDesc()));
4  ...
5  for (i = P.biList.iterator(); i.hasNext(); ) {
6          bi = (ProcessBI) i.next();
7          el1 = new Element("bi");
8          el1.setAttribute(new Attribute("name",bi.name()));
9          for (j = bi.fieldList.iterator(); j.hasNext(); ) {
10                 bif = (ProcessBIField) j.next();
11                 el2 = new Element("field");
12                 el2.setAttribute(new Attribute("name",bif.name()));
13                 el2.setAttribute(new Attribute("sourcescore",bif.sourceScore.getString()));
14                 el1.addContent(el2);
15         }
16         process.addContent(el1);
17 }
18 ...
19 try {
20         FileOutputStream sourceOut = new FileOutputStream(f);
21         XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
22         outputter.output(procDoc, sourceOut);
23 } catch (Exception e) {
24         e.printStackTrace();
25 }
```

Figura 4.12: Trecho de código Java do módulo *XML builder*

tag, criando os atributos necessários e encadeando a estrutura conforme padronizado no XML *Schema* definido para esta implementação.

Depois de equalizadas as estruturas armazenadas por *P* e pela nova estrutura XML, o arquivo XML de saída é gravado através do método *outputter.output(procDoc, sourceOut)*, utilizando a classe de sistema *FileOutputStream* e a classe JDOM *XMLOutputter*.

4.4 Ferramenta de visualização

Para permitir que os ciclos da metodologia proposta se completem, são previstas ferramentas de modelagem de processo e de visualização das complexidades calculadas para cada tarefa. O objetivo é fomentar as atividades de projetistas e analistas.

Com o intuito de avaliar a metodologia, a modelagem dos processos foi efetuada em XML, um formato que facilita a portabilidade para aplicações de modelagem de processos bem difundidas no mercado. Complementando, de maneira a concentrar os resultados e facilitar a sua visualização, sempre focando a avaliação da metodologia, foi desenvolvida uma interface Web para visualização dos resultados.

Esta interface, projetada de acordo com os ciclos da metodologia proposta, permite visualizar o processo e identificar os pontos de maior complexidade. A Figura 4.13 exemplifica uma tela da ferramenta confeccionada para visualização do processo desenhado pelo Graphviz. Pode-se observar os links na lateral, cujo conteúdo direciona a navegação para o diagrama do processo ou para histogramas de complexidade de execução, coordenação e itens de negócio, além da complexidade geral do processo. No centro da imagem, por sua vez, está presente uma partição do processo, que neste exemplo apresenta três entidades definidas:

NetworkValidation, decisão executada pelo papel *PhoneEngineering*, recebe e envia os itens de negócio *SoftswitchBrand*, *SoftswitchModel* e *Network*. Neste caso, a com-

plexidade da decisão foi quantificada em 115.

SoftswitchConfiguration, tarefa executada pelo papel *PhoneEngineering*, recebe de *NetworkValidation* os itens de negócio *SoftswitchBrand*, *SoftswitchModel* e *Network*, e envia os itens de negócio *Network* e *SoftswitchIpAddress*. Sua complexidade foi quantificada em 75.

RoutingConfiguration, tarefa executada pelos papéis *IspEngineering* e *IspSupport*, recebe de *SoftswitchConfiguration* os itens de negócio *Network* e *SoftswitchIpAddress* e os envia sem alteração à próxima entidade do fluxo do processo.

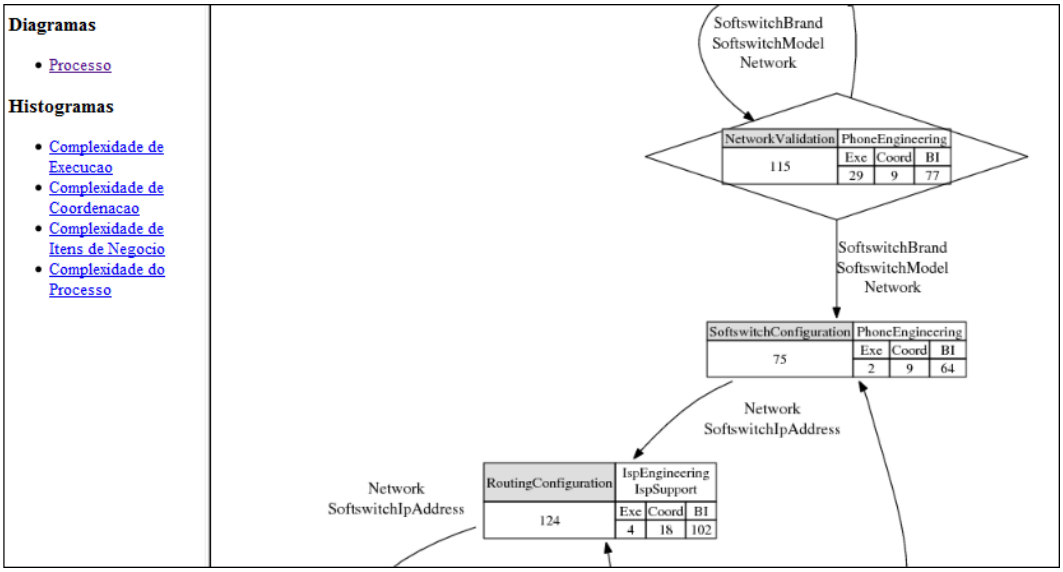

Figura 4.13: Tela de exibição de resultados, com diagrama dos processos

Representando dados do mesmo processo, a Figura 4.14 apresenta outra tela do sistema, esta confeccionada para visualização dos histogramas gerados para cada tipo de complexidade. Nela, percebe-se que a tarefa *VoiceTest* é a de maior esforço, com complexidade acima de 200, enquanto a tarefa *SystemOk* tem complexidade próxima de zero.

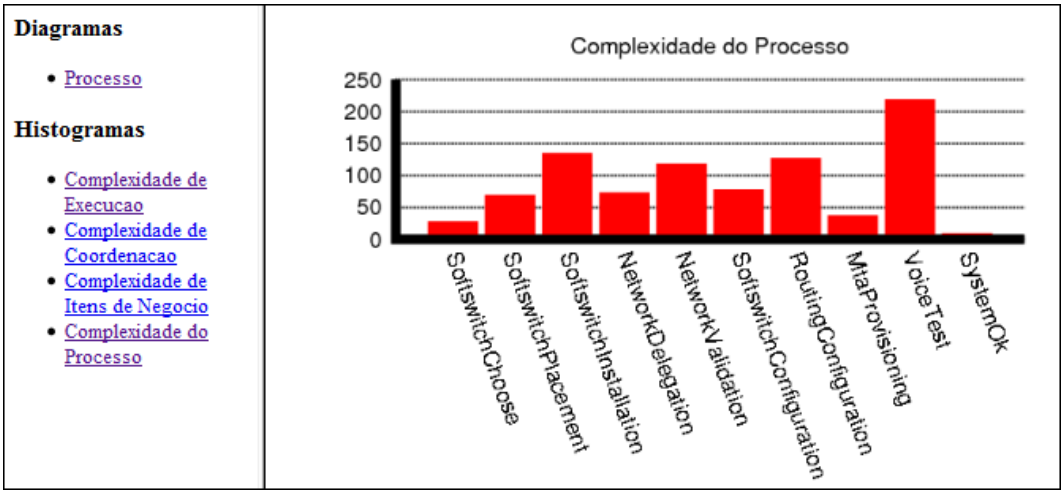

Figura 4.14: Tela de exibição de resultados, com histograma de complexidade

Esta ferramenta é suficiente para avaliar a complexidade de processos de TI interdomínios e exibir resultados compatíveis com a metodologia de quantificação de complexidade de processos de TI. Portanto, ao utilizar este protótipo, podemos avaliar a extensão de metodologia proposta, o que será demonstrado no próximo capítulo.

5 AVALIAÇÃO DA METODOLOGIA

O objetivo deste capítulo é avaliar a metodologia proposta no presente trabalho. Para isto, é avaliada a sua eficácia ao ser aplicada a um processo de TI modelado de um caso de uso real, baseado em um processo de suporte de um serviço de telefonia residencial interligado à rede pública de telefonia. O intuito é demonstrar ser factível a quantificação da complexidade quando um processo atravessa múltiplos domínios de autoridade.

É importante ressaltar que a base da metodologia proposta é um conjunto de métricas focadas na análise de complexidade, e que tais métricas são dependentes dos cenários aos quais são aplicadas e mutáveis conforme a evolução destes(ZUSE, 1999). Neste contexto, os resultados da aplicação da metodologia em um processo fiel ao cenário avaliado são suficientes para mostrar os benefícios da sua implementação, sempre em coerência com as avaliações efetuadas pelos autores em estudos similares aqui referenciados, e este será o cerne da avaliação descrita a seguir.

Inicialmente é detalhado o processo utilizado para avaliar o uso da metodologia estendida proposta. Prosseguindo, esta é aplicada ao processo, resultando nas diferentes versões de processos previstos (LPR, LPU e LP), seguidos das análises de complexidade e da compilação destas para cada domínio participante. Por fim, os resultados obtidos são comparados entre si e com o cenário real, onde todos conhecem todos os detalhes de todos os processos (LPR), mostrando o quanto o resultado obtido se aproxima deste cenário ideal e utópico.

5.1 Processo avaliado

O cenário avaliado – e cuja complexidade deve ser mensurada – exige a presença de diferentes domínios de autoridade em um único processo de TI. Para atender a esta premissa, foi escolhido um processo de suporte a um serviço de telefonia, cujas tarefas são executadas por três empresas distintas, partindo da rede pública de telefonia e de interconexões entre operadoras até o ponto de acesso instalado junto ao usuário final.

O processo é executado na prática e oriundo da experiência profissional do autor do presente trabalho, em uma das empresas envolvidas, porém mapeado com exclusividade para este estudo, sem utilizar os materiais disponíveis internamente nas organizações envolvidas. Portanto, o processo é real, porém o mapeamento foi efetuado novamente, o que visa resguardar ativos ou pessoas envolvidas no processo. O objetivo do serviço prestado pelas empresas envolvidas resumidamente consiste em prover telefonia eficiente e de baixo custo. Esta premissa fomentou a união das partes envolvidas, cada uma com atividades já estabelecidas e com forte *know-how* nas respectivas áreas de atuação.

O serviço utiliza uma rede metropolitana de dados, composta por fibras e cabos coaxiais (HFC, *Hybrid Fiber Coaxial*), uma operadora de telefonia com serviços de discagem

local, nacional e internacional, além de interoperabilidade completa nas redes públicas de telefonia (PSTN, *Public switched telephone network*), uma terceira instituição, contratada pelas duas anteriormente citadas para prestar serviços de atendimento e suporte de primeiro nível ao usuário final. Nesta breve descrição, são identificados os três domínios de autoridade atuantes no processo:

1. Um de gestão da rede de dados e última milha – chamado *last mile* (LM) –, e cujo patrimônio inclui uma rede HFC eficiente e com serviços de TV e de acesso à Internet já estabelecidos. Portanto, apesar de não possuir *know-how* em telefonia, pode absorver tal conhecimento, buscar autonomia e se conectar diretamente a redes de telefonia, sem precisar de terceiros.

2. Outro domínio de autoridade é responsável pela interconexão com redes de telefonia – chamado de *PSTN provider* (PP) –, e cujas atividades incluem uma grande e eficiente rede de interconexão telefônica. De forma análoga ao domínio LM, o PP também pode absorver conhecimentos complementares e buscar autonomia na última milha até o cliente. Assim como o domínio LM em relação ao PP, este também pode se tornar independente e concorrente direto no mesmo nicho de negócio.

3. O terceiro domínio – contratado pelos dois outros, o LM e o PP – é responsável pelo atendimento ao usuário, e, portanto chamado de *service desk* (SD). Este, no entanto, não prove exclusividade em seu serviço. Ou seja, pode também fornecer serviços para empresas potencialmente concorrentes de LM e de PP, assim instituindo conflitos de interesse relevantes.

Estes são os três domínios de autoridade envolvidos no processo. Conforme descrito acima, apesar do forte interesse de todos em preservar uma relação forte e benéfica, é real a presença de interesses conflitantes. Enquanto LM e PP podem buscar independência, cuja trajetória seria facilitada pelo conhecimento dos processos internos do concorrente, SD possuirá informações sensíveis de LM e PP, e sendo potencial fornecedor de outras empresas do mesmo ramo, o vazamento de tais dados pode caracterizar risco excessivo. Estes fatores esclarecem a motivação do presente trabalho e o forte vínculo desta com o exemplo avaliado.

A seguir são definidas as funções de cada domínio, as suas interfaces e as suas responsabilidades dentro dos fluxos do processo, além de outras premissas necessárias para aplicar a metodologia proposta.

5.2 Aplicação da metodologia

A metodologia se resume aos passos do algoritmo de troca de subprocessos, estes listados na Seção 3.3.3 e aplicados a seguir:

5.2.1 Passo (i): pré-requisitos para aplicação da metodologia

Tarefas de negociação são complexas, fora do escopo do presente documento e, portanto, consideradas bem resolvidas. Assim, neste passo da metodologia é pressuposto que todos os domínios interajam com eficiência e definam objetivos, interfaces de comunicação, funções e responsabilidades dentro do processo a ser executado. A seguir, é detalhado um possível resultado destas análises preliminares.

5.2.1.1 Responsabilidades, funções e interfaces

Os três domínios de autoridade responsáveis pela execução do processo são descritos abaixo, conforme necessidades de cada um:

***Service desk* (SD)** é o domínio de autoridade responsável pelo atendimento telefônico ao cliente, efetuando vendas e manutenções remotas de baixa complexidade. Utiliza para isto ferramentas fornecidas pelos outros domínios envolvidos e, quando necessário, direciona a análise para outros níveis de atendimento. Como é composto por uma empresa terceirizada, tem objetivos e estratégias próprios, mas que devem garantir o atendimento conforme SLA estabelecido em contrato.

***Last mile* (LM)** é o domínio responsável pela operação e pela gestão da rede de fibras e cabos coaxiais, meio de comunicação de última milha, ou seja, entregando a conexão de dados ao cliente. Neste processo, é responsável pela análise de segundo nível, incluindo instalações locais, rede metropolitana e central de rede. Não sendo capaz de resolver o problema, repassa a análise ao terceiro nível, o de provisionamento de conexão com a rede de telefonia.

***PSTN provider* (PP)** recebe o tráfego de telefonia e o encaminha às redes públicas, além de gerir o provisionamento dos números telefônicos. É o terceiro nível, encerrando a cadeia de análise. Este e o domínio LM exercem funções de telecomunicações, o que pode gerar conflitos entre suas estratégias. No entanto, para o bom resultado do processo, é necessário que as comunicações entre os dois fluam com qualidade.

As boas práticas recomendadas pelo ITIL definem um modelo para atendimento com múltiplos níveis, o que se enquadra neste cenário, e cujo desenho pode ser visto na Figura 5.1. Inicialmente, o domínio SD atende o cliente, registra a solicitação, filtra conforme o tipo e tenta resolver. Não sendo possível, repassa a necessidade ao segundo nível, o domínio LM. Este analisa seu ambiente e, caso não consiga atender à demanda, repassa para análise de conectividade e provisionamento do acesso à rede de telefonia, responsabilidade do domínio PP. Este realiza suas tarefas e, por fim, caso não consiga dar a solução, reinicia o ciclo do processo, voltando ao domínio LM.

O processo padronizado pelo ITIL possui alto nível de abstração, sendo possível aplicá-lo como está ou desmembrar cada tarefa ou nível em mais níveis e tarefas, detalhando conforme necessário. Logo, é possível a todos os domínios a omissão de informações estratégicas. O processo padrão também é publicamente conhecido e, portanto, amplamente acessível e capaz de endereçar o objetivo do processo avaliado.

Os detalhes de cada entidade deste processo são de responsabilidade de cada domínio de autoridade, que a partir das informações aqui detalhadas devem desenhar seus subprocessos e interagir com outros domínios parceiros até que todos obtenham uma visão adequada do processo completo.

5.2.2 Passo (ii): definição de subprocessos

Após definidas as premissas, é possível prosseguir na aplicação da metodologia proposta. Para isso, é necessário que projetistas de cada domínio transformem em um subprocesso a sua parte local do processo completo e iniciem os passos previstos na Seção 3.3.3 (Algoritmo de troca de subprocessos). Ao longo do processo, sempre ocorrem interações com outros projetistas de outros domínios, em busca de um encaixe de suas interfaces de comunicação. É importante ressaltar que divergências neste passo podem reconduzir ao

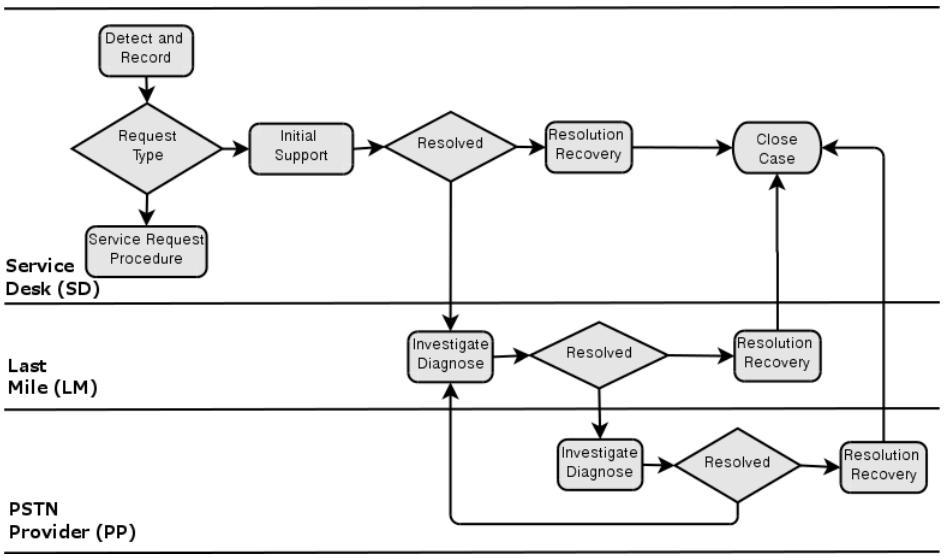

Figura 5.1: Suporte em múltiplos níveis (ITIL, 2009)

anterior para redefinição das premissas, o que caracteriza nova negociação entre domínios que compõem o processo.

Para definir os subprocessos, cada domínio de autoridade modela seu subprocesso LPR, contendo todos os dados necessários para a execução de sua parte do processo. A partir do LPR, cada domínio define um LPU para cada domínio parceiro, resguardando dados confidenciais. Com as duas versões do subprocesso local, LPR e LPU, é possível calcular um IDX para cada domínio parceiro utilizando a Equação 3.2. Apesar da possibilidade de cada domínio confeccionar um LPU e um IDX para cada parceiro – e da importância desta característica – o presente trabalho considera que cada domínio terá o mesmo LPU para todos os parceiros, ou seja, omitirá as mesmas informações não importando o destino. Isto reduz o conjunto de subprocessos analisado sem reduzir a riqueza da avaliação da metodologia.

Domínio SD

A Figura 5.2 representa o subprocesso executado pelo domínio SD, o LPR.

A partir do LPR, o domínio SD compõe um subprocesso simplificado, desejando não divulgar que internamente seu atendimento é composto por dois níveis. Este processo simplificado, o LPU, pode ser visto na Figura 5.3.

Domínio LM

A Figura 5.4 representa o subprocesso executado pelo domínio LM, o LPR.

Baseado no LPR, o domínio LM compõe um subprocesso simplificado, omitindo dos domínios parceiros a tarefa *CPE ok*, cujo objetivo é analisar o equipamento de conexão fornecido aos clientes (CPE). O processo simplificado, o LPU, pode ser visto na Figura 5.5.

Domínio PP

A Figura 5.6 representa o subprocesso executado pelo domínio PP, o LPR.

Com base no LPR, o domínio PP compõe um subprocesso simplificado, omitindo a tarefa *MTA ok*. Esta tarefa avalia um equipamento interno do CPE, o MTA, cuja função

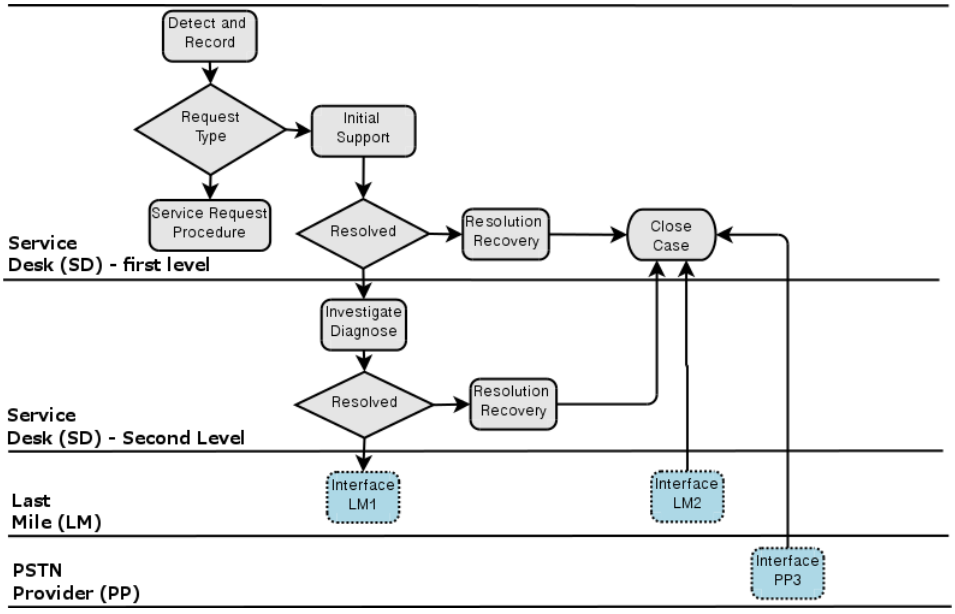

Figura 5.2: Subprocesso LPR do domínio SD

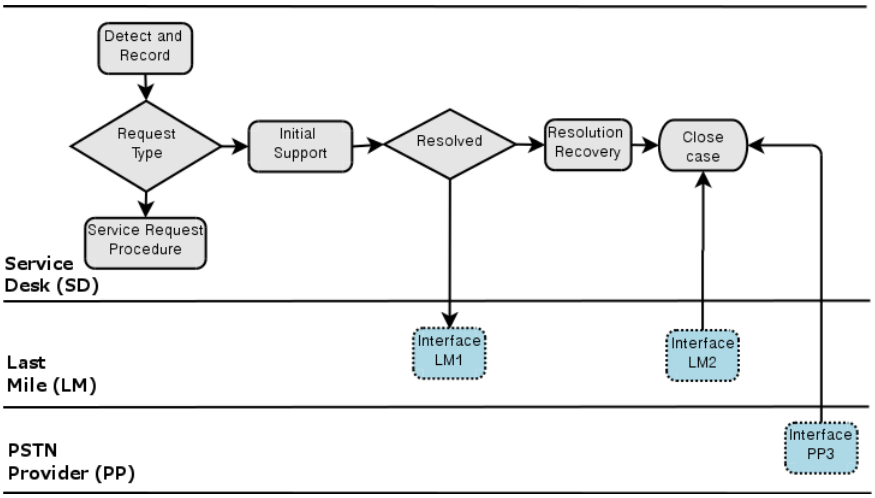

Figura 5.3: Subprocesso LPU do domínio SD

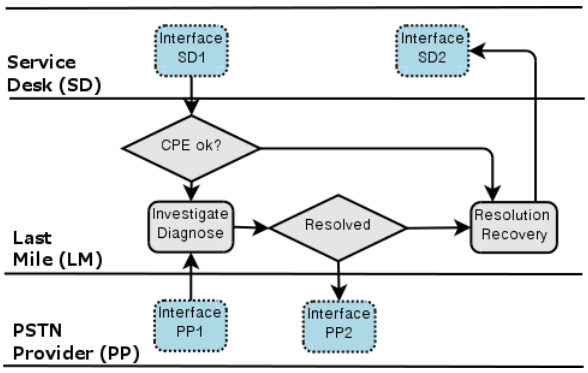

Figura 5.4: Subprocesso LPR do domínio LM

é conectar o telefone do cliente à rede pública de telefonia. O processo simplificado, o LPU, pode ser visto na Figura 5.7.

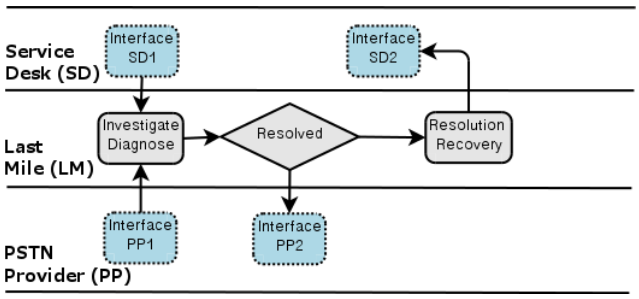

Figura 5.5: Subprocesso LPU do domínio LM

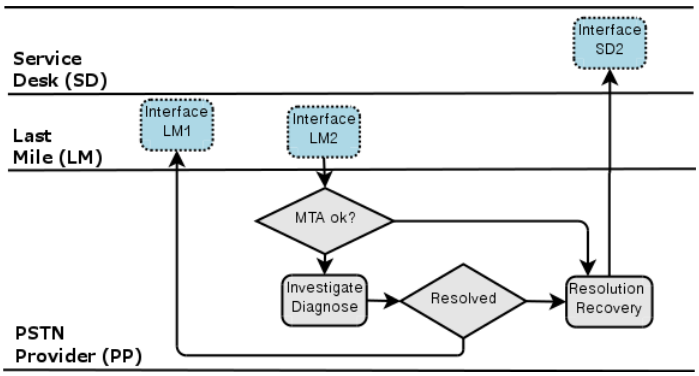

Figura 5.6: Subprocesso LPR do domínio PP

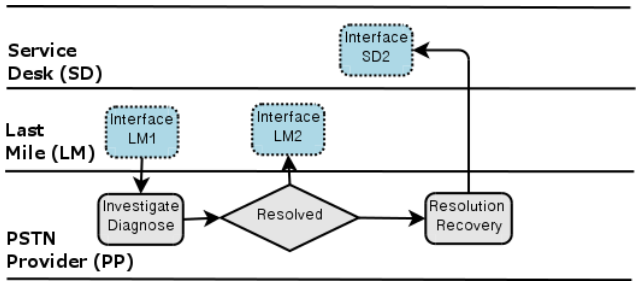

Figura 5.7: Subprocesso LPU do domínio PP

Cada domínio modela seus processos internos, e neles utiliza a metodologia de análise de complexidade para obter um processo eficiente. De posse dos valores numéricos de complexidade resultantes da aplicação do LPR e do LPU ao *complexity analyzer*, é possível calcular os valores de ajustes de complexidade IDX por meio da Equação 3.2. É importante ressaltar que, para um dominio local i, variando o *LPU* conforme o domínio d de destino, serão compostos tantos $LPU_{i,p}$ quantos forem os domínios parceiros p e, portanto, índices de complexidade $IDX_{i,p}$ em mesmo número. Assim, analisando os processos, como fariam os analistas de cada um dos três domínios, obtém-se os resultados dispostos na Tabela 5.2.3.

5.2.3 Passo (iii): complexidade do processo

Após definir os LPUs, cada domínio envia para cada parceiro sua versão referente, processo visto na Figura 3.2. Ao final, todos recebem um LPU de cada parceiro e, portanto, possuem uma versão de cada parte do processo completo.

Com todas as informações necessárias sobre todos os domínios de autoridade, cada

Tabela 5.1: Valores de complexidade e IDX calculados para os subprocessos LPU e LPR dos domínios SD, LM e PP

Domínio origem	Domínio destino	Complexidade LPR	Complexidade LPU	IDX
SD	LM	285,00	202,00	1,411
SD	PP	285,00	202,00	1,411
LM	SD	139,00	76,00	1,829
LM	PP	139,00	76,00	1,829
PP	SD	133,00	70,00	1,9
PP	LM	133,00	70,00	1,9

domínio pode compor sua visão própria do processo completo, o FP. Este, por sua vez, é composto pela união do LPR do domínio local com cada LPU recebido de cada domínio parceiro, o que dá o maior detalhamento na visão deste domínio. Ajustando o FP pelo IDX, obtém-se o FAP, que é estruturalmente igual ao FP, porém ponderando a complexidade de cada tarefa pelo IDX.

Por exemplo, para o domínio SD, o FP é formado pela união dos diagramas dispostos nas Figuras 5.2, 5.5 e 5.7, unindo o LPR local com os LPUs dos domínios LM e PP. O resultado é mostrado na Figura 5.8.

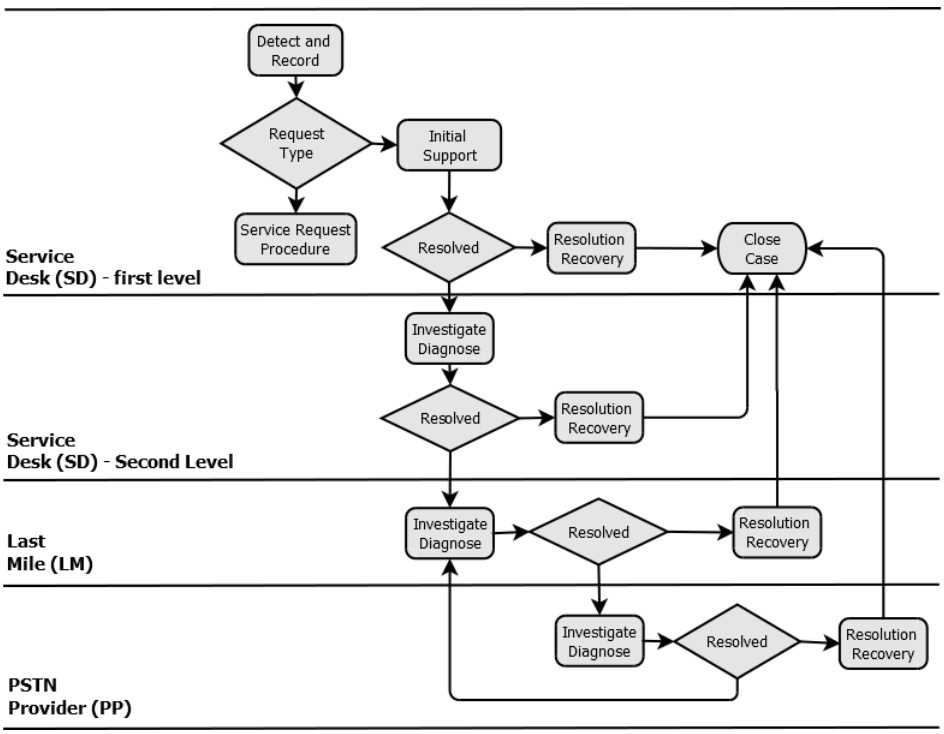

Figura 5.8: Processo FP do domínio SD

Depois de todos os domínios realizarem o mesmo procedimento – modelando subprocessos locais, enviando estes aos domínios parceiros e com eles compilando a visão local do processo completo – todos disporão do processo. Ao analisá-lo no *Complexity Analyzer*, todos os domínios conhecerão a complexidade deste e de suas tarefas, conforme resultados vistos na Tabela 5.2.2. Nela estão dispostas as complexidades calculadas para

os processos FP e FAP de cada domínio, evidenciando a diferença entre o processo sem ajuste e o processo de complexidade ajustada.

Tabela 5.2: Valores de complexidade para os domínios SD, LM e PP

Domínio	Complexidade do FP	Complexidade do FAP
SD	401	515,22
LM	381	513,38
PP	381	508,96

Este passo encerra a aplicação da metodologia proposta no presente documento. Logo, a complexidade do FAP é suficiente para representar a complexidade de forma coerente para todos os domínios. Além disso, serve como base para um futuro modelo de custos que converta valores em tempo ou em dinheiro, e fomente a gestão do negócio. No entanto, para concluir a avaliação da metodologia, é necessário comparar os valores, obtidos a partir do FAP – com o processo ideal, obtido a partir do FPR –, sem omissão de informações.

5.3 Avaliação dos resultados

Para avaliar os resultados, é necessário comparar a complexidade do processo FAP com a do processo FPR, presente na Figura 5.9. O último não omite qualquer informação, contém todos os dados sigilosos de cada domínio e, portanto, é a versão cuja complexidade calculada tende a ser mais próxima da realidade.

Na prática, o uso do FPR é improvável, pois dificilmente serão expostas todas as informações de todos os domínios. No entanto, como este exemplo foi simulado e inteiramente modelado pela mesma equipe de pesquisa, o FPR será concebido sob premissa de conhecer os LPRs de todos os domínios e, assim, será utilizado na avaliação da metodologia proposta. A Tabela 5.3 apresenta os resultados calculados para o FAP de cada domínio comparados com o FPR.

Tabela 5.3: Valores de complexidade – para os domínios SD, LM e PP – comparados com a complexidade do processo ideal (FPR)

Domínio	Complexidade do FAP	Complexidade do FPR	Diferença	Erro (%)
SD	515,22	527	11,78	2,23%
LM	513,38	527	13,62	2,58%
PP	508,96	527	18,04	3,42%

Enquanto o FPR, único para todos, obtém resultado 527, a diferença deste para o FAP dos domínios varia entre 11,78 e 18,04, com o erro variando entre 2,23% e 3,42%. Devido à formação da fórmula de ajuste baseada no IDX, pode-se concluir que a diferença entre FPR e FAP é diretamente proporcional ao seu valor médio para o domínio. Assim, os resultados da Tabela 5.3 mostram tendência de proporcionalidade entre IDX e o erro no cálculo de complexidade, o que pode consolidá-lo como uma nova métrica de confiabilidade entre domínios. Esta nova métrica permitiria a fiscalização dos resultados, uma vez

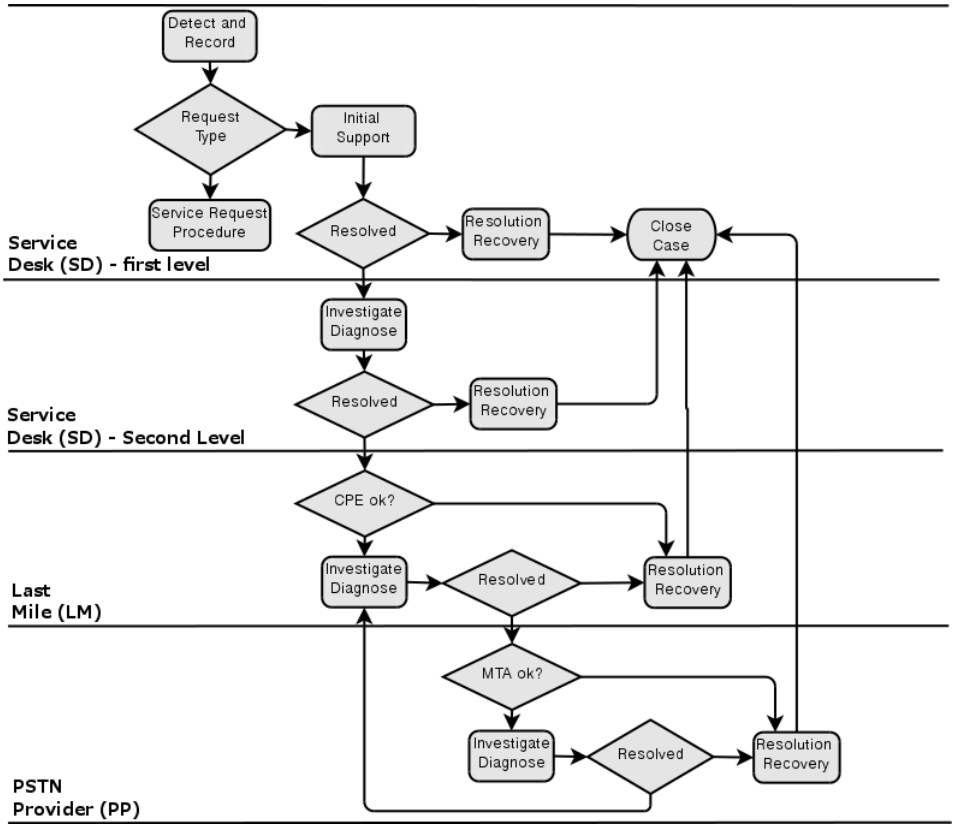

Figura 5.9: Processo FPR

que os domínios não poderão validá-los utilizando o FPR. Esta e outras considerações, além de conclusões e ideias de trabalhos futuros, serão apresentados no próximo capítulo.

6 CONTRIBUIÇÕES

Como contribuições, são descritas neste último capítulo as principais realizações e conclusões resultantes do presente estudo, bem como os trabalhos futuros identificados durante o seu desenvolvimento.

6.1 Realizações e conclusões

Nesta dissertação foi apresentada uma metodologia para a avaliação de processos de TI em casos em que seus fluxos envolvem múltiplos domínios de autoridade. No cenário analisado, os domínios são autônomos, com poder de decisão e estratégias próprias.

Com este intuito, foram definidas extensões de conceitos de processos e subprocessos, além equações complementares para cálculo da complexidade e um índice de ajuste dos resultados, além de métodos de aplicação destas equações e de troca de informações entre os domínios de autoridade. A metodologia foi estendida, implementada em um protótipo completamente desenvolvido para este estudo, e avaliada por meio de sua aplicação na análise de um processo de TI modelado de um caso real de negócio.

O processo avaliado foi mapeado com base em exaustivo trabalho de coleta de informações e de interação propiciada pelas empresas envolvidas no mesmo, sempre de forma autorizada por todas as partes.

Com base nos resultados, pode-se concluir que a extensão proposta é suficiente para alcançar os objetivos do presente trabalho. A partir dos conceitos aqui definidos, é possível fornecer a todos os domínios envolvidos uma visão melhorada do processo completo e de sua complexidade, mensurando-a de forma adequada mesmo quando domínios omitirem informações confidenciais entre si em níveis diferentes.

Por se tratar de um campo de pesquisa em aberto, complexo e com grande número de variáveis, durante este trabalho inúmeros pontos pendentes foram identificados, listados e contornados ou aceitos como premissas para resolução futura. Entre eles, a necessidade de formalizar atividades de negociação, a possibilidade e necessidade de inserir métricas de confiabilidade das informações compartilhadas, bem como o desenvolvimento de interfaces interativas que possibilitem uso prático e ágil, entre outros. Assim, complementando a proposta, os trabalhos futuros aqui listados tratam problemas ainda não resolvidos e essenciais para que o resultado alcançado o agregue valor esperado.

6.2 Trabalhos Futuros

Durante todas as etapas do presente estudo, foram enumerados os trabalhos futuros relacionados a seguir.

1. Para aplicar os passos de cálculo, é necessário que todos os domínios atendam premissas do algoritmo, conforme definido no Capítulo 3. Estas são a base para que cada domínio participante modele seus subprocessos locais e conecte suas tarefas às dos domínios parceiros com harmonia, dentro do objetivo dos fluxos projetados. Como estes passos preliminares são complexos e cruciais, já tratados com mais profundidade (DUNNE; WOOLDRIDGE; LAURENCE, 2005) (DUNG; THANG; TONI, 2008) (GRIFFEL et al., 1997), tarefas de negociação são consideradas bem resolvidas e, portanto, fora do escopo do presente estudo. Contudo, a formalização destas negociações constitui importante trabalho futuro.

2. Os ciclos necessários para apresentação e avaliação da metodologia são baseados no uso do protótipo *Complexity Analyzer*. Este recebe e emite processos em formato XML, além de gerar histogramas e diagramas para auxílio de analistas, projetistas e executores. No entanto, a aplicação na prática exige ferramentas interativas de modelagem dos processos, ou integração com ferramentas já existentes para este fim.

3. O protótipo implementado, quando associado à metodologia, fornece os resultados esperados. No entanto, como este quantifica a complexidade dos processos sem unidade de medida, para a aplicação na gestão de negócios é necessário definir um modelo de custo capaz de converter os resultados em unidades que o negocio entenda, como custo em tempo ou dinheiro. Processo similar já foi realizado para procedimentos, conforme descrito no Capítulo 2.

4. Como domínios podem adotar comportamentos maliciosos ou viciados, manipulando dados e metodologia, é necessário um mecanismo de fiscalização das informações recebidas. Neste contexto, tanto o subprocesso LPU quanto o respectivo IDX devem ser conferidos, e para isto é necessário desenvolver dispositivos que possibilitem medir a confiabilidade de forma eficiente. Um deles pode ser o uso dos IDX recebidos como métrica de confiabilidade, já que apresentam valor proporcional à omissão de informações no exemplo real avaliado, o que pode compor uma nova métrica de confiabilidade ao modelo. Esta e outras possíveis soluções devem ser estudadas futuramente.

Por envolver processos automatizados ou manuais, além de relações humanas e de cooperação entre instituições distintas e autônomas, é elevado o número de variáveis, o que torna o problema complexo e a solução dependente do cenário avaliados. Esta área de estudo apresenta avanços significativos e grandes desafios relacionados a análise de complexidade, métricas, indicadores, relações entre federações, validações estatísticas e desenvolvimento de ferramentas de apoio. Com base nos resultados anteriores e nos aqui apresentados, o presente trabalho é concluído com a certeza de mais um passo dado no sentido de mensurar a complexidade dos processos de TI. Além de agregar valor ao conhecer e controlar a complexidade de processos de TI, sejam interdomínios ou referentes a outro cenário específico, permite que soluções possam ser baseadas ou customizadas conforme o presente trabalho.

REFERÊNCIAS

A Simple Way to Estimate the Cost of Downtime. In: USENIX CONFERENCE ON SYSTEM ADMINISTRATION, 16., 2002, Berkeley, CA, USA. **Proceedings...** USENIX Association, 2002. p.185–188. (LISA '02).

ANATEL. **Regulamento Geral de Portabilidade**. Disponível em: <http://www.anatel.gov.br/Portal/verificaDocumentos/documento.asp?numeroPublicacao=125293&assuntoPublicacao=Apresenta%E7%E3o%20sobre%20Portabilidade%20Num%E9rica&caminhoRel=null&filtro=1&documentoPath=acontece_anatel/palestras/portabilidade_numerica.pdf>. Acesso em: 04 jun., 2012.

BRASIL. **Decreto n. 4.733, de 10 de junho de 2003.** Dispõe sobre políticas públicas de telecomunicações e dá outras providências. Brasília, 2003. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto/2003/d4733.htm>. Acesso em: 04 jun., 2012.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; EVE; YERGEAU, F. (Ed.). **Book title: Extensible Markup Language (XML) 1.0**. 4.ed. [S.l.]: W3C, 2003. (W3C Recommendation).

BROWN, A. B.; HELLERSTEIN, J. L. An approach to benchmarking configuration complexity. In: ACM SIGOPS EUROPEAN WORKSHOP, 11., 2004, New York, NY, USA. **Proceedings...** ACM, 2004. (EW 11).

CAI, W.; TURNER, S.; GAN, B. P. Hierarchical federations: an architecture for information hiding. In: PARALLEL AND DISTRIBUTED SIMULATION, 2001. PROCEEDINGS. 15TH WORKSHIP ON, 2001. **Anais...** [S.l.: s.n.], 2001. p.67 –74.

CHEN, D.; TURNER, S. J.; CAI, W.; THEODOROPOULOS, G. K.; XIONG, M.; LEES, M. Synchronization in federation community networks. **J. Parallel Distrib. Comput.**, Orlando, FL, USA, v.70, n.2, p.144–159, Feb. 2010.

COSTA CORDEIRO, W. Luis da; MACHADO, G.; DAITX, F.; BOTH, C.; PASCHOAL GASPARY, L.; GRANVILLE, L.; SAHAI, A.; BARTOLINI, C.; TRASTOUR, D.; SAIKOSKI, K. A template-based solution to support knowledge reuse in IT change design. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2008. NOMS 2008. IEEE, 2008. **Anais...** [S.l.: s.n.], 2008. p.355 –362.

COUCH, A. L.; WU, N.; SUSANTO, H. Toward a cost model for system administration. In: LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE - VOLUME 19, 19., 2005, Berkeley, CA, USA. **Proceedings...** USENIX Association, 2005. p.13–13. (LISA '05).

DIAO, Y.; KELLER, A. Quantifying the Complexity of IT Service Management Processes. In: STATE, R.; MEER, S. van der; O?SULLIVAN, D.; PFEIFER, T. (Ed.). **Large Scale Management of Distributed Systems**. [S.l.]: Springer Berlin / Heidelberg, 2006. p.61–73. (Lecture Notes in Computer Science, v.4269). 10.1007/11907466_6.

DUNG, P. M.; THANG, P. M.; TONI, F. Towards argumentation-based contract negotiation. In: COMPUTATIONAL MODELS OF ARGUMENT: PROCEEDINGS OF COMMA 2008, 2008., 2008, Amsterdam, The Netherlands, The Netherlands. **Proceedings...** IOS Press, 2008. p.134–146.

DUNNE, P. E.; WOOLDRIDGE, M.; LAURENCE, M. The complexity of contract negotiation. **Artif. Intell.**, Essex, UK, v.164, n.1-2, p.23–46, May 2005.

ELLSON, J.; GANSNER, E.; KOUTSOFIOS, L.; NORTH, S.; WOODHULL, G. Graphviz Open Source Graph Drawing Tools. In: MUTZEL, P.; JüNGER, M.; LEIPERT, S. (Ed.). **Graph Drawing**. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2002. p.594–597. (Lecture Notes in Computer Science, v.2265).

GRIFFEL, F.; TU, M.; MUNKE, M.; MERZ, M.; LAMERSDORF, W.; SILVA, M. da. Electronic contract negotiation as an application niche for mobile agents. In: ENTERPRISE DISTRIBUTED OBJECT COMPUTING WORKSHOP [1997]. EDOC '97. PROCEEDINGS. FIRST INTERNATIONAL, 1997. **Anais...** [S.l.: s.n.], 1997. p.354 –365.

HAGEN, S.; KEMPER, A. Facing the unpredictable: automated adaption of it change plans for unpredictable management domains. In: NETWORK AND SERVICE MANAGEMENT (CNSM), 2010 INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.33 –40.

HAROLD, E. R.; MEANS, W. S. **Book title: XML in a Nutshell**. 3rd.ed. Sebastopol, CA: O'Reilly, 2004.

IBM. **IBM Websphere Business Modeler**. Disponível em: <http://www-306.ibm.com/software/integration/wbimodeler>. Acesso em: dezembro 2011.

ISACA. **Control Objectives for Information and related Technology.** USA, 2005.

ITIL. **Information Technology Infrastructure Library.** Disponível em <http://www.itil-officialsite.com/>. Acesso em: 05 mai. 2011.

KELLER, A.; BROWN, A.; HELLERSTEIN, J. A configuration complexity model and its application to a change management system. **Network and Service Management, IEEE Transactions on**, [S.l.], v.4, n.1, p.13 –27, june 2007.

KELLER, A.; HELLERSTEIN, J.; WOLF, J.; WU, K.-L.; KRISHNAN, V. The CHAMPS system: change management with planning and scheduling. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2004. NOMS 2004. IEEE/IFIP, 2004. **Anais...** [S.l.: s.n.], 2004. v.1, p.395 –408 Vol.1.

MCLAUGHLIN, B. **Book title: Java and XML. Section: JDOM**. [S.l.]: O'Reilly/VVA, 2001.

QUINT, A. SVG: scalable vector graphics. **IEEE Multimedia**, [S.l.], v.10, n.3, p.99–102, July 2003.

RACINE, J. gnuplot 4.0: a portable interactive plotting utility. **Journal of Applied Econometrics**, Department of Economics, McMaster University, Hamilton, Ontario, Canada, v.21, n.1, p.133–141, 2006.

SANTOS, R. dos; WICKBOLDT, J.; LUNARDI, R.; DALMAZO, B.; GRANVILLE, L.; GASPARY, L.; BARTOLINI, C.; HICKEY, M. A solution for identifying the root cause of problems in IT change management. In: INTEGRATED NETWORK MANAGEMENT (IM), 2011 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON, 2011. **Anais...** [S.l.: s.n.], 2011. p.586 –593.

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. **EMF: eclipse modeling framework (2nd edition)**. 2.ed. [S.l.]: Addison-Wesley Professional, 2008.

WHITE, S. **Business Process Modeling Notation (BPMN) Version 1.0**. [S.l.]: BPMI.org, 2004.

ZUSE, H. Validation of Measures and Prediction Model. In: INTERNATIONAL WORKSHOP ON SOFTWARE MEASUREMENT, 9., 1999, Lac-Supérieur, Canada. **Anais...** [S.l.: s.n.], 1999.

ANEXO I - CÓDIGO FONTE DO PROTÓTIPO

Este anexo descreve na integra o código fonte do protótipo implementado.

```
1  import java.io.*;
2  import java.util.*;
3  import org.jdom.*;
4  import org.jdom.input.*;
5  import org.jdom.output.*;
6  import org.jdom.output.Format.TextMode.*;
7
8  class ProcessStats {
9   Process process;
10  public ProcessStats(Process p){
11   this.process = p;
12  }
13
14  public void saveToFile(File f){
15
16   Iterator i;
17   ProcessTask t;
18
19   try {
20
21    FileWriter fstream = new FileWriter(f);
22    BufferedWriter out = new BufferedWriter(fstream);
23
24    out.write("- Execution Coordination Business_Itens Total\n");
25
26    for (i = this.process.taskList.iterator(); i.hasNext(); ) {
27     t = (ProcessTask) i.next();
28
29     int comp_exec = t.execComplexity();
30     int comp_coord = t.coordComplexity();
31     int comp_bi = t.biComplexity();
32
33     out.write(t.name());
34     out.write(" "+comp_exec);
35     out.write(" "+comp_coord);
36     out.write(" "+comp_bi);
37     out.write(" "+(comp_exec+comp_coord+comp_bi)+"\n");
38
39    }
40
41    out.flush();
42
43   } catch(Exception e) {
44    e.printStackTrace();
45   }
46
47  }
48 }
49
50 class ProcessGraph {
51  Process process;
52  public ProcessGraph(Process p){
53   this.process = p;
54  }
55
56  public void saveToFile(File f){
57
58   Iterator i, j, k, l, m;
59   ProcessRole p, task_p;
60   ProcessTask t;
61   ProcessTaskRole tp;
62   ProcessBranch b;
63   ProcessBI bi;
64
65   try {
66
67    FileWriter fstream = new FileWriter(f);
68    BufferedWriter out = new BufferedWriter(fstream);
69
70    out.write("digraph G {\n");
71    out.write(" color=\"lightgrey\";\n");
72    out.write(" rankdir=\"TD\";\n");
73    out.write(" nodesep=\"2\";\n");
```

```
74
75      for (i = this.process.taskList.iterator(); i.hasNext(); ) {
76       t = (ProcessTask) i.next();
77       if (t.branchOutList.size() > 1){
78        out.write(" "+t.name()+" [shape=diamond,label=<<TABLE BORDER=\"0\" CELLBORDER=\"1\" CELLSPACING=\"0\" CELLPADDING
              =\"1\"><TR><TD BGCOLOR=\"#dedede\">"+t.name()+"</TD>");
79       } else {
80        if ( (t.branchOutList.size()>0 && t.branchInList.size()==0) ){
81         out.write(" "+t.name()+" [shape=plaintext,label=<<TABLE BORDER=\"2\" CELLBORDER=\"1\" CELLSPACING=\"0\"
                CELLPADDING=\"1\"><TR><TD BGCOLOR=\"#dedede\">"+t.name()+"</TD>");
82        } else {
83         out.write(" "+t.name()+" [shape=plaintext,label=<<TABLE BORDER=\"0\" CELLBORDER=\"1\" CELLSPACING=\"0\"
                CELLPADDING=\"1\"><TR><TD BGCOLOR=\"#dedede\">"+t.name()+"</TD>");
84        }
85       }
86       out.write("<TD COLSPAN=\"3\">");
87       for (j = t.roleList.iterator(); j.hasNext(); ) {
88        tp = (ProcessTaskRole) j.next();
89        out.write(tp.role().name());
90        if (j.hasNext()){ out.write("<BR/>"); }
91       }
92       int execComp = t.execComplexity();
93       int coordComp = t.coordComplexity();
94       int biComp = t.biComplexity();
95       out.write("</TD></TR><TR><TD ROWSPAN=\"2\">"+(execComp+coordComp+biComp)+"</TD><TD>Exe</TD><TD>Coord</TD><TD>BI</
              TD></TR>");
96       out.write("<TR><TD>"+execComp+"</TD><TD>"+coordComp+"</TD><TD>"+biComp+"</TD></TR>\n");
97       out.write("</TABLE>>];\n");
98
99
100     }
101
102     for (i = this.process.taskList.iterator(); i.hasNext(); ) {
103      t = (ProcessTask) i.next();
104      for (j = t.branchOutList.iterator(); j.hasNext(); ) {
105       b = (ProcessBranch) j.next();
106       if (b.biList.size()>0){
107        out.write(" "+t.name()+" -> "+b.name()+"[label=<<TABLE BORDER=\"0\" CELLBORDER=\"0\" CELLSPACING=\"0\"
                CELLPADDING=\"1\"><TR><TD ALIGN=\"left\"><FONT POINT-SIZE=\"16\">");
108       for (k = b.biList.iterator(); k.hasNext(); ) {
109        bi = (ProcessBI) k.next();
110        out.write(bi.name());
111        if (k.hasNext()){ out.write("<BR/>"); }
112       }
113       out.write("</FONT></TD></TR></TABLE>>];\n");
114      } else {
115       out.write(" "+t.name()+" -> "+b.name()+";\n");
116      }
117     }
118    }
119
120    out.write("}\n");
121    out.flush();
122
123   } catch(Exception e) {
124    e.printStackTrace();
125   }
126  }
127 }
128
129 class ProcessTaskList extends ArrayList<ProcessTask> {
130  public ProcessTask find(String name){
131   ProcessTask o;
132   Iterator i = this.iterator();
133   //System.out.println("xml_load: ProcessTaskList.find(), buscando "+name);
134   while (i.hasNext()){
135    o = (ProcessTask) i.next();
136    if (name.equals(o.name())){
137     //System.out.println("xml_load: ProcessTaskList.find(), achado "+o.name());
138     return o;
139    }
140   }
141   //System.out.println("ERROR: task "+name+" nao existe!");
142   return null;
143  }
144 }
145
146 class ProcessBIFieldList extends ArrayList<ProcessBIField> {
147  public ProcessBIField find(String name){
148   ProcessBIField o;
149   Iterator i = this.iterator();
150   //System.out.println("xml_load: ProcessBIList.find(), buscando "+name);
151   while (i.hasNext()){
152    o = (ProcessBIField) i.next();
153    if (name.equals(o.name())){
154     //System.out.println("xml_load: ProcessBIList.find(), achado "+o.name());
155     return o;
156    }
157   }
158   //System.out.println("ERROR: business item "+name+" nao existe!");
159   return null;
160  }
161 }
162
163 class ProcessBIList extends ArrayList<ProcessBI> {
164  public ProcessBI find(String name){
165   ProcessBI o;
166   Iterator i = this.iterator();
167   //System.out.println("xml_load: ProcessBIList.find(), buscando "+name);
```

```
168     while (i.hasNext()){
169      o = (ProcessBI) i.next();
170      if (name.equals(o.name())){
171       //System.out.println("xml_load: ProcessBIList.find(), achado "+o.name());
172       return o;
173      }
174     }
175     //System.out.println("ERROR: business item "+name+" nao existe!");
176     return null;
177    }
178   }
179
180   class ProcessRoleList extends ArrayList<ProcessRole> {
181    public ProcessRole find(String name){
182     ProcessRole o;
183     Iterator i = this.iterator();
184     //System.out.println("xml_load: ProcessRoleList.find(), buscando "+name);
185     while (i.hasNext()){
186      o = (ProcessRole) i.next();
187      if (name.equals(o.name())){
188       //System.out.println("xml_load: ProcessRoleList.find(), achado "+o.name());
189       return o;
190      }
191     }
192     //System.out.println("ERROR: role "+name+" nao existe!");
193     return null;
194    }
195   }
196
197   class ProcessTaskRoleList extends ArrayList<ProcessTaskRole> {
198    public ProcessTaskRole find(String name){
199     ProcessTaskRole o;
200     Iterator i = this.iterator();
201     //System.out.println("xml_load: ProcessTaskRoleList.find(), buscando "+name);
202     while (i.hasNext()){
203      o = (ProcessTaskRole) i.next();
204      if (name.equals(o.role().name())){
205       //System.out.println("xml_load: ProcessTaskRoleList.find(), achado "+o.name());
206       return o;
207      }
208     }
209     //System.out.println("ERROR: task-role "+name+" nao existe!");
210     return null;
211    }
212   }
213
214   class ProcessBranchList extends ArrayList<ProcessBranch> {
215    public ProcessBranch find(String name){
216     ProcessBranch o;
217     Iterator i = this.iterator();
218     //System.out.println("xml_load: ProcessBranchList.find(), buscando "+name);
219     while (i.hasNext()){
220      o = (ProcessBranch) i.next();
221      if (name.equals(o.getTask().name())){
222       //System.out.println("xml_load: ProcessBranchList.find(), achado "+o.getTask().name());
223       return o;
224      }
225     }
226     //System.out.println("ERROR: branch "+name+" nao existe!");
227     return null;
228    }
229   }
230
231   class MetricExecType {
232    private int value;
233    public static final int AUTOMATIC = 0;
234    public static final int TOOL_ASSISTED = 1;
235    public static final int MANUAL = 2;
236    public String toText(int n){
237     if (n==AUTOMATIC){ return "AUTOMATIC"; }
238     if (n==TOOL_ASSISTED){ return "TOOL_ASSISTED"; }
239     if (n==MANUAL){ return "MANUAL"; }
240     return "UNKNOWN";
241    }
242
243    public int fromText(String n){
244     if (n.equals("AUTOMATIC")){ return AUTOMATIC; }
245     if (n.equals("TOOL_ASSISTED")){ return TOOL_ASSISTED; }
246     if (n.equals("MANUAL")){ return MANUAL; }
247     return 0;
248    }
249
250    public void setValue(int val){
251     if (val==AUTOMATIC||val==TOOL_ASSISTED||val==MANUAL){
252      this.value = val;
253     }
254    }
255
256    public int getInt(){ return this.value; }
257
258    public String getString(){ return toText(this.value); }
259
260    public void setValue(String str){
261     if (str!=null){ int val = fromText(str);
262      if (val==AUTOMATIC||val==TOOL_ASSISTED||val==MANUAL){
263       this.value = val;
264      }
265     }
266    }
```

```
267
268   public MetricExecType(){
269     this.value = MANUAL;
270   }
271  }
272
273  class MetricGFactor {
274   private int value;
275   public static final int RECOMENDATION = 1;
276   public static final int INFORMATION = 2;
277   public static final int INTERNAL = 3;
278   public static String toText(int n){
279    if (n==RECOMENDATION){ return "RECOMENDATION"; }
280    if (n==INFORMATION){ return "INFORMATION"; }
281    if (n==INTERNAL){ return "INTERNAL"; }
282    return "UNKNOWN";
283   }
284
285   public static int fromText(String n){
286    if (n.equals("RECOMENDATION")){ return RECOMENDATION; }
287    if (n.equals("INFORMATION")){ return INFORMATION; }
288    if (n.equals("INTERNAL")){ return INTERNAL; }
289    return 0;
290   }
291
292   public void setValue(int val){
293    if (val==RECOMENDATION||val==INFORMATION||val==INTERNAL){
294     this.value = val;
295    }
296   }
297
298   public int getInt(){ return this.value; }
299
300   public String getString(){ return toText(this.value); }
301
302   public void setValue(String str){
303    if (str!=null){
304     int val = fromText(str);
305     if (val==RECOMENDATION||val==INFORMATION||val==INTERNAL){
306      this.value = val;
307     }
308    }
309   }
310
311   public MetricGFactor(){
312    this.value = INTERNAL;
313   }
314
315  }
316
317  class MetricCFactor {
318   private int value;
319   public static final int NEGLIGIBLE = 1;
320   public static final int MODERATE = 2;
321   public static final int SEVERE = 3;
322   public static String toText(int n){
323    if (n==MODERATE){ return "MODERATE"; }
324    if (n==NEGLIGIBLE){ return "NEGLIGIBLE"; }
325    if (n==SEVERE){ return "SEVERE"; }
326    return "UNKNOWN";
327   }
328
329   public static int fromText(String n){
330    if (n.equals("NEGLIGIBLE")){ return NEGLIGIBLE; }
331    if (n.equals("MODERATE")){ return MODERATE; }
332    if (n.equals("SEVERE")){ return SEVERE; }
333    return 0;
334   }
335
336   public void setValue(int val){
337    if (val==NEGLIGIBLE||val==MODERATE||val==SEVERE){
338     this.value = val;
339    }
340   }
341
342   public int getInt(){ return this.value; }
343
344   public String getString(){ return toText(this.value); }
345
346   public void setValue(String str){
347    if (str!=null){ int val = fromText(str);
348     if (val==NEGLIGIBLE||val==MODERATE||val==SEVERE){
349      this.value = val;
350     }
351    }
352   }
353
354   public MetricCFactor(){
355    this.value = SEVERE;
356   }
357
358  }
359
360  class MetricVFactor {
361   private int value;
362   public static final int IMEDIATE = 1;
363   public static final int SHORT_TERM = 2;
364   public static final int LONG_TERM = 3;
365   public static String toText(int n){
```

```
366     if (n==IMEDIATE){ return "IMEDIATE"; }
367     if (n==SHORT_TERM){ return "SHORT_TERM"; }
368     if (n==LONG_TERM){ return "LONG_TERM"; }
369     return "UNKNOWN";
370   }
371
372   public static int fromText(String n){
373     if (n.equals("IMEDIATE")){ return IMEDIATE; }
374     if (n.equals("SHORT_TERM")){ return SHORT_TERM; }
375     if (n.equals("LONG_TERM")){ return LONG_TERM; }
376     return 0;
377   }
378
379   public void setValue(int val){
380     if (val==IMEDIATE||val==SHORT_TERM||val==LONG_TERM){
381       this.value = val;
382     }
383   }
384
385   public int getInt(){ return this.value; }
386
387   public String getString(){ return toText(this.value); }
388
389   public void setValue(String str){
390     if (str!=null){ int val = fromText(str);
391       if (val==IMEDIATE||val==SHORT_TERM||val==LONG_TERM){
392         this.value = val;
393       }
394     }
395   }
396
397   public MetricVFactor(){
398     this.value = LONG_TERM;
399   }
400
401 }
402
403 class MetricSourceScore {
404   private int value;
405   public static final int INTERNAL = 0;
406   public static final int FREE_CHOICE = 1;
407   public static final int DOCUMENTATION_DIRECT = 2;
408   public static final int DOCUMENTATION_ADAPTED = 3;
409   public static final int BEST_PRATICE = 4;
410   public static final int ENVIRONMENT_FIXED = 5;
411   public static final int ENVIRONMENT_CONSTRAINED = 6;
412   public static String toText(int n){
413     if (n==INTERNAL){ return "INTERNAL"; }
414     if (n==FREE_CHOICE){ return "FREE_CHOICE"; }
415     if (n==DOCUMENTATION_DIRECT){ return "DOCUMENTATION_DIRECT"; }
416     if (n==DOCUMENTATION_ADAPTED){ return "DOCUMENTATION_ADAPTED"; }
417     if (n==BEST_PRATICE){ return "BEST_PRATICE"; }
418     if (n==ENVIRONMENT_FIXED){ return "ENVIRONMENT_FIXED"; }
419     if (n==ENVIRONMENT_CONSTRAINED){ return "ENVIRONMENT_CONSTRAINED"; }
420     return "UNKNOWN";
421   }
422
423   public static int fromText(String n){
424     if (n.equals("INTERNAL")){ return INTERNAL; }
425     if (n.equals("FREE_CHOICE")){ return FREE_CHOICE; }
426     if (n.equals("DOCUMENTATION_DIRECT")){ return DOCUMENTATION_DIRECT; }
427     if (n.equals("DOCUMENTATION_ADAPTED")){ return DOCUMENTATION_ADAPTED; }
428     if (n.equals("BEST_PRATICE")){ return BEST_PRATICE; }
429     if (n.equals("ENVIRONMENT_FIXED")){ return ENVIRONMENT_FIXED; }
430     if (n.equals("ENVIRONMENT_CONSTRAINED")){ return ENVIRONMENT_CONSTRAINED; }
431     return 0;
432   }
433
434   public void setValue(int val){
435     if (val==INTERNAL||val==FREE_CHOICE||val==DOCUMENTATION_DIRECT||val==DOCUMENTATION_ADAPTED||val==BEST_PRATICE||val==
            ENVIRONMENT_FIXED||val==ENVIRONMENT_CONSTRAINED){
436       this.value = val;
437     }
438   }
439
440   public int getInt(){ return this.value; }
441
442   public String getString(){ return toText(this.value); }
443
444   public void setValue(String str){
445     if (str!=null){ int val = fromText(str);
446       if (val==INTERNAL||val==FREE_CHOICE||val==DOCUMENTATION_DIRECT||val==DOCUMENTATION_ADAPTED||val==BEST_PRATICE||val
              ==ENVIRONMENT_FIXED||val==ENVIRONMENT_CONSTRAINED){
447         this.value = val;
448       }
449     }
450   }
451
452   public MetricSourceScore(){
453     this.value = ENVIRONMENT_CONSTRAINED;
454   }
455
456 }
457
458 class MetricLinkType {
459   private int value;
460   public static final int AUTO_LINK = 0;
461   public static final int CONTROL_LINK = 1;
462   public static final int DATA_TRANSFERED = 2;
```

```
463   public static final int DATA_ADAPTED = 3;
464   public static String toText(int n){
465    if (n==AUTO_LINK){ return "AUTO_LINK"; }
466    if (n==CONTROL_LINK){ return "CONTROL_LINK"; }
467    if (n==DATA_TRANSFERED){ return "DATA_TRANSFERED"; }
468    if (n==DATA_ADAPTED){ return "DATA_ADAPTED"; }
469    return "UNKNOWN";
470   }
471
472   public static int fromText(String n){
473    if (n.equals("AUTO_LINK")){ return AUTO_LINK; }
474    if (n.equals("CONTROL_LINK")){ return CONTROL_LINK; }
475    if (n.equals("DATA_TRANSFERED")){ return DATA_TRANSFERED; }
476    if (n.equals("DATA_ADAPTED")){ return DATA_ADAPTED; }
477    return 0;
478   }
479
480   public void setValue(int val){
481    if (val==AUTO_LINK||val==CONTROL_LINK||val==DATA_TRANSFERED||val==DATA_ADAPTED){
482     this.value = val;
483    }
484   }
485
486   public int getInt(){ return this.value; }
487
488   public String getString(){ return toText(this.value); }
489
490   public void setValue(String str){
491    if (str!=null){
492     int val = fromText(str);
493     if (val==AUTO_LINK||val==CONTROL_LINK||val==DATA_TRANSFERED||val==DATA_ADAPTED){
494      this.value = val;
495     }
496    }
497   }
498
499   public MetricLinkType(){
500    this.value = DATA_ADAPTED;
501   }
502
503  }
504
505  class MetricTaskType {
506   private int value;
507   public static final int NOT_SHARED = 0;
508   public static final int SHARED = 1;
509   public static final int BI_CONSUMED = 2;
510   public static final int BI_PRODUCED = 3;
511   public static String toText(int n){
512    if (n==NOT_SHARED){ return "NOT_SHARED"; }
513    if (n==SHARED){ return "SHARED"; }
514    if (n==BI_CONSUMED){ return "BI_CONSUMED"; }
515    if (n==BI_PRODUCED){ return "BI_PRODUCED"; }
516    return "UNKNOWN";
517   }
518
519   public static int fromText(String n){
520    if (n.equals("NOT_SHARED")){ return NOT_SHARED; }
521    if (n.equals("SHARED")){ return SHARED; }
522    if (n.equals("BI_CONSUMED")){ return BI_CONSUMED; }
523    if (n.equals("BI_PRODUCED")){ return BI_PRODUCED; }
524    return 0;
525   }
526
527   public void setValue(int val){
528    if (val==NOT_SHARED||val==SHARED||val==BI_CONSUMED||val==BI_PRODUCED){
529     this.value = val;
530    }
531   }
532
533   public int getInt(){ return this.value; }
534
535   public String getString(){ return toText(this.value); }
536
537   public void setValue(String str){
538    if (str!=null){ int val = fromText(str);
539     if (val==NOT_SHARED||val==SHARED||val==BI_CONSUMED||val==BI_PRODUCED){
540      this.value = val;
541     }
542    }
543   }
544
545   public MetricTaskType(){
546    this.value = BI_PRODUCED;
547   }
548
549  }
550
551  class MetricMeetingIndicator {
552   private int value;
553   public static final int MEETING_NOT_NEEDED = 0;
554   public static final int MEETING_NEEDED = 1;
555   public static String toText(int n){
556    if (n==MEETING_NEEDED){ return "MEETING_NEEDED"; }
557    if (n==MEETING_NOT_NEEDED){ return "MEETING_NOT_NEEDED"; }
558    return "UNKNOWN";
559   }
560
561   public static int fromText(String n){
```

```
562     if (n.equals("MEETING_NEEDED")){ return MEETING_NEEDED; }
563     if (n.equals("MEETING_NOT_NEEDED")){ return MEETING_NOT_NEEDED; }
564     return 0;
565   }
566
567   public void setValue(int val){
568     if (val==MEETING_NEEDED||val==MEETING_NOT_NEEDED){
569       this.value = val;
570     }
571   }
572
573   public int getInt(){ return this.value; }
574
575   public String getString(){ return toText(this.value); }
576
577   public void setValue(String str){
578     if (str!=null){ int val = fromText(str);
579       if (val==MEETING_NEEDED||val==MEETING_NOT_NEEDED){
580         this.value = val;
581       }
582     }
583   }
584
585   public MetricMeetingIndicator(){
586     this.value = MEETING_NEEDED;
587   }
588
589 }
590
591
592
593 class Process {
594   private String desc;
595   public ProcessTaskList taskList;
596   public ProcessBIList biList;
597   public ProcessRoleList roleList;
598   private ProcessTask startTask;
599   public String getDesc(){
600     return this.desc;
601   }
602
603   public void setDesc(String desc){
604     this.desc = desc;
605   }
606
607   public Process(){
608     this.desc = "";
609     this.taskList = new ProcessTaskList();
610     this.taskList.clear();
611     this.biList = new ProcessBIList();
612     this.biList.clear();
613     this.roleList = new ProcessRoleList();
614     this.roleList.clear();
615   }
616 }
617
618 class ProcessBranch {
619   private ProcessTask task;
620   public ProcessBIList biList;
621   public MetricLinkType linkType;
622   public ProcessBranch(ProcessTask t){
623     this.biList = new ProcessBIList();
624     this.linkType = new MetricLinkType();
625     this.biList.clear();
626     this.task = t;
627   }
628
629   public ProcessTask getTask(){
630     return this.task;
631   }
632
633   public String name(){
634     return this.task.name();
635   }
636 }
637
638 class ProcessTask {
639
640   private String name;
641   public ProcessTaskRoleList roleList;
642   public ProcessBranchList branchOutList;
643   public ProcessBranchList branchInList;
644
645   /* complexidade de execucao */
646   public MetricGFactor gFactor;
647   public MetricCFactor cFactor;
648   public MetricVFactor vFactor;
649
650   /* complexidade de coordenacao */
651   public MetricTaskType taskType;
652   public MetricMeetingIndicator meetingIndicator;
653
654   public ProcessTask(String n) {
655     this.name = n;
656     this.roleList = new ProcessTaskRoleList();
657     this.roleList.clear();
658     this.branchOutList = new ProcessBranchList();
659     this.branchOutList.clear();
660     this.branchInList = new ProcessBranchList();
```

```
661     this.branchInList.clear();
662
663     // seta metricas default
664     this.gFactor = new MetricGFactor();
665     this.cFactor = new MetricCFactor();
666     this.vFactor = new MetricVFactor();
667     this.taskType = new MetricTaskType();
668     this.meetingIndicator = new MetricMeetingIndicator();
669   }
670
671   public String name(){
672    return this.name;
673   }
674
675   public int biComplexity(){
676    Iterator i, j, k, l;
677    ProcessBranch b;
678    ProcessBI bi;
679    ProcessBIField bif;
680    int sourceScore = 0;
681
682    for (i = this.branchOutList.iterator(); i.hasNext(); ) {
683     b = (ProcessBranch) i.next();
684     for (j = b.biList.iterator(); j.hasNext(); ) {
685      bi = (ProcessBI) j.next();
686      for (k = bi.fieldList.iterator(); k.hasNext(); ) {
687       bif = (ProcessBIField) k.next();
688       sourceScore += bif.sourceScore.getInt();
689      }
690     }
691    }
692
693    for (i = this.branchInList.iterator(); i.hasNext(); ) {
694     b = (ProcessBranch) i.next();
695     for (j = b.biList.iterator(); j.hasNext(); ) {
696      bi = (ProcessBI) j.next();
697      for (k = bi.fieldList.iterator(); k.hasNext(); ) {
698       bif = (ProcessBIField) k.next();
699       sourceScore += bif.sourceScore.getInt();
700      }
701     }
702    }
703
704    return (sourceScore+this.branchOutList.size()+this.branchInList.size())*this.roleList.size();
705
706   }
707
708   public int coordComplexity(){
709
710    Iterator i, j, k, l;
711    ProcessBranch b;
712    int val = 0;
713    int linkType = 0;
714
715    for (i = this.branchOutList.iterator(); i.hasNext(); ) {
716     b = (ProcessBranch) i.next();
717     linkType += b.linkType.getInt();
718    }
719
720    for (i = this.branchInList.iterator(); i.hasNext(); ) {
721     b = (ProcessBranch) i.next();
722     linkType += b.linkType.getInt();
723    }
724
725    val = linkType*this.roleList.size();
726
727    val = val + (this.roleList.size()*this.taskType.getInt()*(this.meetingIndicator.getInt()-1));
728
729    return val;
730   }
731
732   public int execComplexity(){
733    ProcessTaskRole tp;
734    Iterator i, j, k, l;
735
736    int val = 0;
737
738    // execType
739    for (i = this.roleList.iterator(); i.hasNext(); ) {
740     tp = (ProcessTaskRole) i.next();
741     val = val + tp.execType.getInt();
742    }
743
744    // decision
745    if (this.branchOutList.size()>1){
746     val = val + (this.roleList.size() * (this.branchOutList.size()-1) * this.cFactor.getInt() * this.cFactor.getInt() *
             this.vFactor.getInt());
747    }
748
749    return val;
750   }
751
752  }
753  class ProcessBIField {
754   private String name;
755   public MetricSourceScore sourceScore;
756   public ProcessBIField(String n) {
757    // seta metrica default
758    this.sourceScore = new MetricSourceScore();
```

```
759
760     this.name = n;
761   }
762
763   public String name(){
764     return this.name;
765   }
766
767 }
768
769 class ProcessBI {
770
771   public ProcessBIFieldList fieldList;
772
773   private String name;
774   public ProcessBI(String n) {
775     this.fieldList = new ProcessBIFieldList();
776     this.name = n;
777   }
778
779   public String name(){
780     return this.name;
781   }
782 }
783
784
785 class ProcessRole {
786   private String name;
787   public ProcessRole(String n) {
788     this.name = n;
789   }
790
791   public String name(){
792     return this.name;
793   }
794 }
795
796
797 class ProcessTaskRole {
798   private ProcessRole role;
799   public MetricExecType execType;
800   public ProcessTaskRole(ProcessRole p) {
801     this.role = p;
802     this.execType = new MetricExecType();
803   }
804
805   public ProcessRole role(){
806     return this.role;
807   }
808 }
809
810 class ProcessXML {
811   private Process P;
812   public ProcessXML(Process p){
813     this.P = p;
814   }
815
816   public void saveToFile(File f){
817
818     Element el1, el2, el3;
819     PrintStream out = System.out;
820     Iterator i, j, k;
821     ProcessTask t;
822     ProcessRole p;
823     ProcessTaskRole tp;
824     ProcessBranch b;
825     ProcessBI bi;
826     ProcessBIField bif;
827
828     /* GERA NOVO XML */
829     Element process = new Element("process");
830     Document procDoc = new Document(process);
831
832     process.addContent(new Element("desc").setText(P.getDesc()));
833
834     /* carrega BIs */
835     for (i = P.biList.iterator(); i.hasNext(); ) {
836       bi = (ProcessBI) i.next();
837       el1 = new Element("bi");
838       out.print("xml_build: business item "+bi.name()+"\n");
839       el1.setAttribute(new Attribute("name",bi.name()));
840       for (j = bi.fieldList.iterator(); j.hasNext(); ) {
841         bif = (ProcessBIField) j.next();
842         el2 = new Element("field");
843         out.print("xml_build: business item field "+bi.name()+"."+bif.name()+"\n");
844         el2.setAttribute(new Attribute("name",bif.name()));
845         // metricas
846         el2.setAttribute(new Attribute("sourcescore",bif.sourceScore.getString()));
847         el1.addContent(el2);
848       }
849       process.addContent(el1);
850     }
851
852     /* carrega Roles */
853     for (i = P.roleList.iterator(); i.hasNext(); ) {
854       p = (ProcessRole) i.next();
855       el1 = new Element("role");
856       out.print("xml_build: role "+p.name()+"\n");
857       el1.setAttribute(new Attribute("name",p.name()));
```

```
858     process.addContent(el1);
859    }
860
861    /* carrega Tasks */
862    for (i = P.taskList.iterator(); i.hasNext(); ) {
863     t = (ProcessTask) i.next();
864     out.print("xml_build: task "+t.name()+"\n");
865     el1 = new Element("task");
866     el1.setAttribute(new Attribute("name",t.name()));
867
868     // metricas
869     el1.setAttribute(new Attribute("cfactor",t.cFactor.getString()));
870     el1.setAttribute(new Attribute("vfactor",t.vFactor.getString()));
871     el1.setAttribute(new Attribute("gfactor",t.gFactor.getString()));
872
873     // metricas
874     el1.setAttribute(new Attribute("tasktype",t.taskType.getString()));
875     el1.setAttribute(new Attribute("meetingindicator",t.meetingIndicator.getString()));
876
877     for (j = t.roleList.iterator(); j.hasNext(); ){
878      tp = (ProcessTaskRole) j.next();
879      out.print("xml_build: task.role "+tp.role().name()+"\n");
880      el2 = new Element("role");
881      el2.setAttribute("name",tp.role().name());
882      el2.setAttribute("exectype",tp.execType.getString());
883      el1.addContent(el2);
884     }
885     for (j = t.branchOutList.iterator(); j.hasNext(); ){
886      b = (ProcessBranch) j.next();
887      out.print("xml_build: task.branch "+b.name()+"\n");
888      el2 = new Element("branch");
889      el2.setAttribute("name",b.name());
890      el2.setAttribute(new Attribute("linktype",b.linkType.getString()));
891      for (k = b.biList.iterator(); k.hasNext(); ){
892       bi = (ProcessBI) k.next();
893       out.print("xml_build: task.branch.bi "+bi.name()+"\n");
894       el3 = new Element("bi");
895       el3.setAttribute("name",bi.name());
896       el2.addContent(el3);
897      }
898      el1.addContent(el2);
899     }
900     process.addContent(el1);
901    }
902
903    try {
904     FileOutputStream sourceOut = new FileOutputStream(f);
905     XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
906     outputter.output(procDoc, sourceOut);
907    } catch (Exception e) {
908     e.printStackTrace();
909    }
910
911   }
912
913  }
914
915  class ProcessHistogram {
916   public ProcessHistogram(File f, String eps, String title, String dat_file, int data_id){
917    try {
918
919     FileWriter fstream = new FileWriter(f);
920     BufferedWriter out = new BufferedWriter(fstream);
921
922     out.write("set terminal postscript eps color lw 15 \"Helvetica\" 36\n");
923     out.write("set output '"+eps+"'\n");
924     out.write("set size '1.7,1.7'\n");
925     out.write("set border 3 front linetype -1 linewidth 1.000\n");
926     out.write("set boxwidth 0.8 absolute\n");
927     out.write("set style fill solid 1.00\n");
928     out.write("set grid nopolar\n");
929     out.write("set grid noxtics nomxtics ytics nomytics noztics nomztics nox2tics nomx2tics noy2tics nomy2tics nocbtics
            nomcbtics\n");
930     out.write("set grid layerdefault linetype 0 linewidth 1.000, linetype 0 linewidth 1.000\n");
931     out.write("set nokey\n");
932     out.write("set style histogram rowstacked title offset character 2, 0.25, 0\n");
933     out.write("set datafile missing '-'\n");
934     out.write("set style data histograms\n");
935     out.write("set xtics border in scale 1,0.5 nomirror rotate by -70 offset character 0, 0, 0\n");
936     out.write("set ytics border in scale 0,0 mirror norotate offset character 0, 0, 0 autofreq\n");
937     out.write("set ztics border in scale 0,0 nomirror norotate offset character 0, 0, 0 autofreq\n");
938     out.write("set cbtics border in scale 0,0 mirror norotate offset character 0, 0, 0 autofreq\n");
939     out.write("set title \""+title+"\"\n");
940     out.write("set xlabel \"\"\n");
941     out.write("set xlabel offset character 0, -2, 0 font \"\" textcolor lt -1 norotate\n");
942     out.write("plot '"+dat_file+"' using "+data_id+":xtic(1) t "+data_id+"\n");
943
944     out.flush();
945
946    } catch(Exception e) {
947     e.printStackTrace();
948    }
949
950   }
951  }
952
953  public class ComplexityAnalyzer {
954
955   /* MAIN */
```

```
956  public static void main(String[] args) {
957
958   /*
959   CONFIG INICIAL
960    */
961
962   /* stream de saída */
963   PrintStream out = System.out;
964
965   Process P = new Process();
966
967   ProcessTask t;
968   ProcessTaskRole tp;
969   ProcessRole p;
970   ProcessBranch b;
971   ProcessBI bi;
972   ProcessBIField bif;
973   Element el1, el2, el3;
974
975   Iterator i, j, k;
976
977   /* check parametros */
978   if (args.length < 1) {
979    out.println("Usage: analyzer work_dir");
980    return;
981   } try {
982
983    String work_dir = args[0];
984    File input_xml = new File(work_dir+"/process.xml");
985    File output_xml = new File(work_dir+"/output.xml");
986    File output_dot = new File(work_dir+"/process.dot");
987    File output_dat = new File(work_dir+"/complexity.dat");
988    File output_gnu_exec = new File(work_dir+"/histogram_exec.gnu");
989    File output_gnu_coord = new File(work_dir+"/histogram_coord.gnu");
990    File output_gnu_bi = new File(work_dir+"/histogram_bi.gnu");
991    File output_gnu_total = new File(work_dir+"/histogram_total.gnu");
992    String output_eps_exec = work_dir+"/www/histogram_exec.eps";
993    String output_eps_coord = work_dir+"/www/histogram_coord.eps";
994    String output_eps_bi = work_dir+"/www/histogram_bi.eps";
995    String output_eps_total = work_dir+"/www/histogram_total.eps";
996
997    new ProcessHistogram(output_gnu_exec,output_eps_exec,"Complexidade de Execucao",work_dir+"/complexity.dat",2);
998    new ProcessHistogram(output_gnu_coord,output_eps_coord,"Complexidade de Coordenacao",work_dir+"/complexity.dat",3);
999    new ProcessHistogram(output_gnu_bi,output_eps_bi,"Complexidade de Itens de Negocio",work_dir+"/complexity.dat",4);
1000   new ProcessHistogram(output_gnu_total,output_eps_total,"Complexidade do Processo",work_dir+"/complexity.dat",5);
1001
1002   /*
1003   XML
1004    */
1005
1006   /* cria documento sem validacao */
1007   SAXBuilder builder = new SAXBuilder(false);
1008   Document xmlDoc = builder.build(input_xml);
1009
1010   /* busca elemento raiz do arquivo */
1011   Element root = xmlDoc.getRootElement();
1012
1013   /* busca <description> */
1014   Element desc = root.getChild("desc");
1015
1016   if (desc!=null){
1017
1018    out.print("xml_load: process description \""+desc.getText()+"\"\n");
1019
1020    /* seta descricao do processo */
1021    P.setDesc(desc.getText());
1022
1023   } else {
1024
1025    P.setDesc("Not Available");
1026
1027   }
1028
1029   /*
1030   ROLES
1031    */
1032
1033   /* busca <role/> */
1034   List proc_roles = root.getChildren("role");
1035
1036   /* lista roles */
1037   i = proc_roles.iterator();
1038   while (i.hasNext()) {
1039    Element proc_role = (Element) i.next();
1040    out.print("xml_load: carregando role "+proc_role.getAttributeValue("name")+"\n");
1041
1042    /* adiciona role ao processo */
1043    P.roleList.add(new ProcessRole(proc_role.getAttributeValue("name")));
1044
1045   }
1046
1047   /*
1048   BUSINESS ITENS
1049    */
1050
1051   /* busca <bi/> */
1052   List proc_bis = root.getChildren("bi");
1053
1054   /* lista roles */
```

```
1055      i = proc_bis.iterator();
1056      while (i.hasNext()) {
1057       Element proc_bi = (Element) i.next();
1058       out.print("xml_load: carregando business item "+proc_bi.getAttributeValue("name")+"\n");
1059
1060       bi = new ProcessBI(proc_bi.getAttributeValue("name"));
1061
1062       List proc_bi_fields = proc_bi.getChildren("field");
1063       j = proc_bi_fields.iterator();
1064       while (j.hasNext()) {
1065        Element proc_bi_field = (Element) j.next();
1066        bif = new ProcessBIField(proc_bi_field.getAttributeValue("name"));
1067        out.print("xml_load: carregando business item field "+proc_bi.getAttributeValue("name")+"."+proc_bi_field.
              getAttributeValue("name")+"\n");
1068        bif.sourceScore.setValue(proc_bi_field.getAttributeValue("sourcescore"));
1069        bi.fieldList.add(bif);
1070       }
1071
1072       /* adiciona business item ao processo */
1073       P.biList.add(bi);
1074
1075      }
1076      /*
1077      TASKS
1078       */
1079
1080      /* tasks */
1081      List proc_tasks = root.getChildren("task");
1082
1083      i = proc_tasks.iterator();
1084
1085      while (i.hasNext()){
1086
1087       Element proc_task = (Element) i.next();
1088
1089       out.print("xml_load: carregando task "+proc_task.getAttributeValue("name")+"\n");
1090
1091       /* cria task */
1092       t = new ProcessTask(proc_task.getAttributeValue("name"));
1093       t.gFactor.setValue(proc_task.getAttributeValue("gfactor"));
1094       t.cFactor.setValue(proc_task.getAttributeValue("cfactor"));
1095       t.vFactor.setValue(proc_task.getAttributeValue("vfactor"));
1096       t.taskType.setValue(proc_task.getAttributeValue("tasktype"));
1097       t.meetingIndicator.setValue(proc_task.getAttributeValue("meetingindicator"));
1098
1099       /* task-roles */
1100       List proc_task_roles = proc_task.getChildren("role");
1101       j = proc_task_roles.iterator();
1102       while (j.hasNext()) {
1103        Element proc_task_role = (Element) j.next();
1104        out.print("xml_load: carregando task.role "+proc_task_role.getAttributeValue("name")+"\n");
1105
1106        /* adiciona role à task */
1107
1108        tp = new ProcessTaskRole(P.roleList.find(proc_task_role.getAttributeValue("name")));
1109        if (proc_task_role.getAttributeValue("exectype")!=null){
1110         tp.execType.setValue(proc_task_role.getAttributeValue("exectype"));
1111        }
1112        t.roleList.add(tp);
1113
1114       }
1115
1116       /* adiciona task ao processo */
1117       P.taskList.add(t);
1118
1119      }
1120      /* task-branches */
1121      /* tasks */
1122      i = proc_tasks.iterator();
1123
1124      while (i.hasNext()) {
1125
1126       Element proc_task = (Element) i.next();
1127       out.print("xml_load: carregando branches da task "+proc_task.getAttributeValue("name")+"\n");
1128
1129       t = P.taskList.find(proc_task.getAttributeValue("name"));
1130
1131       List proc_task_branches = proc_task.getChildren("branch");
1132
1133       j = proc_task_branches.iterator();
1134
1135       while (j.hasNext()) {
1136
1137        Element proc_task_branch = (Element) j.next();
1138        out.print("xml_load: carregando task.branch "+proc_task_branch.getAttributeValue("name")+"\n");
1139
1140        /* cria branch */
1141        b = new ProcessBranch(P.taskList.find(proc_task_branch.getAttributeValue("name")));
1142        b.linkType.setValue(proc_task_branch.getAttributeValue("linktype"));
1143
1144        /* task-branch-bis */
1145
1146        List proc_task_branch_bis = proc_task_branch.getChildren("bi");
1147
1148        k = proc_task_branch_bis.iterator();
1149        while (k.hasNext()) {
1150         Element proc_task_branch_bi = (Element) k.next();
1151         out.print("xml_load: carregando task.branch.bi "+proc_task_branch_bi.getAttributeValue("name")+"\n");
1152
```

```
1153            bi = P.biList.find(proc_task_branch_bi.getAttributeValue("name"));
1154            b.biList.add(bi);
1155
1156           }
1157
1158           out.print("xml_load: adicionando branch "+b.name()+"\n");
1159           P.taskList.find(proc_task.getAttributeValue("name")).branchOutList.add(b);
1160           P.taskList.find(proc_task_branch.getAttributeValue("name")).branchInList.add(b);
1161
1162          }
1163         }
1164
1165         /* Salva XML */
1166         ProcessXML pxml = new ProcessXML(P);
1167         pxml.saveToFile(output_xml);
1168
1169         /* cria saida para gerar grafico */
1170         ProcessGraph graph = new ProcessGraph(P);
1171         graph.saveToFile(output_dot);
1172
1173         /* cria saida para histograma */
1174         ProcessStats stats = new ProcessStats(P);
1175         stats.saveToFile(output_dat);
1176
1177       } catch (Exception e){
1178                 e.printStackTrace();
1179             }
1180        }
1181    }
```

ANEXO II - XML SCHEMA

Este anexo descreve o XML *Schema* definido para composição e validação dos processos modelados para análise de complexidade.

```
<!-- SCHEMA -->
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- PROCESS -->
  <xs:element name="process">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="desc"/>
        <xs:element ref="bi"/>
        <xs:element ref="role"/>
        <xs:element ref="task"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="desc" type="xs:string"/>
  <!-- TASK -->
  <xs:element name="task">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="role"/>
        <xs:element ref="branch"/>
      </xs:sequence>
      <xs:attribute name="compExec" type="xs:integer"/>
      <xs:attribute name="compCoord" type="xs:integer"/>
      <xs:attribute name="compBI" type="xs:integer"/>
      <xs:attribute name="cfactor">
           <xs:simpleType name="cfactor">
            <xs:restriction base="xs:string">
             <xs:enumeration value="NEGLIGIBLE"/>
                 <xs:enumeration value="MODERATE"/>
                 <xs:enumeration value="SEVERE"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
      <xs:attribute name="gfactor">
                        <xs:simpleType>
            <xs:restriction base="xs:string">
             <xs:enumeration value="RECOMENDATION"/>
                  <xs:enumeration value="INFORMATION"/>
                  <xs:enumeration value="INTERNAL"/>
            </xs:restriction>
               </xs:simpleType>
          </xs:attribute>
      <xs:attribute name="meetingindicator">
           <xs:simpleType>
            <xs:restriction base="xs:string">
             <xs:enumeration value="MEETING_NOT_NEEDED"/>
                 <xs:enumeration value="MEETING_NEEDED"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
      <xs:attribute name="name" type="xs:string"/>
```

```
53        <xs:attribute name="tasktype">
54             <xs:simpleType>
55              <xs:restriction base="xs:string">
56               <xs:enumeration value="SHARED"/>
57                    <xs:enumeration value="NOT_SHARED"/>
58                    <xs:enumeration value="BI_PRODUCED"/>
59                    <xs:enumeration value="BI_CONSUMED"/>
60              </xs:restriction>
61             </xs:simpleType>
62            </xs:attribute>
63        <xs:attribute name="vfactor">
64             <xs:simpleType>
65              <xs:restriction base="xs:string">
66               <xs:enumeration value="IMEDIATE"/>
67                    <xs:enumeration value="SHORT_TERM"/>
68                    <xs:enumeration value="LONG_TERM"/>
69              </xs:restriction>
70             </xs:simpleType>
71            </xs:attribute>
72      </xs:complexType>
73    </xs:element>
74    <xs:element name="branch">
75      <xs:complexType>
76        <xs:sequence>
77          <xs:element ref="bi"/>
78        </xs:sequence>
79        <xs:attribute name="linktype">
80             <xs:simpleType>
81              <xs:restriction base="xs:string">
82               <xs:enumeration value="AUTO_LINK"/>
83                    <xs:enumeration value="DATA_TRANSFERED"/>
84                    <xs:enumeration value="DATA_ADAPTED"/>
85                    <xs:enumeration value="CONTROL_LINK"/>
86              </xs:restriction>
87             </xs:simpleType>
88            </xs:attribute>
89       <xs:attribute name="name" type="xs:string"/>
90      </xs:complexType>
91    </xs:element>
92    <!-- BUSINESS ITEM -->
93    <xs:element name="bi">
94      <xs:complexType>
95        <xs:sequence>
96          <xs:element ref="field"/>
97        </xs:sequence>
98        <xs:attribute name="name" type="xs:string"/>
99      </xs:complexType>
100   </xs:element>
101   <xs:element name="field">
102     <xs:complexType>
103       <xs:attribute name="name" type="xs:string"/>
104       <xs:attribute name="sourcescore">
105            <xs:simpleType>
106             <xs:restriction base="xs:string">
107              <xs:enumeration value="INTERNAL"/>
108                   <xs:enumeration value="FREE_CHOICE"/>
109                   <xs:enumeration value="DOCUMENTATION_DIRECT"/>
110             <xs:enumeration value="DOCUMENTATION_ADAPTED"/>
111                  <xs:enumeration value="BEST_PRATICE"/>
112                  <xs:enumeration value="ENVIRONMENT_FIXED"/>
113             <xs:enumeration value="ENVIRONMENT_CONSTRAINED"/>
114             </xs:restriction>
115            </xs:simpleType>
116           </xs:attribute>
117     </xs:complexType>
118   </xs:element>
119   <!-- ROLE -->
120   <xs:element name="role">
121     <xs:complexType>
122       <xs:attribute name="IDX" type="xs:integer"/>
123       <xs:attribute name="exectype">
124            <xs:simpleType>
125             <xs:restriction base="xs:string">
```

```
126                 <xs:enumeration value="AUTOMATIC"/>
127                     <xs:enumeration value="TOOL_ASSISTED"/>
128                     <xs:enumeration value="MANUAL"/>
129                 </xs:restriction>
130               </xs:simpleType>
131             </xs:attribute>
132         <xs:attribute name="name" type="xs:string"/>
133       </xs:complexType>
134     </xs:element>
135 </xs:schema>
```