COLLABRDL: UMA EXTENSÃO DA RDL PARA ESPECIFICAÇÃO DE PROCESSOS DE REUTILIZAÇÃO COLABORATIVOS

Edson Mello Lucas

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Toacy Cavalcante de Oliveira

Rio de Janeiro
Maio de 2013

COLLABRDL: UMA EXTENSÃO DA RDL PARA ESPECIFICAÇÃO DE PROCESSOS DE REUTILIZAÇÃO COLABORATIVOS

Edson Mello Lucas

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Toacy Cavalcante de Oliveira, D.Sc.

_____
Prof.ª Claudia Maria Lima Werner, D.Sc.

_____
Elder José Reioli Cirilo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2013

Lucas, Edson Mello

CollabrRDL: Uma extensão da RDL para especificação de processos de reutilização colaborativos/ Edson Mello Lucas. – Rio de Janeiro: UFRJ/COPPE, 2013.

XII, 150 p.: il.; 29,7 cm.

Orientador: Toacy Cavalcante de Oliveira

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 102-112.

1. Reutilização de Software. 2. Especificação de Processos Colaborativos. I. Oliveira, Toacy Cavalcante de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

A Deus pela vida e presença, e aos meus pais, Lecy e Edma, pelo ensinamento dos valores que me fortalecem a cada conquista.

A minha esposa, Nina, e aos meus filhos, Lucas e Mateus, por entenderem a minha falta física em alguns momentos e pelo apoio.

Aos professores Francisco Moura, Helio Souto, Pedro Watts e Ivan Bastos, do Instituto Politécnico da Uerj, por pelo menos, pelas palavras de motivação. E aos diversos alunos, professores e técnico-administrativos que tive o prazer de conviver nesses muitos anos de dedicação à Uerj, em especial ao Hilton Guaraldi e ao Anderson Cordeiro.

A todos os professores e companheiros de trabalho de grupo em disciplinas que tive o privilégio de conviver no PESC/COPPE. E também ao pessoal administrativo e de suporte que sempre deram o apoio necessário para o cumprimento das minhas obrigações.

Aos meus colegas do grupo Prisma, André Campos, Raquel Pillat, Talita Gomes, Ivens Portugal, Chessman Correa, Renata Mesquita e Alciléia Rocha pelo compartilhamento dos problemas e soluções e , em especial, ao Fábio Basso que colaborou na execução da prova de conceito deste trabalho.

Ao professor e ser humano Toacy pela paciência e dedicação que sempre foram a mais do que eu imaginava um dia ser.

Um agradecimento especial aos professores Guilherme Horta, Claudia Werner e Ana Regina pelas frases de efeito que me fizeram refletir. E ao Daniel Schneider pela imensa colaboração na escrita do artigo que descreve parte deste trabalho.

Aos membros do grupo de Engenharia Experimental do PESC, Breno França e Jobson Silva, pela disponibilidade para o diálogo sobre experimentação.

À Prof. Claudia Werner e ao Dr. Elder Cirilo por aceitarem o convite para a minha banca de defesa.

Finalmente, a todas as pessoas que de alguma forma contribuíram socialmente e/ou tecnicamente.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

COLLABRDL: UMA EXTENSÃO DA RDL PARA ESPECIFICAÇÃO DE PROCESSOS DE REUTILIZAÇÃO COLABORATIVOS

Edson Mello Lucas

Maio/2013

Orientador: Toacy Cavalcante de Oliveira

Programa: Engenharia de Sistemas e Computação

Sistemas de Software modernos são desenvolvidos por pessoas trabalhando em equipe e reutilização de software consiste no reaproveitamento de artefatos e conhecimento adquiridos em projetos anteriores durante um projeto atual. A Reutilização de Software Colaborativa une os conceitos de trabalho colaborativo e de reutilização de software com o objetivo de produzir software com melhor qualidade e em menos tempo.

Esta dissertação estende uma linguagem para representação de reutilização, RDL, com conceitos de colaboração para permitir a realização das atividades de reutilização em equipe e quando possível em paralelo. Para isto foram criados os comandos ROLE, PARALLEL e DOPARALLEL, além da implementação de um ambiente de execução baseado em uma ferramenta que implementa BPMN. A avaliação deste trabalho foi realizada através de padrões de workflow e de uma prova de conceito.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COLLABRDL: AN EXTENSION OF RDL TO SPECIFY COLLABORATIVE REUSE PROCESSES

Edson Mello Lucas

May/2013

Advisor: Toacy Cavalcante de Oliveira

Department: Systems and Computing Engineering

Modern software systems are developed by people working in teams and software reuse is the reuse of artifacts and knowledge acquired in previous projects for the current project. Collaborative Software Reuse unites concepts of collaborative work and software reuse in order to produce better quality software in less time.

This dissertation extends a language for representing reuse, RDL, with concepts of collaboration to enable the realization of reuse activities in a team and when possible in parallel. For that, the commands ROLE, PARALLEL, DOPARALLEL and a runtime environment based on BPMN tool were created. The evaluation of this work was conducted through workflow patterns and a proof of concept.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1 - INTRODUÇÃO

Este capítulo apresenta a motivação para este trabalho, os seus objetivos, a metodologia de pesquisa e a organização desta dissertação.

1.1 - Motivação

Sistemas de Software modernos são desenvolvidos por pessoas trabalhando em equipe, uma vez que a complexidade destes sistemas exige o conhecimento em várias disciplinas como Programação, Interface Homem-máquina e Banco de Dados, além do conhecimento do domínio de aplicação do sistema. Neste contexto, a colaboração entre pessoas aflora como fator importante para o sucesso de um projeto de desenvolvimento de software e, portanto, ferramentas para apoiar o trabalho colaborativo são necessárias (BARTHELMESS e ANDERSON, 2002). Um outro fator determinante para a construção de sistemas de software é a Reutilização de Software (FRAKES e KANG, 2005). O conceito de reutilização permite o reaproveitamento do conhecimento adquirido em projetos anteriores durante um projeto atual de desenvolvimento, permitindo assim ganhos de qualidade e a economia nos recursos envolvidos. Neste cenário, a Reutilização de Software Colaborativa (MENDONÇA *et al*., 2008, NOOR *et al*, 2007) une os conceitos de trabalho colaborativo e de desenvolvimento de software a partir de artefatos reutilizáveis, de modo que o processo de desenvolvimento possa ser harmonioso (MOHAGHEGHI e CONRADI, 2007).

A reutilização sistemática que possui um alto grau de gerenciamento necessita de um processo de reutilização formalizado (ROTHENBERGER *et al.*, 2003), possibilitando assim a sua repetição, ou seja, gerando produtos diferentes a partir do mesmo conjunto de artefatos reutilizáveis. Além disso, precisa ser planejada e coordenada (MALONE e CROWSTON, 1994). Nesse contexto, pessoas realizam atividades que fazem uso de artefatos reutilizáveis de forma coordenada para atingirem um objetivo: o software a ser gerado. A realização dessas atividades caracteriza um processo de reutilização de software colaborativo, que pode ser documentado e também reutilizado.

Por outro lado, a documentação de um processo de reutilização de software colaborativo coordena atividades que podem ser interativas. Assim, várias execuções do mesmo processo poderão produzir softwares diferentes, pois são consequências das escolhas e respostas nas atividades interativas. Dessa forma, a documentação pode apoiar na construção de softwares para um mesmo domínio com características diferentes. Por exemplo, quando uma equipe decide reutilizar o framework Portlet (BELLAS, 2004, HEPPER, 2008).

A Figura 1.1 mostra um Sistema de Gerenciamento de Projetos (SGP, 2013) construído a partir da reutilização do framework portlet. Em detalhe, a aba *Metas* é composta por 4 portlets: *Metas, Detalhes da Meta, Atividades da Meta, Detalhes da Atividade.* O portlet *Metas* exibe todas as metas do projeto selecionado informando o nome, andamento, o responsável e a data/hora da última atualização de cada meta. O portlet *Detalhes da Meta* exibe a descrição, data início e fim da meta selecionada no portlet *Metas*. O portlet *Atividades da Meta* lista as atividades da meta selecionada no portlet *Metas* informando o nome, o andamento e data da última edição de cada atividade necessária para alcançar a meta em questão. E, finalmente, o portlet *Detalhes da Atividade* mostra a descrição, data de início e fim, peso percentual sobre a meta, percentual executado, observações e, quando necessário, uma subdivisão da atividade em itens necessários para a sua execução.

Cada portlet do Sistema da Figura 1.1 é criado a partir da herança da classe GenericPortlet, e a equipe decide pela implementação ou não dos seus métodos *init*, *render, doDispatch, getTitle, doEdit, doHelp, destroy e* obrigatoriamente implementará outros, tais como *doView e doHeaders.* Os filtros são opcionais e são criados pela implementação das *interfaces ActionFilter, EventFilter, RenderFilter e ResourceFilter.* Já o arquivo portlet.xml tem as informações necessárias para registrar os portlets no contêiner de execução. Além disso, podemos ver que os portlets possuem funcionalidades comuns, tais como, modo de edição, visualização, e funções de minimizar e maximizar como tantas outras.

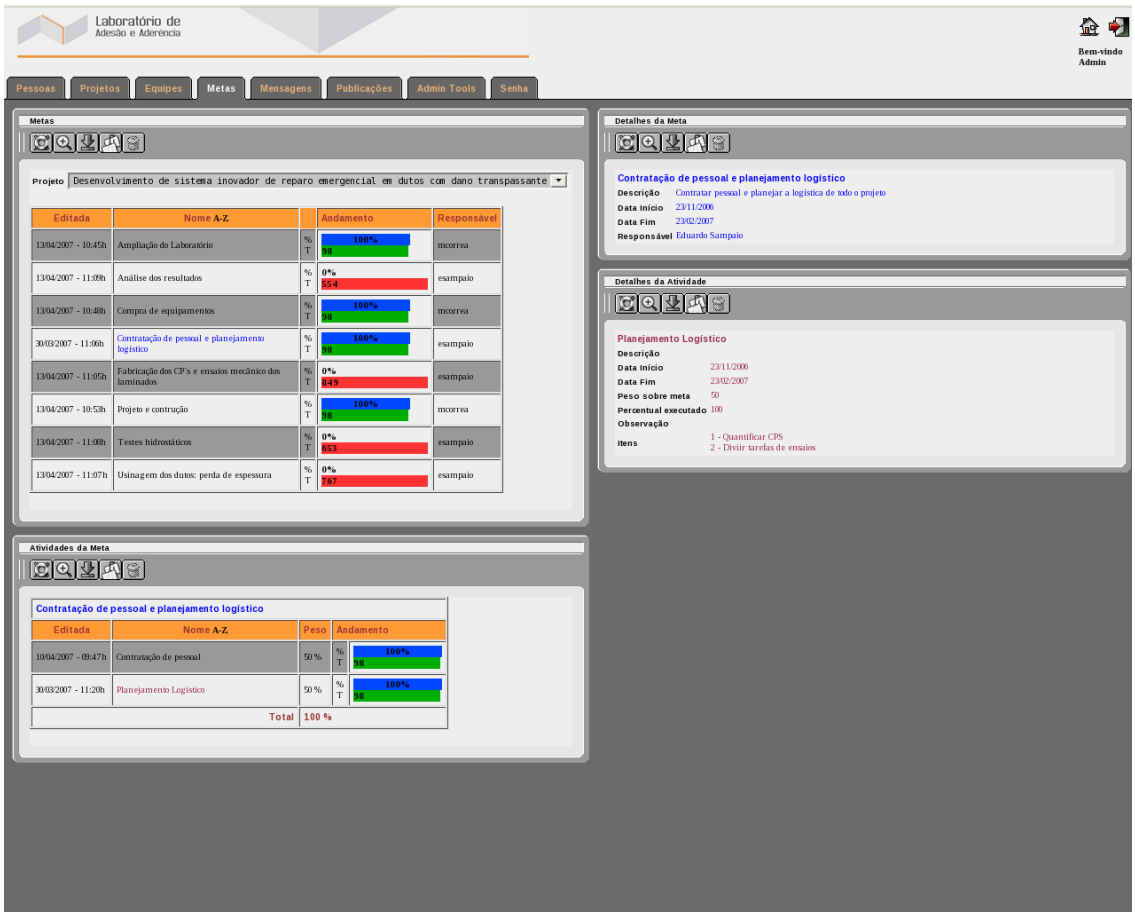


Figura 1.1: Sistema de Gerenciamento de Projetos construído com portlets

É grande a variedade de artefatos de software que podem ser reutilizados, sendo os mais conhecidos código fonte, modelos, especificações, análise, projeto, casos de teste e casos de uso (ALI e DU, 2004). Inicialmente, e de forma rudimentar, a reutilização se baseia em copiar e colar trechos de um código de software para gerar outro. A organização melhora quando se utiliza o conceito de bibliotecas para reutilização e, em um nível mais abstrato, a reutilização de modelos e ativos. Ativos são o agrupamento de artefatos que oferecem uma solução para um problema em um contexto (OMG, 2005). Um ativo é classificado quanto a sua *granularidade*, que em geral está relacionada ao seu tamanho e complexidade, *variabilidade,* o quanto ele pode ser alterado, e o grau de *completude* dos seus artefatos, i.e., se oferecem especificação e implementação.

Na literatura encontramos algumas abordagens para reutilização de software em Instanciação de Framework, Linha de Produtos de Software (LPS), Cadeias de Transformação (MDD) e *Staged Configuration*. CECHTICKY *et al.* (2003) descrevem um ambiente para instanciar frameworks que utiliza uma linguagem de transformação para gerar código a partir da especificação formal dos componentes do framework, fazem uso de processo mas não tratam a questão do trabalho em equipe. *Collaborative Product Configuration* (CPC) permite configurar um novo produto em uma Linha de Produtos de Software de forma colaborativa (MENDONÇA *et al.,* 2007 e 2008), embora tenha um plano de execução, este não descreve o processo de escolha das características do novo produto.

ThinkLets são usados para descrever a engenharia de colaboração que é o desenvolvimento de processos de colaboração repetíveis conduzido pelos próprios praticantes. NOOR *et al.* (2007) apresentam uma abordagem colaborativa para expressar um processo no escopo de Linha de Produtos utilizando *thinklets,* mas não explora a possibilidade de atividades em paralelo, mesma limitação da abordagem de RABISER *et al.* (2007). CZARNECKI *et al. (*2004) criam uma nova configuração a partir de sucessivos passos de especialização do modelo de características (KANG *et al.*, 1990), em cada passo é gerado um modelo mais próximo do produto final, essa técnica é conhecida como *staged configuration*, mas não abordam colaboração nesse trabalho. No entanto, mais tarde (CZARNECKI *et al.,* 2005), é proposto o conceito *Multi-level Staged Configuration* (MLSC), onde mencionam a participação de pessoas com diferentes papéis na configuração, mesmo que ainda sequencialmente. As linguagens *Variability Modelling Language* (VML), para sincronizar o modelo de características ao modelo de ativos reutilizáveis, que por exemplo pode ser um modelo de componentes representado em UML (LOUGHRAN *et al.*, 2008) e ATL (ATLAS *Transformation Language*) para transformação de modelos no escopo de MDE (*Model Driven Engineering*) não possuem comandos para expressar colaboração (JOUAULT e KURTEV, 2006).

Finalmente, HUBAUX *et al.* (2009) definiram *Feature Configuration Workflows* (FCW) que usa o conceito de workflow para descrever o processo de configuração de um novo produto a partir de um modelo de características, permite a distribuição de

atividades entre pessoas sendo possível realizá-las em paralelo. Em (ABBASI, HUBAUX e HEYMANS, 2011) é apresentada uma ferramenta para FCW.

Com exceção da abordagem de HUBAUX *et al.* (2009), todas as demais possuem deficiências quanto às características necessárias para facilitar o trabalho em equipe. Assim, fica entendido que investir na melhoria dessas ferramentas nesse sentido se justifica já que softwares são desenvolvidos por pessoas trabalhando em equipes. Este trabalho de mestrado se compromete em suprir essas deficiências na RDL.

1.2 - Objetivos

Esta dissertação tem como objetivo apoiar o processo de reutilização de software colaborativo. E para isto será utilizada a *Reuse Description Language* (RDL), RDL é objeto de estudo do nosso grupo de pesquisa, ela é para descrever o processo de reutilização, o faz de forma sistemática e para instanciar frameworks orientados a objetos, porém não de forma colaborativa (OLIVEIRA *et al*., 2007, OLIVEIRA, 2001). Além disso, ela é textual e executável, permite descrever as atividades do processo de reutilização de forma explícita (OLIVEIRA et al., 2007) seguindo uma ordem. RDL também é uma linguagem interativa onde os reutilizadores, que fazem a instanciação de framework, precisam entrar com respostas durante o processo de reutilização.

A Figura 1.2 ilustra um trecho da reutilização de um framework orientado a objetos modelado em UML utilizando RDL, este exemplo é baseado no framework *Shape,* visto com mais detalhes em Oliveira *et al.* (2011). O produto deste trecho de reutilização é a classe *Circle* que estende da classe *Shape* redefinindo os seus métodos *addConnectionAnchor*, *getFigure* e *getIcon* em um novo pacote chamado *org.reusetool.myshapeseditor*. A parte superior da Figura 1.2 mostra o trecho de programa escrito em RDL para guiar este processo de reutilização. Na linha 1, descreve-se a criação do pacote *org.reusetool.myshapeseditor*, a linha 2 indica o início de um bloco de atividades que pode ser repetido em tempo de execução quantas vezes o reutilizador necessitar. Assim ao executar está linha, ele receberá a pergunta "*Criar outro shape?*", caso responda sim, ele entrará no bloco do comando LOOP. A linha 4 descreve a criação de uma nova classe que estende de *Shape* no pacote *org.reusetool.myshapeseditor*, que é para onde a variável *packA* aponta, e o parâmetro

"?" indica que em tempo de execução o reutilizador irá receber uma pergunta para entrar com o nome da nova classe, neste exemplo o reutilizador escolheu *Circle*. As linhas 6, 7 e 8 redefinem na classe apontada pela variável *shapeClass* os métodos *addConnectionAnchor*, *getFigure*, *getIcon* da classe pai *Shape*.

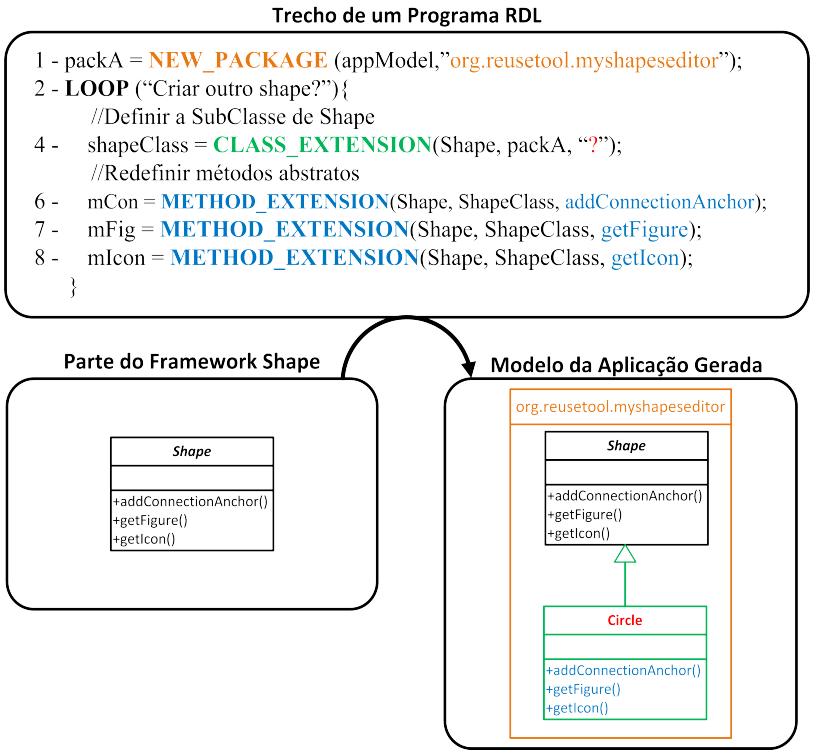


Figura 1.2: Trecho de reutilização utilizando RDL.

Atualmente, um programa em RDL expressa um processo de reutilização sequencial e monousuário, e portanto inadequado para várias situações complexas de reutilização. Logo, um programa RDL necessita incorporar conceitos de colaboração para permitir a realização das atividades de reutilização em equipe e quando possível em paralelo. Este objetivo é dividido em 5 objetivos específicos:

1. Permitir a definição de grupos de reutilizadores;
2. Delegar atividades de reutilização a grupo de pessoas;
3. Expressar atividades de reutilização em paralelo;
4. Definir um ambiente executável com base em BPMN (2011);
5. Avaliar através de alguns padrões de workflow (VAN DER AALST *et al.*, 2003, , RUSSELL *et al.* 2006);

6. Avaliar através de uma prova de conceito.

O primeiro objetivo específico está relacionado à criação e gerenciamento de grupos de pessoas. O segundo atende a necessidade de delegar atividades a reutilizadores de acordo com a sua área de conhecimento, experiência e responsabilidade, por exemplo, associando atividades de modelagem a analistas e atividades de programação a programadores. O terceiro atende a necessidade de distribuir atividades para serem realizadas simultaneamente no intuito de reduzir o tempo total de execução do processo de reutilização. O quarto surge da necessidade de reutilizar um ambiente de execução de workflow compatível com BPMN para executar o processo de reutilização de forma colaborativa. O quinto e o sexto objetivos se destinam à avaliação deste trabalho descrevendo alguns padrões de workflow (VAN DER AALST *et al.*, 2003, RUSSELL *et al.* 2006) e pela realização de uma prova de conceito. A extensão da RDL que implementa esses objetivos é chamada de CollabRDL.

Cabe ressaltar que este trabalho não tem como objetivo tratar dos problemas relacionados com as linguagens que se propõem representar colaboração. Da mesma forma que não irá investigar soluções para melhorar a percepção do usuário em ambientes computacionais de reutilização.

1.3 - Metodologia de Pesquisa

Não foi realizada uma revisão quasi- sistemática, mas sim a utilização de motores de busca e a verificação de artigos de conferências. Inicialmente, foi investigado o conceito de colaboração na literatura. No escopo de Trabalho Cooperativo Suportado por Computador, foi feita uma busca por artigos que descrevessem linguagens para representar colaboração ou cooperação nas bases de dados científicas IEEE, Scopus e ACM, com o objetivo de identificar mecanismos para expressar colaboração/cooperação. A busca utilizou os termos *collaboration*, *collaborative*, *coordination*, *cooperation* e *workflow*. Os artigos que continham uma forma para representar colaboração/cooperação em seus resumos foram selecionados para leitura. Sempre que possível, utilizamos a fonte original da linguagem na sua versão atual. Além disso, as seguintes conferências foram consideradas: Computer-supported

Cooperative Work, CSCW (2012, 2011, 2010, 2008, 2006, 2004, 2002, 2000, 1998, 1996, 1994, 1992, 1990); Computer Supported Cooperative Work in Design, CSCWD, (2012, 2011, 2010, 2009, 2008, 2007); European Conference on Computer-Supported Cooperative Work, ECSCW (2011, 2009, 2007, 2005, 2003, 2001, 1999, 1997, 1995, 1993, 1991, 1989). Cabe ressaltar que em alguns anos, algumas conferências não aconteceram, caso da ECSCW 2010, e em outras, pesquisamos nas mais recentes, caso da CSCWD.

Já no escopo de Reutilização de Software, similarmente como foi a busca por linguagens, foram feitas buscas nas bases de dados científicas IEEE, Scopus e ACM, utilizando as palavras: *software reuse*, *framework instantiation*, *software process*, *collaborative system, collaborative approach.* Também foram consideradas as conferências: *International Conference on Model Transformation* (ICMT, anos de 2010, 2011 e 2012), *IEEE International Conference and Workshops on the Engineering of Computer Based Systems* (ECBS) 2012, *International Conference on the Design of Cooperative Systems* (COOP) 2012, *Symposium on Engineering Interactive Computing Systems* (SIGCHI), *International Workshop on Model Driven Web Engineering* (MDWE, anos 2006, 2007), *International Conference on Web Engineering* (ICWE, anos 2008, 2009, 2010). Nosso objetivo aqui não foi cobrir todas as conferências, mas obter material suficiente para ajudar e argumentar as pesquisas, por isso, algumas conferências não foram consultadas.

O objetivo foi encontrar artigos que abordassem reutilização de software no caminho de implementar colaboração nas áreas de Instanciação de Framework, Linha de Produtos de Software (LPS), Cadeias de Transformação (MDD) e *Staged Configuration*. Sobre a linguagem RDL, foram estudados os seus conceitos, o seu ambiente de desenvolvimento e sua ferramenta de execução monousuário (OLIVEIRA *et al*., 2011). E por último, foi preciso entender a notação BPMN para fazer o mapeamento RDL e CollabRDL para BPMN, com o objetivo de utilizar um ambiente de execução multiusuário BPMN para executar processos expressos em CollabRDL.

1.4 - Organização da Dissertação

Esta dissertação está organizada em sete capítulos. Este capítulo apresentou a motivação para este trabalho, seus objetivos, geral e específicos, prosseguiu com a metodologia de pesquisa, e termina com esta organização de texto.

O Capítulo 2, Fundamentação Teórica, caracteriza colaboração evidenciando os seus conceitos e requisitos, a reutilização de software vista como um processo, descreve os pontos de extensão em frameworks orientados a objetos, e finaliza com a apresentação de cinco linguagens para representar colaboração.

O Capítulo 3, RDL- Uma Linguagem para instanciar frameworks, apresenta a estrutura da RDL com seus comandos para instanciação de frameworks orientados a objetos, descreve as suas fases de desenvolvimento e execução, e termina com a apresentação de um exemplo ilustrativo.

O Capítulo 4, Apresentação da CollabRDL, apresenta CollabRDL como uma extensão da RDL, descreve os seus comandos para atender aos objetivos deste trabalho, mostra como foi feito a implementação do ambiente de execução para CollabRDL baseado em uma ferramenta BPMN e descreve o mapeamento de CollabRDL para BPMN-Activiti.

O Capítulo 5, Avaliação da CollabRDL, expressa em CollabRDL alguns padrões de workflow e descreve a realização de uma prova de conceito.

O Capítulo 6, Trabalhos Relacionados, apresenta alguns trabalhos identificados na literatura em Reutilização de Software, estando organizados em Instanciação de Framework, Linha de Produtos de Software (LPS), Cadeias de Transformação (MDD) e *Staged Configuration.*

Finalmente, no Capítulo 7, listam-se as contribuições e as limitação deste trabalho, os trabalhos futuros e finaliza-se esta dissertação com as considerações finais.

CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA

Este capítulo caracteriza colaboração evidenciando os seus conceitos e requisitos, mostra a reutilização de software vista como um processo, descreve os pontos de extensão em frameworks orientados a objetos, apresenta cinco linguagens para representar colaboração e finaliza com as considerações finais.

2.1 - Introdução

O entendimento de colaboração requer o entendimento de *comunicação* e *coordenação*, por isso esses conceitos são vistos nas seções 2.2 e 2.3. Logo em seguida , na Seção 2.4, apresentamos então *colaboração.* Na seção 2.5, discutimos Processo de Reutilização de Software e, na 2.6, descrevemos frameworks orientados a objetos. Finalmente, na seção 2.7, apresentamos linguagens para representar colaboração pesquisadas na literatura com o objetivo de identificar caraterísticas que precisam estar presente em linguagens que se propõem a alcançar este objetivo.

2.2 - Comunicação

Comunicação é uma atividade coletiva e é guiada pelo processo chamado *grounding* que é definido como atualização do conhecimento comum (CLARK e BRENNAN, 1991). No diálogo entre pessoas é preciso existir um propósito para que o diálogo aconteça. Um exemplo é quando um desenvolvedor de uma equipe de projeto de software tem uma dúvida sobre um requisito e procura esclarecê-la com o analista de negócios. O propósito é esclarecer a dúvida, sendo que ao final do diálogo certamente o conhecimento comum ficou mais rico, o analista de negócio possuirá mais informações para melhorar a sua especificação de requisitos e o desenvolvedor agora tem as informações necessárias para desempenhar as suas atividades.

A comunicação entre duas pessoas pode ser afetada por oito restrições de acordo com o meio utilizado para realizar a comunicação (CLARK e BRENNAN, 1991). *Co-presença* é quando os participantes compartilham um ambiente físico, conversa face a face. *Visibilidade* caracteriza o cenário onde os participantes podem se ver, conversa

face a face e videoconferência são exemplos. Quando os participantes podem ouvir um ao outro, denominamos como *Audibilidade*, exemplos são face a face e telefone. *Contemporaneidade* para quando um receptor B recebe de A "no mesmo tempo". *Simultaneidade* é o caso particular de contemporaneidade onde os canais de recebimento e envio são diferentes, possibilitando interrupções para ajudar no entendimento da mensagem, que é o caso face a face e telefone. Para caracterizar uma *sequência no diálogo*, isto é, em uma conversa face a face, o diálogo segue uma sequência, quando não acontece a interrupção por outras pessoas. Por fim, características de *rever* as mensagens trocadas com possibilidade de *revisá-las* antes de enviá-las, que é o caso das mensagens eletrônicas (e-mail, SMS).

A Internet passou a ser um poderoso meio de comunicação possibilitando o surgimento de novas ferramentas para facilitar a comunicação entre pessoas e "coisas" tais como chat, mensagens eletrônicas, videoconferência, redes sociais e telepresença. No escopo de negócios eletrônicos, o OASIS (*Organization for the Advancement of Structured Information Standards*, Organização para o Avanço de Padrões Estruturados de Informação, http://www.oasis-open.org) é responsável pelo desenvolvimento do ebXML que é um pacote modular para comércio eletrônico baseado no XML (eXtensible Markup Language). XML tem um formato de texto flexível especialmente adequado para a troca de informações de estruturas diferentes por meio da internet.

O conceito de *comunicação*, desde sua forma mais simples, face a face entre pessoas, e apoiada por ferramentas computacionais funcionando sobre a internet para pessoas e também para objetos, nos situa para entender a importância da comunicação quando o objetivo é trabalhar em equipe. Neste cenário, a contribuição deste trabalho fará uso de comunicação face a face e, de forma mais elaborada, mediada por computador.

2.3 - Coordenação

O conceito de coordenação é outro aspecto importante, e é definido como *gerenciamento das dependências entre as atividades* (MALONE e CROWSTON, 1994). MALONE e CROWSTON (1990) exploraram o conceito de coordenação e chegaram a conclusão que coordenação é composta por objetivos, atividades, atores, e

interdependências, e é preciso também especificar processos para unir as atividades a atores gerenciando as interdependências para alcançar os objetivos. Objetivos são decompostos em atividades, aqui podemos identificar os processos *identificar objetivos* e *decompor objetivo em atividades*. Em seguida, mais dois processos, *selecionar os atores* e *associar atores a atividades*. Ao associar um ator a uma atividade é preciso considerar as interdependências entre as atividades, sendo portanto necessário os processos de *alocação de recursos* e *sincronização de atividades*, por exemplo.

MALONE e CROWSTON (1994) listam algumas dependências: *recursos compartilhados, associação de tarefas, relações de produtor e consumidor, restrições de prerrequisitos, transferência, usabilidade, projeto participativo, restrições de simultaneidade, e tarefa/subtarefa*. Em seguida, iremos fazer alguns comentários sobre elas sempre relacionando-as com *colaboração*, que será tratado com mais ênfase na seção seguinte.

No escopo da engenharia de software, em projetos de desenvolvimento de software, podemos identificar muitos recursos necessários para que sistemas sejam produzidos (FUGGETTA, 2000). O primeiro e mais importante é a pessoa, qualificada, bem treinada e alinhada com o ambiente de desenvolvimento, que pode ser um *recurso compartilhado* entre projetos. Assim, uma possibilidade é dividir o tempo de dedicação de uma pessoa em dois ou mais projetos. Outro exemplo é o compartilhamento de hardware, normalmente um computador é alocado para um membro da equipe, mas existem casos em que há o compartilhamento, cenário comum em laboratórios de pesquisa nas instituições de ensino onde alunos têm dedicação parcial nos projetos.

No mesmo sentido de alocar pessoas para vários projetos ao longo do tempo, tarefas devem ser delegadas para pessoas em um espaço de tempo. Cabe aqui ressaltar que ao *associar uma tarefa a uma pessoa*, estamos de certa forma solicitando uma cooperação, ao contrário, quando delegamos uma atividade a duas ou mais pessoas, há um claro incentivo à *colaboração*. Caso coloquemos como um objetivo a realização da tarefa e a pessoa que a recebeu interagir com outras no sentido de realizá-la, pelo fato de que as outras pessoas não estão associadas a mesma, ou seja não compartilham do objetivo comum, a ajuda se dará de forma *cooperativa*. Por outro lado, caso coloquemos como objetivo comum a construção do software, mesmo ao associar uma pessoa a uma

atividade, esta pode, mesmo sem incentivo, buscar colaboradores, ou seja, a ajuda de pessoas que estão no mesmo projeto de software.

As relações de *produtor e consumidor entre atividades* é caracterizada quando o(s) produto(s) gerado(s) por uma atividade produtora é(são) utilizado(s) como entrada na atividade consumidora. Claramente há a relação de *cooperação* entre as atividades produtora e consumidora. A atividade produtora tem como objetivo gerar um ou mais produtos que serão utilizados pela atividade consumidora (*transferência*), isto significa que a atividade produtora coopera com a atividade consumidora, pois só desta forma a atividade consumidora poderá ser realizada. Além disso, a atividade produtora precisa terminar antes da atividade consumidora iniciar (*restrições de prerrequisitos*), e é necessário pensar em padrões de *usabilidade* já que a atividade consumidora precisa usar os produtos gerados pela atividade produtora.

Uma outra característica presente em atividades produtoras e consumidoras com uma abordagem *colaborativa* é o *projeto participativo*, ele se dá na especificação dos produtos gerados. Por exemplo, portlets são mini aplicações web interativas, local ou remota ao portal, que processam fragmentos de marcação que o portal pode agregar em uma página (BELLAS, 2004, HEPPER, 2008). Um portal é um contêiner onde os portlets podem ser hospedados. A OASIS, com o objetivo de fazer com que os portlets locais pudessem ser consumidos como serviço, ou seja, tornarem-se remotos, criou o padrão WSRP em setembro de 2003. Com este padrão os portlets podem ser consumidos independente das plataformas dos portais produtor e consumidor. Na prática, portais construídos sobre a plataforma J2EE podem consumir portlets de portais da plataforma .NET e vice-versa. A OASIS desenvolve padrões de forma colaborativa com muitas empresas de Tecnologia da Informação e Comunicação.

Atividades podem ser realizadas sequencialmente ou em paralelo, essa dependência é denominada de *restrições de simultaneidade*, o caso das atividades produtoras e consumidoras apresentadas anteriormente é um exemplo de atividades que precisam acontecer sequencialmente, uma após a outra ao longo do tempo. Atividades em paralelo são aquelas que podem ser realizadas ao mesmo tempo, preferencialmente por pessoas diferentes, na linha do tempo.

E finalizando as dependências entre atividades, temos a relação entre *tarefa/subtarefa*. É comum ouvirmos a frase "vamos fazer por partes", essa expressão vem da necessidade, que pode ser entendida por estratégia, de dividirmos um objetivo em partes menores, que podemos chamar de atividades ou tarefas. Desta forma, o objetivo pode ser visto como um conjunto de atividades com um grau de complexidade menor, que se realizadas nos levarão ao objetivo inicial, essa estratégia é conhecida como *top-down*. Por outro lado, é possível também investigar atividades que estão sendo feitas para elaborar um novo objetivo, estratégia *botton-up*. Um exemplo seria a criação de um novo programa de pós-graduação multidisciplinar (por exemplo Bioinformática) baseado nas atividades de pesquisa realizadas por programas tradicionais.

2.4 - Colaboração

Colaboração pode ser entendida como *o ato de trabalhar em conjunto* (SOLIMAN *et al.*, 2005). Essa definição é tão genérica que SOLIMAN *et al.* (2005) definiram ingredientes para auxiliar na sua identificação: *pessoas, espaço compartilhado, tempo, objetivo comum, foco no objetivo, linguagem comum, conhecimento na área do objetivo, e interação.*

Pessoas podem trabalhar em conjunto ou isoladamente, sendo preciso pelo menos duas trabalhando em conjunto para existir colaboração. *Espaço compartilhado* pode ser um local com objetos reais tais como quadro, tabelas, canetas etc, ou um espaço virtual onde objetos reais e os seus manuseios são simulados. Ambos os espaços fazem uso de canais de comunicação (e-mail, telefone, etc), cada um com o seu ponto de interação (texto, fala, etc). Ferramentas virtuais são muito utilizadas em ambientes de trabalho distribuídos fisicamente, elas criam um cenário onde pessoas distantes podem trabalhar em conjunto.

Colaboração pode ser *síncrona* quando o canal de comunicação é síncrono (por exemplo, telefone, videoconferência), ou *assíncrono* (e-mail, texto). Esses recursos são utilizados para alcançar o objetivo de trabalho, por exemplo, gerentes, na maior parte das vezes, preferem agendar reuniões por e-mail, enquanto a realização da mesma se dá

por fala face a face. Nesse sentido, pessoas precisam investir parte do seu *tempo* em colaboração independente dos recursos utilizados.

Pessoas precisam estar unidas por um *objetivo comum* que contenha um elemento de novidade para os participantes. Por exemplo, em uma LPS – sigla para Linha de Produto de Software, Seção 2.5 desta dissertação, o produto a ser construído é o objetivo comum, já em um processo de reutilização de um framework (FILHO *et al*., 2004, OLIVEIRA *et al.*, 2007), o novo modelo de framework ou software representa o objetivo comum. As pessoas precisam *focar no objetivo comum* pois existem muitos fatores que podem atrapalhar as atividades que as levam a alcançá-lo. Esses fatores podem ser de origem tecnológica, social, ou motivos pessoais (KOLLOCK, 1998).

Colaboração requer uma *linguagem comum* para comunicação entre as pessoas. Por exemplo, cada país possui pelo menos uma língua oficial (Português, Inglês, Francês), e na engenharia de software modelos são construídos usando linguagens, exemplo UML. É por meio delas que as pessoas podem receber e passar informações com o objetivo de realizar as atividades para alcançar o objetivo comum.

Pessoas precisam ter *conhecimento na área do objetivo,* quanto melhor e detalhado for este conhecimento, melhor será o trabalho de realizar as atividades que estão alinhadas com o objetivo comum. Finalmente, *interação* é definida como *uma ação realizada por pessoa que estimula uma outra ação em outra pessoa*. Então, interação usa os outros sete ingredientes para alcançar o objetivo comum.

Uma outra forma de entender colaboração é comparando-a com cooperação. Trabalho cooperativo é constituído por processos de trabalhos que estão relacionados com um propósito, isto é, processos relacionados com o objetivo de produzir um produto ou serviço (BANNON e SCHMIDT, 1989). Isso implica que um trabalho cooperativo é planejado e coordenado e que existe a ideia de pessoas trabalhando em conjunto para atingir um objetivo (produto ou serviço). Contudo, as pessoas não precisam interagir para cooperar umas com as outras, isso significa que um grupo de pessoas pode receber uma determinada tarefa e executá-la isoladamente, isto é, sem interação com outros grupos, em espaços diferentes e sem comunicação durante a sua realização, contribuindo de forma cooperativa e pontual. Podemos então concluir que

colaboração é um tipo de cooperação onde necessariamente há a interação entre pessoas de forma organizada e planejada para alcançar um objetivo comum.

Por outro lado, FUKS *et al.* (2007) exploram o modelo 3C que foi originalmente apresentado por ELLIS *et al*. (1991). O modelo 3C de FUKS *et al*. (2007) é composto por **C**omunicação, **C**oordenação e **C**ooperação. Coordenação faz o link entre Comunicação e Cooperação com o objetivo de fomentar a colaboração, e Cooperação, no contexto de groupware (ELLIS *et al*., 1991) , *é um conjunto de operações durante uma sessão em um ambiente de trabalho compartilhado*. Nesse sentido, o modelo 3C ressalta a importância de *awareness*, isto é, a percepção/consciência das pessoas em relação às suas atividades e das outras pessoas em um contexto de colaboração.

A Figura 2.2 foi criada com o intuito de resumir e de certa forma caracterizar colaboração para os nossos propósitos; ao centro então presentes os conceitos de *comunicação*, pois sem comunicação implica no isolamento de pessoas ou "coisas", impossibilitando então qualquer tentativa no sentido de colaboração. Também ao centro temos juntamente com a comunicação o conceito de *coordenação*, isso se justifica pois até em pequenos diálogos é preciso que as pessoas cheguem a um consenso de quando é a vez de falar e quando é a vez de ouvir. *Cooperação*, também ao centro indica que pessoas podem trabalhar em atividades isoladamente. Logo fica entendido que comunicação, coordenação, e cooperação são imprescindíveis para a existência de colaboração.

Circundando os conceitos de comunicação, coordenação, cooperação, e por isso também baseados neles, estão os ingredientes de colaboração apresentados anteriormente neste tópico. Finalmente, de forma transversal, temos o conceito de processo iniciando do mais simples no núcleo, sem a necessidade de formalização, e de forma mais complexa em colaboração, isso porque para garantir colaboração é preciso expressar as relações de forma que o trabalho colaborativo realmente aconteça, garantindo a inclusão de todos os conceitos e ingredientes.

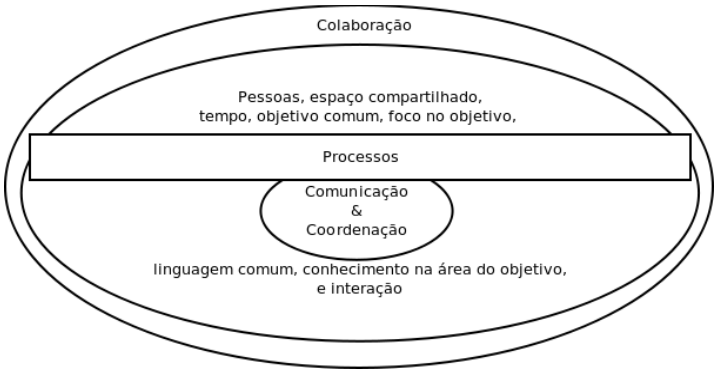


Figura 2.1: Caracterização de colaboração

2.5 - Processo de Reutilização de Software

FUGGETTA (2000) define processo de software *como o conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, implantar e manter um software*. Portanto, existe a necessidade de definir mecanismos para que as pessoas possam colaborar de forma harmoniosa em atividades para atingir os objetos do processo de construção do software. Por exemplo, em um projeto de desenvolvimento várias atividades precisam ser delegadas para profissionais com habilidades específicas, tais como administrador de banco de dados, programadores e analistas, denominados de papéis. Essa abordagem não muda quando o desenvolvimento é através da reutilização de software já desenvolvido com o objetivo de ser reutilizado, que é o caso dos frameworks.

A reutilização de software é uma prática importante no processo de desenvolvimento de software, *consiste no uso de um software existente ou conhecimento de software para construir um novo software com o objetivo de melhorar a qualidade e aumentar a produtividade* (FRAKES e KANG,. 2005). Estudos experimentais indicam que a reutilização de software contribui para a melhoria da qualidade do software e do aumento da produtividade na sua construção. Como exemplo, em (LIM, 1994), dois estudos de caso foram realizados, o primeiro reduziu em 51% os defeitos (melhoria de qualidade) e aumentou em 57% a produtividade, o segundo obteve 24% na redução dos defeitos e 40% no aumento de produtividade. No

entanto, em outros estudos, FRAKES e SUCCI (2001) não conseguem provar que reutilização aumenta a produtividade.

Historicamente, o processo de utilização de um framework Model-View-Controller foi descrito por meio de CookBook (KRASNER e POPE, 1988), utilizando linguagem natural para documentar os passos para a sua instanciação. *Hook* foi definido por FROEHLICH *et al.* (1997) e pode ser visto como uma evolução de CookBook já que possui uma descrição estruturada, tais como nome que identifica o *hook*, tipo, requisito, área etc, embora ainda mantenha a linguagem natural para a descrição. E, MDD, desenvolvimento guida por modelo, é uma abordagem de desenvolvimento de software onde o sistema é construído a partir da especificação de modelos mais abstratos, próximos do entendimento humano, que são transformados de forma semiautomática em modelos mais específicos, terminando com a geração de código (LANO, 2009).

O Processo de Reutilização de Software se apresenta com dois objetivos, o de desenvolver para reutilização e o de desenvolver com reutilização (DE ALMEIDA *et al.*, 2005). O primeiro foca na construção de artefatos reutilizáveis (ARANGO e PRIETO-DIAZ, 1991), e o segundo na construção de aplicações , também conhecida como engenharia de aplicação. Inicialmente e até 1998, a reutilização estava focada na engenharia de domínio (DE ALMEIDA *et al.*, 2005). Após esse período, o foco mudou para Linha de Produtos de Software, LPS.

LPS *é um conjunto de sistemas que compartilham características comuns e gerenciáveis que satisfazem as necessidades de um específico segmento de mercado ou missão que são desenvolvidos a partir de um conjunto comum de ativos principais de forma predeterminada* (NORTHROP, 2002). Ativos principais ou núcleo de ativos, do original em inglês *core assets*, é a forma abstrata para se referir à arquitetura, aos componentes de software reutilizáveis, modelos do domínio, declaração de requisitos, documentação e especificação, modelos de desempenho, cronogramas, orçamentos, planos de testes, planos de trabalho, e descrições de processos. Ou seja, tudo aquilo que se pode especificar como uma característica pertencente a um determinado domínio de sistemas. Um sistema gerado a partir de uma LPS é construído seguindo um plano de reutilização que documenta como os ativos gerenciáveis podem ser reutilizados.

Embora o caso ótimo aconteça quando o sistema faz uso somente dos ativos predeterminados, outros podem ser adicionados para atender a objetivos específicos, podendo no futuro serem disponibilizados como ativos compartilhados (*core assets*).

O SEI, Software Enginnering Institute, estabelece três atividades essenciais para uma LPS, *desenvolvimento de ativos principais*, *desenvolvimento do produto*, e *gerenciamento organizacional*:

> 1 - O *desenvolvimento de ativos principais* precisa considerar as informações sobre as restrições de produto no sentido de especificar as partes comuns e as diferentes dos produtos que fazem parte de uma linha de produtos. Um exemplo é uma linha de produtos que estabelece que a camada de persistência pode ser em banco de dados relacional, XML, ou arquivo texto. Assim um sistema gerado para ser hospedado em um computador pode preferir usar um banco de dados relacional, enquanto que um sistema para um dispositivo móvel pode preferir uma forma de persistência em texto. Neste exemplo estamos considerando as restrições de hardware onde o aplicativo será executado. Para atingir esse objetivo, o arquiteto de software faz uso de estilos, padrões e frameworks para produzir os ativos com essas possibilidades de escolha. Além disso, é preciso tratar as restrições de produção (comercial, atendimento a um padrão específico), escolher a estratégia de produção que pode ser *top down* - quando se inicia o desenvolvimento a partir de um conjunto de ativos preliminares - ou *bottom up* - quando examina os sistemas para a produção de componentes genéricos. Após isso, defini-se o escopo da linha de produto, especifica os ativos principais, e cria um plano de produção para descrever como um produto deve ser gerado a partir deles.
>
> 2- Na atividade *desenvolvimento do produto* faz-se uso do plano de produção, dos ativos reutilizáveis, e do escopo da linha de produtos criados pela atividade *desenvolvimento dos ativos principais*. Na construção de um novo produto seguindo o plano de produção, pode surgir a necessidade de desenvolver outros ativos tanto para o sistema em produção, como para os outros que virão, tornado assim necessário o

desenvolvimento de um novo ativo a partir de uma necessidade particular de um produto gerado que até então não havia sido descoberta. Uma outra situação, é a alteração de um ativo para aceitar novas funcionalidades.

3- A atividade de *gerenciamento organizacional* é crítica, podendo levar ao sucesso ou ao fracasso a linha de produtos. É preciso gerenciar bem as atividades e a relação entre os grupos de desenvolvedores de ativos principais e os desenvolvedores de produtos. O gerente deve manter sob controle a evolução dos ativos e garantir que os produtos gerados estão seguindo o plano de produção.

Uma LPS é um bom exemplo de reutilização de software, existem os conceitos de desenvolver para reutilização e desenvolver com reutilização. O primeiro conceito está presente na atividade de desenvolvimento de ativos principais, o segundo está na atividade de desenvolvimento do produto, existindo também o direcionamento para plano de reutilização com forte viés para o gerenciamento das atividades. O SEI é referência em LPS, este Instituto criou um guia (CHASTEK e MCGREGOR, 2002) para ajudar na criação de plano de produção de linha de produtos, discute o gerenciamento de projetos na abordagem de LPS (CLEMENTS *et al.*, 2005), e descreve os conceitos necessários para criar ativos incluindo variabilidades (BACHMANN e CLEMENTS, 2005). Finalizando, LPS foi muito bem aceita pela indústria, na página da SPLC, Conferências sobre Linha de Produto de Software (SPCL, 2012), são relatados 18 casos de sucesso.

LPS também é na sua essência um trabalho colaborativo (MENDONÇA *et al.* 2007). O produto gerado precisa atender a diferentes visões, por exemplo, de segurança, projeto de interface, limitações de hardware, portabilidade, e custo de venda. Logo, a decisão se um ativo vai ou não pertencer ao produto gerado precisa ser uma decisão feita por pessoas que defendem visões diferentes e que precisam chegar a um consenso. Nesse contexto, as pessoas precisam, em conjunto, e visando um objetivo comum, o software a ser gerado pela LPS, interagir seguindo um planejamento, o plano de produção.

Uma outra abordagem colaborativa é a identificação e desenvolvimento de ativos principais. Um ativo principal, pela sua definição, é para ser reutilizado em vários

sistemas pertencentes a um domínio. Surge então a necessidade de especificar um ativo que atenda a diferentes perspectivas dentro de um domínio. Por exemplo, em uma LPS de dispositivo móvel, um ativo de comunicação precisa ser modelado para atender a diversos requisitos de hardware, segurança, custo, e protocolos de comunicação. Assim, a especificação desse ativo precisa considerar a opinião de vários especialistas, surgindo então a forte necessidade de colaboração entre eles.

2.6 - Framework

Um *framework* é uma aplicação incompleta que pode ser reutilizada para produzir aplicações customizadas (MOHAMED e SCHMIDT, 1997), tradicionalmente, baseado no paradigma orientado a objetos (OO), no início com as linguagens Smalltalk, Lisp e C++, com várias iniciativas, de interface gráfica a compiladores (JOHNSON e FOOTE, 1988). Segundo a TIOBE Programming Community (TIOBE, 2012), Java e C# são as linguagens mais utilizadas, seguidas de C++. Frameworks podem ser construídos utilizando outros frameworks, JEE e .Net (Java EE, 2012, DOT NET, 2012) são exemplos destes frameworks tipo guarda-chuva que oferecem uma infraestrutura para desenvolvimento de aplicações.

Os pontos de extensão de um framework são chamados de *hot-spots* (PREE e SIKORA, 1997) e é através deles que os desenvolvedores produzem aplicações. *Hot-spots* em frameworks orientados a objetos podem ser criados por instanciação, por herança, por implementação de interfaces, ou configuração (BECK, 2007). Reutilização via instância acontece quando criamos um objeto a partir de uma classe pertencente a um framework e acessamos os seus métodos. Por herança, quando declaramos uma nova classe filho a partir de uma classe existente pai. Dessa forma a classe filho pode ser referenciada como uma classe pai, herdando os seus comportamentos e atributos. Por outro lado, os comportamentos da classe pai, que foram herdadas pela classe filho, podem ser redefinidos, criando uma identidade na classe filho, justificando a sua criação. Interfaces são utilizadas para limitar o acesso aos serviços de uma classe. Uma classe pode implementar muitas interfaces, as classes que irão utilizá-las podem, portanto, escolher a interface correta para comunicação. E por último, a configuração

pode acontecer por definição em arquivos, por passagem de parâmetros para objetos do framework ou, mais recentemente, por anotação.

A Tabela 2.1 lista os passos necessários para criarmos um exemplo de reutilização que será utilizado para ilustrar alguns casos de reutilização em frameworks orientados a objetos. Na versão Indigo do Eclipse, uma IDE para desenvolvimento de sistemas, seguindo os passos da tabela, iremos ter como resultado um projeto de software em Java no ambiente Eclipse, que será referenciado a partir de agora como Pteste. Outros códigos exemplos podem ser criados somente mudando a escolha do template no passo 4.

Tabela 2.1: Criando um *Plug-in* em Java para Eclipse (versão Indigo)

1 – No item de menu File, escolha *new plug-in project*;
2 – Entre com o nome do projeto (por exemplo *Pteste*)
3- Aceite as sugestões;
4- Escolha um template (neste caso foi escolhido *plug-in com um editor)*;
5- Ao final, o projeto será criado;
6- Com o botão direito do mouse sobre o projeto criado, escolha executar como uma aplicação Eclipse, uma instância do ambiente Eclipse será criado com o *plug-in* recém gerado;
7 – Na instância do ambiente Eclipse, crie um arquivo .xml e com o botão direito sobre ele, escolha abrir com o nome do editor especificado no passo 2 (*PTeste*);

Reutilização por instância: O Código 2.1 ilustra um código em Java de reutilização do framework GEF (*Graphical Editing Framework da Fundação Eclipse*). O método createGraphViewer cria uma instância da classe do GEF *ScrollingGraphicalViewer* que pertence ao pacote *org.eclipse.gef.ui.parts* e manipula os seus métodos. A reutilização por instância se baseia nesse princípio, instanciar e manipular, e em alguns casos passados a diante, caso deste exemplo, como está expresso em *editDomain.addViewer(viewer)*.

Código 2.1: Trecho de código em Java para reutilização do framework GEF (*The Eclipse Foundation*)

```
private void createGraphViewer(Composite parent) {
      ScrollingGraphicalViewer viewer = new
                              ScrollingGraphicalViewer();
      viewer.setRootEditPart(new
                      ScalableFreeformRootEditPart());
      viewer.createControl(parent);
      viewer.getControl().
                      setBackground(ColorConstants.white);
      viewer.setEditPartFactory(new
                                  BlockEditPartFactory());
      viewer.setContents(contents);
      editDomain.addViewer(viewer);
      getSite().setSelectionProvider(viewer);
}
```

Reutilização por herança: *Herança* é um conceito largamente difundido no paradigma orientado a objetos e consiste em expressar que uma nova classe irá incluir na sua definição o comportamento de seus ancestrais. Não está no escopo deste trabalho detalhar os conceitos deste paradigma mas, no foco de reutilização de framework, é ilustrado um exemplo no Código 2.2. Neste, em Java, a classe XMLEditor do projeto Pteste herda de TextEditor do pacote org.eclipse.ui.editors.text, define o seu construtor (XMLEditor) fazendo referência para o construtor de *TextEditor* (comando *super*), e redefine o método *dispose*.

Código 2.2: Reutilização por herança em Java (Fundação Eclipse)

```
public class XMLEditor extends TextEditor {
 private ColorManager colorManager;
 public XMLEditor() {
      super();
      colorManager = new ColorManager();
      setSourceViewerConfiguration(new
                      XMLConfiguration(colorManager));
      setDocumentProvider(new XMLDocumentProvider());
 }
 public void dispose() {
      colorManager.dispose();
      super.dispose();
 }
}
```

Reutilização por interface: Interface é um ou mais serviços (métodos) que as classes que a implementam precisam oferecer. Na Código 2.3, trecho de código em java, a interface IWhitespaceDetector do pacote org.eclipse.jface.text.rules especifica o método isWhitespace e a classe XMLWhitespaceDetector implementa este serviço. Classes que implementam interfaces do framework são para serem utilizadas por outras classes do framework que esperam utilizar os serviços implementados.

Código 2.3: Reutilização por interface em Java (The Eclipse Foundation)

```
public class XMLWhitespaceDetector implements
                                    IWhitespaceDetector{
      public boolean isWhitespace(char c) {
            return (c == ' ' || c == '\t' || c == '\n' ||
                                                c == '\r');
      }
}
```

Reutilização por configuração: Pode ser via arquivo XML, código ou anotação. Primeiro serão apresentados alguns conceitos do ambiente Eclipse, depois será mostrado um trecho de arquivo de configuração para expressar reutilização. *Workbench* é a área de trabalho para o desenvolvimento de software no ambiente Eclipse e pode ter uma ou mais perspectivas. Uma perspectiva define um conjunto inicial de *layout* e *views* na janela do Workbench (Eclipse documentation, 2012). Views são partes bem definidas de visualização para atender a certas necessidades, um exemplo é o Project Explorer.

O Código 2.4 ilustra a reutilização expressa em formato XML para criar uma nova perspectiva em um ambiente Eclipse. A marcação *extension point* define o tipo de extensão a configurar, neste caso, de uma perspectiva. Aninhada a esta, podem ter marcações *perspective.* Neste exemplo, foi expresso somente uma, e ela indica a classe que implementa a perspectiva, o ícone, um id e, finalmente, o nome da perspectiva que aparece no workbench, conforme Figura 2.2.

Código 2.4: Reutilização por configuração via arquivo XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
<extension point="org.eclipse.ui.perspectives">
      <perspective
            class="myClass.perspectives.MyPerspective"
            icon="myIcon.gif"
            id="MyPerspective"
            name="My Perspective">
      </perspective>
</extension>
</plugin>
```

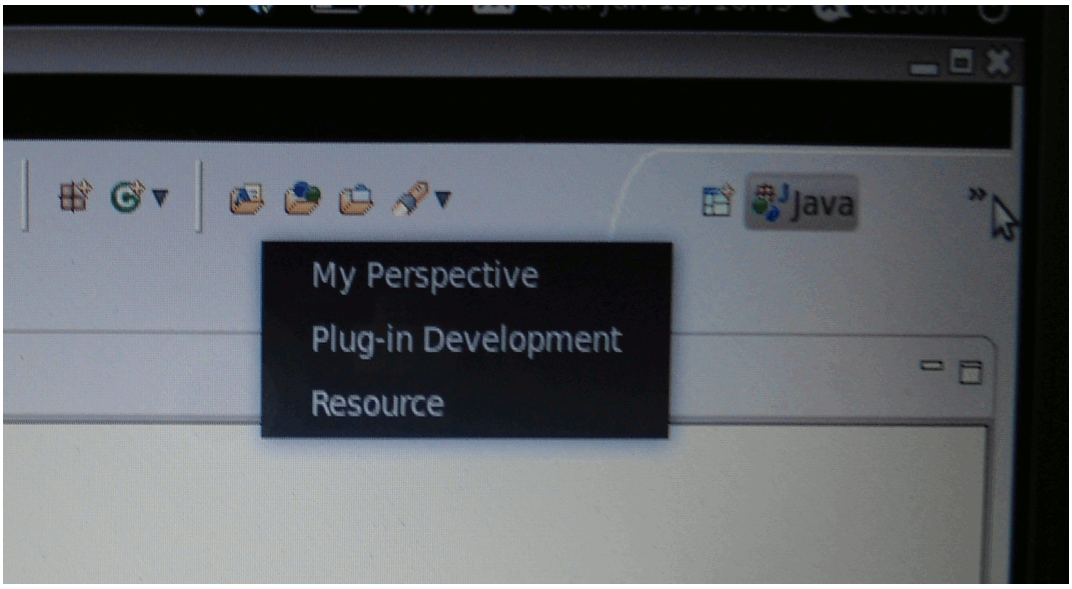


Figura 2.2: A perspectiva criada pela configuração descrita na Tabela 2.5.

Um outro exemplo de reutilização por configuração é o por *anotação*. É feito no código e facilita muito o entendimento, já que expressa em código de forma mais limpa e simples o que é expresso em arquivos XML. Para ilustrar este caso, será utilizado o desenvolvimento com *portlets*. O Código 2.5 mostra um arquivo .jsp que em tempo de execução produz uma página que pode ser entendida pelos browsers web (é comum que seja gerado código HTML). Este código possui código HTML (<form) com código da biblioteca *portlet* (<portlet:actionURL var="sendMessage">). Esta visualização ainda é feita por linguagem de marcação, foi explicada aqui somente para facilitar o entendimento do conceito de anotação a seguir.

Código 2.5: Reutilização por anotação (Visão)

```
<%@ taglib uri="http://java.sun.com/portlet_2_0"
prefix="portlet" %>
<portlet:defineObjects />
This is the <b>SendEmail</b> portlet.
<portlet:actionURL var="sendMessage">
         <portlet:param name="javax.portlet.action"
value="actionSendMessage" />
</portlet:actionURL>
<form action="<%=sendMessage%>" method="POST">
          <trecho de código omitido>
</form>
```

Caso a página HTML ilustrada acima seja exibida em um browser no lado cliente após uma requisição ao endereço em um servidor que hospeda o portlet, o usuário poderá enviar um *request* com uma *action* com valor *actionSendMessage*. E o servidor poderá responder a esta requisição. Uma forma é mostrada no Código 2.6, ilustrando a anotação @ProcessAction (name="actionSendMessage"), que expressa que o método handleActionSendMessage deverá ser chamado para atender a requisição actionSendMessage. Este por sua vez pode fazer alguns processamentos e repassar para o médodo doSuccess anotado com @RenderMode (name = "print"), que renderiza uma página para ser enviada para o cliente como resposta.

Código 2.6: Reutilização por anotação (Controle)

```
public class SendEmail extends GenericPortlet {
    @RenderMode (name = "print")
    public void doSuccess(RenderRequest renderRequest,
                          RenderResponse renderResponse)
                throws IOException, PortletException {
        include(successJSP , renderRequest, renderResponse);
    }
    @ProcessAction (name="actionSendMessage")
    public void handleActionSendMessage(ActionRequest
                     request, ActionResponse response)
                throws PortletException, IOException {
      <trecho de código omitido>
       response.setPortletMode(new PortletMode("print"));
    }
}
```

2.7 - Linguagens para representar colaboração

Esta dissertação tem como objetivo apoiar o processo de reutilização de software colaborativo. Nesse sentido, uma pesquisa foi realizada no escopo de CSCW (Trabalho Cooperativo Apoiado por Computador) nas bases de dados científicas IEEE, Scopus e ACM para estudar linguagens que se propõem a oferecer este tipo de suporte. A busca utilizou os termos *collaboration*, *collaborative*, *coordination*, *cooperation* e *workflow*. Os artigos que continham uma forma para representar colaboração/cooperação em seus resumos foram selecionados para leitura. Sempre que possível, utilizamos a fonte original da linguagem na sua versão atual. Além disso, as seguintes conferências foram consideradas: Computer-supported Cooperative Work, CSCW (2012, 2011, 2010, 2008, 2006, 2004, 2002, 2000, 1998, 1996, 1994, 1992, 1990); Computer Supported Cooperative Work in Design, CSCWD, (2012, 2011, 2010, 2009, 2008, 2007); European Conference on Computer-Supported Cooperative Work, ECSCW (2011, 2009, 2007, 2005, 2003, 2001, 1999, 1997, 1995, 1993, 1991, 1989). O resultado da pesquisa foi o artigo *A survey of languages to represent collaboration as a means of designing CSCW facilities in RDL* (LUCAS *et al*., 2013), que descreve as linguagens CSDL, ebXML, BPMN, DCWPL, COCA e ThinkLets.

CSDL é a sigla em inglês que significa linguagem para projetar sistemas cooperativos (*Cooperative Systems Design Language*). Ela é uma linguagem orientada a objetos projetada para dar suporte ao desenvolvimento de sistemas cooperativos, partindo da especificação para a implementação. Em CSDL, um sistema cooperativo tem um ambiente compartilhado onde usuários podem interagir sincronamente e regras podem ser definidas para a coordenação das atividades (DE PAOLI e TISATO, 1994).

A ebXML é um pacote modular para negócios eletrônicos baseado em XML (eXtensible Markup Language). XML tem um formato de texto flexível especialmente adequado para a troca de informações com diferentes estruturas através da Internet. OASIS conduz o desenvolvimento do ebXML. O esquema de especificação de processos de negócios do ebXML usa um Perfil de Protocolo para Colaboração (CPP – *Collaboration Protocol Profile*) ou um Contrato de Protocolo de Colaboração (CPA - *Collaboration Protocol Agreement*) para suportar a execução de Colaboração de

Negócios, definido como um conjunto de atividades de negócios executando transações de negócios (unidade de trabalho atômica) entre parceiros de negócios ou partes da colaboração (OASIS, 2006).

A OMG também criou a especificação da BPMN e controla a sua revisão e melhoria contínua. BPMN é a sigla em inglês para *Business Process Model and Notation*, e seu principal objetivo é oferecer um padrão de linguagem visual fácil para fazer a ligação entre o projeto de processo de negócio e o processo de implementação, facilitando portanto o intercâmbio entre diferentes ferramentas (BPMN, 2011).

DCWPL é a sigla em inglês para Linguagem de Programa para Descrever Trabalho Colaborativo e foi desenvolvida para representar coordenação no desenvolvimento de aplicações para *groupware* (CORTES e MISHRA, 1996). DCWPL oferece facilidades para coordenação tais como artefatos, papéis, armazenamento, funções de coordenação, políticas, e gerenciamento de sessão, desacopladas de qualquer linguagem de programação tradicional.

COCA (*Collaborative Objects Coordination Architecture*) foi proposta para modelar e suportar colaboração através da Internet (LI e MUNTZ, 2000). COCA separa as políticas de coordenação da interface com o usuário, oferece uma linguagem de especificação executável para descrever políticas de coordenação e participantes na mesma colaboração, assumindo papéis em tempo de execução.

ThinkLets são usados para descrever a engenharia de colaboração que é o desenvolvimento de processos de colaboração repetíveis conduzido pelos próprios praticantes. BRIGGS *et. al.* (2003) definiram um *thinkLet* como *a menor unidade de capital intelectual necessário para criar um padrão repetível e previsível de colaboração entre as pessoas que trabalham para alcançar um objetivo*. Eles identificam cinco padrões de colaboração: Diverge; Converge; Organizar, Avaliar e Construir consenso. Um thinklet deve ter um nome, um padrão de colaboração associada a ele, a especificação das ferramentas usadas para apoiá-lo com suas configurações, e um *script* para descrever a sequência de eventos e instruções.

Finalmente, LUCAS *et al.* (2013) constataram que essas linguagens, independente do seu domínio de atuação e paradigma associado à sua estrutura, oferecem suporte aos conceitos de *papéis*, *paralelismo* e *atividades*.

2.8 - Considerações Finais

Este capítulo mostrou que é preciso haver comunicação, cooperação e coordenação para existir colaboração, sendo necessário pelo menos duas pessoas com um objetivo comum e conhecimento na área do objetivo, interagindo em um ambiente compartilhado e fazendo uso de pelo menos uma linguagem para comunicação. Esclareceu processo de software e processo de reutilização de software, evidenciando as características de uma LPS, mostrando que é importante definir mecanismos para que as pessoas possam colaborar para atingir os objetos do processo de construção de software. Mostrou a definição de frameworks e esclareceu os seus pontos de extensão: instanciação; herança; implementação de interfaces; ou configuração. E finalizou, apresentando algumas características das linguagens que se propõem a representar colaboração, concluindo que elas implementam os conceitos de papéis, paralelismo e atividades que serão utilizados nos próximos capítulos.

Este capítulo é importante para este trabalho porque mostrou que os novos comandos e ambiente de execução devem oferecer suporte à comunicação, cooperação e coordenação para dar suporte à instanciação de frameworks orientados a objetos em grupo, além de mostrar a necessidade de considerar os conceitos de papéis, paralelismo e atividades.

Finalmente, o conceito de *coordenação* esclareceu como a RDL, hoje executada sequencialmente e por um reutilizador, precisará se comportar quando passar a ser executada em paralelo por vários reutilizadores. De fato, podemos classificar a atual RDL e também a CollabRDL como linguagens coordenativas de acordo com a definição de coordenação: *gerenciamento das dependências entre as atividades* (MALONE e CROWSTON, 1994).

CAPÍTULO 3 - RDL

Este capítulo apresenta a estrutura da RDL com seus comandos para instanciação de frameworks orientados a objetos, descreve as suas fases de desenvolvimento e execução, apresenta um exemplo ilustrativo e finaliza com as considerações finais.

3.1 - Introdução

RDL (*Reuse Description Language*) é a sigla em inglês de Linguagem para Descrever Reutilização. RDL descreve o processo de especificação da instanciação de framework orientado a objetos. Este processo de reutilização é para fazer reutilização de software de forma explícita, representando as atividades de reutilização (OLIVEIRA *et al*., 2007, OLIVEIRA, 2001). RDL também é uma linguagem interativa onde o reutilizador – quem faz a instanciação de framework – precisa entrar com respostas sobre a reutilização, por exemplo, a execução de RDL gera a pergunta "Qual é o nome da nova classe?", o reutilizador pode responder algo parecido como "Pessoa". Os programas em RDL são executados em um ambiente chamado ReuseTool (OLIVEIRA *et al*., 2011).

A Figura 3.1 mostra a visão geral de utilização da RDL, adaptada de (OLIVEIRA *et al.,* 2011), em um cenário de reutilização de frameworks orientados a objetos. O primeiro passo, marcado com o número 1 na figura, indica que é preciso entrar com o modelo de classes do framework orientados a objetos no formato de aquivo .uml, sendo permitido um ou mais arquivos. Depois, passo 2, é preciso entrar com o programa RDL escrito para instanciar o framework, entrada do passo 1. O passo 3, também indicado na figura, mostra que um reutilizador precisa responder às perguntas de execução do programa RDL fazendo uso do seu conhecimento sobre o framework e sobre os requisitos da nova aplicação, indicado com o número 4. Finalmente o passo 5, mostra que a execução do programa RDL, juntamente com as respostas do reutilizador produzem o modelo de saída, o modelo da aplicação, uma instância do framework de entrada.

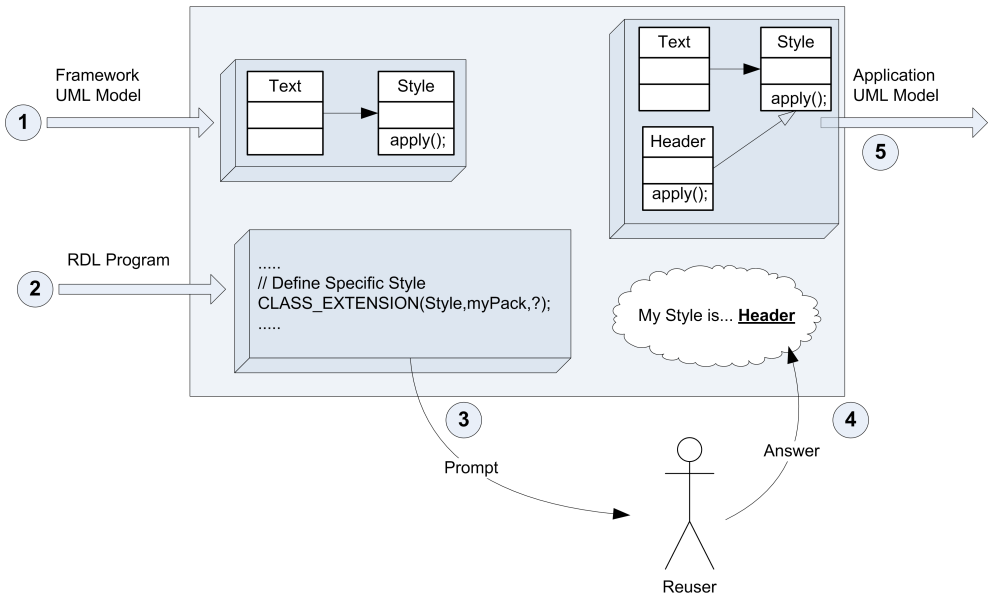


Figura 3.1: Visão geral de utilização da RDL

Já a Figura 3.2 ilustra um trecho da reutilização de um framework orientado a objetos modelado em UML, este exemplo é baseado no framework Shape, visto com mais detalhes em (OLIVEIRA *et al.,* 2011). O produto deste trecho de reutilização é a classe *Circle* que estende da classe *Shape* redefinindo os seus métodos *addConnectionAnchor*, *getFigure* e *getIcon* em um novo pacote chamado org.reusetool.myshapeseditor. A Figura 3.1, parte superior, mostra o trecho de programa escrito em RDL para guiar este processo de reutilização. A linha 1 descreve a criação do pacote org.reusetool.myshapeseditor, a linha 2 indica o início de um bloco de atividades que pode ser repetido em tempo de execução quantas vezes o reutilizador necessitar. Assim, ao executar está linha, ele receberá a pergunta “Criar outro shape?”, caso responda sim, ele entrará no bloco do comando LOOP. A linha 4 descreve a criação de uma nova classe que estende de *Shape* no pacote org.reusetool.myshapeseditor, que é para onde a variável *packA* aponta, e o parâmetro “?“ indica que em tempo de execução o reutilizador irá receber uma pergunta para entrar com o nome da nova classe, neste exemplo o reutilizador escolheu *Circle*. As linhas 6, 7 e 8 redefinem na classe apontada pela variável *shapeClass* os métodos *addConnectionAnchor*, *getFigure*, *getIcon* da classe pai *Shape*.

É importante ressaltar que várias execuções do trecho do programa em RDL da Figura 3.1 pode gerar vários modelos diferentes porque há duas descrições que podem variar de execução para execução, resultando portanto em modelos diferentes. A primeira é com relação ao número de classes que podem ser geradas a partir da extensão de *Shape*, isso se deve ao comando LOOP, pois permite ao reutilizador, em tempo de execução, determinar o número de classes que estenderá de *Shape*. E a outra está na escolha dos nomes das novas classes que herdam de *Shape*.

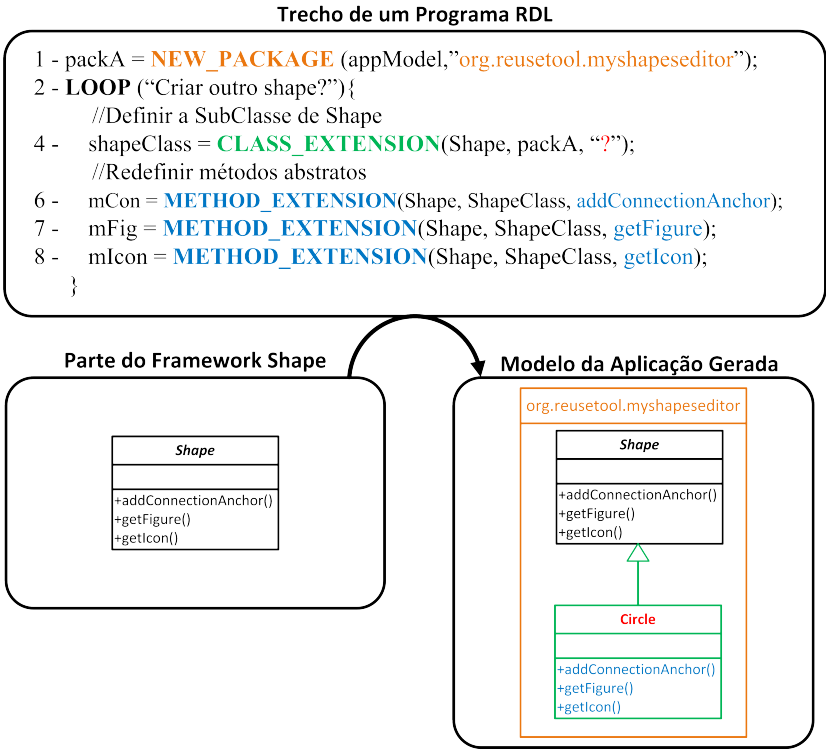


Figura 3.2: Trecho de reutilização de framework orientado a objetos em UML

3.2 - Comandos RDL

A Tabela 3.1 apresenta os comandos disponíveis na atual versão da RDL (OLIVEIRA *et al*., 2007), mostrando a sua sintaxe e descrição, e como já dito, limitada a frameworks orientados a objetos descritos em UML.

Tabela 3.1: Comandos RDL

Sintaxe	Descrição
class **new_class** (paramName)	Expressa a criação de uma classe no modelo com nome *paramName*. Retorna a classe criada.
method **new_method** (className, metName)	Expressa a criação do método *metName* na classe *className*. Retorna o método criado.
attribute **new_attribute** (className, attName)	Expressa a criação do atributo *attName* na classe *className*. Retorna o atributo criado.
void **new_inheritance** (superClass, subClass)	Expressa a relação de herança entre a classe pai *superClass* e a classe filha *subClass*.
void **add_code** (class, method, code_string)	Expressa a associação do código *code_string* ao método *method* da classe *class*.
boolean **element_choice** (anelement)	Emite uma pergunta ao usuário sobre a presença do elemento no projeto da aplicação.
class **class_extension** (superClass, className)	Expressa a criação da classe *className* a partir da herença da classe *superClass*.
class **select_class_extension** (classC)	Lista as subclasses concretas da classe *classC* para o executor escolher uma para estar presente no projeto da aplicação.
void **method_extension** (superC,metName,subC)	Expressa a criação do método *metName* na classe filha *subC*, e a classe *superC*, que tem declarado o método *metName*, é classe pai de *subC*.
void **value_assigment** (cName, attName, value)	Expressa a associação do valor *value* no atributo *attName* da classe *cName*.
value **value_assigment** (cName, attName, list)	Emite uma lista de valores em *list* para ser escolhida um para ser associado ao *attName* da classe *cName*.
void **pattern_class_extension** (cName,	Expressa a criação da classe *cName* através

patName, list)	do padrão *patName* que faz uso das informações de *list*.
void **pattern_method_extension** (superClass, metName, subClass, patName, list)	Expressa a redefinição do método *metName* da classe *superClass* na classe *subClass* através do padrão *patName* que faz uso das informações de *list*. O pré-requisito é que a classe *subClass* herde de *superClass*.
var = comando1	Indica que o resultado da execução do *comando1* estará na variável var.
?	Indica que o ambiente de execução da RDL precisa obter informações do reutilizador, isso é, emitir uma pergunta.
RECIPE name (parameter_list) recipe_body END_RECIPE;	Expressa um grupo de comandos que produzem algo que poder ser nomeado em *name*. Similar a funções e procedimentos em linguagens estruturadas.

3.3 - Fases da RDL

RDL apresenta duas fases bem definidas, a Fase de Desenvolvimento que tem como produto um programa RDL, e a Fase de Execução, onde os produtos são os modelos de aplicações em UML. O programa RDL pode ser visto como um *plano de execução* de uma Linha de Produto de Software que tem como domínio o framework e os produtos como sendo aplicações para o domínio.

Na Fase de Desenvolvimento, os reutilizadores especialistas no framework, olhando para a documentação e exemplos de aplicações, escrevem o programa RDL para guiar processos de reutilização. Um programa RDL expressa em ordem os pontos de extensão do framework, solicitando quando necessário, decisões do usuário para questões de extensões opcionais e alternativas, situações baseadas no modelo de características (FILHO *et al.*, 2004).

A Fase de Execução guia passo a passo as atividades de reutilização do framework através do programa RDL para gerar uma aplicação. Os reutilizadores não necessitam ser especialistas no framework como os que desenvolvem a RDL, mas precisam ter conhecimento do framework, acesso a sua documentação e da especificação da aplicação que se pretende gerar.

Como descrito em (MENDONÇA *et al*., 2008, NOOR *et al*., 2007), processos de reutilização podem ser executados por várias pessoas com o objetivo de construir software com melhor qualidade e menor tempo. No entanto, a atual versão da RDL descreve o processo de reutilização de forma mono usuária, não permitindo que vários reutilizadores possam executar trechos de RDL em uma mesma reutilização de framework. Em outras palavras, não há em sua especificação os mecanismos necessários para expressar atividades em paralelo, vinculando-as a pessoas com determinadas habilidades. Neste contexto se faz necessária a definição da CollabRDL, que estende RDL com conceitos relacionados à colaboração.

3.4 - Exemplo Ilustrativo

A Figura 3.3 é o modelo simplificado em UML do framework Portlet 2.0 (BELLAS, 2004) com o objetivo de mostrar os seus pontos de extensão. Embora o framework Portlet ofereça a possibilidade de reutilização por implementação de interfaces, recomenda-se a reutilização a partir da herança da classe *GenericPortlet* que tem uma implementação padrão para as interfaces necessárias, isso facilita a evolução dos portlets.

A Figura 3.3 também ilustra o modelo de uma aplicação baseada em portlets, o pacote MySystemModel. Basicamente novos portlets herdam de *GenericPortlet* e filtros são construídos a partir da implementação das interfaces *ActionFilter*, *EventFilter*, *RenderFilter* e *ResourceFilter*. O Código 3.1 mostra o trecho de RDL que quando executado produz os portlets, iremos agora realizar uma execução para produzir esta aplicação exemplo. A linha 4, produz o pacote *MySystemModel*. O LOOP que se inicia na linha 5 e termina na linha 40 contém um bloco com todas as atividades de reutilização para produzir portlets. A execução da linha 5 emite a pergunta "Deseja criar um portlet?", deve-se responder sim, a próxima linha de execução será a linha 6, esta

cria uma nova classe portlet que herda de *GenericPortlet,* o nome desta classe será informado pelo reutilizador, o parâmetro "?" indica esta situação. Neste exemplo, deve-se responder *MyPortletOne*, e a variável *myNewPortlet* apontará para esta nova classe.

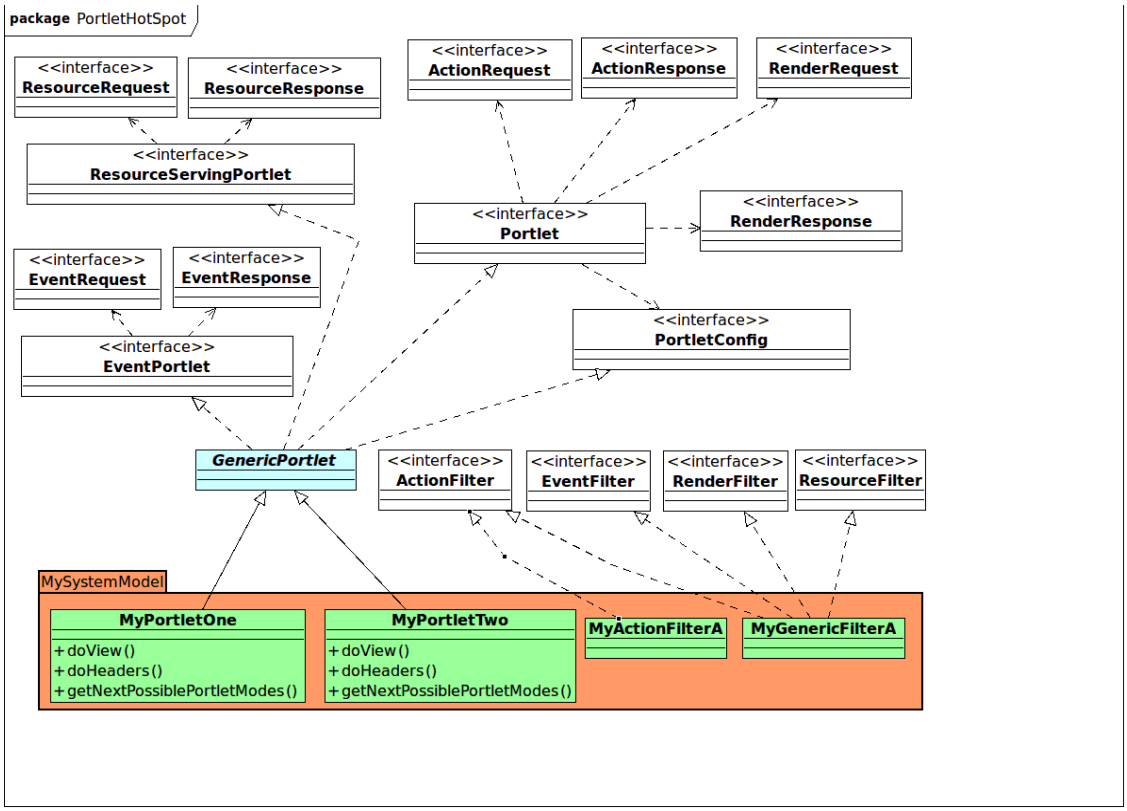


Figura 3.3: Framework Portlet descrito em UML.

Código 3.1: Trecho de código RDL para gerar portlets

```
4 - packA = NEW_PACKAGE(appModel,"MySystemModel");
5 - LOOP ("Deseja criar um portlet?"){
6 - myNewPortlet = CLASS_EXTENSION(GenericPortlet, packA,"?");
7 - IF ("Deseja implementar o método: public void init()?") THEN{
8 -   m = METHOD_EXTENSION(GenericPortlet, myNewPortlet,init);
9 - }
10 - LOOP ("Deseja fazer processamento de acao em myNewPortlet?"){
11 -  m = NEW_METHOD(myNewPortlet, "?");
12 -  ADD_CODE(myNewPortlet, m, "Criar a anotacao:
                              @ProcessAction(name=<nomeDaAcao>)");
13 - }
14 - IF ("Deseja redefinir o método: public void render(RenderRequest
                  request, RenderResponse       response)?") THEN{
15 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,render);
16 - }
```

```
17 - IF ("Deseja redefinir o método: protected void
                                    doDispatch(RenderRequest request,
                                    RenderResponse response)?") THEN{
18 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,doDispatch);
19 - }
20 - IF ("Deseja redefinir o método: protected java.lang.String
                        getTitle(RenderRequest request)?")  THEN{
21 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,getTitle);
22 - }
23 - METHOD_EXTENSION(GenericPortlet,myNewPortlet,doView);
24 -
25 - IF ("Deseja redefinir o método: protected void
                        doEdit(RenderRequest request, RenderResponse
                                              response)?") THEN{
26 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,doEdit);
27 - }
28 - IF ("Deseja redefinir o método: protected void
                        doHelp(RenderRequest request, RenderResponse
                                              response)?") THEN{
29 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,doHelp);
30 - }
31 - IF ("Deseja redefinir o método: public void destroy()?") THEN{
32 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,destroy);
33 - }
34 - LOOP ("Deseja tratar evento em myNewPortlet?"){
35 -  m = NEW_METHOD(myNewPortlet, "?");
36 -  ADD_CODE(myNewPortlet, m, "Criar a anotacao:
                              @ProcessEvent(qname=<nomeDoEvento>)");
37 - }
38 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,doHeaders);
39 -  METHOD_EXTENSION(GenericPortlet,myNewPortlet,
                                    getNextPossiblePortletModes);
40 - } // Comentário: END LOOP ("Deseja criar um portlet?"){
```

O método *init* é opcional em novos portlets, o comando IF permite emitir uma pergunta ao reutilizado solicitando a decisão de execução de atividades opcionais. A linha 7 emite a pergunta "Deseja implementar o método: public void init()?", para este exemplo deve-se responder não, assim a linha 8 não será executada. Casos semelhantes são vistos nas linhas 14-16, 17-19, 20-22, 25-27, 28-30, 31-33, que respectivamente deixam a decisão de redefinir ou não os métodos *render, doDispatch, getTitle, doEdit, doHelp e destroy* para o reutilizador em tempo de execução, para este exemplo devemos responder não para essas perguntas.

As linhas 23, 38 e 39 são executadas automaticamente quando a linha de execução chegam a essas linhas, isto porque elas são mandatórias, sem precisar portanto da interação com o reutilizador. Elas respectivamente redefinem na nova classe portlet

os métodos *doView*, *doHeaders* e *getNextPossiblePortletModes*. Finalmente, quando a linha de execução chega em 40, ela é remetida para a linha 5, emitindo novamente a pergunta de decisão do LOOP, "Deseja criar um portlet?", para este exemplo devemos responder sim e na linha 6 responder *MyPortletTwo*, seguindo respondendo não a todos os métodos opcionais. Para o próximo teste de LOOP, devemos responder não já que geramos os dois portlets necessários para este exemplo: *MyPortletOne* e *MyPortletTwo.*

O Código 3.2 exibe o código para produzir filtros. O comando LOOP, início de bloco na linha 42 e fim de bloco na linha 56, expressa que podem ser criados zero ou mais filtros e eles podem ser criados através da realização das interfaces *ActionFilter*, *EventFilter*, *RenderFilter* e *ResourceFilter*, respectivamente nas linhas 45, 48, 51 e 54. Para gerar os filtros para a nossa aplicação exemplo, pacote MySystemModel da Figura 3.3, no primeiro teste do LOOP devemos responder sim para a pergunta da linha 43, entrar com resposta *MyActionFilter,* sim para a pergunta da linha 44, "Deseja implementar a interface ActionFilter?", e responder não para o restante. Para o segundo teste do LOOP, responder sim, e entrar com *MyGenericFilterA* na linha 43. Para as linhas 44, 47, 50, e 53 responder sim, pois assim as linhas 45, 48, 51 e 54 serão executadas fazendo com que o filtro *MyGenericFilterA* implemente as interfaces *ActionFilter*, *EventFilter*, *RenderFilter* e *ResourceFilter,* respectivamente.

Código 3.2: Trecho de código RDL para gerar filtros para portlets

```
42 - LOOP ("Deseja criar um filtro?"){
43 -  myNewFilter = NEW_CLASS(packA, "?");
44 -  IF ("Deseja implementar a interface ActionFilter?"){
45 -        NEW_REALIZATION(myNewFilter, ActionFilter);
46 -  }
47 -  IF ("Deseja implementar a interface EventFilter?"){
48 -        NEW_REALIZATION(myNewFilter, EventFilter);
49 -  }
50 -  IF ("Deseja implementar a interface RenderFilter?"){
51 -        NEW_REALIZATION(myNewFilter, RenderFilter);
52 -  }
53 -  IF ("Deseja implementar a interface ResourceFilter?"){
54 -        NEW_REALIZATION(myNewFilter, ResourceFilter);
55 -  }
56 -}
```

Além disso, o programa RDL completo para a reutilização do framework Portlet possui uma atividade externa para marcar a configuração do arquivo portlet.xml que é necessário para toda aplicação baseada em portlet. O comando RDL utilizado para expressar é: EXTERNAL_TASK("Configure o arquivo portlet.xml."). Finalmente, omitimos o comando *import* que aponta para o framework Portlet expresso em UML e o comando *export* que aponta para um arquivo onde serão gravados os modelos de cada execução do programa RDL, também no formato UML.

3.5 - Considerações Finais

Este capítulo mostrou a visão geral de utilização da RDL, apresentou um exemplo simples, mostrou os seus comandos e ressaltou que a RDL tem dois momentos bem distintos, o de escrita do programa RDL e o de sua execução. Além disso, mostrou como pode ser feita a instanciação do framework Portlet através de um programa RDL.

Com isso, podemos concluir que, para frameworks simples, a utilização de RDL de forma monousuária é possível, porém, em ambientes de produção, os frameworks e os requisitos da nova aplicação são complexos, necessitando da colaboração de mais pessoas.

CAPÍTULO 4 - APRESENTAÇÃO DA CollabRDL

Este capítulo apresenta CollabRDL como uma extensão da RDL, descreve os seus comandos para atender aos objetivos deste trabalho, mostra como foi feito a implementação do ambiente de execução para CollabRDL baseado em uma ferramenta BPMN, descreve o mapeamento de CollabRDL para BPMN-Activiti, e finaliza com as considerações finais.

A Figura 4.1 mostra a visão geral de utilização da CollabRDL, esta figura adiciona à Figura 3.1, visão geral da RDL, vários reutilizadores. Assim como na RDL, é preciso entrar com o modelo do framework, marcado como item 1. O item 2 marca a entrada do programa CollabRDL, isto é, os comandos da RDL mais os novos comandos ROLE, PARALLEL e DOPARALLEL. Os itens 3 e 4, que em RDL permitiam somente a interação de um reutilizador, agora com CollabRDL, é possível a interação com vários reutilizadores ao mesmo tempo. Já a saída, item 5, continua como em RDL, um modelo da aplicação.

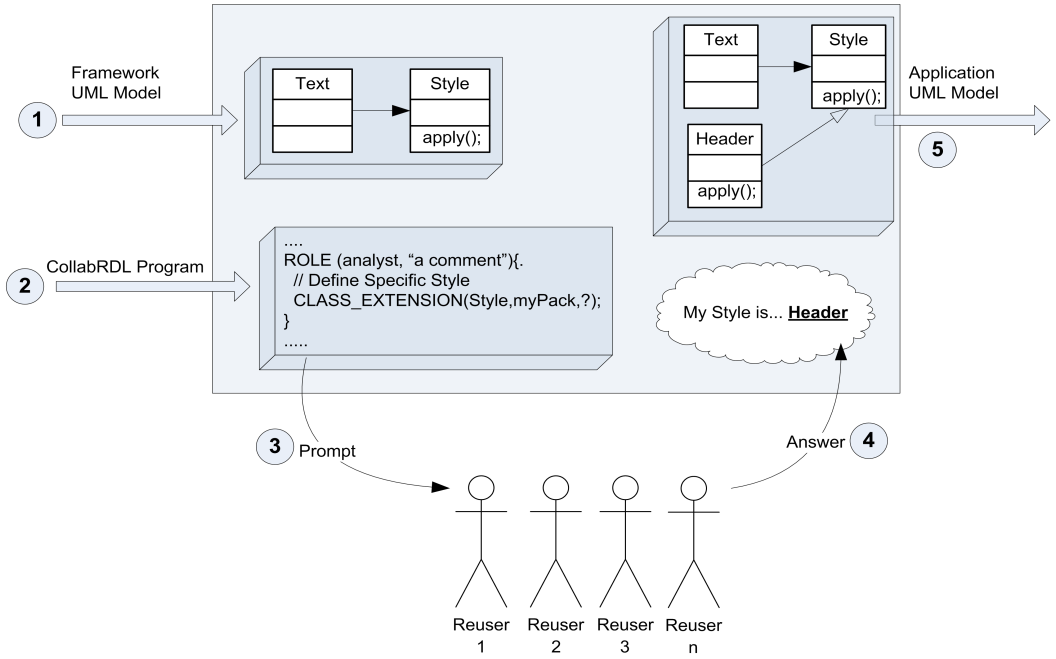
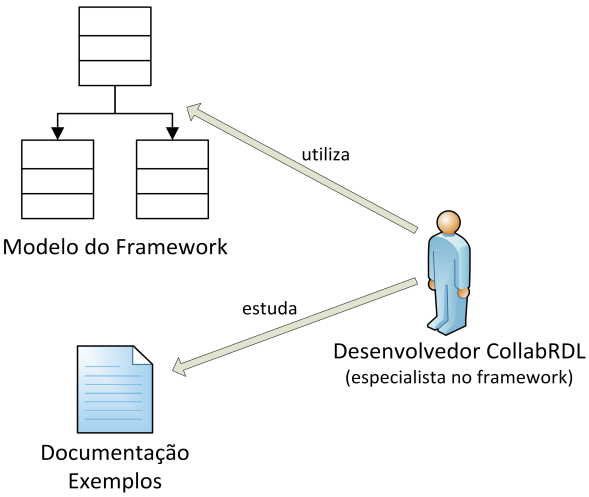


Figura 4.1: Visão geral de utilização da CollabRDL

A Figura 4.2 ilustra os passos que o desenvolvedor CollabRDL pode seguir para criar um programa CollabRDL. O primeiro passo é “Identificar as atividades de reutilização”, para isso, o desenvolvedor, consultando o modelo do framework a ser instanciado, sua documentação e exemplos, identifica as atividades de reutilização, podendo usar como estratégia a delimitação de regiões no framework (por exemplo, o padrão MVC, modelo, visualização e controle), ou por funcionalidades (descrição de casos de uso). O segundo passo é “Identificar as atividades que podem ser realizadas em paralelo”, levando em consideração as suas interdependências, a sugestão aqui é criar uma tabela para ajudar na organização do paralelismo entre as atividades. Neste exemplo, as atividades 1, 2 e 3 podem ser executadas em paralelo, logo após, as atividades 9 e 10 podem ser executadas em paralelo com 11 e 12. O terceiro passo é “Definir os grupos de reutilização” de acordo coma as habilidades/responsabilidades necessárias para a realização das atividades identificadas no primeiro passo. O quarto passo, “Associar atividades a grupos”, é alcançado quando toda atividade identificada no primeiro passo é associada a um grupo que foi definido no terceiro passo. E finalmente, o quinto passo,“Expressar em CollabRDL”, é preciso descrever as atividades identificadas, considerando o paralelismo entre elas e sua associação com grupos utilizando os comandos herdados da RDL e os novos comandos CollabRDL.



1 – Identificar as atividades (1ª, 2ª, 3ª ...)

2 – Organizar as atividades em paralelo

	Atividades
Paralelo1	1,2,3
Paralelo2	{9,10},{11,12}

3 – Definir os grupos de reutilizadores

Administrador: grupo responsável pela configuração do ambiente de execução e da aplicação
...

4 – Associar atividades a grupos

Atividades	Administrador	Analista
{1,2}	X	
3		X
{9,10}	X	
{11,12}		X

5 – Expressar em CollabRDL

Figura 4.2: Passos para criar um programa CollabRDL

Embora o quinto passo da Figura 4.2 tenha como produto um programa CollabRDL, ainda é preciso rever o código para analisar as possíveis regiões críticas (SCOTT, 2009) geradas pelos comandos PARALLEL e DOPARALLEL. No Capítulo 5, na avaliação através de uma prova de conceito, iremos descrever esta situação em um caso real. Além disso, programas RDL podem ser alterados para CollabRDL seguindo os passos da Figura 4.2.

4.1 - Fundamentação da CollabRDL

A RDL atualmente é uma linguagem textual assim como C e Java, e é também sequencial porque um comando é executado após o término do anterior, e por isso classificada como *imperativa* (OLIVEIRA *et al.*, 2007, JOUAULT e KURTEV, 2006). Além disso, também possui comandos para *decisão* (por exemplo IF ELSE) e iteração (LOOP), por isso também é *estruturada* (SCOTT, 2009). Existem comandos para solicitar ao reutilizador respostas, assim podemos dizer que ela também é interativa. E finalmente, é executável, existe um ambiente em que um programa RDL pode ser executado pelo reutilizador (OLIVEIRA, ALENCAR e COWAN, 2011).

Contudo, RDL ainda não oferece ao desenvolvedor o poder de associar atividades a grupo de pessoas com habilidades específicas (administrador de banco de dados, programador etc) e expressar atividades em paralelo. Esses são os dois objetivos da primeira versão da CollabRDL, o nome da versão da RDL que tem como objetivo geral ser uma linguagem para representar processo de reutilização de software de forma colaborativa.

O Capítulo dois desta dissertação sugere alguns elementos que CollabRDL deve oferecer para atender ao seu propósito. A Tabela 4.1 associa esses elementos aos novos comandos e ao ambiente de execução. O elemento *comunicação* deve ser suportado pelo ambiente de execução através de softwares para chat e videoconferência. A *coordenação* deve ser suportada pelos três novos comandos: ROLE, PARALLEL e DOPARALLEL. O comando ROLE para dar suporte à alocação e seleção dos autores, já para as restrições de simultaneidade e sincronização de atividades, os comandos PARALLEL e DOPARALLEL, todos, devidamente suportados pelo ambiente de execução. O comando ROLE e o ambiente de execução são necessários para

implementar a distribuição de atividades e a sessão compartilhada do elemento *cooperação*. Finalmente, o elemento *awareness* dever ser suportado pelo comando ROLE juntamente com o ambiente de execução.

Tabela 4.1: Elementos para CollabRDL

Elementos	**CollabRDL**
Comunicação	► Canais de comunicação entre os reutilizadores tais como chat e videoconferência. (Suporte do Ambiente de Execução)
Coordenação	► Selecionar os atores; (Comando ROLE + Ambiente de Execução) ► Alocar atores; (Comando ROLE + Ambiente de Execução) ► Sincronização de atividades. (Comandos PARALLEL e DOPARALLEL + Ambiente de Execução) ► Restrições de simultaneidade; (Comandos PARALLEL e DOPARALLEL)
Cooperação	► Distribuição de atividades; (Comando ROLE + Ambiente de Execução) ► Sessão compartilhada; (Comando ROLE + Ambiente de Execução)
Awareness	Percepção de: ► Quem deve fazer; (Comando ROLE + Ambiente de Execução) ► Quem fez; (Comando ROLE + Ambiente de Execução) ► Quem está fazendo; (Comando ROLE + Ambiente de Execução)

4.2 - Comandos CollabRDL

CollabRDL deve suportar pessoas trabalhando juntas no processo de reutilização. Fica implícito que é preciso expressar *coordenação* já que é preciso organizar os esforços individuais para produzir um resultado único. A teoria da coordenação relata que coordenação é "um conjunto de princípios sobre como as atividades podem ser coordenadas" (MALONE e CROWSTON, 1990), conceitos vistos em detalhe no item 2.3, logo a

> *CollabRDL precisa permitir ao desenvolvedor o poder de expressar atividades em paralelo e associar atividades a grupos.*

Antes de definir os novos comandos, vamos olhar com detalhe para o trecho de um programa em RDL exibido no Código 4.1. Na linha 1, o comando NEW_PACKAGE precede o comando CLASS_EXTENSION da linha 2, e é representado textualmente em uma linha com término em ";". Dessa forma, é garantido que o comando CLASS_EXTENSION será iniciado somente após o término de NEW_PACKAGE. E o comando CLASS_EXTENSION pode por isso usar o resultado de NEW_PACKAGE, uma referência para o pacote criado (packA). Nesse caso, o comando NEW_PACKAGE é pré-requisito para CLASS_EXTENSION. Esse é o paradigma das linguagens de programação imperativas (JOUAULT e KURTEV, 2006).

Código 4.1: Trecho de um Programa em RDL

```
1. packA = NEW_PACKAGE(appModel,"my.domain.pachName");
2. shapeClass = CLASS_EXTENSION(Shape, packA,"?");
3. METHOD_EXTENSION(Shape,shapeClass,addConnectionAnchor);
4. IF ("Add Feature - Restrict Connections?") THEN {
5.         m = METHOD_EXTENSION(Shape,shapeClass,canConnect);
6. }
```

Um exemplo de comando interativo é o apresentado na linha 2 do Código 4.1. O comando descreve a criação de uma nova classe no pacote packA a partir de uma herança da classe Shape do framework (OLIVEIRA, ALENCAR e COWAN, 2011). O sinal "?" indica que em tempo de execução, o reutilizador precisará entrar com o nome

da nova classe. São esses comandos, os interativos, que serão delegados aos grupos de reutilizadores. Portanto, RDL é composta por comandos que são executados automaticamente pelo ambiente de execução e outros por reutilizadores, pois necessitam da sua intervenção.

Os comandos RDL (OLIVEIRA *et al.*, 2007) somados aos comandos ROLE, PARALLEL e DOPARALLEL formam a primeira versão da CollabRDL, que possibilita ao desenvolvedor expressar atividades em paralelo e associá-las a grupos de reutilizadores. Por outro lado, em tempo de execução, ajuda aos reutilizadores na coordenação das atividades de reutilização em equipe. A seguir, iremos apresentar esses 3 novos comandos.

O comando ROLE: Para permitir ao desenvolvedor CollabRDL associar atividades a grupos, nos inspiramos na definição de roles, *a forma como uma entidade participa de um relacionamento* (USCHOLD *et al.*, 1998). Role é uma invenção humana, é através dela que uma pessoa pode interagir na sociedade de diferentes formas (SMITH *et al.*, 1998). Além disso, as linguagens que se propõem a representar colaboração implementam-na (DE PAOLI e TISATO, 1994, OASIS, 2006, BPMN, 2011, CORTES e MISHRA, 1996, LI e MUNTZ, 2000, BRIGGS *et al.*, 2003).

O Código 4.2 apresenta a sintaxe do novo comando ROLE. Este comando implementa o item *associar atores a atividades,* conforme visto no elemento *coordenação* da Tabela 4.1. Assim, em tempo de execução, todas as atividades que precisam da intervenção do usuário que estejam no bloco delimitado por chaves serão delegadas para os reutilizadores do grupo *analista,* indicado no primeiro parâmetro, passando uma informação "Um comentário!". Um membro do grupo analista poderá então retirar a próxima atividade para execução.

Código 4.2: O comando ROLE

```
ROLE (analista, "Um comentário!"){
    shapeClass = CLASS_EXTENSION(Shape, packA,"?");
    m = METHOD_EXTENSION(Shape,shapeClass,addConnectionAnchor);

    ADD_CODE(shapeClass,m,"return new ChopboxAnchor(iFigure);");
    m = METHOD_EXTENSION(Shape,shapeClass,getFigure);

    ADD_CODE(shapeClass,m,"return  new IFIGURE_SUBCLASS);");
```

```
    m = METHOD_EXTENSION(Shape,shapeClass,getIcon);
    ADD_CODE(shapeClass,m,"return createImage(\"NEW_SHAPE.gif\");");
    ...
}
```

O Código 4.3, linhas 3-7, mostra a BNF que identifica o comando ROLE. Este comando tem dois parâmetros identificados pelos símbolos não terminais *role* e *comment*. O parâmetro *role* identifica o grupo e o parâmetro *comment* o comentário a ser passado para o grupo. Além disto a BNF, Código 4.3, apresenta como este comando é tratado pelo gerador de código BPMN. Na linha 5 há a expressão *codeGen.addBeginRoleBlock($role.text, $comment.text)* que representa a chamada do método *addBeginRoleBlock* com os parâmetros *$role.text*, *$comment.text*. É importante ressaltar que o método em questão pertence à classe *BPMNCodeGenerator* do pacote RDL-BPMN-Activiti, que é responsável por gerar o código BPMN em formato XML para todos os comandos, como apresentado na Seção 4.3, passo 1 da Figura 4.3. O Apêndice I mostra o BNF da RDL com os novos comandos CollabRDL.

Código 4.3: Gramática do comando ROLE

```
1- //Tokens
2 - BEGIN_ROLE = 'ROLE';
// ------
3 - role_statement
4 -    : BEGIN_ROLE LEFT_PARENTHESIS role COMMA comment
5 -        RIGHT_PARENTHESIS { codeGen.addBeginRoleBlock
                                    ($role.text, $comment.text); }
6 -       statement_block { codeGen.addEndRoleBlock(); }
7 -     ;
```

O comando PARALLEL: O Código 4.4 mostra o comando PARALLEL para agrupar blocos de comandos que podem ser executados em paralelo. Este comando implementa os itens *restrições de simultaneidade, sincronização de atividades e alocar atores a atividades* do elemento *coordenação* mostrado na Tabela 4.1. O primeiro bloco FLOW indica que o reutilizador irá delegar as atividades interativas que estão entre chaves para o grupo *analista*, passando também uma orientação através do segundo parâmetro, da mesma forma que o comando ROLE. E sem esperar alguém do grupo *analista* realizar essas atividades do primeiro bloco FLOW, poderá delegar as atividades

do segundo bloco FLOW para o grupo *programadorVisual*. O fecha chaves do comando PARALLEL (“}”) indica que o reutilizador irá esperar a realização das atividades dos dois blocos de FLOW para prosseguir com o processo de reutilização. O comando PARALLEL permite 2 ou mais blocos FLOW.

Código 4.4: O comando PARALLEL

```
PARALLEL {
   FLOW (analista, "Um comentário!"){
       A;
   }
   FLOW (programadorVisual, "Outro comentário!"){
       B;
   }
}
```

O Código 4.5, linhas 3-8, mostra a BNF que identifica o comando PARALLEL. Este comando possui um bloco com duas ou mais declarações *flow_statement.* A linha 5 marca o início do bloco com chamada do método *addFork*, responsável por criar linhas de execução em paralelo para as declarações *flow_statement.* E a linha 7 declara a chamada ao método *addJoin*, no final do bloco, para juntar essas linhas de execução. A declaração *flow_statement,* linhas 9-13, similar ao comando ROLE em seus parâmetros, declara um bloco de atividades que pode ser executado em paralelo no comando PARALLEL. Novamente, os métodos *addFork* e *addJoin* pertencem a classe *BPMNCodeGenerator* do pacote RDL-BPMN-Activiti, Seção 4.3, passo 1 da Figura 4.3.

Código 4.5: Gramática do comando PARALLEL

```
1- //Tokens
2 - PARALLEL = 'PARALLEL';
// ------
3 - parallel_statement
4 -  : PARALLEL
5 -    LEFT_CURLY { codeGen.addFork(); }
6 -    flow_statement flow_statement (flow_statement)*
7 -   { codeGen.addJoin(); } RIGHT_CURLY
8 -    ;
// ------
9 - flow_statement
10 -  : BEGIN_FLOW LEFT_PARENTHESIS role COMMA comment
11-     RIGHT_PARENTHESIS { codeGen.addBeginFlowBlock
                              ($role.text, $comment.text); }
12 -    statement_block { codeGen.addEndFlowBlock(); }
13 -   ;
```

O Código 4.6 mostra a combinação do comando LOOP da RDL com o novo comando ROLE da CollabRDL, enquanto o reutilizador responder sim à pergunta "Repetir?", essa combinação gerará um bloco ROLE para execução sequencialmente, isto é, após a primeira iteração, o reutilizador irá esperar alguém do grupo *analista* realizar todas as atividades interativas que estão no bloco ROLE, somente após será emitida a pergunta "Repetir?" novamente. Caso a resposta seja positiva, o bloco ROLE será novamente delegado para o grupo *analista,* e como na primeira iteração, espera-se a realização das suas atividades para novamente emitir a pergunta do LOOP.

Código 4.6: Combinação do comando LOOP com o ROLE

```
LOOP ("Repetir?"){
   ROLE (analista, "Passe um comentário!"){
      atividadeA;
      ...
   }
}
```

O comando DOPARALLEL: Em outras situações, diferente do comportamento serial obtido com o comando LOOP, existe a necessidade de executar os blocos em paralelo. O comando DOPARALLEL expressa esse comportamento, ele implementa os itens *restrições de simultaneidade* e *sincronização de atividades* do elemento *coordenação* mostrados na Tabela 4.1. Já o Código 4.7 expressa que em tempo de execução a atividade A será oferecida para o grupo *analista*, e sem a necessidade de esperar o término de A, a pergunta "Executar novamente o bloco?" será emitida, se a resposta for positiva, colocará mais uma instância da atividade A para o grupo *analista*, e se a resposta for negativa, o fluxo irá para o final do DOPARALLEL, o ponto e vírgula (";"), onde irá esperar o término de todas as atividades contidas no bloco do comando DOPARALEL.

Código 4.7: O comando DOPARALLEL

```
DOPARALLEL{
   ROLE (analista, "Um comentário!"){
      A;
   }
}WHILE ("Executar novamente o bloco?");
```

O Código 4.8, linhas 4-9, mostra a BNF que identifica o comando DOPARALLEL. A linha 5 marca o início do bloco com chamada do método *addBeginDoparallelBlock.* A linha 6 declara que o bloco deste comando pode conter um *statement_block*, um bloco genérico que empacote comandos de reutilização e estruturas da linguagem (por exemplo IF). As linhas 7 e 8 indicam o teste para continuar ou não a iteração, a expressão addEndDoparallelBlock(*$condition.text*) representa a chamada do método addEndDoparallelBlock com o parâmetro *$condition.text.* Como acontece em todos os comandos, os métodos *addBeginDoparallelBlock* e *addEndDoparallelBlock* pertencem a classe *BPMNCodeGenerator* do pacote RDL-BPMN-Activiti, Seção 4.3, passo 1 da Figura 4.3.

Código 4.8: Gramática do comando DOPARALLEL

```
1- //Tokens
2 - BEGIN_DOPARALLEL = 'DOPARALLEL';
3 - END_DOPARALLEL = 'WHILE';
// ------
4 - doparallel_statement
5 - :BEGIN_DOPARALLEL
            { codeGen.addBeginDoparallelBlock(); }
6 -   statement_block
7 - END_DOPARALLEL LEFT_PARENTHESIS condition RIGHT_PARENTHESIS
8 -   SEMICOL{ codeGen.addEndDoparallelBlock($condition.text);}
9 - ;
```

4.3 - Ambiente de Execução

O ambiente de execução para processos de reutilização expressados em CollabRDL precisa minimamente oferecer facilidades para carregar, iniciar e executar processos, além de permitir as funcionalidades de workflow (WFMC, 1999). No contexto de BPMN existem mais de 70 ambientes que oferecem essas funcionalidades (BPMN, 2013). Portanto, ao converter um programa CollabRDL para BPMN, passaremos a usufruir dessas funcionalidades. A nossa escolha foi pelo ambiente Activiti (ACTIVIT, 2012) por ter código livre e aberto e porque também utiliza a tecnologia Java, assim como o núcleo da linguagem RDL.

O ambiente de execução CollabRDL-Activiti-Explorer é baseado no workflow e gerenciador de processos de negócios Activiti (ACTIVIT, 2012). Ele usa o ambiente Activiti e as funcionalidades para manipulação de artefatos UML da RDL. Para utilizar este ambiente de execução, é preciso converter os processos descritos em CollabRDL para o formato XML no padrão BPMN com marcações BPMN-Activiti.

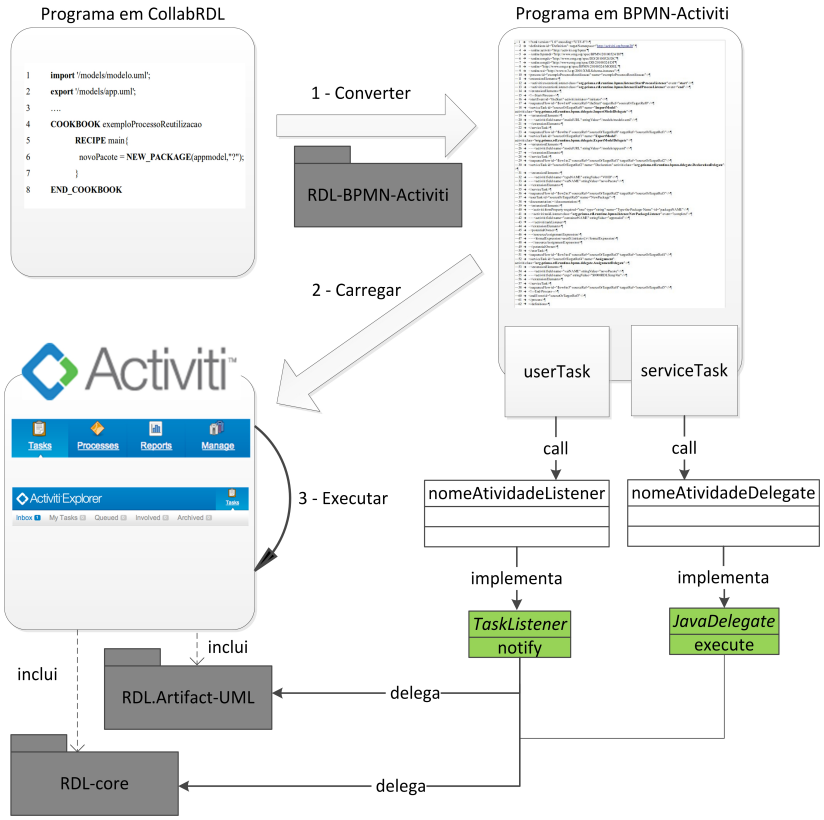


Figura 4.3: Ambiente de execução para CollabRDL

A Figura 4.3 ilustra como utilizar o ambiente CollabRDL-Activiti-Explorer para executar um programa CollabRDL. O passo 1 converte o programa escrito em CollabRDL para BPMN-Activiti utilizando o pacote RDL-BPMN-Activiti. As atividades interativas são convertidas para atividades do tipo *userTask* e as não interativas em *serviceTask*. Atividades tipo *userTask* são aquelas que serão realizadas por reutilizadores (pessoas) e as *serviceTask* são executadas automaticamente pelo

ambiente de execução (BPMN, 2011). As *userTasks* precisam de classes que implementam a interface *TaskListener* para delegar a execução das atividades de reutilização aos pacotes *RDL-core* e *RDL.Artifact-UML*, já as serviceTasks precisam de classes que implementam *JavaDelegate*. Esta conversão é tratada em detalhes na Seção 4.4.

O passo 2 carrega o programa BPMN-Activiti convertido no Activiti-Explorer incluindo os pacotes RDL-core e RDL.Artifact-UML. O Activiti-Explorer é um software desenvolvido para Web que permite a criação de usuários e grupos pela sua aba *Manager* e iniciar e terminar processos através da aba *Process.* As atividades que foram delegadas em CollabRDL por ROLE ou FLOW, em tempo de execução, irão aparecer no item de menu *Queued* separados por grupo e as que não foram irão aparecer no item *involved* para ser executada por quem iniciou o processo.

4.4 - Mapeamento para BPMN-Activiti

CollabRDL acrescenta à atual RDL os comandos ROLE, PARALLEL e DOPARALLEL, e por isso o seu desenvolvimento é baseado no ambiente da RDL. Sendo assim, para executar um programa CollabRDL no ambiente CollabRDL-Activiti, precisamos converter toda a estrutura de um programa RDL para BPMN-Activiti, além dos novos comandos mencionados. A Figura 4.4 mostra o diagrama de classes resumido que, combinado com o Parser e Lexer, faz chamadas a uma instância da classe *BPMNCodeGenerator* para gerar o código intermediário, contendo todos os comandos necessários para produzir o XML executável BPMN-Activiti. Este diagrama mostra o modelo para o comando DOPARALLEL, a classe *BeginDoparallelBPMN* herda de *BPMNCode* e redefine comportamentos através de *BeginDoparallelBPMN* (construtor), *setVarCondition*, *getCode*. E a classe *EndDoparallelBPMN* redefine *EndDoparallelBPMN* (construtor), *getCode e getTransitions.* Essa forma é repedida para implementar todos os comandos, criando assim um mapeamento de CollabRDL para BPMN-Activiti.

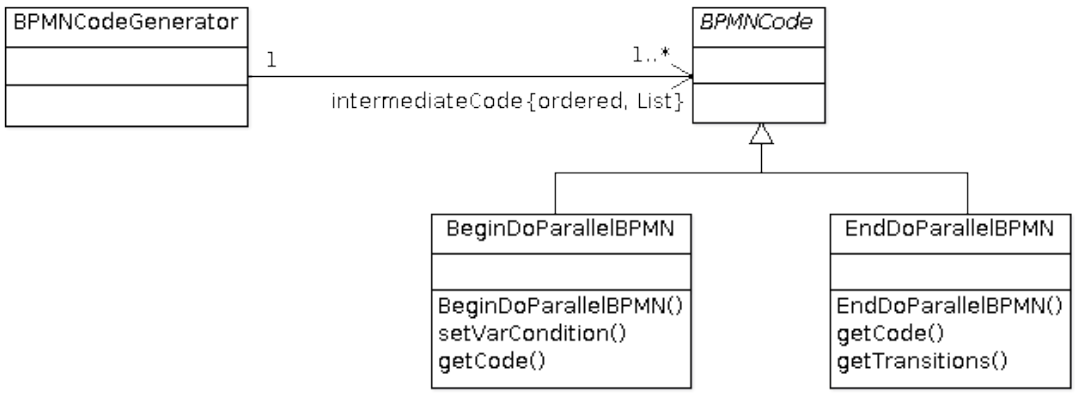


Figura 4.4: Modelo de Classes resumido do conversor, comando DOPARALLEL.

Uma vez produzido o XML, resultado da conversão do programa CollabRDL para BPMN, faz-se necessário explicar como, em tempo de execução, as atividades de reutilização são delegadas para o núcleo da RDL. Como os novos comandos CollabRDL são de workflow, ou seja, não são associados às atividades de reutilização, usaremos o comando NEWPACKAGE da RDL como exemplo. A Tabela 4.2 mostra o comando NEWPACKAGE em CollabRDL com o seu mapeamento para BPMN-Activiti. O comando NEWPACKAGE da RDL é mapeado para *userTask* da BPMN porque se trata de um comando que precisa de uma interação com o usuário, o parâmetro *appmodel* foi mapeado para *activiti:field* e através da marcação *activiti:taskListener* a atividade de reutilização, depois da interação com o usuário, é delegada para a classe *NewPackageListener.* Esse mecanismo é possível no ambiente de execução CollabRDL-Activiti-Explorer.

Tabela 4.2: Mapeamento do comando NEWPACKAGE para BPMN-Activiti

Comando NEWPACKAGE em RDL

```
NEW_PACKAGE(appmodel,"?");
```

Comando NEWPACKAGE em BPMN-Activiti

```
<userTask id="sourceOrTargetRef4" name="NewPackage" >
<documentation></documentation>
  <extensionElements>
    <activiti:formProperty required="true" type="string" name="Type
the Package Name"  id="packageNAME" />
    <activiti:taskListener
class="org.prisma.rdl.runtime.bpmn.listener.NewPackageListener"
event="complete" >
      <activiti:field name="containerNAME" stringValue="appmodel" />
    </activiti:taskListener>
  </extensionElements>
  <potentialOwner>
    <resourceAssignmentExpression>
      <formalExpression>user(${initiator})</formalExpression>
    </resourceAssignmentExpression>
  </potentialOwner>
</userTask>
```

A Figura 4.5 mostra o modelo para mapear o comando NEWPACKAGE para BPMN-Activiti. Ele mostra que é preciso criar a classe *NewPackageListener* que implementa a interface *TaskListener* com um atributo *containerName* que tem referência para uma classe que implementa a interface *Expression,* utilizado quando o valor está expresso em XML, caso de *appmodel*. A referência para uma classe que implementa *DelegateTask* é usada para obter as respostas dos usuários, caso de *packageNAME*. Após, cria-se uma classe NewPackage, esta do núcleo da RDL, passando os parâmetros, um expresso em XML (*appmodel*) e o outro como resultado da interação com o usuário, resposta à pergunta *Type the Package Name.* A Figura 4.6 mostra o diagrama de sequência para esta situação.

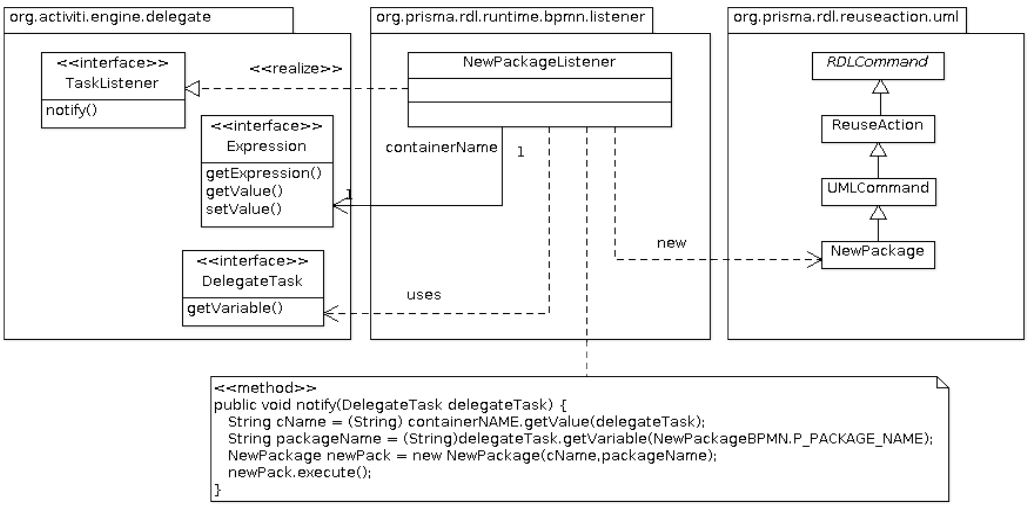


Figura 4.5: Modelo do mapeamento do comando NEWPACKAGE para BPMN-Activiti

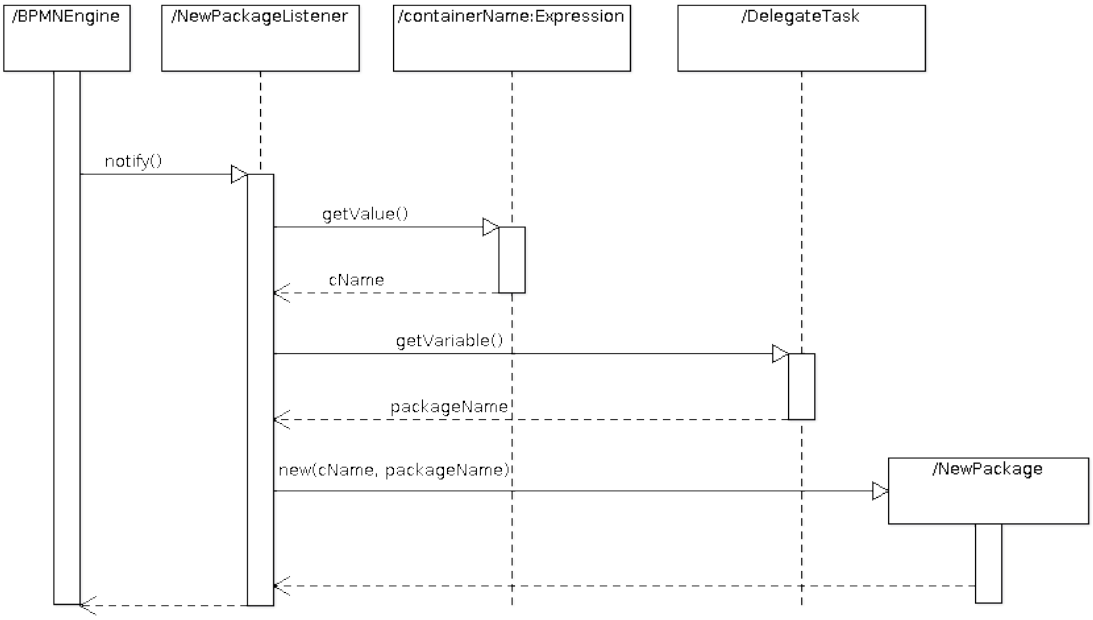


Figura 4.6: Diagrama de sequência para o modelo da Figura 4.5

O Código 4.9, linhas 6-8, mostra como utilizar o comando ROLE em CollabRDL, ele indica que o comando interativo NEW_PACKAGE será oferecido para o grupo *arquiteto* com o comentário "Siga o padrão de nomes de pacotes." em tempo de execução. A Figura 4.7 mostra como é representado visualmente em BPMN, e o Código 4.10 mostra as marcações em negrito que precisam ser feitas para todas as atividades interativas que estiverem nos blocos dos comandos ROLEs. Além disso, o ambiente CollabRDL-Activiti irá delegar para a classe *pacote.NewPackageListener* a responsabilidade de acessar a infraestrutura da RDL para realizar a criação do novo pacote no modelo *app.uml*.

Código 4.9: Exemplo dos comandos ROLE, PARALLEL e DOPARALLEL em CollabRDL:

```
1- import '/models/modelo.uml';
2- export '/models/app.uml';
3-
4- COOKBOOK exemploProcessoReutilizacao
5-    RECIPE main{
6-    ROLE (arquiteto, "Siga o padrão de nomes de pacotes."){
7-       pacoteA = NEW_PACKAGE(appmodel,"?");
8-    }
9-
10-   PARALLEL {
11-      FLOW (analistaJunior, "Comentário"){
12-           climaPortlet = CLASS_EXTENSION(GenericPortlet,
                                                    pacoteA,"?");
13-           ...
14-      }
15-      FLOW (analistaSenior, "Comentário"){
16-         cambioPortlet = CLASS_EXTENSION(GenericPortlet,
                                                    pacoteA,"?");
17-         ...
18-      }
19-   }
20-
21-   DOPARALLEL ("Deseja criar outro filtro?"){
22-      ROLE (analistaSenior, "Comentário"){
23-        filtro = NEW_CLASS(pacoteA, "?");
24-        IF ("Deseja implementar a interface ActionFilter?"){
25-           NEW_REALIZATION(myNewFilter, ActionFilter);
26-        }
27-        ...
28-      }
29-   }
30-}
31- END_COOKBOOK
```

Figura 4.7: Representação em BPMN do comando ROLE

Código 4.10: Marcação em BPMN-Activiti para o comando ROLE das linhas 6-8 do Código 4.9

```
<userTask id="sourceOrTargetRef3" name="NewPackage" >
<documentation>"Siga o padrão de nomes de
                                    pacotes."</documentation>
  <extensionElements>
    <activiti:formProperty required="true" type="string"
      name="Type the Package Name"  id="packageNAME" />
    <activiti:taskListener
            class="org.prisma.rdl.runtime.bpmn.
            listener.NewPackageListener" event="complete" >
      <activiti:field name="containerNAME"
                                    stringValue="appmodel" />
    </activiti:taskListener>
  </extensionElements>
  <potentialOwner>
    <resourceAssignmentExpression>
      <formalExpression>group(arquiteto)</formalExpression>
    </resourceAssignmentExpression>
  </potentialOwner>
</userTask>
```

O comando PARALLEL, linhas 10-19 do Código 4.9, expressa que a criação do Portlet Clima será oferecido para o grupo *analistaJunior* e o Portlet Câmbio para o grupo *analistaSenior*. Essas atividades poderão ser feitas em paralelo já que não existem dependências entre elas por se tratarem de portlets com conteúdos diferentes. A Figura 4.8 apresenta a representação deste trecho em BPMN, são utilizadas duas lanes, uma para atividades delegadas para *analistaJunior* e outra para *analistaSenior.* Além disso, ao meio estão os elementos de início e fim dos caminhos em paralelo.

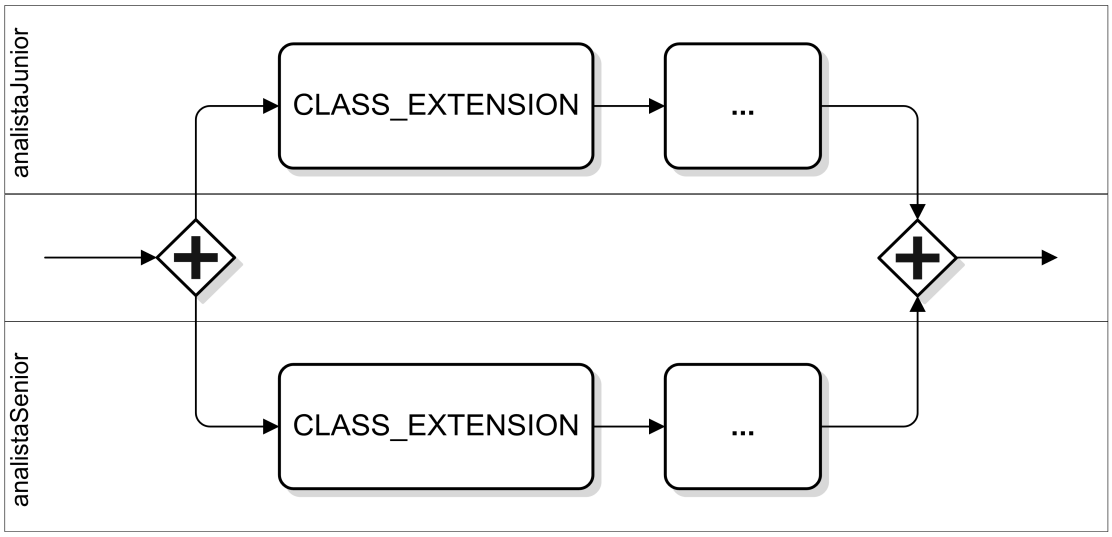


Figura 4.8: Representação em BPMN do comando PARALLEL, linhas 10-19 do Código 4.9

O Anexo I mostra as marcações para o comando PARALLEL em BPMN-Activiti. O abre chaves de PARALLEL ("{") gera a marcação **parallelGateway,** em negrito na primeira linha, divergindo para os fluxos em paralelo demarcados pelo FLOW, e o fecha chaves gera a marcação **parallelGateway,** última marcação em negrito, convergindo para o fluxo seguinte. As marcações para o FLOW são as mesmas explicitadas anteriormente para ROLE.

Por último, o comando DOPARALLEL, linhas 21-29 do Código 4.9, expressa que as atividades interativas do bloco ROLE serão oferecidas para o grupo *analistaSenior*, e o reutilizador irá delegar esses blocos quantas vezes desejar e esperará a realização das atividades para prosseguir, neste exemplo, para finalizar o processo de reutilização. No entanto, iremos fazer uso do comando DOPARALLEL em sua versão mais simples para facilitar o entendimento do mapeamento para BPMN-Activiti. O Código 4.11 ilustra o comando DOPARALLEL, onde não temos marcação ROLE e o bloco de atividades contém somente a atividade ExternalTask.

Código 4.11: Comando DOPARALLEL

```
DOPARALLEL{
        ExternalTask;
}WHILE ("Executar novamente o bloco?");
```

BPMN não suporta diretamente a implementação do comando DOPARALLEL, portanto, faz-se necessário combinar vários elementos, como é representado na Figura 4.9. Os gateways foram numerados de cima para baixo e da esquerda para a direita com o objetivo de facilitar a identificação da correspondência com o Código 4.11. O gateway 1 do tipo XOR implementa o "DOPARALLEL {", isso significa que ele passará o fluxo para o gateway 2 assim que receber um fluxo do comando que antecede o DOPARALLEL, entrada no comando, ou do gateway 4 quando a resposta da pergunta "Executar novamente o bloco?" for positiva. As variáveis numCreatedInstances e numExecutedInstances serão inicializadas com zero quando o fluxo vir do comando que antecede o DOPARALLEL. O gateway 2, do tipo Parallel, divide o fluxo em dois, um segue para a atividade ExternalTask e o outro para EvalCondition. Finalmente, o Anexo II mostra o resultado da conversão do Código 4.11 para BPMN-Activiti.

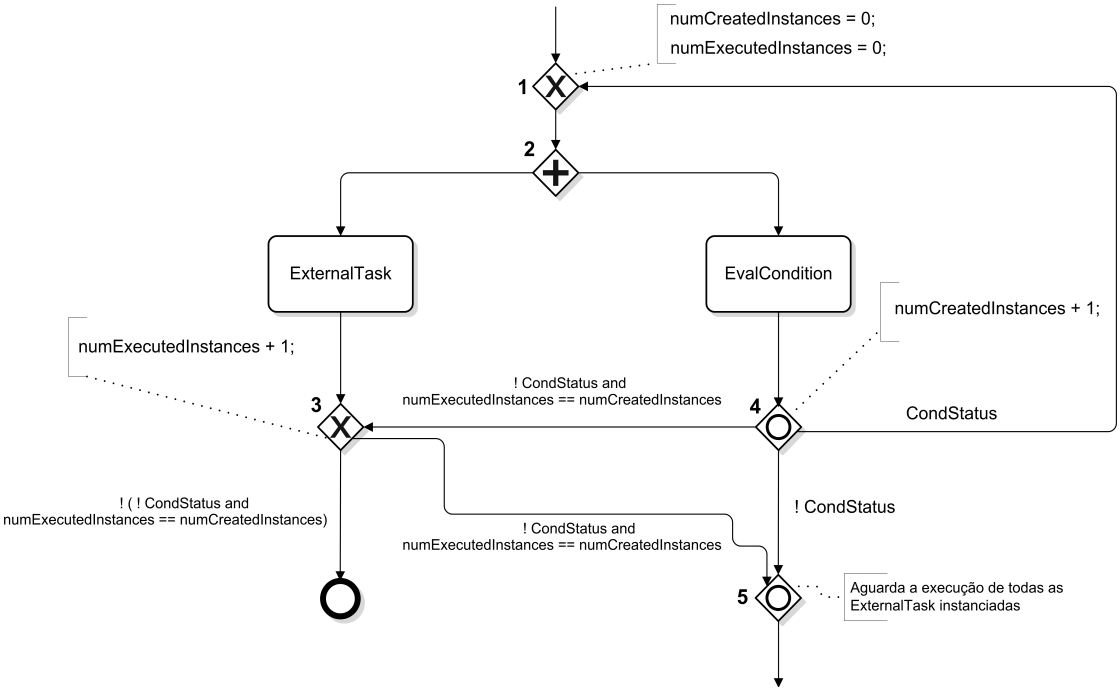


Figura 4.9: Representação em BPMN do Código 4.7

4.5 - Considerações Finais

Este capítulo mostrou como são implementados os elementos do conceito colaboração em CollabRDL, apresentou os novos comandos ROLE, PARALLEL, DOPARALLEL e o ambiente de execução CollabRDL-Activiti-Explorer, e finalizou mostrando o mapeamento dos novos comandos para BPMN.

Cabe ressaltar que os comandos PARALLEL e DOPARALLEL produzem regiões críticas, tema tratado em linguagens para programação concorrente (SCOTT, 2009). Além disso, o mapeamento para BPMN foi feito com o objetivo de execução, não sendo portanto, somente notacional.

CAPÍTULO 5 - AVALIAÇÃO DA CollabRDL

Este capítulo expressa em CollabRDL alguns padrões de workflow, descreve a realização de uma prova de conceito e finaliza com as considerações finais.

5.1 - Introdução

O interesse e a necessidade pela Engenharia de Software Experimental tem crescido e é considerado uma boa prática a escolha de um método para a realização dos estudos (EASTERBROOK *et al.*, 2007, RUNESON e HÖST, 2009). A Engenharia de Software Baseada em Evidências foi introduzida por KITCHENHAM, DYBÅ e JØRGENSEN (2004) com o objetivo de fornecer meios para integrar evidências nas pesquisas com as experiências práticas e os valores humanos na tomada de decisão sobre o desenvolvimento e manutenção de software. Ela é baseada no sucesso das evidências na área médica. Um exemplo da importância de se relatar as evidências nas pesquisas pode ser vista no artigo "Workflow patterns put into context", onde VAN DER AALST e TER HOFSTEDE (2012), embora argumentem a validação dos padrões de workflow, admitem que estudos experimentais mais elaborados são bem-vindos.

MOHAGHEGHI e CONRADI (2007) fizeram uma revisão das evidências na indústria entre os anos de 1994 e 2005 com o objetivo de caracterizar a reutilização de software quanto à qualidade, produtividade e benefícios econômicos, e chegaram a conclusão que existem evidências positivas na melhoria da qualidade e aumento da produtividade, mas não é conclusivo quanto aos benefícios econômicos. No contexto da RDL, existe um estudo de caso sobre a reutilização de 5 frameworks orientados a objetos (OLIVEIRA, ALENCAR e COWAN, 2011) e um experimento (SALVADOR, 2009) comparando o processo de instanciação de framework com RDL e técnica manual no foco de produtividade e qualidade, ambos os estudos indicam que é preciso melhorar o ambiente de execução da RDL, ReuseTool, principalmente quanto à percepção do ambiente, awareness.

A ferramenta ReuseTool já apresentava limitações para experimentos com RDL em ambiente monousuário, no caso da CollabRDL a complexidade e exigências sobre o

ambiente de execução aumentam, já que é preciso que atendam às necessidades de multiusuários, principalmente nas questões de percepção dos usuários com relação ao contexto de trabalho em grupo, awareness (MANGAN, BORGES e WERNER, 2004). O ambiente CollabRDL-Activiti-Explorer usa as funcionalidades do Activiti-Explorer para permitir trabalho em grupo, no entanto, a percepção do ambiente, awareness, ainda é limitada.

A estratégia de avaliação de CollabRDL nesta dissertação de mestrado foi avaliar os seus comandos no contexto de alguns padrões de workflow na Seção 5.2. Estes padrões são utilizados para comparar ferramentas de workflow, indicando se a ferramenta implementa ou não determinado padrão (WOHED *et. al*, 2009, AVALWF, 2013, AVALCOMWF, 2013, AVALOSWF, 2013). Também foi realizada uma prova de conceito para exercitar os seus comandos na descrição e execução do processo de reutilização de um framework orientado a objetos no ambiente CollabRDL-Activiti-Explorer, apresentada na Seção 5.3.

Prova de Conceito pode ser a construção de uma ferramenta para a avaliação de uma nova técnica/abordagem, como uma fase anterior a de um estudo experimental para validação (HAYES e OFFUTT, 2006). Além disso, prova de conceito, também chamada de implementação de conceito, aparece como um dos métodos de pesquisa mais utilizados nos anos 90 (GLASS *et. Al*, 2002), e mais tarde, em outro estudo com artigos publicados entre 1998 e 2003, corresponde somente à 7% , ainda com um percentual superior aos métodos de revisão da literatura e metanálise que aparecem com 6% (SEGAL *et. al*, 2005). Finalmente, entendemos que a implementação de uma ferramenta para avaliar os comandos CollabRDL como uma prova de conceito é importante porque nos ajudará na identificação das características e nos refinamentos do ambiente necessários para que se possa conduzir novos estudos de reutilização em grupo.

5.2 - Padrões para workflow

Workflow é definido como *sistemas que ajudam as organizações a especificar, executar, monitorar e coordenar o fluxo de casos de trabalho dentro de um ambiente de escritório distribuído* (BULL CORPORATION, 1992). Um workflow apresenta dois

momentos, o primeiro momento é para expressar o processo de trabalho, a sua modelagem em si, evidenciando as atividades, o segundo é a sua execução, onde o processo expresso, ao ser executado, gera o trabalho que tem como consequência o produto ou serviço. Embora se possa modelar todos os processos e atividades, alguns, pela sua natureza ainda precisam ser realizados fora do meio computadorizado. A Tabela 5.1 lista algumas das definições presentes em sistemas de workflow (WFMC, 1999), o objetivo aqui foi listar somente aquelas que estão relacionadas com o objetivo desta dissertação de mestrado.

Tabela 5.1: Workflow e suas definições (WFMC, 1999)

Conceito	Definição
Sistema de Gerenciamento de Workflow	Um sistema que define, cria e gerencia a execução de workflows através do uso de software, executando um ou mais *workflow engines* (um serviço de software), que está apto a interpretar a definição de processo, interagir com os participantes do workflow, e quando requisitado, invocar diretamente as aplicações ou os seus usuários.
Processo de Negócio	Um conjunto de um ou mais procedimentos ou atividades interligadas que coletivamente realizam um objetivo de negócio ou meta política, normalmente dentro do contexto de uma estrutura organizacional que define papéis funcionais e relacionamentos.
Processo	Uma visão formal de um processo de negócio, representada de forma coordenada (paralelo e / ou sequencial) como um conjunto de atividades que estão conectadas, a fim de alcançar um objetivo comum.
Atividade	Uma descrição de uma peça de trabalho que

Conceito	Definição
	constitui um passo lógico dentro de um processo. Uma atividade pode ser manual, que não suporta automatização de computador, ou de workflow (automatizada). Uma atividade de workflow requer recursos humanos ou computacionais para suportar a execução do processo; onde recurso humano é necessário para associar uma atividade a um participante do workflow.
Instância de processo	A representação de uma simples execução (enactment) de um processo.
Distribuir em paralelo	Uma parte de uma instância de processo em execução por um sistema de gerenciamento de workflow, onde duas ou mais instâncias de atividades estão sendo executadas em paralelo dentro do workflow, dando origem a múltiplos segmentos de controle.
AND-Split	Um ponto no workflow, onde um único segmento de controle se divide em dois ou mais segmentos que são executados em paralelo dentro do workflow, permitindo que múltiplas atividades possam ser executadas simultaneamente.
AND-Join	Um ponto no workflow onde dois ou mais segmentos de atividades sendo executadas em paralelo convergem para um segmento único de controle.
OR-Split	Um ponto no workflow, onde um único segmento de controle faz com que uma decisão sobre qual ramo a tomar quando encontrou com várias ramificações de workflow alternativos.
OR-Join	Um ponto no workflow onde duas ou mais

Conceito	Definição
	atividades alternativas de workflow convergem para uma única atividade comum como o próximo passo dentro do workflow.
Papel (Organizational Role)	Um grupo de participantes apresentando um específico conjunto de atributos, qualificações e/ou habilidades.

A iniciativa de criar Padrões para Workflow, ou do termo original em inglês, Workflow Patterns, foi com o objetivo de delinear os requisitos fundamentais que surgem durante a modelagem de processos de negócios de forma recorrente e descrevê-los de forma imperativa (VAN DER AALST *et al.*, 2003). VAN DER AALST *et al*. (2003) apresentaram 20 padrões, e alguns anos mais tarde, RUSSELL *et al.* (2006) fizeram uma revisão. Para os propósitos desta dissertação de mestrado, serão apresentados 7 padrões que estão alinhados com coordenação e trabalho em grupo, sempre descrevendo a sua representação em CollabRDL.

Além desses padrões apresentados aqui, é possível expressar com CollabRDL outros padrões de workflow, tais como o WCP7, Mesclagem Sincronizada Estruturada, e o WCP12, Múltiplas Instâncias sem Sincronização, no entanto, outros ainda não, o WCP39, sobre região crítica e o WRP1, sobre Distribuição.

5.2.1 - Padrão Divisão em Paralelo (WCP2)

É a divisão de um ramo em um ou mais que podem ser executados em paralelo. Também conhecido como AND-split, *parallel routing, parallel split e fork*. A Figura 5.1 ilustra esse padrão na notação do BPMN 2.0 e o Código 5.1 expressa esse padrão em CollabRDL.

Simulação:

http://www.workflowpatterns.com/patterns/control/basic/wcp2_animation.php

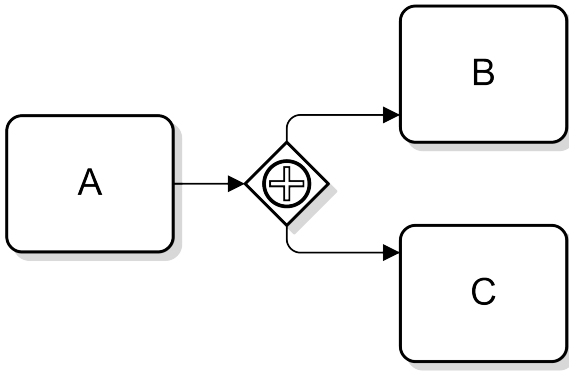


Figura 5.1: Padrão Divisão em Paralelo (AND-split) na notação BPMN 2.0, adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.1: Padrão Divisão em Paralelo (AND-split) expressado em CollabRDL

```
atividadeA;
PARALLEL{
    FLOW (...){
        atividadeB;
    }
    FLOW (...){
        atividadeC;
    }
}
```

O comando PARALLEL é a implementação deste padrão combinado com o padrão Distribuição Baseadas em Papéis (WRP2) apresentado na Seção 5.2.6. A atividade *A* em BPMN é mapeada para *atividadeA* em CollabRDL. Após a sua execução , a divisão de ramos é indicada por PARALLEL em CollabRDL e pelo símbolo de um quadrado com um círculo e cruz em BPMN. Os ramos das atividades B e C em BMPN possuem correspondência direta com os blocos de FLOW em CollabRDL.

5.2.2 - Padrão para Sincronização (WCP3)

A convergência de dois ou mais ramos em um único subsequente ramo tal que o segmento de controle é passado para o ramo posterior quando todos os ramos de entrada forem habilitados. Seus sinônimos são AND-join, *rendezvous, synchronizer*. A Figura

5.2 ilustra esse padrão em BPMN 2.0 e o Código 5.2 expressa esse padrão em CollabRDL.

Simulação:

http://www.workflowpatterns.com/patterns/control/basic/wcp3_animation.php

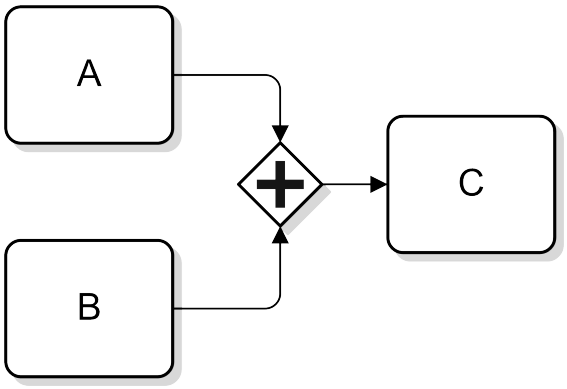


Figura 5.2: Padrão para Sincronização (AND-join) na notação BPMN 2.0 , adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.2: Padrão para Sincronização (AND-join) expressado em CollabRDL

```
PARALLEL{
   FLOW (...){
      atividadeA;
   }
   FLOW (...){
      atividadeB;
   }
}
atividadeC;
```

A *atividadeC* em CollabRDL tem correspondência direta com a atividade *C* em BPMN, elas são executadas somente após a execução do blocos de FLOW em CollabRDL e pelas atividades *A* e *B* em BPMN, respectivamente. Assim, concluímos que CollabRDL implementa o padrão Sincronização (WCP3), AND-join.

5.2.3 - Padrão Escolha Exclusiva (WCP4)

A divisão de um ramo em dois ou mais ramos. Quando o ramo de entrada é habilitado, o segmento de controle é imediatamente passado para um dos ramos de saída com base no resultado de uma expressão lógica associada com o ramo. Seus sinônimos são XOR-split, exclusive OR-split, *conditional routing, switch, decision, case statement*. A Figura 5.3 ilustra esse padrão em BPMN 2.0 e o Código 5.3 expressa esse padrão em CollabRDL, o comando IF-ELSE faz parte da RDL e foi herdado pela CollabRDL sem alterações.

Simulação:

http://www.workflowpatterns.com/patterns/control/basic/wcp4_animation.php

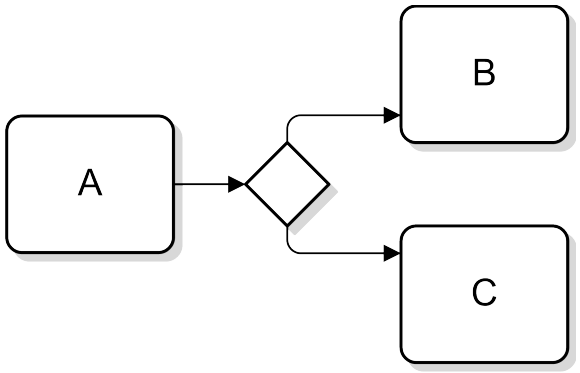


Figura 5.3: Padrão Escolha Exclusiva (XOR-split) na notação BPMN 2.0, adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.3: Padrão Escolha Exclusiva (XOR-split) expressado em CollabRDL

```
atividadeA;
IF ("Você desejá executar a atividadeB?") THEN{
      atividadeB;
}ELSE{
      atividadeC;
}
```

5.2.4 - Padrão Mesclagem Simples (WCP5)

A convergência de dois ou mais ramos em um único e subsequente ramo. As saídas dos ramos são entradas para o segmento de controle do ramo subsequente. A próxima atividade irá utilizar somente uma das saídas dos ramos anteriores. Seus sinônimos são XOR-join, *exclusive* OR-join, *asynchronous join, merge*. A Figura 5.4 ilustra esse padrão em BPMN 2.0 e o Código 5.4 expressa esse padrão em CollabRDL.

Simulação:

http://www.workflowpatterns.com/patterns/control/basic/wcp5_animation.php

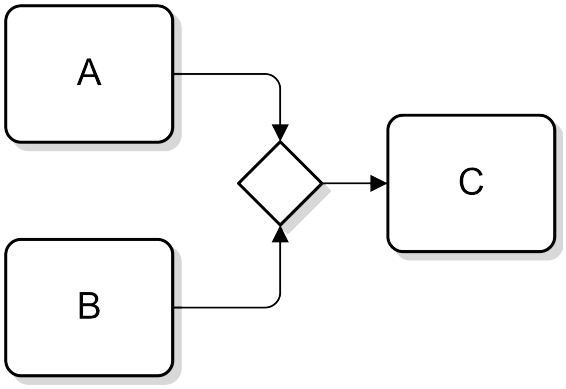


Figura 5.4: Padrão Mesclagem Simples (XOR-join) na notação BPMN 2.0, adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.4: Padrão Mesclagem Simples (XOR-join) expressado em CollabRDL

```
IF ("Você desejá executar a atividadeA?") THEN{
      atividadeA;
}ELSE{
      atividadeB;
}
atividadeC;
```

Contudo, esse código não expressa que as atividades *A* e *B* foram iniciadas e que o término de uma das duas passa o fluxo para a *atividade C*. No entanto, é preciso haver evidências que esse tipo de comando é necessário em uma reutilização colaborativa

antes de adicioná-lo à CollabRDL. Assim concluímos que CollabRDL ainda não implementa totalmente o padrão Mesclagem Simples (WCP5).

5.2.5 - Padrão Escolha Múltipla (WCP6)

A divisão de um ramo em dois ou mais ramos. Quando o ramo de entrada é habilitado, o segmento de controle é imediatamente passado para um ou mais ramos de saída com base no resultado da expressão lógica associada com o ramo. Seus sinônimos são *Conditional routing, selection,* OR-split, *multiple choice*. A Figura 5.5 ilustra esse padrão em BPMN 2.0 e o Código 5.5 expressa esse padrão em CollabRDL.

Simulação:

http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp6_animation.php

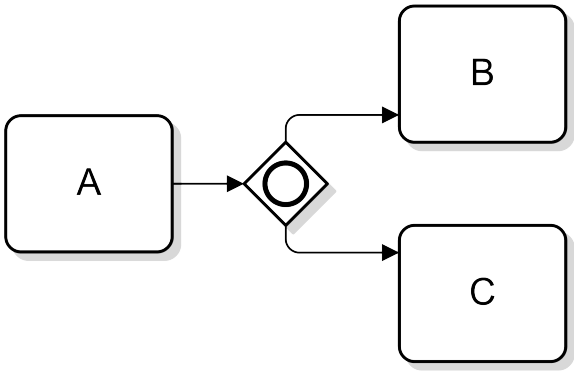


Figura 5.5: Padrão Escolha Múltipla na notação BPMN 2.0, adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.5: Padrão Escolha Múltipla expressado em CollabRDL

```
atividadeA;
PARALLEL{
    FLOW (...){
      IF ("Você desejá executar a atividadeB?") THEN{
            atividadeB;
      }
    }
    FLOW (...){
      IF ("Você desejá executar a atividadeC?") THEN{
            atividadeC;
      }
    }
}
```

O Código 5.5 indica que os blocos de FLOW só serão executados se obtiverem uma resposta positiva na avaliação do comando IF. Essa combinação com IF torna o comando PARALLEL, originalmente AND-split, em OR-split. Assim, concluímos que CollabRDL implementa o padrão Escolha Múltipla (WCP6).

5.2.6 - Distribuição Baseada em Papéis (WRP2)

Este padrão expressa em tempo de desenvolvimento uma ou mais *roles* (papéis) para que as atividades sejam distribuídas em tempo de execução. *Roles* servem para agrupar pessoas em grupos que possuem características/habilidades comuns. Quando uma atividade é associada assim, ela em tempo de execução é oferecida para os grupos (roles) associados. A Figura 5.6 ilustra este padrão em BPMN 2.0 utilizando a *lane analista* para marcar o papel *analista*. O Código 5.6 exibe o comando ROLE criado para atender esse padrão.

Simulação:

http://www.workflowpatterns.com/patterns/resource/creation/wrp2_animation.php

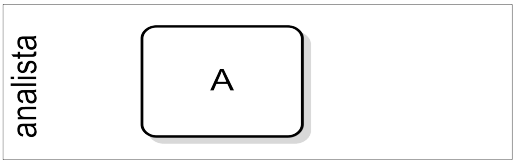


Figura 5.6: Padrão Escolha Múltipla na notação BPMN 2.0, adaptado de RUSSELL *et al.* (2006) e de WHITE (2004)

Código 5.6: Padrão Distribuição Baseada em Papéis expressado em CollabRDL

```
ROLE (analista, "Comentário"){
   atividadeA;
}
```

A Figura 5.6 ilustra a *lane,* retângulo com o papel *analista* escrito à esquerda na vertical, para indicar que todas as atividades que estão dentro do retângulo serão

executadas por pessoas do grupo *analista,* neste exemplo somente a atividade *A*. Em CollabRDL, como visto no Código 5.6, faz o mesmo quando indica no primeiro parâmetro de ROLE o grupo *analista* para realizar as atividades contidas no bloco abre e fecha chaves, neste exemplo somente a *atividadeA*. Concluímos então que o padrão Distribuição Baseada em Papéis (WRP2) é implementado pelo comando ROLE em CollabRDL.

5.2.7 - Múltiplas Instâncias com decisão em tempo de execução (WCP14)

Este padrão descreve a criação de várias instâncias de uma atividade em um processo. Elas são independentes uma das outras, por isso podem ser executadas em paralelo, e o número de instâncias será decidido em tempo de execução. É necessário sincronizar as instâncias ao final antes de passar o fluxo para as próximas atividades. As Figuras 5.7 e 5.8 ilustram este padrão nas suas variantes, em paralelo e em série na notação BPMN 2.0. O Código 5.7 mostra o comando DOPARALLEL para expressar a realização de múltiplas instâncias de uma atividade em paralelo com CollabRDL. Por outro lado, para expressar a criação de várias instâncias de uma atividade para serem executadas serialmente, a combinação do comando LOOP e ROLE produz o efeito desejado, veja a Tabela 5.9.

Simulação:

http://www.workflowpatterns.com/patterns/control/multiple_instance/wcp14_animation.php

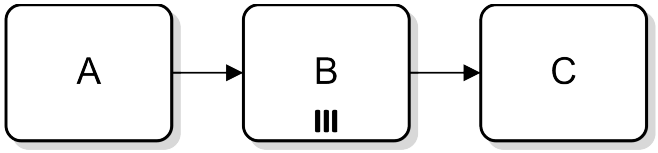


Figura 5.7: Múltiplas Instâncias de B em paralelo

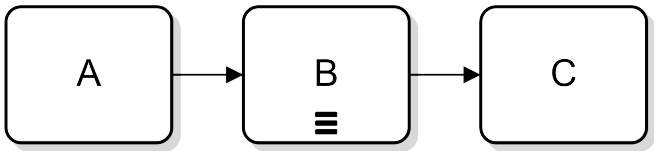


Figura 5.8: Múltiplas Instâncias de B em série

Código 5.7: Padrão Múltiplas Instâncias com decisão em tempo de execução expresso paralelamente em CollabRDL

```
atividadeA;
DOPARALLEL{
   ROLE (analista, "Um comentário!"){
      atividadeB;
   }
}WHILE (“Executar novamente o bloco?”);
atividadeC;
```

Em algumas situações, ao descrever um processo de reutilização, existe a necessidade de executar os blocos em paralelo. O comando DOPARALLEL expressa esse comportamento, o Código 5.7 expressa que em tempo de execução, após a execução da *atividadeA*, a *atividadeB* será oferecida para o grupo *analista*, e sem a necessidade de esperar o seu término, a pergunta “Executar novamente o bloco?” será emitida, se a resposta for positiva colocará mais uma instância da *atividadeB* para o grupo *analista*, e se a resposta for negativa, o fluxo irá para o final do DOPARALLEL, marcado pelo ponto e vírgula (“;”), onde irá esperar o término de todas as atividades contidas no bloco do comando DOPARALEL.

Código 5.8: Padrão Múltiplas Instâncias com decisão em tempo de execução expressado serialmente em CollabRDL

```
atividadeA;
LOOP (“Executar o bloco do loop?”){
     ROLE (...){
          atividadeB;
     }
}
atividadeC;
```

Em outras situações, diferente do comportamento paralelo obtido com o comando DOPARALLEL, existe a necessidade de repetir uma sequência de atividades por um número de vezes definido em tempo de execução. Uma forma de expressar isso é através do comando LOOP. O Código 5.8 ilustra esta situação, em tempo de execução é emitida a pergunta "Executar o bloco do loop?", caso a resposta seja positiva, a *atividadeB* será posta para execução, e somente após a sua execução, será novamente emitida a pergunta do LOOP. Este comportamento indica que o comando LOOP produz a execução de n blocos de código de foma serial com a determinação de n em tempo de execução.

Concluímos então que CollabRDL implementa o padrão Múltiplas Instâncias com decisão em tempo de execução nas suas variantes em paralelo e serial. A variante paralelo através dos comandos DOPARALLEL e ROLE, já a serial com LOOP e ROLE. A próxima Seção ilustra a utilização desses comandos na instanciação de um framework orientados a objetos, mostrando portanto a sua necessidade em um caso específico de reutilização. No entanto, acreditamos que esse tipo de estrutura também seja útil na descrição de reutilização de artefatos genéricos.

5.3 - Prova de Conceito

Esta prova de conceito é para avaliar os comandos ROLE, PARALLEL E DOPARALLEL quanto às suas funcionalidades com o objetivo de realizar atividades de reutilização em equipe e quando possível em paralelo. As questões para este objetivo são 1 – O comando ROLE consegue oferecer um bloco de atividades para um determinado grupo? 2 - O comando PARALLEL consegue criar fluxos de execução de blocos de atividades em paralelo? 3 - O comando DOPARALLEL consegue criar fluxos de execução de um mesmo bloco de atividades? Para responder estas questões iremos fazer uso das seguintes métricas: 1- número de atividades contidas nos blocos das atividades; 2 - número de papéis; 3 – número de fluxos em paralelo.

Para isso utilizamos o framework Openswing, Seção 5.3.1, criamos um modelo de uma aplicação alvo, 5.3.2, um programa em CollabRDL para gerar aplicações deste domínio, 5.3.3, e finalmente realizamos uma execução com duas pessoas distantes fisicamente utilizando o ambiente de execução CollabRDL-Activiti-Explorer, descrita

em detalhes na Seção 5.3.4. A primeira iniciou e conduziu o processo de reutilização e realizou as atividades delegadas para o grupo *designer* e *Administrador de Banco de Dados*, enquanto que a segunda realizou as atividades do grupo *analista*. Ambas as pessoas têm conhecimento de RDL. A primeira, o autor desta dissertação, que, portanto, tem conhecimento dos novos comandos e do ambiente de execução. Já o segundo participante possui conhecimento da RDL e recebeu, por escrito, a descrição dos novos comandos CollabRDL antecipadamente, já do ambiente, somente no dia da execução da prova de conceito.

Esta prova de conceito foi idealizada desta forma porque os novos comandos da CollabRDL são de workflow. Além disso, aplicações em Openswing sugerem a participação de pessoas com habilidades/responsabilidades diferentes, sendo portanto adequado para simular os elementos da Tabela 4.1.

5.3.1 - Framework Openswing

Swing é uma biblioteca de componentes gráficos para aplicações Java (SWING, 2013). Já Openswing (OPENSWING, 2013), além de oferecer novos componentes baseados em Swing, também é um framework para suportar o desenvolvimento de aplicações utilizando o paradigma MVC (Modelo, Visão e Controle). Com Openswing é possível desenvolver aplicações tipo desktop, para a internet (em três camadas: Swing + HTTP + Java Servlet + banco de dados), ou aplicações distribuídas (três camadas: Swing + RMI + Java Beans + banco de dados).

Openswing suporta interface MDI (*Multiple Document Interface*) que pode conter várias outras estruturas de telas. A interface MDI permite a customização de várias características, tais como título, menu, seleção de linguagem, autenticação e janelas com menu. A nossa prova de conceito é baseada no exemplo 10 do Openswing (TUTORIAL OPENSWING, 2013), que descreve a criação de uma aplicação MDI cliente servidor. Este exemplo implementa as funcionalidades de cadastro de departamento, empregados e tarefas. A Figura 5.9 apresenta o Demo10 em execução.

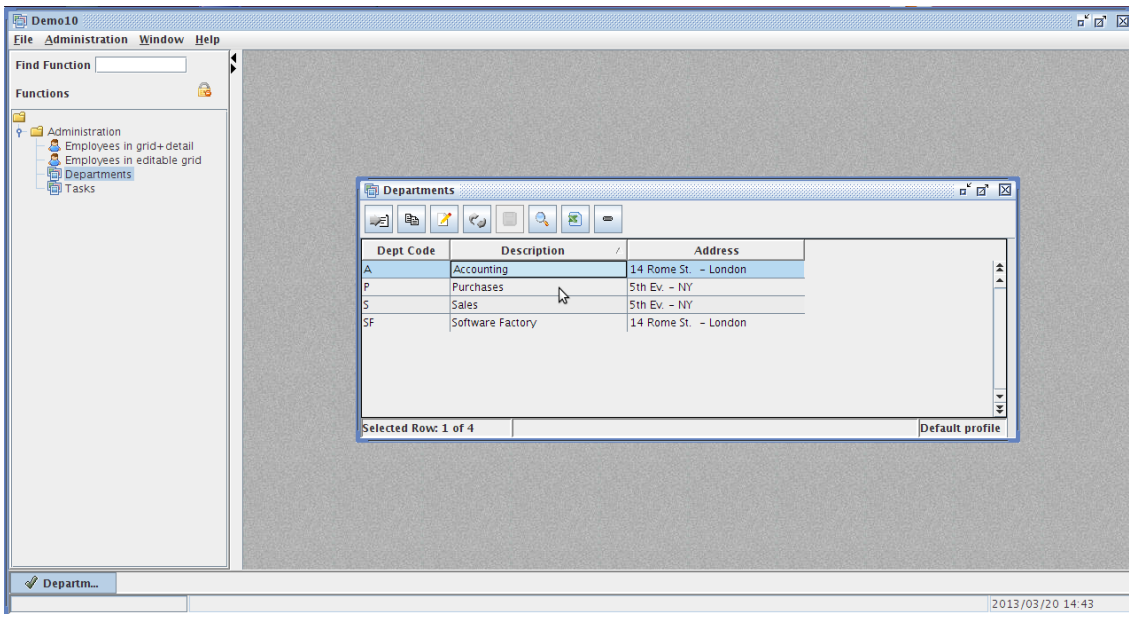


Figura 5.9: Demo10 em execução

O framework OPENSWING permite a criação de aplicações completas na arquitetura MVC, e por isso é organizado para facilitar a distribuição das atividades de reutilização entre equipes com perfis/responsabilidades diferentes, tais como analista, designer, administrador de banco de dados etc. E como consequência disso, bloco de atividade independentes podem ser identificados para serem executados em paralelo, com o objetivo de reduzir o tempo total de criação da aplicação. Por isso, este framework é adequado para ser utilizado em um cenário de reutilização colaborativa.

A Figura 5.10 mostra o modelo de classes em UML para o demo10 do Openswing (TUTORIAL OPENSWING, 2013). As classes do Demo10 são marcadas com “from demo10”. Para facilitar a visualização do modelo, omitimos os atributos e os métodos das classes e interfaces do Openswing e dos atributos do Demo10. O modelo mostra as implementações de interface, as heranças e as dependências entre as classes. Como exemplo, a classe *ClientApplication* implementa a Interface *LoginController,* para adicionar à aplicação as funcionalidades de autenticação e autorização, e MDIController para adicionar as funcionalidades de MDI.

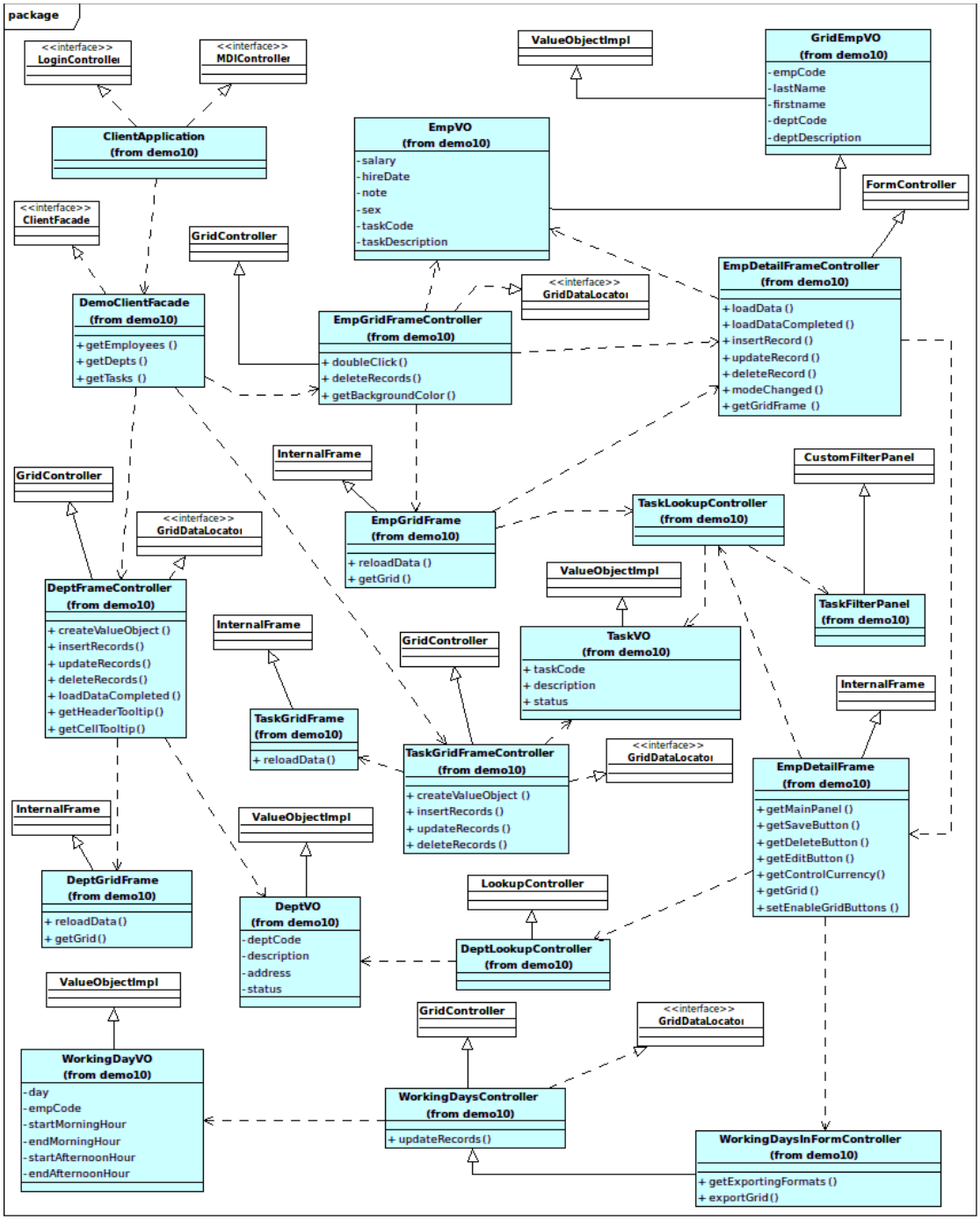


Figura 5.10: Modelo de Classes em UML do Demo10 do Openswing

A classe *DemoClientFacade* é instanciada por *ClientApplication* e através dos seus métodos instancia a criação das classes controles. As classes de controle precisam herdar de *GridController* e implementar a interface *GridDataLocator* para controlar as

ações nas janelas, ligando a camada de dados a camada de visão. Todas as classes que herdam de *InternalFrame*, o fazem para obter o comportamento de janelas para interagir com o usuário, como visto em *Departaments* na Figura 5.10. E as classes que são mapeadas para o banco de dados precisam herdar de *ValueObjectImpl* diretamente, caso de *GridEmpVO,* e indiretamente como é o caso de *EmpVO*.

A Tabela 5.2 mostra os pontos de extensão do Demo10. A classe *GridController* com dois atributos e 48 métodos que podem ser redefinidos apresenta a extensão com maior complexidade. A interface *MDIController* define a necessidade de implementação de 16 métodos. E a interface *ClientFacade* é criada somente para servir como um super tipo, por isso não define serviços. A soma de todos os atributos dos pontos de extensão é 49 e dos que podem ser redefinidos métodos é 90. A conta não inclui os atributos é métodos herdados. Esses números indicam a complexidade de reutilização, os atributos estão associados a manipulação de dados e os métodos ao comportamento da aplicação. Além disso, uma aplicação, para implementar os seus requisitos normalmente o faz através de várias classes de controle, modelo e visão.

Tabela 5.2: Pontos de Extensão do Demo10

Pontos de Extensão	**Atributos**	**Métodos***
ClientFacade	0	0
CustomFilterPanel	0	7
GridController	2	48
GridDataLocator	0	1
InternalFrame	9	2
LoginController	0	4
LookupController	38	11
MDIController	0	16
ValueObjectImpl	0	1
Total	**49**	**90**

* Métodos que podem ser redefinidos

5.3.2 - Nova Aplicação

A nova aplicação será a Demo10 menos as classes *WorkingDayVO, WorkingDaysControlller, WorkingDaysInFormController e TaskFilterPanel.* A construção do modelo dessa aplicação será guiada por um programa CollabRDL em grupo, explorando a realização de atividades em paralelo. A Figura 5.11 mostra o modelo UML marcando as classes que devem ser construídas pelos grupos. As classes *ClientApplication* e *DemoClientFacade,* marcadas com a letra I à esquerda da classe*,* devem ser construídas por quem inicia o processo de instanciação, uma pessoa que além de ter conhecimento do framework, pode delegar tarefas a grupo de pessoas. As classes *EmpGridFrame, TaskGridFrame, DeptGridFrame e EmpDetailFrame,* marcadas com a letra D à esquerda*,* devem ser oferecidas para o grupo *designer*. Já as classes de controle, letra A, para o grupo *analista*.

Cabe ressaltar que, como a nova aplicação é baseada no Demo10, e este no framework Openswing, que por sua vez é organizado por MVC (classes de Modelo, de Visualização e Controle), faz com que a nova aplicação também siga esta organização. Desta forma, as tarefas de reutilização podem assim ser organizadas, necessitando, portanto, de grupos especialistas nestas áreas do framework. Isto fortalece e ajuda a organização da instanciação do framework em grupo.

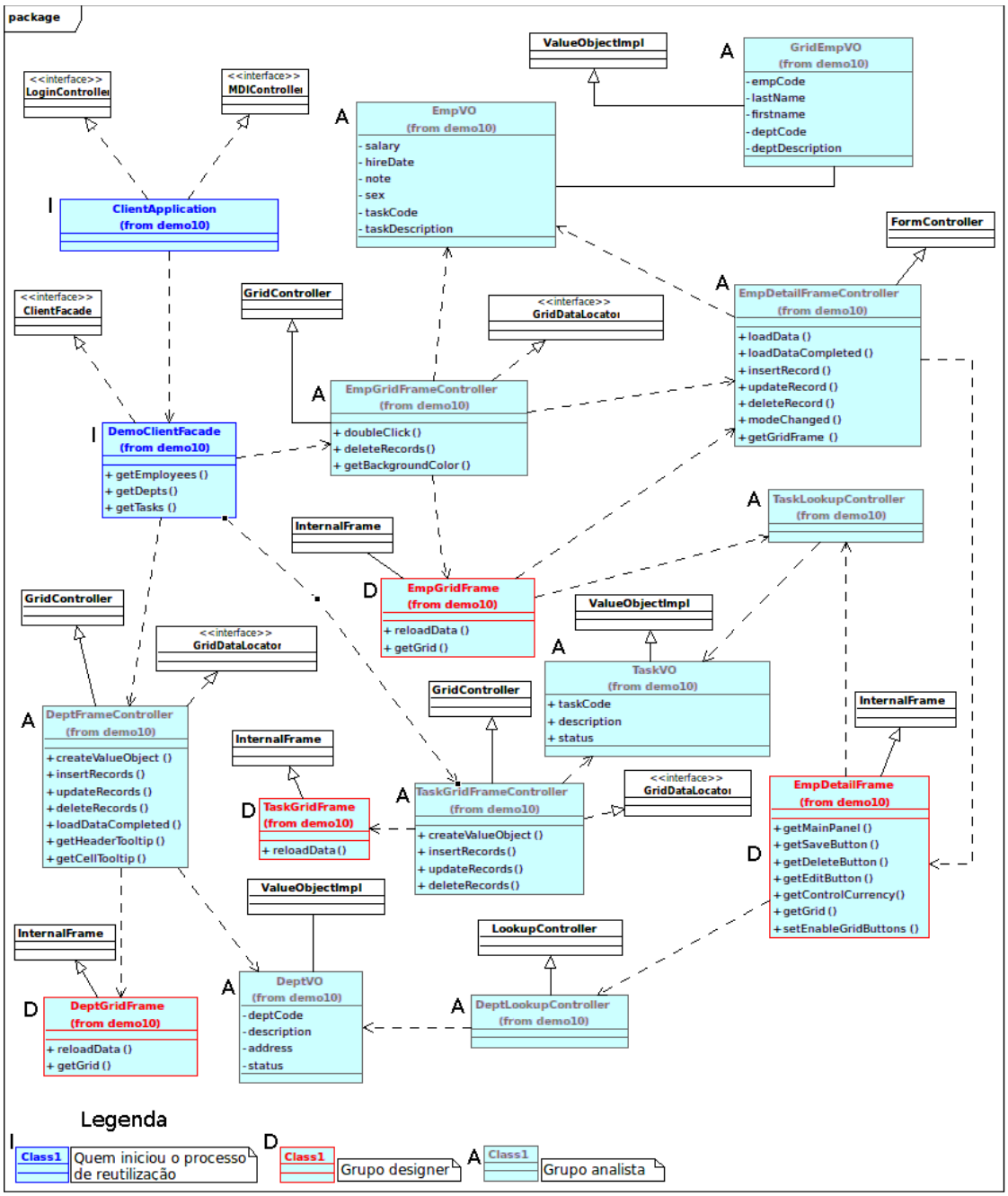


Figura 5.11: Nova aplicação baseada no domínio de aplicações do Demo10

5.3.3 - Programa CollabRDL

Um programa CollabRDL foi escrito para guiar o processo de criação em grupo de novas aplicações para o Demo10 fazendo uso dos novos comando ROLE, PARALLEL e DOPARALLEL (Anexo III). Este programa não inclui a criação das classes *WorkingDayVO, WorkingDaysControlller, WorkingDaysInFormController e TaskFilterPanel,* pois não são necessárias para gerar a aplicação desta prova de conceito.

A Tabela 5.3 mostra em números o programa CollabRDL criado para esta prova de conceito (Anexo III). Consideramos como atividades todos os comandos que adicionam alguma informação ao modelo da aplicação, por exemplo, NEW_REALIZATION que indica que uma classe irá implementar uma interface, e atividades interativas sendo aquelas que para adicionar algo ao modelo precisam que o usuário entre com uma informação, caso do comando CLASS_EXTENSION. Sendo assim, para efeito de comparação, não contamos o IF como uma atividade interativa, pois ele somente solicita uma resposta de sim ou não ao usuário, mas pela sua relevância, criamos uma coluna para explicitar este tipo de interação. Além disso, o comando FLOW que é parte do comando PARALLEL produz o mesmo resultado do comando ROLE, delegar bloco de atividades a grupos, no entanto fizemos a sua conta de forma diferenciada.

Tabela 5.3: Programa CollabRDL em números (aplicações baseadas no Demo10)

	Quantidade	**Atividades (Interativas)**	**FLOW**	**IF**	**LOOP**
DOPARALLEL	1	70 (5)	-	53 (20 aninhados)	0
PARALLEL	1	6 (2)	2	3	3 (1 aninhado)
ROLE	4 (2 aninhados)	- (6)	-	53 (20 aninhados)	0

A Tabela 5.3 nos informa que o maior esforço para instanciar aplicações baseadas no domínio do Demo10 está concentrada na execução do bloco de atividades do comando DOPARALLEL. Ele expressa 70 atividades, sendo 5 interativas e contém

53 IFs, sendo 20 destes aninhados, necessitando portanto muita intervenção do reutilizador na decisão de adicionar ou não itens ao modelo da aplicação. Vale ressaltar que a contagem de IFs do comando ROLE é a mesma para o comando DOPARALLEL, pois o ROLE está contida no seu bloco. E finalmente, os comandos ROLEs e FLOWs associam blocos de atividades aos grupos *analista*, *designer* e *Administrador de Banco de Dados*.

5.3.4 - Executando um processo de reutilização em CollabRDL

O processo Demo10 em BPMN-Activiti, resultado da conversão do programa CollabRDL para guiar o processo de instanciação de aplicações baseadas no domínio do Demo10 (Anexo III), foi carregado no ambiente de execução CollabRDL-Activiti-Explorer. Neste ambiente foram cadastrados dois usuários e os grupos *analista*, *designer* e *Administrador de Banco de Dados*. Um usuário foi associado ao grupo *analista*, e o outro a todos os outros.

Um usuário inicia a execução do processo. O objetivo é, olhando para o modelo da Figura 5.11, reproduzi-lo em grupo utilizando quatro papéis. O usuário executa o primeiro trecho de atividades como mostrado no Código 5.9. No primeiro comando, o ambiente de execução pergunta o nome do pacote onde o modelo da nova aplicação será gerado. Em seguida pergunta o nome da classe cliente da aplicação, onde o usuário deve responder ClienteApplication, como visto na Figura 5.11. Após, sem necessidade de intervenção do usuário, cria duas associações para a classe criada, ambas indicando a implementação das interfaces *MDIController* e *LoginController*. Depois, solicita ao usuário para entrar com o nome da classe fachada, que deve ser DemoClientFacade, e em seguida, associa a nova classe à interface *ClientFacade*, sempre de acordo com o modelo da Figura 5.11.

Código 5.9: Trecho I do processo para gerar uma nova aplicação baseada no Demo10

```
appPack = NEW_PACKAGE(appmodel,"?");
clientApplicationClass = NEW_CLASS(appPack, "?");
NEW_REALIZATION(clientApplicationClass, .      .      .      .      .      .
       .      .      org.openswing.swing.mdi.client.MDIController);
NEW_REALIZATION(clientApplicationClass, .      .      .      .
       org.openswing.swing.permissions.client.LoginController);
clientFacadeClass = NEW_CLASS(appPack, "?");
NEW_REALIZATION(clientFacadeClass, .    .      .      .      .      .      .
       .      org.openswing.swing.mdi.client.ClientFacade);
```

O Código 5.10 mostra o trecho de código reduzido do comando DOPARALLEL. O trecho completo (veja o Anexo III) expressa as atividades necessárias para criar todas as classes de controle e janelas, que são as classes cujos nomes terminam em FrameController e Frame, respectivamente. O primeiro comando do bloco DOPARALLEL pergunta, ainda a quem iniciou o processo, o nome do novo método de fachada para a classe referenciada por *clientFacadeClass*, classe criada no Trecho I. A resposta deve ser, para este exemplo, *getEmployees*, *getDepts* ou *getTasks*, o importante é seguir as dependências como vista no modelo da Figura 5.11.

Código 5.10: Trecho II do processo para gerar uma nova aplicação baseada no Demo10

```
DOPARALLEL {
    newFunction = NEW_METHOD(clientFacadeClass, "?");
    ROLE (analyst, "Create frameControllerClass,frameClass and
                                    detailFrameControllerClass."){
      frameControllerClass =
      CLASS_EXTENSION(org.openswing.swing.table.client.GridController,
                                                        appPack, "?");
      NEW_REALIZATION(frameControllerClass,
                  org.openswing.swing.table.java.GridDataLocator);
      IF ("Do you want redefine enterButton method?") THEN {
            m = METHOD_EXTENSION(org.openswing.swing.table.
                  client.GridController,
                  frameControllerClass,enterButton);
      }
      IF ("Do you want redefine doubleClick method?") THEN {
            m = METHOD_EXTENSION(org.openswing.swing.table.
                              client.GridController,
                              frameControllerClass,doubleClick);
      }
         Código suprimido
      ROLE (designer, "Use visual plugin for Eclipse or Netbeans!"){
            frameClass = CLASS_EXTENSION(org.openswing.swing.mdi.
            client.InternalFrame, appPack, "?");
            EXTERNAL_TASK("Edit frameClass using visual plugin for
                                          Eclipse or Netbeans.");
      }
         Código suprimido
   }//End ROLE (analyst....
} WHILE("Create another function?"); //End DOPARALLEL
```

As atividades que criam as classes que têm o nome com final LookupController e VO, respectivamente, são independentes entre si. Assim, como visto no Código 5.11, um bloco de FLOW foi escrito para gerar as classes tipo LookupController e outro para

gerar as VOs, todas oferecidas para o grupo *analyst.* O primeiro FLOW é um LOOP para criar classes LookupController é dependendo da resposta do analista, incluir ou não os métodos *validateCode, loadData e getTreeModel* ao modelo da aplicação. O segundo FLOW apresenta um LOOP para a criação de classes tipo VOs e outro para criar métodos em classes VOs.

Código 5.11: Trecho III do processo para gerar uma nova aplicação baseada no Demo10

```
PARALLEL {
     FLOW (analyst, ""){
     LOOP ("Create another LookupController?"){
     lookupControllerClass = CLASS_EXTENSION(org.openswing.swing.
                        lookup.client.LookupController, appPack, "?");
     IF ("Redefine validateCode method?") THEN {
        m = METHOD_EXTENSION(org.openswing.swing.
                        lookup.client.LookupController,
                        lookupControllerClass,validateCode);
     }
     IF ("Redefine loadData method?") THEN {
        m = METHOD_EXTENSION(org.openswing.swing.
                  lookup.client.LookupController,
                  lookupControllerClass,loadData);
     }
     IF ("Redefine getTreeModel method?") THEN {
        m = METHOD_EXTENSION(org.openswing.swing.
                        lookup.client.LookupController,
                        lookupControllerClass,getTreeModel);
     }
     }//End LOOP
     }//End FLOW
     FLOW (analyst, ""){
     LOOP ("Create another VOClass?"){
      VOClass =   CLASS_EXTENSION(org.openswing.swing.
                  message.receive.java.ValueObjectImpl,
                  appPack, "?");
      LOOP ("Create another attribute in VOClass?"){
            NEW_ATTRIBUTE (VOClass,"?",int);
      }
      }//End LOOP
      }//End FLOW
}// End PARALLEL
```

E o quarto e último trecho, Código 5.12, adiciona à lista das atividades do grupo *databaseAdministrator* a tarefa de criar um banco de dados, externa por não manipular o modelo da aplicação em UML, mas sim de se tratar de uma tarefa que precisa de outro ambiente de execução.

Código 5.12: Trecho IV do processo para gerar uma nova aplicação baseada no Demo10

```
ROLE (databaseAdministrator, "See vOClass created."){
      EXTERNAL_TASK("Create Database.");
}
```

As Figura 5.12 e 5.13 são as fotos de telas da execução desta prova de conceito. Os usuários estavam fisicamente distantes e utilizaram o Skype para comunicação durante todo o processo de criação do modelo da aplicação. Embora o ambiente permita salvar o processo e retornar em outro horário, a execução foi feita durante uma sessão de trabalho e durou aproximadamente 1h e 30 minutos. O usuário que executou o bloco de atividades do comando DOPARALLEL tirou as suas dúvidas no momento de execução das atividades. Desta forma, o modelo da aplicação foi gerado nesta prova de conceito de forma colaborativa, isso porque a *coordenação* foi expressa em CollabRDL, houve *comunicação* via Skype e textual através do ambiente de execução, e houve *cooperação* já que duas pessoas geraram partes do modelo com diferentes papéis, alcançando o modelo completo para a aplicação planejada.

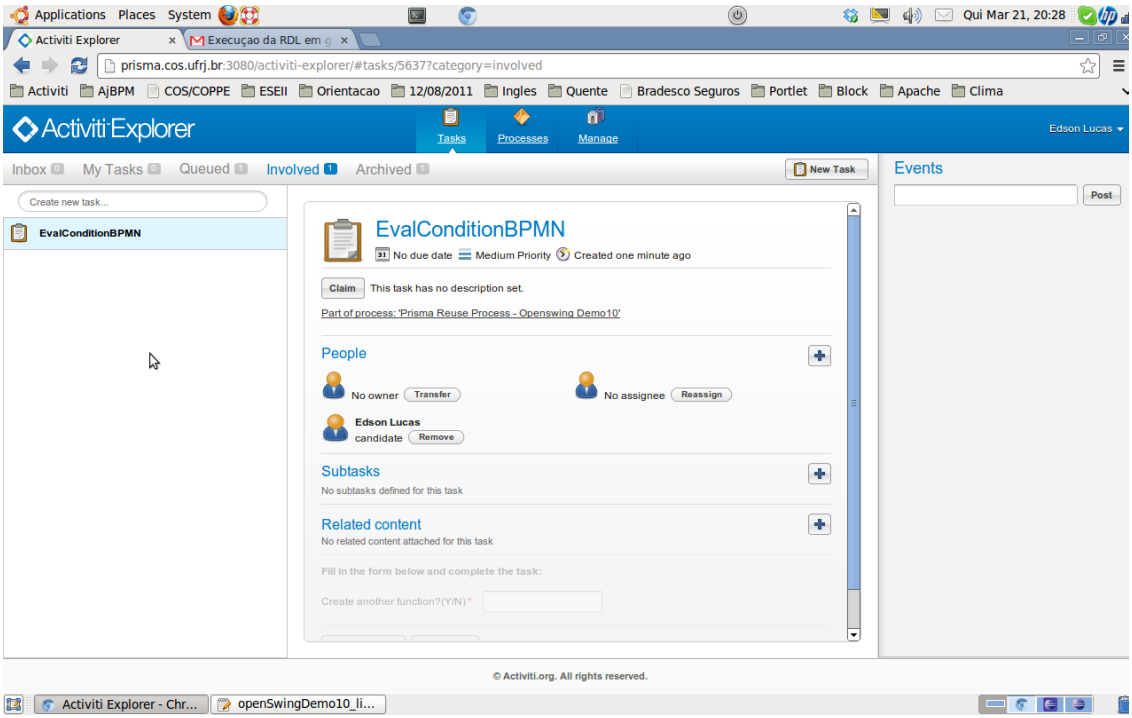


Figura 5.12: Tela de execução CollabRDL-Activiti-Explorer - usuário I

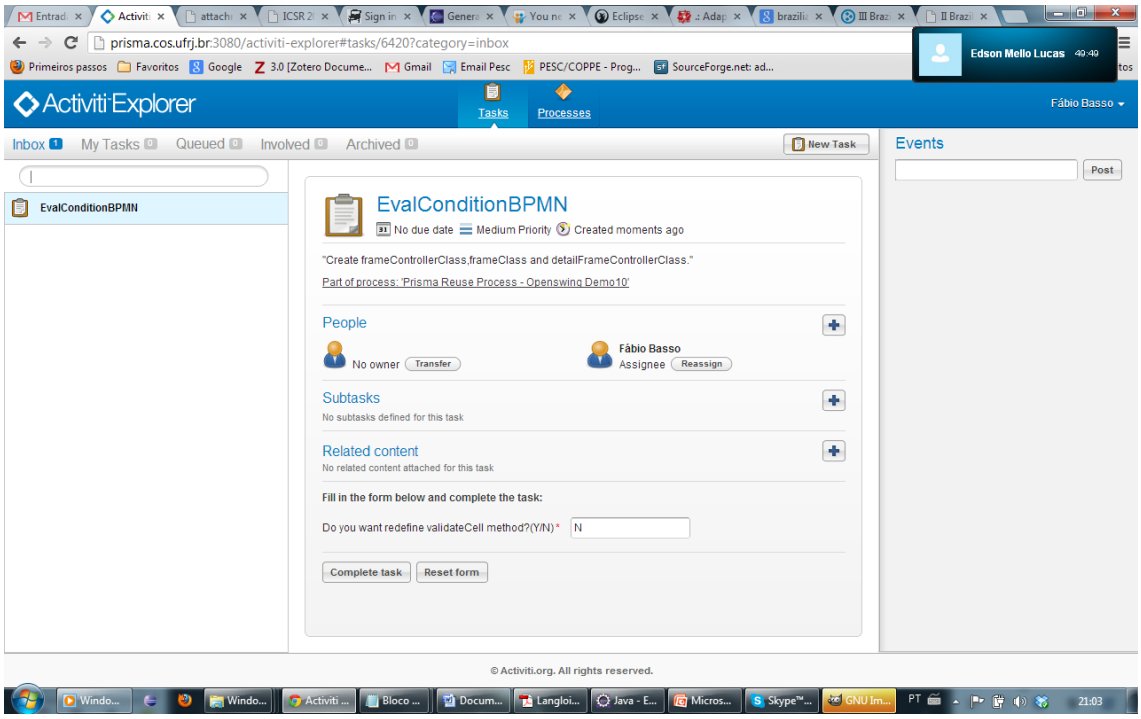


Figura 5.13: Tela de execução CollabRDL-Activiti-Explorer - usuário II

A Tabela 5.4 compara o modelo alvo de aplicação com o modelo da aplicação gerada pela execução desta prova de conceito. Com exceção da classe *EmpDetailFrameController* , todas as demais estão presentes no modelo gerado. Como resultado geral, o modelo apresentou 24 elementos a mais e 13 a menos em comparação ao modelo de aplicação alvo, Figura 5.11. Para efeito de contagem, são considerados elementos no modelo os atributos, as operações, a implementação de interface, descrição de herança, e a própria presença da classe. Na segunda coluna da Tabela 5.4, apresentamos o número de elementos esperados por classe, na terceira, o número de elementos criados através da execução das atividades de reutilização por classe descritas em CollabRDL no Demo 10, e na última coluna identificamos as causas que geraram as diferenças, quarta coluna.

Tabela 5.4: Comparação do modelo de aplicação gerado com o modelo alvo

Classes	Elementos no modelo alvo (Figura 6.11)	Elementos no modelo gerado	Diferenças	Causas
ClientApplication	3	3	0	
DemoClientFacade	5	5	0	
EmpGridFrameController	6	13	►8 elementos a mais; ►uma a menos;	►Awarene ►Não expressado em CollabRDL;
EmpDetailFrameController	9	0	9	Desconhecido
DeptFrameController	10	18	8 elementos a mais	Awareness
TaskGridFrameController	7	15	8 elementos a mais	Awareness
EmpGridFrame	4	2	2 a menos	Atividade Externa
EmpDetailFrame	9	2	7 a menos	Atividade Externa
DeptGridFrame	4	2	2 a menos	Atividade Externa
TaskGridFrame	3	2	1 a menos	Atividade Externa
DeptLookupController	2	2	0	
TaskLookupController	1	2	1 a mais	Erro modelo alvo
EmpVO	8	8	uma herança errada	Expressado errado em CollabRDL;
GridEmpVO	7	7	0	
DeptVO	6	6	0	
TaskVO	5	5	0	
Total	**87**	**92**	**24 elementos a mais e 13 a menos. E um errado.**	

As classes *ClientApplication*, *DemoClientFacade*, *DeptLookupController*, *GridEmpVO*, *DeptVO* e *TaskVO* foram criadas assim como foram especificadas no modelo alvo. As classes de controle *EmpGridFrameController*, *DeptFrameController* e *TaskGridFrameController* são geradas por iterações do bloco de atividades do comando DOPARALLEL e, por isso, apresentam o mesmo número de elementos a mais. O Código 5.13 mostra o trecho em CollabRDL que produziu esses elementos, isso comprova que o usuário respondeu NÃO à pergunta "Drag event is enabled, do you want set to false?" para todas as essas classes de controle. Esse equívoco aconteceu porque o usuário só consultou o modelo alvo, Figura 5.11, e não olhou para o programa CollabRDL. Isto nos leva a marcar como causa os problemas de percepção no ambiente de execução, *awareness*.

Código 5.13: Trecho de código que gerou marcações a mais

```
IF ("Drag event is enabled, do you want set to false?") THEN {
      dragEnabledMethod = METHOD_EXTENSION(org.openswing.swing.
                              table.client.GridController,
                              frameControllerClass,dragEnabled);
      ADD_CODE(frameControllerClass,dragEnabledMethod,
                                                  "return false;");
} ELSE {
      m = METHOD_EXTENSION(org.openswing.swing.
                              table.client.GridController,
                              frameControllerClass,dragEnter);
      m = METHOD_EXTENSION(org.openswing.swing.
                              table.client.GridController,
                              frameControllerClass,dragExit);
      m = METHOD_EXTENSION(org.openswing.swing.
                              table.client.GridController,
                              frameControllerClass,dragOver);
      m = METHOD_EXTENSION(org.openswing.swing.
                        table.client.GridController,
                        frameControllerClass,dropActionChanged);
      m = METHOD_EXTENSION(org.openswing.swing.
                        table.client.GridController,
                        frameControllerClass,dragDropEnd);
      m = METHOD_EXTENSION(org.openswing.swing.
                        table.client.GridController,
                        frameControllerClass,dropEnter);
      m = METHOD_EXTENSION(org.openswing.swing.
                        table.client.GridController,
                        frameControllerClass,dropExit);
      m = METHOD_EXTENSION(org.openswing.swing.
                        table.client.GridController,
                        frameControllerClass,dropOver);
}
```

A falta do método *getBackgroundColor* na classe *EmpGridFrameController* é porque a atividade para criá-lo não foi descrita no programa CollabRDL. E a ausência da classe *EmpDetailFrameController* não foi devido a uma resposta negativa à pergunta "Create Detail Frame?", isso porque a classe EmpDetailFrame foi criada e faz parte do bloco desta condição.

As classes com término em Frame são criadas pelo grupo *designer* e são expressas em CollabRDL como um atividade externa ao ambiente de execução, por isso os seu métodos não aparecem no modelo alvo. Esse tipo de atividade é muito bem realizado com editores visuais para construção de janelas, comuns em ferramentas IDE como Eclipse e Netbeans.

A classe *TaskLookupController* apresenta um elemento a mais em relação ao modelo alvo, ela apresenta a herança de *LookupController* no modelo gerado, essa atividade foi descrita em CollabRDL e está correta, o modelo alvo foi quem omitiu esta herança. E a classe *EmpVO* herda de *GridEmpVO* mas em CollabRDL foi expressa uma herança para *ValueObjectImpl.*

Por fim, cabe uma discussão a respeito das causas que geraram as diferenças entre o modelo alvo e o gerado. O erro de programação em CollabRDL, ou seja, o programador CollabRDL expressou errado ou não expressou corretamente uma atividade de reutilização, foi uma causa responsável por 2 diferenças. Sobre a causa com relação às atividades externas, o modelo alvo identifica os elementos necessários para construir as janelas da aplicação, no entanto, o programa CollabRDL expressou este tipo de atividade para ser feita utilizando um programa de programação visual que gera as janelas, e o ambiente de execução não constrói no modelo gerado o resultado dessa atividade, foram 12 diferenças desse tipo. Consideramos o erro de especificação no modelo alvo como um erro de projeto da aplicação, responsável por uma diferença. Nove diferenças possuem causa desconhecida. E 24 diferenças têm causa a percepção do usuário com relação do ambiente com relação às atividades de reutilização, *awareness.* O número elevado com causa em *awareness* se justifica porque o ambiente de execução é baseado em uma ferramenta direcionada a processos de negócios e com limitações de implementação para ambiente multiusuário.

5.4 - Considerações Finais

O objetivo de realizar atividades de reutilização em equipe e quando possível em paralelo foi alcançada, o Anexo IV exibe o modelo, em formato .uml, produzido pela execução desta prova de conceito. Para responder a questão 1, “O comando ROLE consegue oferecer um bloco de atividades para um determinado grupo?”, esta prova de conceito fez uso de quatro comandos ROLEs com 6 atividades interativas e 53 respostas a comandos IF executados pelos grupos que foram declarados nos comandos ROLEs (analyst, designer e databaseAdministrator). A resposta para a questão 2, “O comando PARALLEL consegue criar fluxos de execução de blocos de atividades em paralelo?”, é sim porque um comando PARALLEL, declarado com duas atividades interativas, dois fluxos em paralelo, contendo 3 comandos IF e 3 comandos LOOP, conseguiu criar dois fluxos de atividades em paralelo. E a resposta para a questão 3, “O comando DOPARALLEL consegue criar fluxos de execução de um mesmo bloco de atividades?”, também é sim, pois um comando PARALLEL, declarado com 5 atividades interativas e 53 comandos IF, executou por mais de uma vez o seu bloco de atividades.

A prova de conceito foi realizada somente com duas pessoas, sendo que uma foi também o autor deste trabalho, e a segunda teve pouco tempo de ambientação no ambiente de execução da CollabRDL. Em função disso, não registramos em um survey as opiniões dos participantes. Quanto ao conhecimento do framework Openswing, um já o utilizou em um desenvolvimento software e o outro não. Um fato importante é que ambos os participantes possuem uma boa experiência no desenvolvimento de software. Cabe ressaltar que os problemas relacionados à percepção do ambiente ao tratamento de regiões críticas ficariam mais evidentes caso a prova de conceito fosse realizada com mais pessoas, e outros, como comunicação entre as pessoas, poderiam aparecer.

Embora, funcionalmente, o comando DOPARALLEL permita a quem iniciou o processo delegar o bloco de atividades do bloco ROLE ao grupo *analyst,* responder sim à pergunta de teste "Create another function?", e novamente criar um novo método através do primeiro comando do bloco DOPARALLEL sem esperar a execução do primeiro bloco ROLE, assim não foi feito porque iria gerar erros devido ao acesso simultâneo a uma região crítica. Este problema é uma uma limitação deste trabalho.

Esse problema acontece porque a referência à classe de controle *frameControllerClass* é utilizada no bloco de ROLE oferecidos para o grupo *analyst*, se a segunda iteração iniciar, irá alterar esta variável, prejudicando portanto a realização das atividades da primeira iteração. Por isso, quem iniciou o processo, criou um método na classe fachada, esperou a realização de todas as demais atividades do bloco DOPARALLEL, e somente após respondeu sim à pegunta "Create another function?", evitando assim acesso simultâneo à região crítica.

Outra limitação é percebida com relação a identificação de quais são as atividades de cada iteração. Por exemplo, o comando “IF ("Do you want redefine doubleClick method?") THEN” irá aparecer três vezes na fila de atividades para o grupo *analyst*, uma para cada iteração. A ferramenta Activiti-Explorer que nos oferece essas funcionalidades de execução cria uma fila única para cada grupo, não adequada para o comando DOPARALLEL.

Finalmente, o ambiente de execução da CollabRDL faz chamadas às funções de reutilização do núcleo da linguagem RDL, que foi concebido para reutilização monousuária. Além disso, este núcleo não permite que o processo de reutilização possa ser salvo em uma sessão de trabalho e carregado em outra.

CAPÍTULO 6 - TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos identificados na literatura em Reutilização de Software, eles estão organizados em Instanciação de Framework, Linha de Produto de Software (LPS), Cadeias de Transformação (MDD) e Staged Configuration. E finaliza com as considerações finais.

6.1 - Instanciação de Framework

Inicialmente, o processo de utilização de um framework Model-View-Controller foi descrito por meio de CookBook utilizando linguagem natural documentando os passos para a sua instanciação (KRASNER e POPE, 1988). *Hook* foi definido por FROEHLICH *et al.* (1997) e pode ser visto como uma evolução de CookBook já que possui uma descrição estruturada tais como nome que identifica o *hook*, tipo, requisito, área etc, embora ainda mantenha a linguagem natural para a descrição.

RDL (Reuse Description Language) é uma linguagem para descrever o processo de reutilização, o faz de forma sistemática e para instanciar frameworks orientados a objetos, porém não de forma colaborativa (OLIVEIRA *et al.*, 2007), ela foi vista em detalhes no Capítulo 3 desta dissertação. CECHTICKY *et al.* (2003) descrevem um ambiente para instanciar frameworks que utiliza uma linguagem de transformação para gerar código a partir da especificação formal dos componentes do framework, fazem uso de processo mas não tratam a questão do trabalho em equipe.

A abordagem de CECHTICKY *et al.* (2003) possui um ambiente de composição de componentes que tem como entrada *Visual Proxy Beans* e a *especificação da aplicação.* Os *Beans,* implementados como JavaBeans*,* são gerados a partir dos componentes do framework descritos em XML, para isso usa-se a linguagem de transformação XSLT. A saída é uma descrição formal de configuração da aplicação em XML que aplicando programa XSLT produz o código da aplicação.

6.2 - Linha de Produto de Software (LPS)

A prática de reutilização sugere uma solução colaborativa, um exemplo é a abordagem de MENDONÇA *et al.* (2007) para permitir que usuários possam de forma colaborativa configurar um novo produto por meio de um processo dentro de uma Linha de Produto de Software. MENDONÇA *et al.* (2008) adicionaram melhorias a este processo, denominado Collaborative Product Configuration, com a apresentação detalhada de algoritmos. Uma ferramenta baseada nesta abordagem está sendo desenvolvida, a S.P.L.O.T. - *Software Product Lines Online Tools*, que pode ser baixada gratuitamente, inclusive o código fonte (SPLOT, 2012).

Collaborative Product Configuration (CPC) tem como entrada um diagrama de características (KANG *et al.*, 1990). Este é rotulado em regiões de configuração associadas a grupos levando em consideração o conhecimento na área de domínio (administrador de banco de dados, especialista em segurança etc) e autoridade de decisão sobre o projeto (gerentes, supervisores etc). Em seguida é convertido para BPMN, e deste para BPEL com o objetivo de executar, ou seja, nesse momento se tem o plano de execução de um CPC. A execução pode produzir conflitos de decisão, por exemplo, um participante pode selecionar uma característica que depende de outra que não foi selecionada em um área de responsabilidade de outros participantes, para resolvê-los foram criados algorítimos para dar suporte à validação do plano de execução (MENDONÇA *et al.,* 2008).

CPC permite associar tarefas a grupos de pessoas, realizar tarefas em paralelo sincronizando-as no decorrer do processo de configuração do novo produto, e produz um plano executável. Contudo, esta abordagem, não descreve o processo de escolha das características do novo produto, deixando para o grupo responsável por uma região de configuração essa responsabilidade. Para uma região de configuração com muitas regras de reutilização, isso pode ser um problema.

RABISER *et al.* (2007) apresentam um abordagem para apoiar a configuração de um novo produto por ampliação e adaptação do modelo de variabilidades (características), no sentido de eliminar ou acrescentar características ao modelo de acordo com as informações de um projeto específico, ou seja, do produto que se

pretende criar. Eles também apresentam um metamodelo com novos elementos que não fazem parte do modelo de variabilidades, mas são importantes na configuração do produto. Basicamente, o metamodelo expressa que um produto possui uma ou mais propriedades, sendo que cada propriedade está relacionada com a decisão de adicionar ou não um artefato do modelo de variabilidades no produto. Um role (papel) está relacionado com uma ou mais decisões, e finalmente, o guia, que pode estar relacionado com uma ou mais decisões, para auxiliar com dicas e recomendações aos envolvidos na configuração do novo produto. No entanto, esta abordagem não torna explícito as atividades em paralelo.

NOOR *et al.* (2007) apresentam uma abordagem colaborativa para expressar um processo no escopo de Linha de Produto utilizando *thinklets*, mas não explora a possibilidade de atividades em paralelo, mesma limitação da abordagem de RABISER *et al.* (2007). *Thinklet* é definido como a menor unidade de capital intelectual necessário para criar um repetitivo, previsível padrão de colaboração entre as pessoas que trabalham para alcançar um objetivo (BRIGGS et al. 2003). Eles são utilizados para descrever a engenharia de colaboração que é o desenvolvimento de repetidos processos colaborativos conduzidos pelos próprios praticantes.

A abordagem de NOOR *et al.* (2007) é organizada em três camadas: a camada de processo; a camada de padrões; e a camada de *thinklets*. Na camada de processo são definidos os objetivos e as tarefas relevantes para alcançá-los, além da identificação e seleção dos participantes. A camada de padrões faz uso dos padrões de engenharia de colaboração discutidos em (BRIGGS et al., 2003): Divergente (por exemplo, *Brainstorming*), Convergente (juntando ou eliminando conceitos), Organização, Avaliação, e Consenso. As tarefas são mapeadas para estes padrões com o objetivo de sua realização. E por último, na camada de *thinklets*, estão os *thinklets* que podem ser executados. A abordagem colaborativa de NOOR *et al.* (2007) expressa as tarefas e sua ordem para apoiar o processo de criação de um produto em grupo, mas não explora a possibilidade de atividades em paralelo.

HUBAUX *et al.* (2009) definiram *Feature Configuration Workflows* (FCW) que usa o conceito de workflow para descrever o processo de configuração de um novo produto a partir de um modelo de características, permitindo a distribuição de atividades

entre pessoas sendo possível realizá-las em paralelo. Em (ABBASI *et al.,* 2011) é apresentada uma ferramenta para FCW.

FCW permite associar atividades de um worflow a parte de modelos de características e delegá-las a pessoas de um grupo. Dessa forma, ao executar o workflow, pessoas aprovam ou desaprovam características para o novo produto e o modelo é sincronizado, respeitando as permissões de alteração dos grupos. Além disso, o workflow pode ser construído com o objetivo de analisar decisões conflitantes entre grupos sobre as características.

6.3 - Cadeias de Transformação

MDA, Arquitetura Guiada por Modelo, especificada pela OMG, tornou-se referência em MDD. Essa arquitetura é composta por modelos em três níveis diferentes de abstração, com mapeamento do nível mais abstrato para o mais específico até a geração de código. CIM (*Computation Independent Models*) são modelos que pertencem ao nível mais abstrato e por isso mais próximo do entendimento humano, isto é, sem a presença de tecnologia. PIM (*Platform Independent Models*) são modelos para especificar a estrutura e a funcionalidade do sistema sem referenciar detalhes técnicos, adicionando mais informações aos modelos do CIM sem depender de plataforma e, de certa forma, descrevendo os componentes e suas interfaces que deverão ser descritos em uma camada mais específica, a PSM (*Platform Dependent Models*). É na PSM que são construídos os modelos para uma plataforma especifica (por exemplo, CORBA, JAVA, .Net), com base na transformação dos modelos PIMs.

Assim podemos descrever a arquitetura MDA, no topo temos os modelos CIMs. Estes após sofrerem transformações do tipo T1 e a adição de novas especificações formam os modelos PIMs, que por sua fez podem sofrer transformações do tipo T2 até TN para uma plataforma específica, por exemplo, T2 pode significar transformações para a plataforma JEE. Por fim, a geração de código é feita a partir de modelos específicos de uma plataforma. Assim, de forma resumida, podemos ver esta arquitetura por camadas, onde CIM gera PIM e PIM pode gerar uma ou mais PSMs.

Transformações são baseadas nos mapeamentos entre os diferentes níveis de abstração de modelos. Um mapeamento é definido como um conjunto de regras e

técnicas utilizadas para modificar um modelo com o objetivo de criar outro. Transformação PIM para PIM é realizada para melhorar, filtrar ou especializar os modelos sem a necessidade de embutir componentes que dependam de uma plataforma específica. A transformação PIM para PSM acontece quando os modelos do PIM estão refinados e podem ser projetados para uma plataforma de execução. Essa transformação é baseada nas características da plataforma destino, e por isso dependente dela, fazendo uso de seus conceitos e componentes. Já a transformação do tipo PSM para PSM é para a realização de implantação de componente. Um exemplo é a configuração e implantação de um componente da uma plataforma para atender as especificações do sistema. E a transformação PSM para PIM, que é gerar modelos abstratos a partir de uma plataforma específica, embora seja de difícil automação, pode ser suportada por ferramentas. Embora a abordagem MDA tenha uma característica de trabalho colaborativo, principalmente nas transformações, essa abordagem não explora essa questão.

Desenvolvimento Guiado por Restrições (DGR, ou CDD, do inglês Constraint-Driven development) é uma abordagem MDD em UML (LANO, 2008). Ela define em sua camada mais abstrata (PIM), a utilização de casos de uso, diagramas de classe, máquinas de estado, adicionado restrições para descrever as funcionalidades do sistema. A partir dessas descrições, modelos específicos de plataforma (PSMs) são gerados através de processos sistemáticos com a ajuda de ferramentas automatizadas. E finalmente o código é gerado a partir dos modelos do nível PSM, sendo portanto similar à abordagem MDA.

UML-RSDS (*Reactive System Design Support*) é uma extensão da UML para dar suporte ao DGR. Ela redefine diagrama de classes e digramas de estado, fazendo uso de uma OCL simplificada para expressar as restrições. CDD, assim como MDA, não aborda as questões de colaboração. Um exemplo de desenvolvimento que utiliza esta abordagem pode ser descrita em cinco passos:

1 - Construção dos casos de uso, diagramas de classe e diagramas de estado para o nível PIM utilizando uma ferramenta UML-RSDS;

2 Análise de consistência e completude dos modelos do PIM;

3.- Transformação dos modelos para realizar a análise semântica;

4.- Transformação de modelos para melhorar a qualidade ou para se adequar a uma plataforma (PSM). Isto implica em transformar modelos do PIM para um PSM, neste caso Java;

5.- Gerar código Java a partir da especificação PSM Java;

Na literatura existem outras abordagens: MAIA *et al.* (2007) apresentam Oddyssey-MDA, que permite transformar modelos PIMs para modelo PSMs utilizando Meta-Object Facility (MOF, 2002), XML Metadata Interchange (XMI, 2002), e Java Metadata Interface (JMI, 2002); ATLAS Transformation Language (ATL) é uma linguagem de transformação de modelos que permite definir transformações de forma declarativa e imperativa (JOUAULT e KURTEV, 2006); e em (DI RUSCIO *et al.,* 2012) é proposta uma introdução à classificação das abordagens de transformação e linguagens.

6.4 - Staged Configuration

CZARNECKI *et al.* (2004) criam uma nova configuração a partir de sucessivos passos de especialização do modelo de características (KANG et al., 1990), gerado em cada passo um modelo mais próximo do produto final. Essa técnica é conhecida como *staged configuration*, mas não abordam colaboração nesse trabalho. No entanto, mais tarde (CZARNECKI et al., 2005) é proposto o conceito Multi-level Staged Configuration (MLSC), onde mencionam a participação de pessoas com diferentes papéis na configuração, mesmo que ainda sequencialmente.

6.5 - Considerações Finais

Este capítulo apresentou alguns trabalhos relacionados nas áreas de Instanciação de Framework, Linha de Produtos de Software, Cadeias de Transformação e Staged Configuration. A Tabela 6.1 apresenta um comparativo dos trabalhos relacionados. A área de Instanciação de Framework, itens 1,2 e 3 da Tabela, iniciou-se com (KRASNER e POPE, 1988) utilizando uma linguagem natural e não tratou a instanciação de forma colaborativa, já OLIVEIRA *et al* (2007) apresentaram uma nova linguagem para

descrever reutilização de forma imperativa, mas não trataram colaboração. Em Linha de Produtos de Software, itens 4-7, MENDONÇA *et al. (*2008) apresentaram algoritmos para apoiar o trabalho em grupo, mas não descreveram o processo de escolha das características do novo produto. RABISER *et al.* (2007) atuaram na ampliação e adaptação do modelo de características, mas não tornaram explícito as atividades em paralelo, já NOOR *et al.* (2007) fizeram uso dos *thinklets,* mas não exploraram atividades em paralelo. HUBAUX *et al.* (2009), finalizando a área de LPS, fizeram uso de workflow para descrever o processo de configuração de um novo produto de forma colaborativa. Na área de Cadeias de Transformação, item 8 da Tabela, LANO (2008) apresentou uma extensão da UML para dar suporte ao Desenvolvimento Guiado por Restrições, usando OCL, mas não abordou as questões de colaboração. E, finalmente, CZARNECKI *et al.* (2005), item 9 da Tabela, na área de *Staged Configuration,* em *Multi-level Staged Configuration*, mencionaram a participação de pessoas com diferentes papéis, mas ainda de forma sequencial.

Tabela 6.1: Comparativo dos Trabalhos Relacionados

1 -	KRASNER e POPE, 1988	Instanciação de Framework	CookBook utilizando linguagem natural	não tratam colaboração
2 -	OLIVEIRA *et al.*, 2007	Instanciação de Framework	Linguagem de forma imperativa	não fazem de forma colaborativa
3 -	CECHTICKY *et al*., 2003	Instanciação de Framework	Linguagem de transformação	não fazem de forma colaborativa
4 -	MENDONÇA *et al.*,2008	Linha de Produtos de Software	apresentação detalhada de algoritmos	não descrevem o processo de escolha das características
5 -	RABISER *et al.*, 2007	Linha de Produtos de Software	ampliação e adaptação do modelo de características	Não tornam explícito as atividades em paralelo
6 -	NOOR *et al.* , 2007	Linha de Produtos de Software	utilizando *thinklets*	não exploram atividades em paralelo

7 -	HUBAUX *et al.* , 2009	Linha de Produtos de Software	Workflow para descrever o processo de configuração de um novo produto	colaborativa
8 -	LANO, 2008	Cadeias de Transformação	extensão da UML para dar suporte ao Desenvolvimento Guiado por Restrições. Usa OCL.	Não aborda as questões de colaboração
9 -	CZARNECKI *et al.*, 2005	Staged Configuration	Multi-level Staged Configuration , onde mencionam a participação de pessoas com diferentes papéis	Mas ainda de forma sequencial.

CollabRDL estende RDL com os conceitos de colaboração, de forma imperativa e seguindo a estrutura da RDL. Além disso, assim como (HUBAUX *et al.,*2009), faz uso de ambientes de workflow. Em relação aos outros trabalhos, uns não suportam a realização de atividades em paralelo, enquanto outros não tratam a questão de colaboração. Por isso, concluímos que CollabRDL agrega valor nesta questão.

CAPÍTULO 7 - CONCLUSÃO

Esta conclusão lista as contribuições e as limitações deste trabalho, os trabalhos futuros e finaliza esta dissertação com as considerações finais.

Esta dissertação teve como objetivo estender RDL com conceitos de colaboração. Denominamos de CollabRDL esta extensão que determinou os seguintes objetivos específicos:

1. Permitir a definição de grupos de reutilizadores;
2. Delegar atividades de reutilização a grupo de pessoas;
3. Expressar atividades de reutilização em paralelo;
4. Definir um ambiente executável com base em BPMN (2011);
5. Avaliar através de alguns padrões de workflow (VAN DER AALST *et al.*, 2003, , RUSSELL *et al.* 2006) e de uma prova de conceito.

O primeiro objetivo especifico foi alcançado utilizando o ambiente de execução Activiti. O segundo foi através dos comandos ROLE e FLOW em execução no ambiente de execução CollabRDL-Activiti-Explorer. Já o terceiro é alcançado utilizando os comandos PARALLEL e DOPARALLEL. O quarto foi atingido através da configuração do ambiente de execução do Activiti, denominado por CollabRDL-Activiti-Explorer. E finalmente, a avaliação, quinto e último objetivo específico, descreveu em CollabRDL alguns padrões de wokflow e realizou uma prova de conceito, seções 6.2 e 6.3, respectivamente.

7.1 - Contribuições

Os comandos ROLE, PARALLEL e DOPARALLEL são contribuições deste trabalho de mestrado, que embora tenham sido avaliados com atividades de instanciação de frameworks orientados a objetos, são genéricos e podem ser utilizados para outras atividades de reutilização, pois são comandos de workflow. Além disso, foi criado um conversor CollabRDL para BPMN-Activiti que pode servir como uma base de mapeamento para outras implementações BPMN, e também foi configurado um ambiente de execução baseado na ferramenta Activiti (ACTIVIT, 2012). Com essas

contribuições, programas escritos em RDL para a instanciação de frameworks orientados a objetos para serem instanciados por uma pessoa podem ser alterados, adicionando os comandos acima citados, com o objetivo de suportar a instanciação de forma colaborativa.

Além disso, o estudo sobre como as linguagens representam colaboração gerou o artigo (LUCAS *et al.,* 2013) que mostra como as linguagens, independente do seu domínio de atuação e paradigma associado à sua estrutura, oferecem suporte aos conceitos de *papéis*, *paralelismo* e *atividades* quando têm o objetivo de representar colaboração.

7.2 - Limitações

O comando DOPARALLEL permite distribuir um bloco de atividades a grupo de pessoas, por isso pode gerar regiões críticas de acesso. O primeiro programa em CollabRDL, escrito para a prova de conceito deste trabalho, já apresenta esse problema. Assim, é preciso que o programador CollabRDL atente para este problema para evitá-lo sempre que possível.

O comando ROLE atribui um bloco de atividades para um grupo de pessoas, não o faz para vários grupos, e também não permite em tempo de execução selecionar uma pessoa para atribuir atividades. Além disso, foi identificada a necessidade de atribuir todas as atividades de um bloco ROLE a mesma pessoa que pegou a primeira atividade.

O comando DOPARALLEL é para expressar em paralelo blocos de atividades que podem ser executados em paralelo, por isso sem a presença de regiões críticas. No entanto, quando duas ou mais marcações FLOWs oferecem atividades para o mesmo grupo de pessoas, o ambiente de execução apresentou-se confuso, as atividades dos FLOWs apareceram na lista do grupo em questão, gerando problemas de percepção das atividades pelos seus integrantes.

O ambiente de execução CollabRDL-Activiti-Explorer configurado a partir da ferramenta Activiti (ACTIVIT, 2012) apresenta limitações com relação a percepção do usuário na reutilização de software colaborativo. Por exemplo, não há como visualizar os requisitos da nova aplicação, indicando as responsabilidades dos grupos, e nem o modelo da aplicação em construção, indicando que grupo gerou o quê.

7.3 - Trabalhos Futuros

A CollabRDL bem como o seu ambiente de execução precisam melhorar o contexto de 1- Percepção do ambiente na reutilização de software colaborativa; e 2– Tratamento de Regiões Críticas. Estes dois tópicos são necessários para a realização de estudos mais elaborados para explorar todas as funcionalidades dos novos comandos com a participação de mais pessoas. O primeiro item, *awareness*, pode ser resolvido através de técnicas de visualização (BOTTERWECK *et. al.*, 2008), já o segundo item está relacionado com às linguagens concorrentes (SCOTT, 2009).

7.4 - Considerações Finais

Este trabalho adicionou à RDL três novos comandos e configurou um ambiente executável com base em BPMN. No entanto, encontramos problemas relacionados com a percepção do usuário no ambiente de reutilização e com problemas relacionados às regiões críticas, tema tratado nas linguagens concorrentes. Nesse sentido, é preciso investir esforços para solucionar esses problemas, para logo após, investir tempo em experimentos controlados.

REFERÊNCIAS BIBLIOGRÁFICAS

ABBASI, E.K.; HUBAUX, A.; HEYMANS, P.; 2011, "A Toolset for Feature-Based Configuration Workflows," *Software Product Line Conference* (SPLC), 2011 15th International , vol., no., pp.65-69, 22-26.

ACTIVIT, http://www.activiti.org/, Acessado em 29/10/2012.

ALI, F.M. & DU, W. 2004, "Toward reuse of object-oriented software design models", *Information and Software Technology*, vol. 46, no. 8, pp. 499-517.

ARANGO, G. E PRIETO-DIAZ, R., 1991. "Introduction and overview: Domain analysis concepts and research directions". In: *Domain Analysis and Software Systems Modeling*. IEEE Press.

AVALCOMWF, C*ommercial Product Evaluation, http://www.workflowpatterns.com/evaluations/commercial/index.php,* visitado em 10 de abril 2013

AVALOSWF, Open Source Product Evaluation, http://www.workflowpatterns.com/evaluations/opensource/index.php visitado em 10 de abril 2013

AVALWF, Avaliação de produtos de workflow, http://www.workflowpatterns.com/vendors/index.php, visitado em 10 de abril 2013

BACHMANN, F., CLEMENTS, P. C, 2005, "Variability in Software Product Lines". In: *ECHNICAL REPORT CMU/SEI-2005-TR-012 ESC-TR-2005-012.*

BANNON, L., AND SCHMIDT, K., "CSCW: Four Characters in Search of a Context," in *Proceedings of the European Conference on Computer Supported*

Cooperative Work, pp. 358--372, Gatwick UK,setembro de 1989.

BARTHELMESS, P. & ANDERSON, K.M. 2002, "A view of software development environments based on activity theory", *Computer Supported Cooperative Work: CSCW: An International Journal*, vol. 11, no. 1-2, pp. 13-37.

BECK, K. 2007. *Implementation Patterns*. Addison-Wesley.

BELLAS, F., 2004, "Standards for second-generation portals", *IEEE Internet Computing*, vol. 8, no. 2, pp. 54-60.

BOTTERWECK, G., THIEL, S., NESTOR, D., ABID, S.B. & CAWLEY, C. 2008, "Visual tool support for configuring and understanding software product lines", *Proceedings - 12th International Software Product Line Conference, SPLC 2008*, pp. 77.

BPMN , http://www.bpmn.org/, Acessado em 29/01/2013.

BPMN , Object Management Group. Business Process Modeling and Notation, Version 2.0, formal/2011-01-03 January 2011.

BRIGGS, R.O., DE VREEDE, G. & NUNAMAKER JR., J.F., 2003, "Collaboration engineering with thinklets to pursue sustained success with group support systems", *Journal of Management Information Systems*, vol. 19, no. 4, pp. 31-64.

BULL CORPORATION, FlowPath Functional Specification. Bull S. A., Paris, France, September 1992.

CECHTICKY, V., CHEVALLEY, P., PASETTI, A. & SCHAUFELBERGER, W. 2003, “A generative approach to framework instantiation”, Lecture Notes in *Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2830 , pp. 267-286.

CHASTEK, G. & MCGREGOR, J. D., 2002, *Guidelines for Developing a Product Line Production Plan*, In: TECHNICAL REPORT CMU/SEI-2002-TR-006 ESC-TR-2002-006.

CLARK, H. H. AND BRENNAN, “Grounding in Communication”, In Perspectives on Socially Shared Cognition (1991), pp. 127-149.

CLEMENTS, P.C., JONES, L.G., NORTHROP, L.M. & MCGREGOR, J.D. 2005, "Project management in a software product line organization", *IEEE Software*, vol. 22, no. 5, pp. 54-62.

CORTES, M. & MISHRA, P. 1996, "DCWPL: A programming language for describing collaborative work", *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 21-29, novembro de 1996, BoNston, Massachusetts, United States.

CZARNECKI, K. ,HELSEN, S. e EISENECKER, U. W. , 2005. “Staged configuration through specialization and multi-level configuration of feature models”. *Software Process: Improvement and Practice*, 10(2):143–169.

CZARNECKI, K., HELSON, S., EISENECKER, U., "Staged configuration using feature models," in *Proc. of the 3rd International Software Product Line Conference (SPLC 2004)*, Boston, MA, USA, 2004, pp. 266--283.

DE ALMEIDA, E.S., ALVARO, A., LUCRÉDIO, D., GARCIA, V.C. & DE LEMOS MEIRA, S.R. 2005, "A survey on software reuse processes", *Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on. , vol., no., pp.66,71, 15-17 Aug. 2005*.

DE PAOLI, F. & TISATO, F. 1994, "CSDL: A language for cooperative systems design", *IEEE Transactions on Software Engineering*, vol. 20, no. 8, pp. 606-616.

DE VRIES, K., OMMERT, O.: *Advanced workflow patterns in practice (1)*: experiences based on pension processing (in Dutch). Bus. Process Mag. 7(6), 15–18 (2001)

DI RUSCIO, D., ERAMO, R. & PIERANTONIO, A, 2012, “Model transformations*”*. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7320 LNCS, pp. 91-136, Bertinoro Itária.

DOT NET, http://www.microsoft.com/net, Acesso em agosto 2012.

EASTERBROOK, S. M., SINGER, J., STOREY, M-A., AND DAMIAN, D., *Selecting Empirical Methods for Software Engineering Research*. In F. Shull, J. Singer and D. Sjøberg(eds), Guide to Advanced Empirical Software Engineering, Springer, 2007.

Eclipse documentation, http://help.eclipse.org/indigo/index.jsp, Acesso em junho 2012.

ELLIS, C. A., GIBBS, S.J., & REIN, G. L. (1991). “Group-ware: Some issues and experiences”. *Communications of the ACM*, 34(1), 38-58.

FILHO, I.M., DE OLIVEIRA, T.C. & DE LUCENA, C.J.P. 2004, "A framework instantiation approach based on the Features Model", *Journal of Systems and Software*, vol. 73, no. 2, pp. 333-349.

FRAKES, W.B. & SUCCI, G. 2001, "An industrial study of reuse, quality, and productivity", *Journal of Systems and Software*, vol. 57, no. 2, pp. 99-106.

FRAKES, W.B. e KANG, K. 2005, "Software reuse research: Status and future", *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 529-536.

FROEHLICH, G., HOOVER, H.J., LIU L. AND SORENSON, P.G., “Hooking into Object-Oriented Application Frameworks”, *Proc. 19th Int'l Conf. on Software*

Engineering, Boston, May 1997, 491-501.

FUGGETTA, A., “Software process: a roadmap”, In *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*. ACM, New York, NY, USA, 25-34, .

FUKS, H., RAPOSO, A., GEROSA, M.A., PIMENTEL, M., LUCENA, C.J.P.: “The 3C Collaboration Model”. In: The Encyclopedia of E-Collaboration, Ned Kock (org), pp. 637-644 (2007).

GLASS, R.L., VESSEY, I., RAMESH, V., 2002, "Research in software engineering: An analysis of the literature", *Information and Software Technology*, vol. 44, no. 8, pp. 491-506.

HAYES, J. H.; OFFUTT, J., 2006. Input validation analysis and testing. *Empirical Softw. Eng. 11*, 4 (December 2006), 493-522.

HEPPER, S., 2008, Java Portlet Specification Version 2.0 ,jsr-286.

HUBAUX, A., CLASSEN, A. e HEYMANS, P., 2009. “Formal modelling of feature configuration workflows”. In *Proceedings of the 13th International Software Product Line Conference (SPLC '09)*. Carnegie Mellon University, Pittsburgh, PA, USA, 221-230.

JAVA EE, Java Platform, Enterprise Edition (Java EE), http://www.oracle.com/technetwork/java/javaee/overview/index.html, Acesso em agosto 2012.

JMI, 2002, Java Metadata Interface Specification, JSR 040 Java Community Process http://www.jcp.org/.

JOHNSON, R.E. & FOOTE, B. 1988, "Designing reusable classes", *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22-30, 35.

JOUAULT, F. & KURTEV, I. 2006, “Transforming models with ATL”. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3844 LNCS, pp. 128-138.*

KANG, K., COHEN, S., HESS, J., NOVAK, W., PETERSON, A., 1990. *Feature-oriented domain analysis (FODA) feasibility study*, SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21.

KITCHENHAM, B.A.; DYBÅ, T.; JØRGENSEN, M.; “Evidence-based Software Engineering”; In *Proc. of 26th Intern. Conf. on Software Engineering (ICSE’04)*; May 2004; Edinburgh, Scotland, United Kingdom, 2004, pp. 273-281.

KOLLOCK, P., 1998, “Social dilemmas: The Anatomy of Cooperation”. Annual Review of Sociology, n. 24, p. 183-214, 1998.

KRASNER, G.E., POPE, S.T., A, “Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80”, *Journal of Object-Oriented Programming* 1(3), 1988.

LANO, K. 2008, “Constraint-driven development”, *Information and Software Technology*, vol. 50, no. 5, pp. 406-423.

LANO, K., *Model-Driven Software Development with UML and Java*, 1.ed., chapter 1, Higth Holborn House, 50-51 Bedford Row, London WC1RLR, Cengage Learning, 2009.

LI, D. & MUNTZ, R.R. 2000, "A collaboration specification language", *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*, vol. 35, no. 1, pp. 149-162.

LIM, W.C , 1994, "Effects of reuse on quality, productivity, and economics", *IEEE*

Software, vol. 11, no. 5, pp. 23-30.

LOUGHRAN, N., SÁNCHEZ, P., GARCIA, A. & FUENTES, L. 2008, *Language support for managing variability in architectural models*. Proceedings of the 7th international conference on Software composition, March 29-30, 2008, Budapest, Hungary.

LUCAS, E.M., SCHNEIDER, D., OLIVEIRA, T.C., MOREIRA DE SOUZA, J., "A survey of languages to represent collaboration as a means of designing CSCW facilities in RDL," *Computer Supported Cooperative Work in Design (CSCWD)*, 2013 IEEE 17th International Conference, Whistler, BC, Canadá.

MAIA, N., BACELO, A.P.T., e WERNER, C.M.L., "Odyssey-MDA: A Transformational Approach to Component Models", *International Conference on Software Engineering and Knowledge Engineering (SEKE'2007)*, Boston, USA, July 2007, pp. 9-14.

MALONE, T.W. & CROWSTON,K., 1990. “What is coordination theory and how can it help design cooperative work systems?”. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work (CSCW '90)*. ACM, New York, NY, USA, 357-370.

MALONE, T.W. & CROWSTON,K., 1994, "Interdisciplinary study of coordination", *ACM Computing Surveys*, vol. 26, no. 1, pp. 87-119.

MANGAN, M. A. S., BORGES, M. R. S., WERNER, C. M. L., "Increasing Awareness in Distributed Software Development Workspaces", In: *International Workshop on Groupware*, pp.84-91, San Carlos, Costa Rica, setembro 2004.

MENDONÇA, M., COWAN, D., MALYK, W. & OLIVEIRA, T. 2008, "Collaborative product configuration: Formalization and efficient algorithms for dependency-

analysis", *Journal of Software*, vol. 3, no. 2, pp. 69-82.

MENDONÇA, M.; COWAN, D; OLIVEIRA, T.; , "A Process-Centric Approach for Coordinating Product Configuration Decisions," *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference* on , vol., no., pp.283a, Jan. 2007.

MOF (2002) Meta Object Facility (MOF) specification, version 1.4, OMG.

MOHAGHEGHI, P. e CONRADI, R., 2007, "Quality, productivity and economic benefits of software reuse: A review of industrial studies", Empirical Software Engineering, vol. 12, no. 5, pp. 471-516.

MOHAMED, F. & SCHMIDT, D.C., 1997. Object-oriented application frameworks. Commun. ACM 40, 10 (October 1997), 32-38.

NOOR, M.A., GRÜNBACHER, P. & BRIGGS, R.O. 2007, "A collaborative approach for product line scoping: A case study in collaboration engineering", *Proceedings of the IASTED International Conference on Software Engineering*, *SE 2007*, pp. 216.

NORTHROP, L.M., 2002, "SEI's software product line tenets", *IEEE Software*, vol. 19, no. 4, pp. 37-40.

OASIS, ebXML Business Process Specification Schema, Technical Specification v2.0.4, December 2006.

OLIVEIRA, T.C., 2001, *Uma Abordagem Sistemática para a Instanciação de Frameworks Orientados a Objetos.*Tese de D.Sc., PUC, Rio de Janeiro. Brasil.

OLIVEIRA, T.C., ALENCAR, P. & COWAN, D. 2011, "ReuseTool - An extensible tool support for object-oriented framework reuse", *Journal of Systems and Software*, vol. 84, no. 12, pp. 2234-2252.

OLIVEIRA, T.C., ALENCAR, P.S.C., DE LUCENA, C.J.P. & COWAN, D.D. 2007,

"RDL: A language for framework instantiation representation", *Journal of Systems and Software*, vol. 80, no. 11, pp. 1902-1929.

OMG, Reusable Asset Specification. In OMG OMG Available Specification, Version 2.2, November 2005.

OPENSWING, 2013, http://oswing.sourceforge.net/

PREE, W. & SIKORA, H. 1997, "Design patterns for object-oriented software development", Proceedings - International Conference on Software Engineering, pp. 663-664, New York, NY, USA.

RABISER, R., GRÜNBACHER, P. & DHUNGANA, D., 2007, "Supporting product derivation by adapting and augmenting variability models", *Proceedings - 11th International Software Product Line Conference, SPLC 2007*, pp. 141, Kyoto, Japan.

ROTHENBERGER, M.A., DOOLEY, K.J., KULKARNI, U.R. & NADA, N., 2003, "Strategies for software reuse: A principal component analysis of reuse practices", *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 825-837.

RUNESON, P. & HÖST, M. 2009, "Guidelines for conducting and reporting case study research in software engineering", *Empirical Software Engineering*, vol. 14, no. 2, pp. 131-164.

RUSSELL, N., TER HOFSTEDE, A.H.M., VAN DER AALST, W.M.P. e. MULYAR, N., *Workflow Control-Flow Patterns: A Revised View*. BPM Center Report BPM-06-22, BPMcenter.org, 2006.

SALVADOR, G., 2009, *Métodos Empíricos para Validação da Reuse Description Language em Instanciação de Frameworks*, Dissertação de M.Sc, PUCRS, Porto Alegre, RS, Brasil.

SCOTT, M.L., *Programming Language Pragmatics*. 3 ed. New York, Morgan

Kaufmann, 2009.

SEGAL, J., GRINYER, A., SHARP, H., 2005, "The type of evidence produced by empirical software engineers", *Proceedings of the 2005 Workshop on Realising Evidence-Based Software Engineering, REBSE '05,* pp. 1-4, New York, NY, USA.

SEI, Software Engineering Institute, http://www.sei.cmu.edu, Acesso em: Acesso em junho 2012.

SGP, Sistema de Gerenciamento de Projetos, contato: lti.iprj.uerj.br, 2013.

SMITH, R.B., HIXON, R. E HORAN, B. 1998, "Supporting flexible roles in a shared space", *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, , pp. 197-206, New York, NY, USA.

SOLIMAN, R., BRAUN, R. & SIMOFF, S., 2005, "The essential ingredients of collaboration", *Proceedings - 2005 International Symposium on Collaborative Technologies and Systems*, pp. 366, Saint Louis, Missouri, USA.

SPCL*, Software Product Line Conferences*, http://www.splc.net, Acesso em junho 2012.

SPLOT - Software Product Lines Online Tools, http://www.splot-research.org, Acesso em maio 2012.

SWING, 2013, http://docs.oracle.com/javase/tutorial/uiswing/start/about.html

TIOBE Programming Community, http://www.tiobe.com, Acesso em junho 2012.

TUTORIAL OPENSWING, 2013, http://oswing.sourceforge.net/tutorial.html

USCHOLD, M., KING, M., MORALEE, S. & ZORGIOS, Y. 1998, "The enterprise

ontology", *Knowledge Engineering Review*, vol. 13, no. 1, pp. 31-89.

VAN DER AALST, W.M.P. & TER HOFSTEDE, A.H.M. 2012, "Workflow patterns put into context", *Software and Systems Modeling*, vol. 11, no. 3, pp. 319-323.

VAN DER AALST, W.M.P., TER HOFSTEDE, A.H.M., KIEPUSZEWSKI, B. e BARROS, A.P. 2003, "Workflow patterns", *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5-51.

WFMC, WORKFLOW MANAGEMENT COALITION, 1999, *Workflow Management Coalition Terminology & Glossary*. Document Number WFMC-TC-1011, Issue 3.0.

WHITE, S. A. , 2004, *Process Modeling Notations and Workflow Patterns,* IBM Corp., United States. Disponível em: http://www.workflowpatterns.com/vendors/documentation/BPMN_wfh.pdf, acessado em 24/10/2012.

WOHED, P., RUSSELL, N., TER HOFSTEDE, A.H.M., ANDERSSON, B. & VAN DER AALST, W.M.P. 2009, "Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark", Information and Software Technology, vol. 51, no. 8, pp. 1187-1216.

XMI (2002) XML Metadata Interchange (XMI) specification, v1.2, OMG.

ANEXO I – Marcação em BPMN-Activiti para o comando PARALLEL (linhas 10-19 do Código 4.9)

```
1 -  <parallelGateway id="sourceOrTargetRef5" name="Gateway"
gatewayDirection="Diverging" />
2 - <sequenceFlow id="flow5to6" sourceRef="sourceOrTargetRef5"
targetRef="sourceOrTargetRef6" />
3 - <sequenceFlow id="flow5to11" sourceRef="sourceOrTargetRef5"
targetRef="sourceOrTargetRef11" />
4 - <serviceTask id="sourceOrTargetRef6" name="BeginThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
5 -  <extensionElements>
6 -      <activiti:field name="currentADDRESS"
stringValue="6" />
7 -  </extensionElements>
8 - </serviceTask>
9 - <sequenceFlow id="flow6to7" sourceRef="sourceOrTargetRef6"
targetRef="sourceOrTargetRef7" />
10 - <serviceTask id="sourceOrTargetRef7" name="Declaration"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.Declaration
Delegate" >
11 -  <extensionElements>
12 -      <activiti:field name="typeNAME" stringValue="VOID" />
13 -      <activiti:field name="varNAME"
stringValue="climaPortlet" />
14 -  </extensionElements>
15 - </serviceTask>
16 - <sequenceFlow id="flow7to8" sourceRef="sourceOrTargetRef7"
targetRef="sourceOrTargetRef8" />
17 - <userTask id="sourceOrTargetRef8" name="ClassExtension" >
18 - <documentation>"Comentário"</documentation>
19 -  <extensionElements>
20 -    <activiti:formProperty required="true" type="string"
name="Type the subsclass name"  id="subNAME" />
21 -    <activiti:taskListener
class="org.prisma.rdl.runtime.bpmn.listener.ClassExtensionListen
er" event="complete" >
22 -      <activiti:field name="subPNAME"
stringValue="pacoteA" />
23 -      <activiti:field name="superNAME"
stringValue="GenericPortlet" />
24 -    </activiti:taskListener>
25 -  </extensionElements>
26 -  <potentialOwner>
27 -    <resourceAssignmentExpression>
28 -
<formalExpression>group(analistaJunior)</formalExpression>
```

```
29 -    </resourceAssignmentExpression>
30 -  </potentialOwner>
31 - </userTask>
32 - <sequenceFlow id="flow8to9" sourceRef="sourceOrTargetRef8"
targetRef="sourceOrTargetRef9" />
33 - <serviceTask id="sourceOrTargetRef9" name="Assignment"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.AssignmentD
elegate" >
34 -  <extensionElements>
35 -      <activiti:field name="varNAME"
stringValue="climaPortlet" />
36 -      <activiti:field name="expr"
stringValue="$0000RDLTempVar" />
37 -  </extensionElements>
38 - </serviceTask>
39 - <sequenceFlow id="flow9to10" sourceRef="sourceOrTargetRef9"
targetRef="sourceOrTargetRef10" />
40 - <serviceTask id="sourceOrTargetRef10" name="EndThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
41 -  <extensionElements>
42 -      <activiti:field name="currentADDRESS" stringValue="10"
/>
43 -  </extensionElements>
44 - </serviceTask>
45 - <sequenceFlow id="flow10to16"
sourceRef="sourceOrTargetRef10"
targetRef="sourceOrTargetRef16" />
46 - <serviceTask id="sourceOrTargetRef11" name="BeginThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
47 -  <extensionElements>
48 -      <activiti:field name="currentADDRESS" stringValue="11"
/>
49 -  </extensionElements>
50 - </serviceTask>
51 - <sequenceFlow id="flow11to12"
sourceRef="sourceOrTargetRef11"
targetRef="sourceOrTargetRef12" />
52 - <serviceTask id="sourceOrTargetRef12" name="Declaration"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.Declaration
Delegate" >
53 -  <extensionElements>
54 -      <activiti:field name="typeNAME" stringValue="VOID" />
55 -      <activiti:field name="varNAME"
stringValue="cambioPortlet" />
56 -  </extensionElements>
57 - </serviceTask>
58 - <sequenceFlow id="flow12to13"
sourceRef="sourceOrTargetRef12"
targetRef="sourceOrTargetRef13" />
```

```
59 - <userTask id="sourceOrTargetRef13" name="ClassExtension" >
60 - <documentation>"Comentário"</documentation>
61 -  <extensionElements>
62 -    <activiti:formProperty required="true" type="string"
name="Type the subsclass name"  id="subNAME" />
63 -    <activiti:taskListener
class="org.prisma.rdl.runtime.bpmn.listener.ClassExtensionListen
er" event="complete" >
64 -      <activiti:field name="subPNAME"
stringValue="pacoteA" />
65 -      <activiti:field name="superNAME"
stringValue="GenericPortlet" />
66 -    </activiti:taskListener>
67 -  </extensionElements>
68 -  <potentialOwner>
69 -    <resourceAssignmentExpression>
70 -
<formalExpression>group(analistaSenior)</formalExpression>
71 -    </resourceAssignmentExpression>
72 -  </potentialOwner>
73 - </userTask>
74 - <sequenceFlow id="flow13to14"
sourceRef="sourceOrTargetRef13"
targetRef="sourceOrTargetRef14" />
75 - <serviceTask id="sourceOrTargetRef14" name="Assignment"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.AssignmentD
elegate" >
76 -  <extensionElements>
77 -      <activiti:field name="varNAME"
stringValue="cambioPortlet" />
78 -      <activiti:field name="expr"
stringValue="$0000RDLTempVar" />
79 -  </extensionElements>
80 - </serviceTask>
81 - <sequenceFlow id="flow14to15"
sourceRef="sourceOrTargetRef14"
targetRef="sourceOrTargetRef15" />
82 - <serviceTask id="sourceOrTargetRef15" name="EndThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
83 -  <extensionElements>
84 -      <activiti:field name="currentADDRESS" stringValue="15"
/>
85 -  </extensionElements>
86 - </serviceTask>
87 - <sequenceFlow id="flow15to16"
sourceRef="sourceOrTargetRef15"
targetRef="sourceOrTargetRef16" />
88 -  <parallelGateway id="sourceOrTargetRef16" name="Gateway"
gatewayDirection="Converging" />
```

```
89 -  <sequenceFlow id="flow16to17"
sourceRef="sourceOrTargetRef16"
targetRef="sourceOrTargetRef17" />
```

ANEXO II – Representação do Código 4.11em XML do BPMN-Activiti

```
1 - <exclusiveGateway id="sourceOrTargetRef433" name="Gateway"
gatewayDirection="Converging" >
2 -  <extensionElements>
3 -    <activiti:executionListener
class="org.prisma.rdl.runtime.bpmn.listener.BeginDoParallelListe
ner" event="start" >
4 -      <activiti:field name="currentADDRESS" stringValue="433"
/>
5 -      <activiti:field name="varCondition"
stringValue="condStatus440" />
6 -    </activiti:executionListener>
7 -  </extensionElements>
8 - </exclusiveGateway>
9 - <sequenceFlow id="flow433to434"
sourceRef="sourceOrTargetRef433"
targetRef="sourceOrTargetRef434" />
10 - <parallelGateway id="sourceOrTargetRef434" name="Gateway"
gatewayDirection="Diverging" />
11 - <sequenceFlow id="flow434to435"
sourceRef="sourceOrTargetRef434"
targetRef="sourceOrTargetRef435" />
12 - <sequenceFlow id="flow434to439"
sourceRef="sourceOrTargetRef434"
targetRef="sourceOrTargetRef439" />
13 - <serviceTask id="sourceOrTargetRef435" name="BeginThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
14 -  <extensionElements>
15 -      <activiti:field name="currentADDRESS"
stringValue="435" />
16 -  </extensionElements>
17 - </serviceTask>
18 - <sequenceFlow id="flow435to436"
sourceRef="sourceOrTargetRef435"
targetRef="sourceOrTargetRef436" />
19 - <userTask id="sourceOrTargetRef436" name="ExternalTask" >
20 - <documentation>"Gere o ícone no formato PNG.""<br/> Criar
ícone 16x16."</documentation>
21 -  <potentialOwner>
22 -    <resourceAssignmentExpression>
23 -      <formalExpression>group(analista)</formalExpression>
24 -    </resourceAssignmentExpression>
25 -  </potentialOwner>
26 - </userTask>
```

```
27 - <sequenceFlow id="flow436to437"
sourceRef="sourceOrTargetRef436"
targetRef="sourceOrTargetRef437" />
28 - <exclusiveGateway id="sourceOrTargetRef437" name="Gateway"
gatewayDirection="Mixed" >
29 -  <extensionElements>
30 -    <activiti:executionListener
class="org.prisma.rdl.runtime.bpmn.listener.IfDoParallelListener
" event="start" >
31 -      <activiti:field name="beginDoparallelAddress"
stringValue="433" />
33 -    </activiti:executionListener>
34 -  </extensionElements>
35 - </exclusiveGateway>
36 - <sequenceFlow id="flowThen437to442"
sourceRef="sourceOrTargetRef437"
targetRef="sourceOrTargetRef442" >
37 -       <conditionExpression xsi:type="tFormalExpression" >$
{!condStatus440 and (numExecInstances433 ==
numCreatedInstances433)}</conditionExpression>
38 - </sequenceFlow>
39 - <sequenceFlow id="flowElse437to438"
sourceRef="sourceOrTargetRef437"
targetRef="sourceOrTargetRef438" >
40 -       <conditionExpression xsi:type="tFormalExpression" >$
{!(!condStatus440 and (numExecInstances433 ==
numCreatedInstances433))}</conditionExpression>
41 - </sequenceFlow>
42 - <endEvent id="sourceOrTargetRef438" />
43 - <serviceTask id="sourceOrTargetRef439" name="BeginThread"
activiti:class="org.prisma.rdl.runtime.bpmn.delegate.NoneDelegat
e" >
44 -   <extensionElements>
45 -      <activiti:field name="currentADDRESS"
stringValue="439" />
46 -  </extensionElements>
47 - </serviceTask>
48 - <sequenceFlow id="flow439to440"
sourceRef="sourceOrTargetRef439"
targetRef="sourceOrTargetRef440" />
49 - <userTask id="sourceOrTargetRef440"
name="EvalConditionBPMN" >
50 - <documentation></documentation>
51 -  <extensionElements>
52 -    <activiti:formProperty required="true" type="string"
name="Criar outro ícone?(Y/N)"  id="condition" />
53 -    <activiti:taskListener
class="org.prisma.rdl.runtime.bpmn.listener.EvalConditionListene
r" event="complete" >
54 -      <activiti:field name="currentADDRESS"
stringValue="440" />
```

```
55 -    </activiti:taskListener>
56 -  </extensionElements>
57 -  <potentialOwner>
58 -    <resourceAssignmentExpression>
59 -      <formalExpression>user($
{initiator})</formalExpression>
60 -    </resourceAssignmentExpression>
61 -  </potentialOwner>
62 - </userTask>
63 - <sequenceFlow id="flow440to441"
sourceRef="sourceOrTargetRef440"
targetRef="sourceOrTargetRef441" />
64 - <inclusiveGateway id="sourceOrTargetRef441" name="Gateway"
gatewayDirection="Mixed" >
65 -  <extensionElements>
66 -    <activiti:executionListener
class="org.prisma.rdl.runtime.bpmn.listener.EndDoParallelListene
r" event="start" >
67 -      <activiti:field name="beginDoParallelAddress"
stringValue="433" />
68 -    </activiti:executionListener>
69 -  </extensionElements>
70 - </inclusiveGateway>
71 - <sequenceFlow id="flow441to433"
sourceRef="sourceOrTargetRef441"
targetRef="sourceOrTargetRef433" >
72 -       <conditionExpression xsi:type="tFormalExpression" >$
{condStatus440}</conditionExpression>
73 - </sequenceFlow>
74 - <sequenceFlow id="flow441to442"
sourceRef="sourceOrTargetRef441"
targetRef="sourceOrTargetRef442" >
75 -       <conditionExpression xsi:type="tFormalExpression" >$
{!condStatus440}</conditionExpression>
76 - </sequenceFlow>
77 - <sequenceFlow id="flow441to437"
sourceRef="sourceOrTargetRef441"
targetRef="sourceOrTargetRef437" >
78 -       <conditionExpression xsi:type="tFormalExpression" >$
{!condStatus440 and (numExecInstances433 ==
numCreatedInstances433)}</conditionExpression>
79 - </sequenceFlow>
80 - <inclusiveGateway id="sourceOrTargetRef442" name="Gateway"
gatewayDirection="Converging" />
81 - <sequenceFlow id="flow442to443"
sourceRef="sourceOrTargetRef442"
targetRef="sourceOrTargetRef443" />
<!-- End Process -->
82 - <endEvent id="sourceOrTargetRef443" />
```

ANEXO III – Programa CollabRDL para aplicações Openswing baseadas no Demo10

```
1 - import '/models/clientos.uml';
2 - import '/models/commonos.uml';
3 - export '/models/app.uml';
4 - //Framework OpenSwing http://oswing.sourceforge.net/
5 - //http://oswing.sourceforge.net/demo10/demo10.jnlp
6 - COOKBOOK Demo10
7 -  RECIPE  main(){
8 -  appPack = NEW_PACKAGE(appmodel,"?");
9 -  clientApplicationClass = NEW_CLASS(appPack, "?");
10 - NEW_REALIZATION(clientApplicationClass,
                org.openswing.swing.mdi.client.MDIController);
11 - NEW_REALIZATION(clientApplicationClass,
     org.openswing.swing.permissions.client.LoginController);
12 - clientFacadeClass = NEW_CLASS(appPack, "?");
13 - NEW_REALIZATION(clientFacadeClass,
                org.openswing.swing.mdi.client.ClientFacade);
14 - DOPARALLEL {
15 -    newFunction = NEW_METHOD(clientFacadeClass, "?");
16 -    ROLE (analyst, "Create frameControllerClass,
                frameClass and detailFrameControllerClass."){
17 -    frameControllerClass = CLASS_EXTENSION(
          org.openswing.swing.table.client.GridController,
                                                appPack, "?");
18 -    NEW_REALIZATION(frameControllerClass,
                org.openswing.swing.table.java.GridDataLocator);
19 -    IF ("Do you want redefine enterButton method?") THEN {
20 -       m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                frameControllerClass,enterButton);
21 -    }
22 -    IF ("Do you want redefine doubleClick method?") THEN {
23 -       m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,doubleClick);
24 -    }
25 -    IF ("Do you want redefine selectedCell method?") THEN {
26 -       m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                frameControllerClass,selectedCell);
27 -    }
28 -    IF ("Do you want redefine validateCell method?") THEN {
29 -       m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                frameControllerClass,validateCell);
30 -    }
```

```
31 -     IF ("Do you want redefine insertRecords method?") THEN
{
32 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,insertRecords);
33 -     }
34 -     IF ("Do you want redefine updateRecords method?") THEN
{
35 -        m = METHOD_EXTENSION(
           org.openswing.swing.table.client.GridController,
                      frameControllerClass,updateRecords);
36 -     }
37 -     IF ("Do you want redefine deleteRecords method?") THEN
{
38 -       m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,deleteRecords);
39 -     }
40 -      IF ("Do you want redefine createValueObject method?")
THEN {
41 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,createValueObject);
42 -     }
43 -      IF ("Do you want redefine loadDataCompleted method?")
THEN {
44 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,loadDataCompleted);
45 -     }
46 -     IF ("Do you want redefine rowChanged method?") THEN {
47 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,rowChanged);
48 -     }
49 -       IF ("Do you want redefine afterReloadGrid method?")
THEN {
50 -     m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,afterReloadGrid);
51 -     }
52 -     IF ("Do you want redefine beforeCopyGrid method?") THEN
{
53 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,beforeCopyGrid);
54 -     }
55 -    IF ("Do you want redefine beforeSaveDataInEdit method?")
THEN {
56 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
```

```
                frameControllerClass,beforeSaveDataInEdit);
57 -     }
58 -        IF  ("Do  you  want  redefine  beforeSaveDataInInsert
method?") THEN {
59 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                frameControllerClass,beforeSaveDataInInsert);
60 -     }
61-      IF ("Do you want redefine beforeEditGrid method?") THEN
{
62 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,beforeEditGrid);
63       }
64 -      IF  ("Do  you  want  redefine  beforeInsertGrid  method?")
THEN {
65 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,beforeInsertGrid);
66 -     }
67 -      IF  ("Do  you  want  redefine  beforeDeleteGrid  method?")
THEN {
68 -     m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,beforeDeleteGrid);
69 -     }
70 -      IF ("Do you want redefine afterEditGrid method?") THEN
{
71-         m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,afterEditGrid);
72 -     }
73 -       IF  ("Do  you  want  redefine  afterInsertGrid  method?")
THEN {
74 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                frameControllerClass,afterInsertGrid);
75 -     }
76 -       IF  ("Do  you  want  redefine  afterDeleteGrid  method?")
THEN {
77 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,afterDeleteGrid);
78 -     }
79 -     IF ("Do you want redefine modeChanged method?") THEN {
80 -        m = METHOD_EXTENSION(
                org.openswing.swing.table.client.GridController,
                     frameControllerClass,modeChanged);
81 -     }
82 -      IF ("Do you want define getFont method (each cell of
the grid)?") THEN {
```

```
83 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,getFont);
84 -     }
85 -     IF ("Do you want define getHeaderTooltip method
                      (identify a grid column)?") THEN {
86 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,getHeaderTooltip);
87 -     }
88 -     IF ("Do you want define getCellTooltip method (
                 specified row and attribute name)?") THEN {
89 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,getCellTooltip);
90 -     }
91 -      IF ("Do you want redefine getInitialQuickFilterValue
method?") THEN {
92 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
           frameControllerClass,getInitialQuickFilterValue);
93 -     }
94 -      IF ("Do you want redefine getInitialQuickFilterValue
method?") THEN {
95 -        m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
           frameControllerClass,getInitialQuickFilterValue);
96 -     }
97 -      IF ("Do you want redefine beforeFilterGrid method?")
THEN {
98 -        m =
METHOD_EXTENSION(org.openswing.swing.table.client.GridController
           ,frameControllerClass,
           beforeFilterGrid);
99 -     }
100 -     IF ("Do you want redefine getExportDialogSize method
                                (dialog size)?") THEN {
101 -       m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,getExportDialogSize);
102      }
103       IF ("Do you want redefine getExportingFormats method
                 (formats allowed for the grid)?") THEN {
104 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,getExportingFormats);
105 -    }
106 -    IF ("Do you want redefine exportGrid method?") THEN {
107 -       m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                      frameControllerClass,exportGrid);
```

```
108 -    }
109 -     IF ("Do you want redefine beforeRetrieveAdditionalRows
method?") THEN {
110 -       m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
            frameControllerClass,beforeRetrieveAdditionalRows);
111 -    }
112 -    IF ("Drag event is enabled, do you want set to false?")
                                                      THEN {
113 -       dragEnabledMethod = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dragEnabled);
114 -       ADD_CODE(frameControllerClass,dragEnabledMethod,
                                             "return false;");
115 -    } ELSE {
116 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dragEnter);
117 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dragExit);
118 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,dragOver);
119 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dropActionChanged);
120 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dragDropEnd);
121 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                       frameControllerClass,dropEnter);
122 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,dropExit);
123 -    m = METHOD_EXTENSION(
                 org.openswing.swing.table.client.GridController,
                 frameControllerClass,dropOver);
124 -    }
125 -    ROLE (designer, "Use visual plugin for Eclipse or
                                             Netbeans!"){
126-        frameClass = CLASS_EXTENSION(
                 org.openswing.swing.mdi.client.InternalFrame,
                       appPack, "?");
127-        EXTERNAL_TASK("Edit frameClass using visual plugin
                                for Eclipse or Netbeans.");
128 -    }
129 -    IF ("Create Detail Frame?") THEN {
130 -       detailFrameControllerClass = CLASS_EXTENSION(
```

```
          org.openswing.swing.client.FormController, appPack,
"?");
131 -       IF ("Implement insertRecord method in detail
                                              frame?") THEN {
132 -               m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                      detailFrameControllerClass,insertRecord);
133 -       }
134 -       IF ("Implement updateRecord method in detail
                                              frame?") THEN {
135 -          m =
METHOD_EXTENSION(org.openswing.swing.client.FormController,detai
lFrameControllerClass,
          updateRecord);
136 -       }
137 -       IF ("Implement deleteRecord method in detail
                                              frame?") THEN {
138 -          m = METHOD_EXTENSION(
                org.openswing.swing.client.FormController,
                detailFrameControllerClass,deleteRecord);
139 -       }
140 -       IF ("Implement createPersistentObject method in
                           detail frame?") THEN {
141 -          m = METHOD_EXTENSION(
                org.openswing.swing.client.FormController,
          detailFrameControllerClass,createPersistentObject);
142 -       }
143 -         IF ("Implement loadDataCompleted method in detail
                                              frame?") THEN {
144 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                detailFrameControllerClass,loadDataCompleted);
145 -       }
146 -       IF ("Implement afterReloadData method in detail
                                              frame?") THEN {
147 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                detailFrameControllerClass,afterReloadData);
148 -       }
149 -       IF ("Implement afterEditData method in detail
                                              frame?") THEN {
150 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                      detailFrameControllerClass,afterEditData);
151 -       }
152 -       IF ("Implement afterInsertData method in detail
                                              frame?") THEN {
153 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                detailFrameControllerClass,afterInsertData);
154 -       }
```

```
155 -       IF ("Implement afterDeleteData method in detail
                                            frame?") THEN {
156 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
                 detailFrameControllerClass,afterDeleteData);
157 -       }
158 -       m =
METHOD_EXTENSION(org.openswing.swing.client.FormController,
                 detailFrameControllerClass,loadData);
159 -        IF ("Implement getObject method in detail frame?")
THEN {
160 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
                       detailFrameControllerClass,getObject);
161 -       }
162 -       IF ("Implement modeChanged method in detail frame?")
                                                      THEN {
163 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
                       detailFrameControllerClass,modeChanged);
164 -       }
165 -       IF ("Redefine beforeSaveDataInEdit method in detail
                                            frame?") THEN {
166 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
          detailFrameControllerClass,beforeSaveDataInEdit);
167 -       }
168 -       IF ("Redefine beforeSaveDataInInsert method in
                               detail frame?") THEN {
169 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
          detailFrameControllerClass,beforeSaveDataInInsert);
170 -       }
171 -       IF ("Redefine beforeEditData method in detail
                                            frame?") THEN {
172 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
                 detailFrameControllerClass,beforeEditData);
173 -       }
174 -       IF ("Redefine beforeInsertData method in detail
                                            frame?") THEN {
175 -          m = METHOD_EXTENSION(
                       org.openswing.swing.client.FormController,
                 detailFrameControllerClass,beforeInsertData);
176 -       }
177 -          IF ("Redefine beforeDeleteData method in detail
frame?") THEN {
178 -          m = METHOD_EXTENSION(
                 org.openswing.swing.client.FormController,
                 detailFrameControllerClass,beforeDeleteData);
179        }
```

```
180 -       IF ("Redefine afterEditData method in detail
                                              frame?") THEN {
181 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                      detailFrameControllerClass,afterEditData);
182         }
183 -       IF ("Redefine afterInsertData method in detail
                                              frame?") THEN {
184 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                 detailFrameControllerClass,afterInsertData);
185 -       }
186 -       IF ("Redefine afterInsertData method in detail
                                              frame?") THEN {
187 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                 detailFrameControllerClass,afterInsertData);
188 -       }
189 -       IF ("Redefine validateControl method?") THEN {
190 -          m = METHOD_EXTENSION(
                      org.openswing.swing.client.FormController,
                 detailFrameControllerClass,validateControl);
191 -       }
192 -       ROLE (designer, "Use visual plugin for Eclipse or
                                                 Netbeans!"){
193 -          detailFrameClass = CLASS_EXTENSION(
                 org.openswing.swing.mdi.client.InternalFrame,
                      appPack, "?");
194 -          EXTERNAL_TASK("Edit detailFrameClass using visual
                           plugin for Eclipse or Netbeans.");
195 -       }
196 -    }
197 -    }//End ROLE (analyst....
198 - } WHILE("Create another function?"); //End DOPARALLEL
199 - PARALLEL {
200 -       FLOW (analyst, ""){
201 -          LOOP ("Create another LookupController?"){
202 -             lookupControllerClass = CLASS_EXTENSION(
          org.openswing.swing.lookup.client.LookupController,
                                                 appPack, "?");
203               IF ("Redefine validateCode method?") THEN {
204                  m = METHOD_EXTENSION(
          org.openswing.swing.lookup.client.LookupController,
                      lookupControllerClass,validateCode);
205               }
206               IF ("Redefine loadData method?") THEN {
207                  m = METHOD_EXTENSION(
          org.openswing.swing.lookup.client.LookupController,
                      lookupControllerClass,loadData);
208               }
209               IF ("Redefine getTreeModel method?") THEN {
```

```
210 -               m = METHOD_EXTENSION(
          org.openswing.swing.lookup.client.LookupController,
                     lookupControllerClass,getTreeModel);
211 -            }
212 -         }
213 -      }
214 -      FLOW (analyst, ""){
215 -         LOOP ("Create another VOClass?"){
216 -            VOClass = CLASS_EXTENSION(
     org.openswing.swing.message.receive.java.ValueObjectImpl,
                     appPack, "?");
217 -            LOOP ("Create another attribute in VOClass?"){
218 -               NEW_ATTRIBUTE (VOClass,"?",int);
219 -            }
220 -            EXTERNAL_TASK("Nothing: LOOP bug.");
221 -         }
222 -      }
223   }// End PARALLEL
224 - ROLE (databaseAdministrator, "See vOClass created."){
225 -    EXTERNAL_TASK("Create Database.");
226 - }
227 -  }
228 -  END_COOKBOOK
```

ANEXO IV – Modelo da Aplicação gerada pela prova de conceito

```
     1     <?xml version="1.0" encoding="UTF-8"?>

     2     <uml:Model xmi:version="2.1"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
xmi:id="_73dcEPgaEeC0HvqCO384og" name="appmodel">

     3       <packagedElement xmi:type="uml:Package"
xmi:id="_OKRewJKIEeKhVeAp7ddhlw" name="demo10ed2103">

     4         <packagedElement xmi:type="uml:Class"
xmi:id="_R0vlYJKIEeKhVeAp7ddhlw" name="ClientApplication"
clientDependency="_R0yBoJKIEeKhVeAp7ddhlw
_R0yosJKIEeKhVeAp7ddhlw">

     5           <interfaceRealization
xmi:id="_R0yBoJKIEeKhVeAp7ddhlw" name="MDIController"
client="_R0vlYJKIEeKhVeAp7ddhlw">

     6             <supplier xmi:type="uml:Interface"
href="../../../../../../models/clientos.uml#_HLiyMELgEeKW2YecvxN
KPQ"/>

     7             <contract
href="../../../../../../models/clientos.uml#_HLiyMELgEeKW2YecvxN
KPQ"/>

     8           </interfaceRealization>

     9           <interfaceRealization
xmi:id="_R0yosJKIEeKhVeAp7ddhlw" name="LoginController"
client="_R0vlYJKIEeKhVeAp7ddhlw">

    10             <supplier xmi:type="uml:Interface"
href="../../../../../../models/clientos.uml#_HMX4oELgEeKW2YecvxN
KPQ"/>

    11             <contract
href="../../../../../../models/clientos.uml#_HMX4oELgEeKW2YecvxN
KPQ"/>

    12           </interfaceRealization>

    13         </packagedElement>
```

```
    14         <packagedElement xmi:type="uml:Class"
xmi:id="_V1T30JKIEeKhVeAp7ddhlw" name="DemoClientFacade"
clientDependency="_V1WUEJKIEeKhVeAp7ddhlw">

    15           <interfaceRealization
xmi:id="_V1WUEJKIEeKhVeAp7ddhlw" name="ClientFacade"
client="_V1T30JKIEeKhVeAp7ddhlw">

    16             <supplier xmi:type="uml:Interface"
href="../../../../../../models/clientos.uml#_HLjZQELgEeKW2YecvxN
KPQ"/>

    17             <contract
href="../../../../../../models/clientos.uml#_HLjZQELgEeKW2YecvxN
KPQ"/>

    18           </interfaceRealization>

    19           <ownedOperation
xmi:id="_mbe_4JKIEeKhVeAp7ddhlw" name="getEmployees"/>

    20           <ownedOperation
xmi:id="_rOZN4JKLEeKhVeAp7ddhlw" name="getDepts"/>

    21           <ownedOperation
xmi:id="_Yx2vMJKNEeKhVeAp7ddhlw" name="getTasks"/>

    22         </packagedElement>

    23         <packagedElement xmi:type="uml:Class"
xmi:id="__N9bAJKIEeKhVeAp7ddhlw" name="EmpGridFrameController"
clientDependency="__OAeUJKIEeKhVeAp7ddhlw">

    24           <generalization
xmi:id="__N9bAZKIEeKhVeAp7ddhlw">

    25             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HNSeoELgEeKW2YecvxN
KPQ"/>

    26           </generalization>

    27           <interfaceRealization
xmi:id="__OAeUJKIEeKhVeAp7ddhlw" name="GridDataLocator"
client="__N9bAJKIEeKhVeAp7ddhlw">

    28             <supplier xmi:type="uml:Interface"
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>
```

```
    29              <contract
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>

    30           </interfaceRealization>

    31           <ownedOperation
xmi:id="_W8UGsJKJEeKhVeAp7ddhlw" name="doubleClick"/>

    32           <ownedOperation
xmi:id="_mSm7QJKJEeKhVeAp7ddhlw" name="deleteRecords"/>

    33           <ownedOperation
xmi:id="_VNHkgJKKEeKhVeAp7ddhlw" name="dragEnter"/>

    34           <ownedOperation
xmi:id="_VNKAwJKKEeKhVeAp7ddhlw" name="dragExit"/>

    35           <ownedOperation
xmi:id="_VNL18JKKEeKhVeAp7ddhlw" name="dragOver"/>

    36           <ownedOperation
xmi:id="_VNO5QJKKEeKhVeAp7ddhlw" name="dropActionChanged"/>

    37           <ownedOperation
xmi:id="_VNR8kJKKEeKhVeAp7ddhlw" name="dragDropEnd"/>

    38           <ownedOperation
xmi:id="_VNU_4JKKEeKhVeAp7ddhlw" name="dropEnter"/>

    39           <ownedOperation
xmi:id="_VNYqQJKKEeKhVeAp7ddhlw" name="dropExit"/>

    40           <ownedOperation
xmi:id="_VNcUoJKKEeKhVeAp7ddhlw" name="dropOver"/>

    41         </packagedElement>

    42         <packagedElement xmi:type="uml:Class"
xmi:id="_gprucJKKEeKhVeAp7ddhlw" name="EmpGridFrame">

    43           <generalization
xmi:id="_gprucZKKEeKhVeAp7ddhlw">

    44             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HLfH0ELgEeKW2YecvxN
KPQ"/>

    45           </generalization>

    46         </packagedElement>
```

```
    47         <packagedElement xmi:type="uml:Class"
xmi:id="_ghSK8JKLEeKhVeAp7ddhlw" name="EmpDetailFrame">

    48           <generalization
xmi:id="_ghSK8ZKLEeKhVeAp7ddhlw">

    49             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HLfH0ELgEeKW2YecvxN
KPQ"/>

    50           </generalization>

    51         </packagedElement>

    52         <packagedElement xmi:type="uml:Class"
xmi:id="_NOi5gJKMEeKhVeAp7ddhlw" name="DeptFrameController"
clientDependency="_NOlVwJKMEeKhVeAp7ddhlw">

    53           <generalization
xmi:id="_NOi5gZKMEeKhVeAp7ddhlw">

    54             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HNSeoELgEeKW2YecvxN
KPQ"/>

    55           </generalization>

    56           <interfaceRealization
xmi:id="_NOlVwJKMEeKhVeAp7ddhlw" name="GridDataLocator"
client="_NOi5gJKMEeKhVeAp7ddhlw">

    57             <supplier xmi:type="uml:Interface"
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>

    58             <contract
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>

    59           </interfaceRealization>

    60           <ownedOperation
xmi:id="_ZRREgJKMEeKhVeAp7ddhlw" name="insertRecords"/>

    61           <ownedOperation
xmi:id="_bPIfYJKMEeKhVeAp7ddhlw" name="updateRecords"/>

    62           <ownedOperation
xmi:id="_cqNWQJKMEeKhVeAp7ddhlw" name="deleteRecords"/>
```

```
    63          <ownedOperation
xmi:id="_ebBMkJKMEeKhVeAp7ddhlw" name="createValueObject"/>

    64          <ownedOperation
xmi:id="_f90cEJKMEeKhVeAp7ddhlw" name="loadDataCompleted"/>

    65          <ownedOperation
xmi:id="_z9lWIJKMEeKhVeAp7ddhlw" name="getHeaderTooltip"/>

    66          <ownedOperation
xmi:id="_1sGtcJKMEeKhVeAp7ddhlw" name="getCellTooltip"/>

    67          <ownedOperation
xmi:id="_AGoLkJKNEeKhVeAp7ddhlw" name="dragEnter"/>

    68          <ownedOperation
xmi:id="_AGqAwJKNEeKhVeAp7ddhlw" name="dragExit"/>

    69          <ownedOperation
xmi:id="_AGr18JKNEeKhVeAp7ddhlw" name="dragOver"/>

    70          <ownedOperation
xmi:id="_AGuSMJKNEeKhVeAp7ddhlw" name="dropActionChanged"/>

    71          <ownedOperation
xmi:id="_AGwucJKNEeKhVeAp7ddhlw" name="dragDropEnd"/>

    72          <ownedOperation
xmi:id="_AGzxwJKNEeKhVeAp7ddhlw" name="dropEnter"/>

    73          <ownedOperation
xmi:id="_AG21EJKNEeKhVeAp7ddhlw" name="dropExit"/>

    74          <ownedOperation
xmi:id="_AG54YJKNEeKhVeAp7ddhlw" name="dropOver"/>

    75        </packagedElement>

    76        <packagedElement xmi:type="uml:Class"
xmi:id="_I8wewJKNEeKhVeAp7ddhlw" name="DeptGridFrame">

    77          <generalization
xmi:id="_I8xF0JKNEeKhVeAp7ddhlw">

    78            <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HLfH0ELgEeKW2YecvxN
KPQ"/>

    79          </generalization>

    80        </packagedElement>
```

```
    81         <packagedElement xmi:type="uml:Class"
xmi:id="_hYEXEJKNEeKhVeAp7ddhlw" name="TaskGridFrameController"
clientDependency="_hYGzUJKNEeKhVeAp7ddhlw">

    82           <generalization
xmi:id="_hYEXEZKNEeKhVeAp7ddhlw">

    83             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HNSeoELgEeKW2YecvxN
KPQ"/>

    84           </generalization>

    85           <interfaceRealization
xmi:id="_hYGzUJKNEeKhVeAp7ddhlw" name="GridDataLocator"
client="_hYEXEJKNEeKhVeAp7ddhlw">

    86             <supplier xmi:type="uml:Interface"
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>

    87             <contract
href="../../../../../../models/commonos.uml#_BS1F0EL1EeK5APE3Kji
XIg"/>

    88           </interfaceRealization>

    89           <ownedOperation
xmi:id="_wMzWsJKNEeKhVeAp7ddhlw" name="insertRecords"/>

    90           <ownedOperation
xmi:id="_xtgUcJKNEeKhVeAp7ddhlw" name="updateRecords"/>

    91           <ownedOperation
xmi:id="_zBazYJKNEeKhVeAp7ddhlw" name="deleteRecords"/>

    92           <ownedOperation
xmi:id="_1NPKsJKNEeKhVeAp7ddhlw" name="createValueObject"/>

    93           <ownedOperation
xmi:id="_NL0qwJKOEeKhVeAp7ddhlw" name="dragEnter"/>

    94           <ownedOperation
xmi:id="_NL2f8JKOEeKhVeAp7ddhlw" name="dragExit"/>

    95           <ownedOperation
xmi:id="_NL4VIJKOEeKhVeAp7ddhlw" name="dragOver"/>

    96           <ownedOperation
xmi:id="_NL6xYJKOEeKhVeAp7ddhlw" name="dropActionChanged"/>
```

```
    97          <ownedOperation
xmi:id="_NL90sJKOEeKhVeAp7ddhlw" name="dragDropEnd"/>

    98          <ownedOperation
xmi:id="_NMAQ8JKOEeKhVeAp7ddhlw" name="dropEnter"/>

    99          <ownedOperation
xmi:id="_NMDUQJKOEeKhVeAp7ddhlw" name="dropExit"/>

   100          <ownedOperation xmi:id="_NMG-
oJKOEeKhVeAp7ddhlw" name="dropOver"/>

   101        </packagedElement>

   102        <packagedElement xmi:type="uml:Class"
xmi:id="_RbnE8JKOEeKhVeAp7ddhlw" name="TaskGridFrame">

   103          <generalization
xmi:id="_RbnsAJKOEeKhVeAp7ddhlw">

   104            <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HLfH0ELgEeKW2YecvxN
KPQ"/>

   105          </generalization>

   106        </packagedElement>

   107        <packagedElement xmi:type="uml:Class"
xmi:id="_DsChIJKPEeKhVeAp7ddhlw" name="DeptLookupController">

   108          <generalization
xmi:id="_DsChIZKPEeKhVeAp7ddhlw">

   109            <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HKbX4ELgEeKW2YecvxN
KPQ"/>

   110          </generalization>

   111        </packagedElement>

   112        <packagedElement xmi:type="uml:Class"
xmi:id="_TvL0oJKPEeKhVeAp7ddhlw" name="TaskLookupController">

   113          <generalization
xmi:id="_TvMbsJKPEeKhVeAp7ddhlw">
```

```
   114             <general xmi:type="uml:Class"
href="../../../../../../models/clientos.uml#_HKbX4ELgEeKW2YecvxN
KPQ"/>

   115           </generalization>

   116         </packagedElement>

   117         <packagedElement xmi:type="uml:Class"
xmi:id="_mZfXoJKPEeKhVeAp7ddhlw" name="EmpVO">

   118           <generalization
xmi:id="_mZfXoZKPEeKhVeAp7ddhlw">

   119             <general xmi:type="uml:Class"
href="../../../../../../models/commonos.uml#_BRaJgEL1EeK5APE3Kji
XIg"/>

   120           </generalization>

   121           <ownedAttribute
xmi:id="_qhNa8JKPEeKhVeAp7ddhlw" name="salary">

   122             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   123             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_qhOCAZKPEeKhVeAp7ddhlw" value="1"/>

   124             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_qhOCAJKPEeKhVeAp7ddhlw"/>

   125           </ownedAttribute>

   126           <ownedAttribute
xmi:id="_tlQawJKPEeKhVeAp7ddhlw" name="hireData">

   127             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   128             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_tlQawpKPEeKhVeAp7ddhlw" value="1"/>

   129             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_tlQawZKPEeKhVeAp7ddhlw"/>

   130           </ownedAttribute>
```

```
   131           <ownedAttribute
xmi:id="_wGC8gJKPEeKhVeAp7ddhlw" name="note">

   132             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   133             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_wGDjkZKPEeKhVeAp7ddhlw" value="1"/>

   134             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_wGDjkJKPEeKhVeAp7ddhlw"/>

   135           </ownedAttribute>

   136           <ownedAttribute
xmi:id="_zCITIJKPEeKhVeAp7ddhlw" name="sex">

   137             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   138             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_zCITIpKPEeKhVeAp7ddhlw" value="1"/>

   139             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_zCITIZKPEeKhVeAp7ddhlw"/>

   140           </ownedAttribute>

   141           <ownedAttribute
xmi:id="_6_aSwJKPEeKhVeAp7ddhlw" name="taskCode">

   142             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   143             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_6_aSwpKPEeKhVeAp7ddhlw" value="1"/>

   144             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_6_aSwZKPEeKhVeAp7ddhlw"/>

   145           </ownedAttribute>

   146           <ownedAttribute
xmi:id="_9npgUJKPEeKhVeAp7ddhlw" name="taskDescription">
```

```
   147             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   148             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_9nqHYJKPEeKhVeAp7ddhlw" value="1"/>

   149             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_9npgUZKPEeKhVeAp7ddhlw"/>

   150           </ownedAttribute>

   151         </packagedElement>

   152         <packagedElement xmi:type="uml:Class"
xmi:id="_F3M7sJKQEeKhVeAp7ddhlw" name="GridEmpVO">

   153           <generalization
xmi:id="_F3NiwJKQEeKhVeAp7ddhlw">

   154             <general xmi:type="uml:Class"
href="../../../../../../models/commonos.uml#_BRaJgEL1EeK5APE3Kji
XIg"/>

   155           </generalization>

   156           <ownedAttribute
xmi:id="_KUjQwJKQEeKhVeAp7ddhlw" name="empCode">

   157             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   158             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_KUjQwpKQEeKhVeAp7ddhlw" value="1"/>

   159             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_KUjQwZKQEeKhVeAp7ddhlw"/>

   160           </ownedAttribute>

   161           <ownedAttribute
xmi:id="_OuvkIJKQEeKhVeAp7ddhlw" name="lastName">

   162             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>
```

```
   163             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_OuwLMJKQEeKhVeAp7ddhlw" value="1"/>

   164             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_OuvkIZKQEeKhVeAp7ddhlw"/>

   165           </ownedAttribute>

   166           <ownedAttribute
xmi:id="_Sb1g8JKQEeKhVeAp7ddhlw" name="firstName">

   167             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   168             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_Sb2IAZKQEeKhVeAp7ddhlw" value="1"/>

   169             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_Sb2IAJKQEeKhVeAp7ddhlw"/>

   170           </ownedAttribute>

   171           <ownedAttribute
xmi:id="_VlW2oJKQEeKhVeAp7ddhlw" name="deptCode">

   172             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   173             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_VlW2opKQEeKhVeAp7ddhlw" value="1"/>

   174             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_VlW2oZKQEeKhVeAp7ddhlw"/>

   175           </ownedAttribute>

   176           <ownedAttribute
xmi:id="_blgbMJKQEeKhVeAp7ddhlw" name="deptDescription">

   177             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   178             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_blgbMpKQEeKhVeAp7ddhlw" value="1"/>
```

```
   179             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_blgbMZKQEeKhVeAp7ddhlw"/>

   180           </ownedAttribute>

   181         </packagedElement>

   182         <packagedElement xmi:type="uml:Class"
xmi:id="_jtLF4JKQEeKhVeAp7ddhlw" name="TaskVO">

   183           <generalization
xmi:id="_jtLF4ZKQEeKhVeAp7ddhlw">

   184             <general xmi:type="uml:Class"
href="../../../../../../models/commonos.uml#_BRaJgEL1EeK5APE3Kji
XIg"/>

   185           </generalization>

   186           <ownedAttribute
xmi:id="_nKVwUJKQEeKhVeAp7ddhlw" name="taskCode">

   187             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   188             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_nKVwUpKQEeKhVeAp7ddhlw" value="1"/>

   189             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_nKVwUZKQEeKhVeAp7ddhlw"/>

   190           </ownedAttribute>

   191           <ownedAttribute
xmi:id="_sjke0JKQEeKhVeAp7ddhlw" name="description">

   192             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   193             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_sjke0pKQEeKhVeAp7ddhlw" value="1"/>

   194             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_sjke0ZKQEeKhVeAp7ddhlw"/>

   195           </ownedAttribute>
```

```
   196           <ownedAttribute
xmi:id="_u7A3UJKQEeKhVeAp7ddhlw" name="status">

   197             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   198             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_u7BeYJKQEeKhVeAp7ddhlw" value="1"/>

   199             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_u7A3UZKQEeKhVeAp7ddhlw"/>

   200           </ownedAttribute>

   201         </packagedElement>

   202         <packagedElement xmi:type="uml:Class"
xmi:id="_5OmnAJKQEeKhVeAp7ddhlw" name="DeptVO">

   203           <generalization
xmi:id="_5OmnAZKQEeKhVeAp7ddhlw">

   204             <general xmi:type="uml:Class"
href="../../../../../../models/commonos.uml#_BRaJgEL1EeK5APE3Kji
XIg"/>

   205           </generalization>

   206           <ownedAttribute
xmi:id="_9dfpUJKQEeKhVeAp7ddhlw" name="deptCode">

   207             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   208             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_9dfpUpKQEeKhVeAp7ddhlw" value="1"/>

   209             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_9dfpUZKQEeKhVeAp7ddhlw"/>

   210           </ownedAttribute>

   211           <ownedAttribute
xmi:id="_AU1p4JKREeKhVeAp7ddhlw" name="description">
```

```
   212             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   213             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_AU1p4pKREeKhVeAp7ddhlw" value="1"/>

   214             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_AU1p4ZKREeKhVeAp7ddhlw"/>

   215           </ownedAttribute>

   216           <ownedAttribute
xmi:id="_EFyL8JKREeKhVeAp7ddhlw" name="address">

   217             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   218             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_EFyL8pKREeKhVeAp7ddhlw" value="1"/>

   219             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_EFyL8ZKREeKhVeAp7ddhlw"/>

   220           </ownedAttribute>

   221           <ownedAttribute
xmi:id="_F48LoJKREeKhVeAp7ddhlw" name="status">

   222             <type xmi:type="uml:DataType"
href="../../../../../../models/commonos.uml#_Bat2gEL1EeK5APE3Kji
XIg"/>

   223             <upperValue
xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_F48LopKREeKhVeAp7ddhlw" value="1"/>

   224             <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_F48LoZKREeKhVeAp7ddhlw"/>

   225           </ownedAttribute>

   226         </packagedElement>

   227       </packagedElement>

   228     </uml:Model>
```

Apêndice I – BNF da RDL com os comandos CollabRDL

A expressividade dos novos comandos ROLE, PARALLEL e DOPARALLEL estão em negrito para facilitar a identificação no BNF da RDL.

tokens{
IMPORT = 'import';
EXPORT = 'export';
COOKBOOK = 'COOKBOOK' ;
END_COOKBOOK = 'END_COOKBOOK';
RECIPE = 'RECIPE';
END_RECIPE = 'END_RECIPE';
BEGIN_LOOP = 'LOOP' ;
END_LOOP = 'END_LOOP';
IF = 'IF';
THEN = 'THEN';
ELSE = 'ELSE';
END_IF = 'END_IF';
NEW_CLASS = 'NEW_CLASS';
NEW_ENUMERATION = 'NEW_ENUMERATION';
NEW_ATTRIBUTE = 'NEW_ATTRIBUTE';
NEW_METHOD = 'NEW_METHOD';
NEW_PACKAGE = 'NEW_PACKAGE';
NEW_INHERITANCE = 'NEW_INHERITANCE';
NEW_INTERFACE = 'NEW_INTERFACE';
NEW_REALIZATION = 'NEW_REALIZATION';
CLASS_EXTENSION = 'CLASS_EXTENSION';
METHOD_EXTENSION = 'METHOD_EXTENSION';
RETRIEVE_CLASS = 'RETRIEVE_CLASS';
RETRIEVE_PACKAGE = 'RETRIEVE_PACKAGE';
ADD_CODE = 'ADD_CODE';
EXTERNAL_TASK = 'EXTERNAL_TASK';
LEFT_PARENTHESIS = '(';
RIGHT_PARENTHESIS = ')';
LEFT_CURLY = '{';
RIGHT_CURLY = '}';
SEMICOL = ';';
COMMA = ',';
QUESTION_MARK = '?';
EQUAL = '=';
DOT = '.';
QUOTE = '"';
BEGIN_ROLE = 'ROLE';
BEGIN_FLOW = 'FLOW';

```
        PARALLEL = 'PARALLEL';
        BEGIN_DOPARALLEL = 'DOPARALLEL';
        END_DOPARALLEL = 'WHILE';
}

rdl_script
        :   (import_decl)+ (export_decl)+ cookbook_decl
        ;

import_decl
        :         IMPORT  url  SEMICOL
        ;

export_decl
  : EXPORT  url  SEMICOL
  ;

cookbook_decl
  : COOKBOOK ID recipe_main recipe_decl* END_COOKBOOK
  ;

recipe_main
  :  RECIPE 'main' LEFT_PARENTHESIS RIGHT_PARENTHESIS statement_block
  ;

recipe_decl
  :  RECIPE ID LEFT_PARENTHESIS RIGHT_PARENTHESIS statement_block
  ;

statement_block
        :  LEFT_CURLY (statement)* RIGHT_CURLY
        ;

statement
        : assignment SEMICOL| reuse_action SEMICOL | recipe_call SEMICOL|
structured_statement
        ;

assignment
        : variable EQUAL expression
         ;

variable: ID
          ;

expression
```

```
        : reuse_action
        ;

reuse_action    :
        ra_new_package
        | ra_new_class
        | ra_new_method
        | ra_new_attribute
        | ra_new_inheritance
        | ra_new_enumeration
        | ra_retrieve_class
        | ra_retrieve_package
        | ra_class_extension
        | ra_method_extension
        | ra_new_interface
        | ra_new_interface_realization
        | ra_add_code
        | ra_external_task
        ;

ra_new_interface
        :  NEW_INTERFACE LEFT_PARENTHESIS qualified_id COMMA STRING
                RIGHT_PARENTHESIS
        ;

ra_new_interface_realization
        : NEW_REALIZATION LEFT_PARENTHESIS id1=qualified_id COMMA
                id2=qualified_id RIGHT_PARENTHESIS
        ;

ra_add_code
        : ADD_CODE LEFT_PARENTHESIS idClass=qualified_id COMMA
                idOper=ID COMMA STRING RIGHT_PARENTHESIS
        ;

ra_external_task
        : EXTERNAL_TASK LEFT_PARENTHESIS STRING
                RIGHT_PARENTHESIS
        ;

recipe_call
        :       ID LEFT_PARENTHESIS RIGHT_PARENTHESIS
        ;

ra_new_class
```

```
        :  NEW_CLASS LEFT_PARENTHESIS pkg=qualified_id COMMA STRING
                RIGHT_PARENTHESIS
        ;

ra_new_enumeration
        :  NEW_ENUMERATION LEFT_PARENTHESIS pkg=qualified_id COMMA
                STRING  RIGHT_PARENTHESIS
        ;

ra_new_method
        :  NEW_METHOD LEFT_PARENTHESIS cls=qualified_id COMMA STRING
                RIGHT_PARENTHESIS
        ;

ra_new_attribute
        : NEW_ATTRIBUTE LEFT_PARENTHESIS cls=qualified_id COMMA
                name=STRING COMMA type=qualified_id RIGHT_PARENTHESIS
        ;

ra_new_package
        : NEW_PACKAGE LEFT_PARENTHESIS pkg=qualified_id COMMA
                STRING RIGHT_PARENTHESIS
        ;

ra_new_inheritance
        : NEW_INHERITANCE LEFT_PARENTHESIS id1=qualified_id COMMA
                id2=qualified_id RIGHT_PARENTHESIS
        ;

ra_retrieve_class
        : RETRIEVE_CLASS LEFT_PARENTHESIS STRING
                RIGHT_PARENTHESIS
        ;

ra_retrieve_package
        : RETRIEVE_PACKAGE LEFT_PARENTHESIS STRING
                RIGHT_PARENTHESIS
        ;

ra_class_extension
        : CLASS_EXTENSION LEFT_PARENTHESIS superID=qualified_id COMMA
                subPackID=qualified_id COMMA STRING RIGHT_PARENTHESIS
        ;

ra_method_extension
```

```
        : METHOD_EXTENSION LEFT_PARENTHESIS idClass=qualified_id
        COMMA idMethod=ID COMMA idSubClass=ID RIGHT_PARENTHESIS
        ;

structured_statement
        : loop_statement
        | if_statement
        | role_statement
        | parallel_statement
        | doparallel_statement
        ;

loop_statement
        : BEGIN_LOOP LEFT_PARENTHESIS condition RIGHT_PARENTHESIS
          statement_block
         ;

if_statement
        : IF LEFT_PARENTHESIS condition RIGHT_PARENTHESIS
           THEN statement_block
          (ELSE statement_block )?
        ;

role_statement
        : BEGIN_ROLE LEFT_PARENTHESIS role COMMA comment
                RIGHT_PARENTHESIS
          statement_block
         ;

parallel_statement
  : PARALLEL
    LEFT_CURLY
    flow_statement  flow_statement (flow_statement)*
    RIGHT_CURLY
   ;

flow_statement
  : BEGIN_FLOW LEFT_PARENTHESIS role COMMA comment
        RIGHT_PARENTHESIS
    statement_block
   ;

doparallel_statement
  : BEGIN_DOPARALLEL
    statement_block
```

```
   END_DOPARALLEL LEFT_PARENTHESIS condition RIGHT_PARENTHESIS
         SEMICOL
  ;

comment
  : STRING
  ;

condition
         : STRING
         ;

role
         : ID
         ;

qualified_id  :
         ID  (DOT ID)*
         ;

url
         :
          ('/')? ID ('/' ID)* model_extension
         ;

model_extension
         : '.uml' | '.xmi'
         ;

ID  :    ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
   ;

INT :   '0'..'9'+
   ;

STRING
   :
    '\"'
    ( ~('\"'|'\\')
    | ESCAPE_SEQUENCE
    )*
    '\"'
      ;

fragment
ESCAPE_SEQUENCE
```

```
    :       '\\' 'b'
    |   '\\' 't'
    |   '\\' 'n'
    |   '\\' 'f'
    |   '\\' 'r'
    |   '\\' '\"'
    |   '\\' '\''
    |   '\\' '\\'
    |       '\\' '0'..'3' OCTAL_DIGIT OCTAL_DIGIT
    |   '\\' OCTAL_DIGIT OCTAL_DIGIT
    |   '\\' OCTAL_DIGIT
            |           UNICODE_CHAR
            ;

fragment
OCTAL_DIGIT
            :           '0'..'7'
            ;

fragment
UNICODE_CHAR
            :           '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
            ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

```
WS  :   (' '|'\t')+ {skip();} ;

COMMENT
    :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    |   '/*' ( options {greedy=false;} : . )* '*/'

            ;
```