UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

LUIZ FERNANDO BATISTA LOJA

Sinfonia: Uma Abordagem Colaborativa e Flexível para Modelagem e Execução de Processos de Negócio

Goiânia
2011

LUIZ FERNANDO BATISTA LOJA

Sinfonia: Uma Abordagem Colaborativa e Flexível para Modelagem e Execução de Processos de Negócio

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Sistemas de Informação.

Orientador: Prof. Juliano Lopes de Oliveira

Goiânia
2011

LUIZ FERNANDO BATISTA LOJA

Sinfonia: Uma Abordagem Colaborativa e Flexível para Modelagem e Execução de Processos de Negócio

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 10 de Fevereiro de 2011, pela Banca Examinadora constituída pelos professores:

Prof. Juliano Lopes de Oliveira
Instituto de Informática – UFG
Presidente da Banca

Prof. Cedric Luiz de Carvalho
Instituto de Informática – UFG

Prof. Ricardo de Almeida Falbo
Departamento de Informática – UFES

Luiz Fernando Batista Loja

Graduou-se em Ciência da Computação na PUC - Pontifícia Universidade Católica de Goiás. Durante sua graduação, foi monitor da disciplina Estrutura de Dados I e II. Especializou-se em Gestão de Software pelo Centro Universitário de Goiás Uni–Anhanguera

A Deus e aos meus pais que me ajudaram a tornar este sonho antigo em realidade.

Agradecimentos

Ao Cristo, pois muitas vezes me via cercado de problemas que julgava sem solução, mas sempre encontrava inspiração para resolvê-los da melhor maneira possível.

Aos meus pais Helena e Luiz Antônio, meus irmãos Bianca e Luiz Antônio, minha avó Amélia e minha madrinha Willma por todo amor e apoio na conclusão deste mestrado.

Ao meu orientador Juliano Lopes de Oliveira, pessoa que possuo profunda admiração, que me aceitou como orientando e sempre acreditou no meu potencial como aluno e ser humano.

À Fabiana Freitas Mendes que é uma grande amiga e irmã, pelo seu empenho, dedicação e sacrifício em me ajudar e principalmente por estar sempre presente, como ninguém, em todos os momentos desta difícil jornada. Não sei o que seria de mim sem os seus ensinamentos.

Ao meu amigo Valdemar Neto pela amizade sincera que me impulsionou a querer melhorar e muitas vezes me inspirou.

À minha tão estimada amiga, tia Edna, que me ajudou a ingressar neste mestrado me dando valiosos conselhos e me incentivando, certamente sem a sua ajuda este trabalho não estaria concluído.

Ao Ministério Público do Estado de Goiás na figura do Dr. José Augusto Falcão, Frederico Guedes Coelho, Luiz Mauro e Wesley Oliveira e a empresa Password Informática representada por Flávio Valente e Ricardo Regis pelo imenso apoio, consideração e principalmente por terem permitido que eu diminuísse a minha carga horária de trabalho, tornando possível minha frequência nas aulas de mestrado e o desenvolvimento e escrita deste trabalho.

À Renata de Oliveira Ferreira e Tanielly Duarte por todo o apoio, paciência, carinho, amor e atenção nesta longa jornada.

Ao meu amigo Renato Meneses, coordenador de trabalho voluntário, que sempre compreendeu a minha dedicação ao mestrado, prorrogando meus prazos e me apoiando.

A todas as pessoas que me ajudaram com os experimentos deste trabalho Sofia Larissa, Marcio Arantes, Leandro Santos, Fernando Mendonça, Iure Guimarães, Saulo Mendonça, Wilker Bueno, Regiane Barbosa, Weuller Jacomini, Augusto Xavier, Cláu-

dio Araujo, Thiago Imolesi, Alexandre Morgado, Leandro Constante, Andrey Oliveira, Guilherme Melo, Max Flávio Cabral, Ramon Marques pela paciência, compreensão e amizade.

Aos meus estimados amigos de mestrado André Rincon, Marcelo Quinta, Elisabete Kowata, Rogério Carvalho, Renan Rodrigues, Jair Alarcon e Adriana Rocha que sempre me apoiaram e não me deixaram caminhar sozinho.

Aos meus amigos Gustavo Costa, Rodrigo Gomide e Shairon Toledo pelos sábios ensinamentos sobre como desenvolver sistemas.

À Cintia, minha professora de português, pela paciência e pelos valiosos ensinamentos.

A toda Superintendência de Informática do Ministério Público de Goiás que como bons amigos sempre estiveram presentes no desenvolvimento deste me incentivando.

E a todas as pessoas que contribuíram de forma direta ou indireta para a conclusão deste trabalho.

Resumo

Loja, Luiz Fernando Batista Loja. **Sinfonia: Uma Abordagem Colaborativa e Flexível para Modelagem e Execução de Processos de Negócio**. Goiânia, 2011. 139p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Para realizar serviços e produtos, as organizações precisam executar processos de negócio. A efetividade destes processos é um fator crítico para o sucesso de qualquer organização. Por essa razão, tem havido grandes esforços para o desenvolvimento de técnicas e ferramentas dirigidas para a melhoria de processos de negócio. Um dos resultados produzidos por esses esforços são os BPMS (*Business Process Management Systems*), *softwares* que auxiliam a definição, análise, e gerenciamento de processos de negócio. Embora existam diversos BPMS disponíveis, os sistemas atuais apenas provêem suporte à execução de um conjunto limitado de processos. Notadamente os BPMS não permitem a execução de processos flexíveis, restringindo sua atuação aos processos definidos. Além disso, eles não proporcionam um ambiente colaborativo de modelagem de processos. Este trabalho apresenta uma arquitetura de software para gerência de processos de negócio que supera essas limitações dos BPMS atuais. A arquitetura foi empregada para a implementação de uma ferramenta de software denominada Sinfonia, que contempla um metamodelo de processos de negócio, uma máquina de execução de processos e um modelador gráfico de processos. Os aspectos inovadores das propostas do presente trabalho abrangem características como o suporte à definição e execução de processos flexíveis, tais como processos empíricos e *ad hoc*, e o apoio à modelagem colaborativa de processos. A ferramenta Sinfonia tem poder de expressão suficiente para definir e executar os principais padrões de processos de negócio descritos na literatura. A capacidade dessa ferramenta de expressar processos flexíveis e de promover a colaboração na modelagem de processos de negócio foram avaliadas em um experimento envolvendo quatorze participantes. Os resultados desse experimento provêem evidências de que Sinfonia contribui para a evolução dos BPMS.

Palavras–chave

Processo de negócio, workflow, BPMS, gerência de processos, WfMS

Abstract

Loja, Luiz Fernando Batista Loja. **Sinfonia: A Colaborative and Flexible Approach to Business Processes Modeling and Execution.**. Goiânia, 2011. 139p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

To offer products and services, organizations need to execute business processes. The effectiveness of these processes is critical to the success of any organization. For this reason, there have been major efforts to develop techniques and tools aimed at the improvement of business processes. One of the results of these efforts are the BPMS (textit Business Process Management Systems), software that help to define, analyze, and manage business processes. Although there are many BPMS available, current systems only provide support for running a limited set of processes. Notably, BPMS do not allow the execution of flexible processes, restricting its operations to defined processes. In addition, they do not provide a collaborative environment for process modeling. This paper presents a software architecture for management of business processes that overcomes these limitations of current BPMS. The architecture was used to implement a software tool called Sinfonia which includes a metamodel for business processes, a process execution engine and a graphical process modeler. The innovative aspects of the proposals of this work include features such as support for defining and implementing flexible processes, such as empirical and textit adhoc processes, and support for modeling collaborative processes. Sinfonia has enough expressive power to define and implement key business process standards described in the literature. The ability of this tool to express flexible processes and to promote collaboration in the modeling of business processes were evaluated in an experiment involving fourteen participants. The results of this experiment provide evidence that Sinfonia contributes to the evolution of BPMS.

Keywords

business process, BPMS, workflow, business process management, BPM, WfMS

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Todo produto ou serviço realizado pelas organizações modernas é necessariamente o resultado de um processo, isto é, de um conjunto de atividades integradas e consistentes [75]. O investimento na padronização e no gerenciamento dos processos tende a aumentar a qualidade de produtos e serviços, pois a maioria dos problemas de qualidade nas organizações decorre de processos mal definidos ou de mecanismos de controle ineficientes [46].

A abordagem de Gerência de Processos de Negócio (*Business Process Management* - BPM) foi criada para otimizar, gerenciar e controlar processos de negócio. A BPM pode ser definida como apoio ao processo de negócio utilizando métodos, técnicas e software para desenhar, organizar, controlar, e analisar processos e operações involvendo humanos, organizações, aplicações, documentos e outras fontes de informação [71].

As organizações possuem processos extensos, complexos, dinâmicos e duradouros [65] e, portanto, necessitam do apoio de Sistemas de Informação (SI) na execução e no controle de seus processos de negócio. Os SIs que apoiam a abordagem BPM são, em geral, chamados de Sistemas Gerenciadores de Processo de Negócio (*Business Process Management System* - BPMS), responsáveis por implementar e gerenciar a lógica dos processos de negócio [45].

1.1 Limitações dos BPMS Atuais

Existem várias ferramentas BPMS disponíveis no mercado. Entretanto, há alguns problemas não solucionados por estas ferramentas [41, 42], tais como:

- A falta de apoio à atividade de modelagem colaborativa, que deve permitir reutilização e compartilhamento de componentes de processo entre os analistas de negócio;
- A falta de automação da distribuição de responsablidades, com base nas características dos participantes dos processos;

- A impossibilidade de alteração de processos durante a sua execução, visando apoiar processos empíricos e *ad hoc* que são tipos de processos importantes em organizações encontram-se em fase de definição de seus processos;
- A deficiência no suporte a padrões de processos, como os propostos por [73, 60].

Segundo Smith e Fingar [65], processos são dependentes das pessoas que os executam, ou seja, é necessário que haja cooperação entre os envolvidos em cada processo para que o negócio atinja seus objetivos.

Ferramentas BPMS têm a finalidade de auxiliar a modelagem e execução de processos, provendo, adicionalmente, um ambiente colaborativo. Tal ambiente deveria proporcionar uma forma de modelar processos colaborativamente, facilitando o compartilhamento de processos e a reutilização de componentes previamente definidos. Porém, a estrutura dos metamodelos de processo adotados nas ferramentas existentes dificulta o compartilhamento de compnentes de processo.

Isso acontece porque, em geral, os artefatos de processo são definidos dentro do contexto de um processo específico. Assim, a granularidade de compartilhamento é definida no nível de processo, e não em componentes menores. Isso limita, por exemplo, a reutilização de partes de um processo para a definição de novos processos.

Além de facilitar a colaboração e a reutilização na modelagem de processos, ferramentas BPMS deveriam prover suporte adequado para a execução dos processos definidos. Para isso, é importante que, durante a execução das tarefas definidas no processo, os colaboradores envolvidos possam cooperar para a realização de um objetivo comum. Um fator crítico para o sucesso dessa cooperação é a seleção e alocação de colaboradores com perfis apropriados, de forma a assegurar que as habilidades necessárias para a execução da tarefa estejam disponíveis na equipe alocada.

Neste sentido, é interessante que as tarefas sejam distribuídas de acordo com o perfil (conhecimentos, experiência e habilidades) de cada participante. Ferramentas BPMS deveriam auxiliar essa atividade de seleção e alocação de participantes, porém as ferramentas pesquisadas por este trabalho não apresentam essa funcionalidade.

Além disso, na abordagem de alocação de participantes utilizada pelas ferramentas pesquisadas, apenas um participante pode ser alocado para executar cada tarefa ou, no melhor caso, um grupo de participantes pode ser alocado, mas com o mesmo tipo de envolvimento na tarefa. Essa abordagem é limitada, pois não permite modelar as cadeias de reponsabilidades que são empregadas no controle de processos das organizações modernas.

Por exemplo, a proposta do modelo COBIT (*Control Objectives for Information and related Technology*) [27] define diversos papéis que podem interagir, com diferentes tipos de envolvimento, durante a execução de uma tarefa. De fato, COBIT propõe quatro tipos de responsabilidade: reponsável (executor), aprovador, interessado e consultado. As

ferramentas atuais não são capazes de oferecer suporte para esse tipo de alocação de responsabilidades.

Como processos de negócio são flexíveis, ou seja, sujeitos a mudanças rápidas e constantes, a possibilidade de alterar um processo durante sua execução é uma característica necessária para ferramentas de BPMS, embora esta característica não seja disponibilizada nas ferramentas pesquisadas.

Outra característica importante para o contexto de modelagem de processos é o suporte a dois tipos de processos: *processos empíricos*, cuja especificação de tarefas é feita durante a própria execução do processo, e *processos definidos*, cuja definição de tarefas é totalmente realizada antes da execução do processo.

Ferramentas atuais de BPMS oferecem suporte apenas para processos definidos e são inadequadas, por exemplo, para representar processos ágeis de desenvolvimento de software, tais como o método *Scrum* [63]. Para lidar com a incerteza, inerente ao desenvolvimento de software, *Scrum* utiliza processos empíricos para desenvolver produtos de software.

Outro tipo de processo que não recebe o devido suporte nas ferramentas de BPMS atuais são os processos *ad hoc*. Esses processos são formados por grupos de atividades que não possuem relações de sequência, ou seja, a ordem do fluxo de execução das atividades de um processo *ad hoc* é indefinida [54]. Essa indefinição dificulta significativamente a implementação deste tipo de processo, pois não há um padrão predefinido para orientar a execução das atividades contidas no processo.

No contexto de modelagem de processos, um conjunto de *padrões de processo* propostos em [73, 60] define os requisitos fundamentais para representação de processos de negócio. Esses padrões deveriam fazer parte do poder de expressão de uma ferramenta de BPMS. No entanto, as ferramentas atuais não implementam todos esses padrões.

Todas essas limitações de ferramentas BPMS atuais representam desafios a serem considerados para o desenvolvimento de uma nova geração de BPMS.

1.2 Objetivos

O principal objetivo deste trabalho é descrever a abordagem utilizada para criar o software Sinfonia, uma ferramenta BPMS que ajuda a superar os desafios propostos na seção anterior. Para tornar isso possível, este trabalho:

- Define um modelo de representação de processos, com base nos padrões propostos em [73, 60], que permite reutilização e compartilhamento de componentes de processo [43];
- Especifica um componente para modelagem gráfica de processos, que facilita a colaboração entre os envolvidos na modelagem do processo de negócio;

- Descreve uma máquina de execução para o modelo de processo definido, permitindo alterar um processo durante a sua execução, além de auxiliar a distribuição de responsabilidades [42].

Sinfonia é um sistema gerenciador de processos que faz parte de um *framework* mais abrangente descrito em [6]. As principais ideias da integração do Sinfonia com o framework são descritas em [13].

1.3 Método de Pesquisa

O método utilizado para o desenvolvimento do presente trabalho está baseado em cinco grandes atividades: (1) Embasamento Teórico, (2) Projeto de Ferramenta de BPMS, (3) Construção da Ferramenta BPMS, (4) Avaliação da Ferramenta e (5) Redação da Dissertação. Dessas atividades, as quatro primeiras foram completadas de forma sequencial, enquanto a quinta atividade foi desenvolvida de maneira concorrente com as demais atividades.

O Embasamento Teórico deste trabalho envolveu uma investigação sobre gerência de processos de negócio, comtemplando padrões de processos, ferramentas de apoio à gerência de processos (BPMS) e notações utilizadas para descrever processos.

Na atividade de Projeto de Ferramenta BPMS foi especificado um metamodelo de processo capaz de lidar com os desafios discutidos na Seção 1.2. Também foi projetado o modelo de uma máquina de execução de processos e a forma de interação dessa máquina com o metamodelo.

A Construção da Ferramenta compreendeu a implementação da máquina de execução, das interfaces com os usuários e do componente gráfico de modelagem de processo.

A Avaliação da Ferramenta foi efetuada por meio da execução dos padrões de processo propostos em [73, 60]. Como estes padrões estabelecem os principais requisitos para representação de processos, quanto maior o número de padrões de processo modelados e executados, maior será seu poder de expressão e a sua capacidade de atender às necessidades de usuários de BPMS.

A atividade de Avaliação contempla, ainda, a análise da aderência da ferramenta aos requisitos estabelecidos para ela. Para cada requisito, foi verificada a viabilidade de avaliá-lo e, em caso positivo, foi definido o teste necessário para comprovar a implementação do requisito na ferramenta.

Ao longo de todo o desenvolvimento do trabalho, artigos descrevendo partes das propostas deste trabalho foram submetidos a conferências nacionais.

Dois artigos foram aceitos no Primeiro Workshop Brasilieiro de Desenvolvimento Dirigido a Modelo [43], [13] e outro foi aceito no Workshop de Teses e Dissertações do Simpósio Brasileiro de Sistemas Colaborativos [42].

1.4 Organização do Trabalho

O Capítulo 2 trata de conceitos fundamentais para o entendimento do trabalho, abordando a gerência de processo de negócio, seus principais conceitos, ferramentas para apoio a seu ciclo de vida e os principais componentes destas ferramentas.

O Capítulo 3 detalha o metamodelo de processos proposto neste trabalho. Além disso, são apresentados os motivos que levaram ao desenvolvimento deste metamodelo, ao invés da utilização de outra forma de representação de processos.

O Capítulo 4 descreve o mecanismo utilizado para executar o processo expresso através do metamodelo de processos. É apresentado seu modelo conceitual e a forma de utilizá-lo para executar processos de negócio.

O Capítulo 5 explica o design arquitetural da ferramenta Sinfonia, discutindo seus componentes e padrões arquiteturais e de projeto utilizados na sua criação.

O Capítulo 6 apresenta dois experimentos realizados com o intuito de avaliar as propostas deste trabalho. O primeiro experimento, realizado com a ferramenta Sinfonia, avalia as funcionalidades disponíveis neste software. O segundo experimento avalia o poder de expressão do metamodelo de processos e da máquina de execução propostos nesta dissertação.

O Capítulo 7 traz as considerações finais do trabalho de pesquisa desenvolvido, comparando-o com trabalhos correlatos e propondo extensões e direções para sua melhoria.

CAPÍTULO 2

Gerenciamento de Processos de Negócio e Ferramentas de Apoio

Os sistemas de informação tradicionais, orientados a dados, não disponibilizam apoio a processos, a menos que estes sejam embutidos no código do sistema. Este tipo de abordagem mescla o código do programa à lógica de processos, resultando em uma arquitetura pouco coesa e sujeita a erros [75].

Ferramentas que gerenciam os processos de negócio (BPMS) surgiram para desacoplar as funcionalidades da aplicação da lógica do processo de negócio [75]. Em geral, esta lógica é transferida para a ferramenta, que permite alterar a lógica do processo sem a necessidade de alteração do código da aplicação. Este capítulo analisa os componentes que formam uma ferramenta responsável por gerenciar processos de negócio.

O capítulo está organizado da seguinte forma: a próxima seção resume as origens da Gerência de Processo de Negócio, enquanto que a Seção 2.2 descreve os principais conceitos a ela relacionados. A Seção 2.3 apresenta várias linguagens de representação de processo. Já a Seção 2.4 descreve mecanismos de execução de processos e suas responsabilidades em uma ferramenta BPMS e finalmente a Seção 2.5 discute padrões de processo.

2.1 Origens do Gerenciamento de Processo de Negócio

Os Sistemas de Informação desenvolvidos nas decadas de 70 e 80 eram orientados a dados [15]. O foco da tecnologia da informação, nesta época, estava em guardar e recuperar informações. Como resultado os dados eram o ponto de partida para construção de sistemas de informação. Portanto o levantamento de requisitos era focado nas informações com as quais os sistemas lidariam [72]. Consequentemente, como o foco do desenvolvimento eram as próprias informações e não o processo responsável por gerá-las, não havia ferramentas que ajudassem no levantamento de processos.

Assim, a lógica de processos das empresas era disseminada entre os vários softwares por elas utilizados. Por exemplo, um processo de aquisição de materiais poderia envolver vários sistemas, como o sistema de compras, de contratos e de material. Primeiramente o processo passa pelo sistemas de compras, depois os dados devem ser passados para o sistema de contratos, e só então incluídos no sistema de material.

Nos anos 90, Hammer e Champy [22] apresentaram a reengenharia de processos de negócio (BPR - *Business Process Reengineering*) como uma reconsideração fundamental e redefinição radical dos processos de uma organização, com o objetivo de otimizar drasticamente o desempenho em custo, serviço e velocidade das empresas. Com essa proposta, os processos das empresas deveriam ser remodelados visando a otimização, clareza, documentação e redução de custos. Com esta ênfase no processo, a indústria de software começou a investir em sistemas orientados a processo. Este tipo de sistema foi chamado de Sistema de Informação voltados a Processo (*Process Aware Information Systems* - PAIS) [15].

Os PAIS separam a lógica de aplicação da lógica de negócios, permitindo alterar a aplicação sem alterar o processo, e vice versa. Logo surgiram os primeiros sistemas que automatizam a execução de processos chamados de Sistemas de Gerenciamento de Fluxo de Trabalho (*Workflow Management Systems* - WfMS) [38].

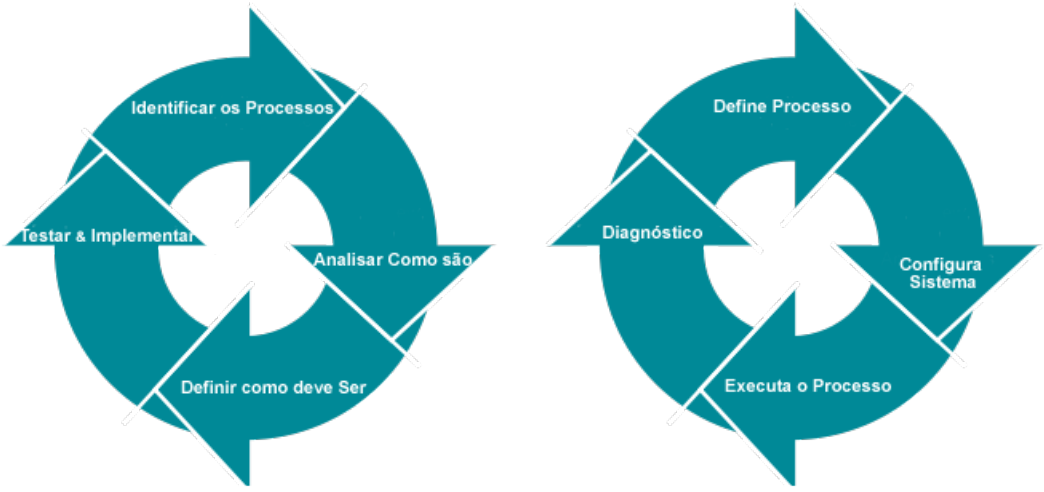
A BPR, apesar de ter iniciado o foco em softwares dirigido a processos, como WfMS, resultou em grande frustração, pois o processo de reengenharia era muito longo e caro para ser implantado na maioria das organizações.

Com a decadência da BPR surgiu o movimento de gerenciamento de processos de negócio (BPM - *Business Process Management*). Ao contrário da BPR, a BPM tem o objetivo de identificar os processos existentes, automatizá-los e gerenciá-los. Portanto, enquanto BPR propõe eliminação dos processos existentes e criação de novos, BPM é mais prático, iterativo e incremental, otimizando os processos existentes [32].

Esta diferença entre as abordagens é facilmente notada através dos ciclos de vida de processo em cada uma. A Figura 2.1(a) [22] ilustra o ciclo de vida da BPR, enquanto a Figura 2.1(b) [71] apresenta o ciclo de vida da BPM.

O ciclo de vida dos processos na BPM tem início com a **definição do processo**, que especifica como ele realmente é em algum sistema gerenciador de processos de negócio (BPMS). Na fase de **configuração do sistema** as regras do sistema são definidas, os papéis envolvidos nas tarefas são configurados e as aplicações que devem ser invocadas durante o andamento do processo são definidas.

Na fase **execução de processo** os processos de negócio são enviados para o servidor e executados pela máquina de execução de processos. Finalmente, na fase de diagnóstico, utilizando as ferramentas necessárias para análise e monitoramento, são identificados pontos de gargalo e otimização dos processos.



(a) *BPR* (b) *BPM*

Figura 2.1: *Ciclos de Vida de Processos de Negócio*

Com a modelagem dos processos é possível visualizar o funcionamento da organização através de seus processos. Além disso, através da análise visual dos modelos dos processos, é mais fácil identificar pontos que tornam o processo mais lento. Estes pontos identificados podem ser candidatos a estudos para possíveis otimizações. Ademais, como os processos podem ser mais facilmente entendidos, o tempo de discussão é reduzido, pois algumas explicações são desnecessárias.

O mapeamento dos processos viabiliza a melhor definição de responsabilidades. Com as atividades mapeadas fica mais fácil definir quais os papéis responsáveis por executar determinada tarefa. Além disso, com os processos definidos é possível auditar suas respectivas execuções, facilitando a detecção de fraudes. Mais ainda, com a auditoria do processo é mais fácil verificar o seu desempenho [32].

2.2 Conceitos de Processos de Negócio

Embora existam várias definições para “Processo de Negócio” na literatura, em geral elas são limitadas, pois são baseadas em algum tipo de metáfora de máquina de exploração de processos [39]. Além disso, a maioria das definições é curta e sucinta, como “Um conjunto de atividades realizadas para servir um cliente” [28]; ou “Um grupo de atividades parcialmente ordenadas que visam atingir um objetivo” [22].

Uma definição mais recente e menos resumida é feita por Wesk [75]: Processo de negócio consiste em um grupo de atividades que são realizadas coordenadamente em um ambiente organizacional e técnico. Estas atividades conjuntamente realizam o objetivo do

negócio. Cada processo de negócio é executado por uma única organização, porém ele pode interagir com processos de negócio de outras organizações.

Ko [32] ressalta as características principais de um processo de negócio, que são: objetivos, temporalidade, localidade, fluxo do processo, atores e a colaboração entre eles. Entretanto, o autor não faz uma definição formal do termo "Processo de Negócio".

Normalmente os processos de negócio são executados de forma manual e guiada pelo conhecimento dos colaboradores das organizações. Além disso, eles são auxiliados pelas regras e procedimentos instaurados [75].

Para automatizar este controle de processos são utilizados WfMS (WorkFlow Management Systems) ou BPMS (Business Process Management Systems). Através destes sistemas é possível definir um processo de negócio e executá-lo. Dessa maneira a lógica de execução do processo é automatizada e transferida dos colaboradores da empresa para o sistema de gerenciamento de processos.

Embora os sistemas BPMS e WfMS gerenciem e executem processos, existe uma diferença entre os dois conceitos. Wesk et. al [74] definem BPMS como um sistema de software genérico orientado a processos explícitos, estruturado para executar e gerenciar processos de negócio operacionais. Sendo assim, um BPMS deve necessariamente ser orientado a processos, ser genérico possibilitando a alteração de seus processos.

Para definir WfMS é necessário antes definir o conceito de workflow. Workflow é a automação do processo de negócio como um todo, ou parte dele, no qual documentos, informações e tarefas são passadas de um participante a outro por ações, de acordo com um grupo de regras de procedimento [25].

Hollingsworth [25] define WfMS como um sistema que define, cria e gerencia a execução de workflows através de um software, envolvendo uma ou mais máquinas de execução, capazes de interpretar a definição de um processo, interagir com os participantes do workflow e invocar ferramentas de tecnologia da informação e aplicativos.

Embora as duas definições citem a execução de processos, este foco em execução é restrito. A definição feita por [74] propõe que BPM engloba o apoio a processo de negócio utilizando métodos, técnicas e software para estruturar, executar, controlar e analisar os processos operacionais envolvendo pessoas, organizações e outras fontes de informação.

Comparando as definições, nota-se que BPM conta com uma propriedade que não está presente na definição de WfMS, que é a análise dos processos. Sendo assim, BPMS é uma extensão de WfMS, com suporte a outras atividades, como diagnóstico do processo [49, 72].

Segundo [38], o núcleo do BPMS é um WfMS. Ou seja, para iniciar uma ferramenta BPMS é necessário que se tenha módulos como: metamodelo de processo, máquina de execução, gerenciador de tarefas (lista de tarefas), administrador da máquina

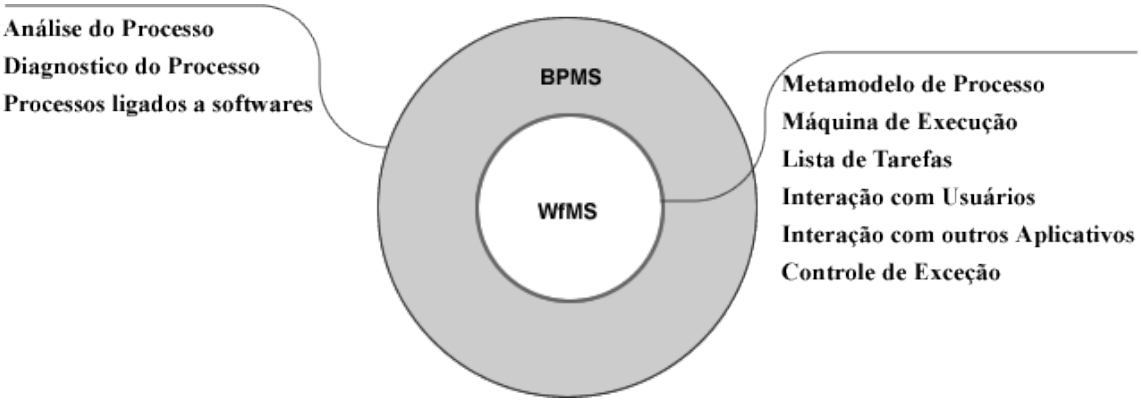


Figura 2.2: *BPMS e WfMS*

de execução, interação com usuários e capacidade de interação com outros aplicativos [15]. A figura 2.2 ilustra esta relação entre estes dois tipos de conceitos, BMPS e WfMS.

Um **metamodelo de processos** representa uma abstração dos conceitos existentes no processo de negócio, permitindo especificar os elementos que compõem um processo, seus atributos, como eles se relacionam e seus respectivos comportamentos. Os metamodelos permitem representar modelos de processo específicos. Um **modelo de processo** consiste em um grupo de modelos de atividades e restrições de execução entre eles.

2.3 Modelagem e Representação de Processos

Utilizar a linguagem natural para expressar processos não é prudente, pois segundo [66], elas são desestruturadas, ambíguas e não apresentam uma forma consistente de representação. Logo, as ferramentas de gerenciamento de processo fazem uso de linguagens de modelagem especificas para representar um processo. Estas linguagens são também conhecidas como metamodelos de processos [53].

Ko [32] estabelece quatro categorias de normas (*standard*) de representação de processo (gráficas, execução, transição e diagnóstico) com base no ciclo de vida da BPM [71]. Cada categoria apoia uma parte deste ciclo de vida.

A fase de definição do processo é apoiada pelas normas gráficas, pois nelas o processo é representado através de um modelo visual. Já as normas de transição proveem suporte à fase de configuração, quando a representação do processo é convertida em uma linguagem de execução. A fase de execução é apoiada pelas normas de execução responsáveis pro criar processos que possam ser interpretados e executados pela máquina. Finalmente, a fase de diagnóstico é apoiada pelas linguagens de consulta e análise de processo.

2.3.1 Notações Gráficas para Representação de Processos

Uma **instância de processo** representa um caso concreto na operação do negócio, composto por instâncias de atividades. Cada modelo de processo de negócio funciona como um molde para um grupo de instâncias do processo de negócio [75].

Ao executar estas instâncias, a máquina de execução distribui as tarefas entre os envolvidos no processo. As tarefas distribuídas para os participantes compõem uma lista de tarefas. Esta lista é administrada para um usuário e contém as tarefas alocadas para os participantes que interagem com estas listas e cumprem as tarefas reservadas a eles.

Geralmente, para cumprir as tarefas os envolvidos no processo necessitam interagir com outros sistemas. Logo, o software de processos invoca outras aplicações com as quais o usuário executor interage.

Para apresentar este modelo de forma clara e concisa é interessante que se use uma notação gráfica [74]. O metamodelo é responsável por definir a sintaxe da notação. Portanto, seus elementos possuem representações visuais distintas. Consequentemente, eles são representados por símbolos, formas, tamanhos e cores diferentes, que ajudam a distinguir entre os elementos.

Selecionar uma notação para modelar sistemas orientados a processo é um tópico frequentemente discutido [7]. Diagrama de Atividades da UML e BPMN são as duas notações mais expressivas, de mais fácil integração nos níveis de transição e execução, e possivelmente as mais influentes em um futuro próximo [33]. Além disso, as duas notações são semelhantes [33, 40], pois ambas notações têm como raiz a teoria de redes de Petri [57].

UML Diagrama de Atividade

O diagrama de atividades é uma notação proposta pela OMG [52]. Este tipo de diagrama foca na sequência de execução do processo em baixo nível e é mais comumente usado para modelagem de processos de maneira gráfica [62].

Este diagrama utiliza ao mesmo tempo técnicas de fluxograma e um caso especial de máquina de estados no qual as atividades são estados e associações interativas representam as transições [24].

Russel [62] destaca alguns pontos fortes desta notação:

- Apoio a envio e recebimento de sinais em um nível conceitual
- Provê suporte a espera e a definição de processamento de estados
- Proporciona um mecanismo para decompor as atividades em sub-atividades. A combinação desta capacidade de decomposição com o apoio a envio de sinal fornece um mecanismo poderoso para tratamento de interrupções.

Business Process Modeling Notation

Esta linguagem é mantida pela OMG e, assim como o diagrama de atividades da UML, é baseada em técnicas de fluxograma, além de adotar a mesma lógica de controle baseada em tokens usada pela rede de Petri [24].

O intuito da BPMN é prover uma notação que seja de fácil entendimento por todos os *stakeholders* do negócio, desde os analistas de negócio até os desenvolvedores responsáveis por implementarem a tecnologia que irá executar os processos, passando pelas pessoas do negócio que devem gerenciar e monitorar estes processos [54]. Além disso, ela tem o objetivo de ser um meio de expressar linguagens de execução, como BPEL [30] e BPML [67], de uma maneira clara.

O foco da BPMN está em modelar processos de negócio. Portanto ela abstrai outros tipos de informações como: estrutura organizacional e recursos, modelo de dados e informações, estratégias e regras de negócio.

Esta notação pode ser utilizada para modelar três tipos de processos: internos, abstratos e colaborativos [54]. Os processos internos (ou privados) são aqueles definidos internamente por uma empresa ou organização. Por exemplo: um processo de locação de dvds em uma locadora, ou uma solicitação de férias a um departamento de recursos humanos. Os processos internos são executáveis (aqueles que podem ser executados por uma máquina de orquestração) ou não executáveis.

Os processos abstratos (ou públicos) são aqueles que interagem com outros processos internos ou com participantes que são externos aos processos internos. Consequentemente, apenas parte do processo é definida, já que a parte externa do processo é abstraída. Um exemplo deste tipo de processo é uma solicitação de compra de material de consumo a um fornecedor. A fase da solicitação é um processo interno, mas o processamento do pedido pelo fornecedor é abstraído.

Processos colaborativos (ou globais) representam a interação de duas ou mais entidades de negócio. O analista que modela este tipo de processo foca apenas nas interações entre as entidades, ou seja, o modelador se preocupa em deixar claro apenas as mensagens entre as entidades envolvidas. Para isso ele abstrai seus respectivos processos internos. Um exemplo deste tipo de processo é um pedido de solicitação de uma empresa desconhecida a um fornecedor, o analista que especifica o processo explicita apenas as mensagens trocadas entre a empresa e o fornecedor.

2.3.2 Linguagens para Transição, Execução e Diagnóstico de Processo

A Transição é uma fase do ciclo de vida do processo que possui dois objetivos bem definidos: traduzir a linguagem gráfica para uma linguagem de execução; e trocar

modelos de processo de negócio entre diferentes BPMS. Existem duas linguagens proeminentes para realizar a tradução das linguagens: *Business Process Definition Metamodel* (BPDM) e *XML Process Definition Language* (XPDL) [33].

BPDM foi proposta pela OMG e define um metamodelo de processos de negócio, porém não especifica uma notação gráfica para representá-lo. BPDM provê suporte à representação de processos de négocio independentes da notação ou metodologia. Entretanto, este metamodelo ainda não foi validado pela OMG. Ademais, ele possui alta complexidade, e não apresenta aplicações que o utilizam [33].

A alternativa para transição entre linguagens é a utilização da XPDL. Esta norma foi criada pela *Workflow Management Coalition* (WfMC) e é utilizada por um grupo de ferramentas BPMS na conversão entre linguagens.

Esta conversão é importante, pois as linguagens que são executadas pelas máquinas de orquestração são orientadas a blocos, enquanto as linguagens gráficas são orientadas a grafos [35]. Linguagens orientadas a grafos representam a progressão temporal e a lógica de fluxo através de nós e conexões entre eles, enquanto que linguagens orientadas a blocos controlam o fluxo aninhando primitivas de algoritmos estruturados. Este é um dos principais motivos da perda de informação durante a conversão entre as linguagens.

Portanto, mesmo utilizando uma norma para fazer a transição entre as linguagens, ocorre a perda de informação [33]. Além disso, fazer a tradução da linguagem gráfica diretamente para linguagem de execução é mais fácil do que converter da linguagem de execução para a gráfica. Ademais, já existem BPMS que podem converter seu modelo de processo em linguagem de execução.

As normas de execução permitem que os processos modelados sejam executados por máquinas de execução nas ferramentas BPMS. Duas linguagens de execução se destacam no mercado: *Business Process Modeling Language* (BPML) e *Business Process Execution Language* (BPEL). Das duas, BPEL é mais amplamente adotada, apesar de BPML possuir uma semântica clara [33].

BPEL é um padrão proposto pela OASIS (*Organization for the Advancement of Structured Information Standards*). Ela especifica ações para serem executadas por *web services* e permite especificar um processo de forma similar a um algoritmo. Ela provê suporte a estruturas de repetição e condicionais, além de permitir a atribuição de valores a variáveis. Portanto, quando a notação gráfica é convertida em uma linguagem de execução, o processo é transformado em um tipo de algoritmo que é executado através da máquina de execução de um *web service*.

BPML é um padrão basedo em XML desenvolvido pela BPMI para representação de processos e sua semântica de execução [24]. Assim como BPEL, BPML foi desenvolvida para ser executada em *web services*. A linguagem é orientada a bloco e tem suas raízes em Pi-calculus [24]. Apesar de possuir várias vantagens em relação a BPEL,

esta linguagem não é mais apoiada pela BPMI que, ao fundir-se com a OMG, parou seu desenvolvimento.

As normas de diagnóstico orientam a administração e o monitoramento dos processos. As linguagens de diagnóstico permitem consultar o processo e auditá-lo, identificando os caminhos críticos do processo, os pontos nos quais ele pode ser otimizado e os pontos deficientes que podem causar atraso na execução do processo. Alguns exemplos deste tipo de linguagem são: *Business Process Runtime Interface* (BPRI) [56] e *Business Process Query Language* (BPQL) [55].

Este tipo de norma não será discutido, pois isto foge do escopo desta dissertação, haja vista que, em princípio, este trabalho tem a finalidade de apoiar apenas as fases de definição, configuração e execução de processos do ciclo da BPM.

2.4 Máquinas de Execução

A **máquina de execução de processos** ou **máquina de orquestração** é o motor do sistema gerenciador de processos. Ela é responsável por instanciar o processo, executar sua instância, distribuir suas atividades entre os usuários, invocar outras aplicações para o cumprimento das tarefas, manter o histórico da execução da instância e gerenciar as exceções que ocorrerem durante a execução da instância do processo.

Em princípio, o modelo do processo representa apenas a definição do processo, ou seja, uma abstração do processo concreto. Para que esta definição seja executada pela máquina de orquestração, é necessário que o processo seja instanciado. A instância do processo representa uma ocorrência real de um determinado processo. Ela descreve a sequência das atividades de uma configuração específica do modelo de um processo e não apresenta qualquer abstração ou generalização [48].

Um modelo de processo pode dar origem a várias instâncias, com diferentes configurações. Por exemplo, a Figura 2.3 representa o modelo de venda de um produto.

O cliente informa os produtos que deseja comprar. A atendente registra os produtos. O cliente informa a forma de pagamento. Caso a forma de pagamento seja dinheiro, então a atendente registra o valor pago e devolve o troco para o cliente. Se a forma for cartão, ela cadastra o número do cartão do cliente e, finalmente, se for escolhido cheque, a atendente registra o nome e telefone do cliente que passou o cheque.

Nota-se que nesta descrição não estão definidos quem é a atendente ou o cliente, nem os produtos comprados pelo cliente. Sendo assim, este modelo pode ser usado para vender qualquer tipo de produto para qualquer cliente por qualquer profissional de vendas. Estas informações serão definidas assim que o processo for instanciado e, algumas delas, durante a sua execução, como por exemplo os produtos que o cliente deseja comprar.

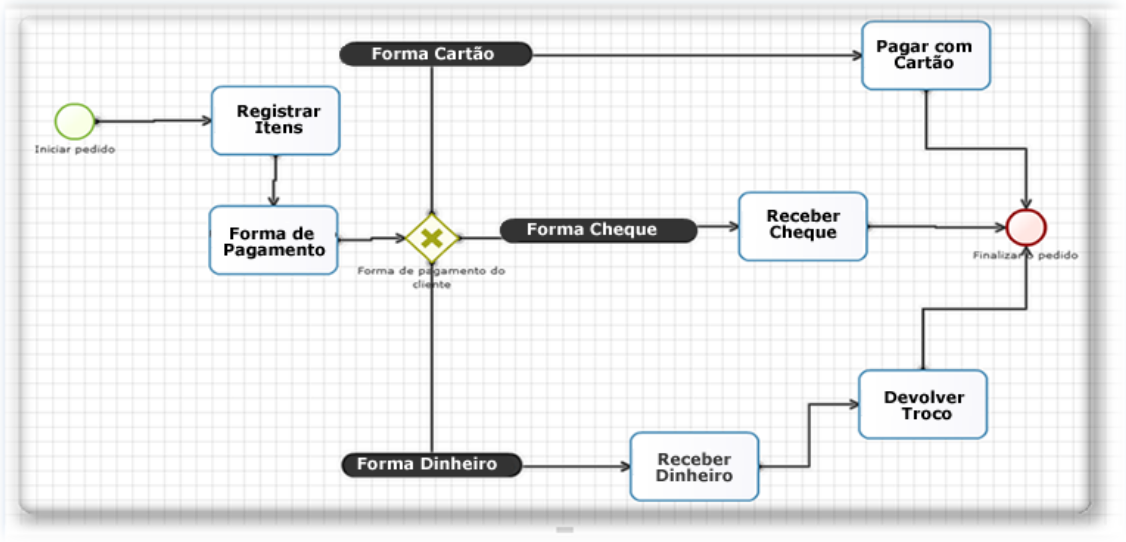


Figura 2.3: *Modelo de Processo de Vendas*

O modelo do processo deve conter todas as informações necessárias sobre o processo para que ele possa ser executado pela máquina de execução. Isso inclui informações sobre o seu início e término, as atividades e as regras para navegar entre elas, as tarefas a serem alocadas para os usuários executarem e, as referências às aplicações que podem ser invocadas [25].

A máquina de execução é responsável por executar a instância do processo criando um ambiente, em tempo de execução, no qual a instância é executada. É responsabilidade do mecanismo de execução [25]:

- Interpretar o modelo do processo;
- Controlar a criação, ativação, suspensão e finalização da instância do processo;
- Navegar entre as atividades do processo, as quais podem envolver operações sequenciais ou paralelas e interpretação de dados relevantes para a instância do processo;
- Adicionar e remover participantes da instância do processo;
- Alocar tarefas para participantes envolvidos na execução da instância;
- Inovar aplicações externas ao sistema gerenciador de processos de negócio;
- Manter dados da execução do processo, possibilitando a transferência destes dados para outras aplicações;
- Supervisionar com objetivo de administrar, controlar e auditar.

A máquina de execução deve ser capaz de interpretar o modelo do processo para executá-lo. Portanto, ela necessita conhecer a linguagem na qual o processo foi modelado ou uma linguagem de execução na qual o modelo do processo pode ser convertido. A máquina deve ser capaz de ler esta linguagem e, a partir dela, executar a instância do processo.

A máquina de execução é responsável por gerenciar todas as instâncias de processos. Isso envolve sua criação, quando o modelo do processo é instanciado. Após a instanciação do processo, ele não é imediatamente executado. A instância fica aguardando sua ativação. Assim que ela é ativada, a máquina inicia sua execução. Durante o andamento do processo ele pode ser suspenso, ou seja, a máquina de execução paralisa sua execução e a instância do processo fica aguardando ser reativada. Finalmente, após a execução de todo fluxo do processo, a máquina de execução deve ser capaz de finalizar a instância do processo e terminar sua execução. A Figura 2.4 mostra os estados pelos quais a instância do processo transita.

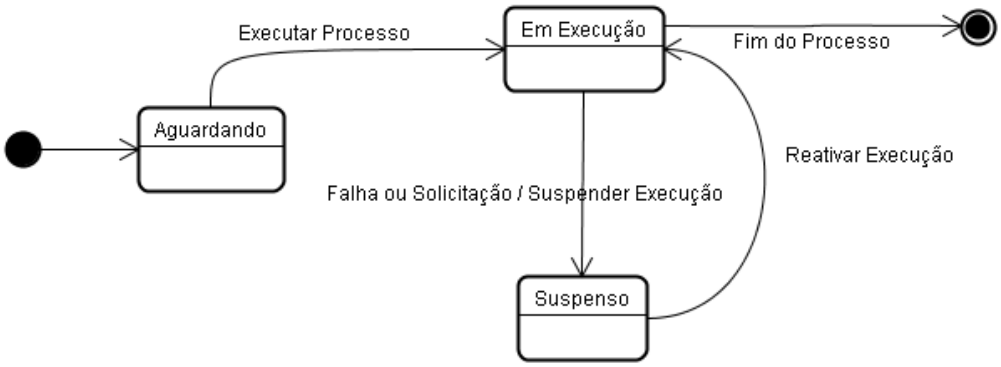


Figura 2.4: *Estados da Execução da Instância do Processo*

Ao executar o processo, a máquina gerencia o fluxo do processo descrito em sua definição. Sendo assim, ela navega entre os elementos do processo através das associações entre eles. A máquina deve ser capaz de interpretar todos os elementos expressos no modelo de processo. Isso significa que ela deve prover suporte à execução de atividades (em paralelo e sequenciais) a pontos de decisão nos quais somente um caminho do processo é executado, e outras características que podem ser expressas através da linguagem de processo.

Além de interpretar o modelo do processo, a máquina deve ser capaz de interpretar os dados enviados por outras aplicações e utilizá-los durante a execução do processo. Estes dados podem ser utilizados para determinar quais caminhos do processo devem ser seguidos.

A máquina de execução deve apoiar a adição e remoção de participantes à instância do processo em tempo de execução. Ou seja, um participante pode ser retirado da execução de um processo, assim como outros participantes podem ser adicionados a ela durante a sua execução.

Para que as atividades pertencentes ao processo sejam executadas, a máquina de orquestração deve ser capaz de definir quais participantes irão executar as atividades do processo. Esta definição de responsabilidade pode ser feita baseada no papel que o participante assume para determinada instância do processo.

Cada participante possui uma lista de atividades. Esta lista apresenta as atividades alocadas para o participante. A máquina de execução utiliza estas listas para definir quais tarefas estão pendentes para cada usuário. Nelas também podem constar atividades que já foram executadas pelo participante. Após selecionar uma tarefa, o usuário pode executá-la para o caso específico da instância do processo que a alocou.

Durante a execução das atividades a máquina de execução deve ser capaz de invocar outros aplicativos que sejam necessários no cumprimento destas atividades. Tais aplicativos podem envolver editores de textos, formulários de dados, planilhas eletrônicas, e-mails e qualquer outro tipo de sistema de informação.

No andamento do processo é importante que a máquina possua algum mecanismo que trate os erros que porventura aconteçam durante do processo. Algumas dessas falhas são devidas a erros na comunicação com o sistema gerenciador do banco de dados, falta de participantes que possuam as habilidades necessárias para cumprirem determinada tarefa, erros no fluxo do processo e assim por diante. Este mecanismo deve paralisar o andamento do processo, caso ocorra alguma falha, e informar ao seu administrador a falha ocorrida. Além disso, o mecanismo deve estar apto a auxiliar o responsável pelo processo a resolver o problema.

As tarefas apresentadas pela lista de trabalho devem conter uma descrição da tarefa a ser realizada pelo usuário. Esta descrição deve guiar o usuário envolvido na tarefa em seu cumprimento. Além disso, em alguns casos é interessante que ela possibilite ao usuário recusar a tarefa, ou passá-la para outro usuário.

2.5 Padrões de Processo

Sistemas gerenciadores de processos de negócio possuem funcionalidades diversificadas e diferentes níveis de poder de expressão, isto é, diferentes capacidades de representar as situações existentes nos processos de negócio.

Situações recorrentes durante a modelagem de um processo de negócio permitem identificar um conjunto de estruturas que frequentemente ocorrem durante esta modelagem. Estas estruturas foram denominadas em [73, 60] de **Padrões de Processo**.

O primeiro estudo sobre estes padrões foi publicado em [73] e identificou vinte situações que se repetem durante a modelagem de um processo de negócio. Três anos mais tardes outros vinte e dois padrões de processo foram identificados [60].

Estes padrões delineiam as principais estruturas existentes em um modelo de processo e possuem um nível de detalhamento e abstratação que forma uma conjunto de estruturas capazes de avaliar linguagens de modelagem de processo [60].

Consequentemente, os padrões de processo são amplamente usados para analisar o poder de expressão das linguagens de modelagem de processos, assim como podem

servir de base para criação de formas de expressão de processos de negócio [62, 78, 77, 29, 31, 76, 68, 69, 1].

Os padrões de processo são divididos em quatro categorias: dados, recursos, tratamento de exceção e controle de fluxo [69]. Os padrões relacionados a dados definem a maneira como os dados devem ser manipulados no modelo de processo. Os padrões de recurso definem a maneira de alocação de recursos dentro do processo, ou seja, como definir os participantes para executar as atividades. Os padrões de tratamento de exceção cuidam das exceções que ocorrem durante o andamento do processo. Finalmente os padrões de fluxo determinam as situações recorrentes nos fluxos de processos.

Os padrões de fluxo são subdivididos em sete categorias: controle de fluxo, sincronização e ramificação, múltipla instância, baseados em estado, cancelamento e forçar conclusão, iteração, terminação e gatilho [60]. Estes padrões estão descritos com mais detalhes no Apêndice B.

CAPÍTULO 3

Metamodelo de Processo

Este capítulo apresenta um novo metamodelo de processo, desenvolvido a partir de conceitos do BPMN [54] e de características essenciais para modelagem de processo de negócio que foram identificadas através da análise dos padrões de processos propostos em [73, 60].

O capítulo esclarece os conceitos do metamodelo e compara esses conceitos com a notação que serviu de base para sua construção.

3.1 Visão Geral do Metamodelo

Um metamodelo é uma especificação completa e precisa de uma Linguagem de Modelagem Específica de Domínio (DSML), que permite definir os modelos suportados pelo domínio [2]. Assim, o metamodelo de processo proposto neste trabalho é uma linguagem de modelagem específica para o domínio de processo de negócio, através do qual é possível expressar modelos de processo de negócio.

O metamodelo proposto por este trabalho sofreu forte influência dos padrões de processo estruturais, descritos no Apêndice B. Foram considerados dezenove padrões descritos em [73] e quatro padrões propostos por Russel [60].

A Figura 3.1 ilustra este metamodelo, cujo conceito principal é **Processo**. Um processo é uma atividade bem definida na qual **Objetos de Fluxo** são ligados entre si através de **Conexões** e **Anexos**.

Objetos de Fluxo são os elementos do processo responsáveis por determinar seu comportamento. Estes elementos podem se associar de duas maneiras: através de uma **Conexão** ou de um **Anexo**. A **Conexão** associa dois **Objetos de Fluxo** formando um fluxo entre eles, enquanto que o **Anexo** relaciona um evento a uma **Atividade** com objetivo de tratamento de erros. Os Objetos de Fluxo são categorizados em: **Eventos**, **Portas** e **Atividades**.

Os Objetos de Fluxo do tipo **Evento** definem os acontecimentos que podem ocorrer durante o fluxo do processo como, por exemplo, o recebimento de uma mensagem, o fim de um ciclo de tempo, um sinal de finalização de processo, entre outros. Os erros

tratados pelo **Anexo** podem ocorrer durante o fluxo do processo, fazendo com que este fluxo sofra alterações de acordo com o tipo de erro.

Os Objetos de Fluxo do tipo **Porta** determinam tomadas de decisão (bifurcações e convergências) no fluxo do processo, ou seja, permitem mudanças no andamento do fluxo baseando-se na avaliação de dados ou de eventos, além de permitir a execução de fluxos em paralelo e o sincronismo do fluxo em um determinado ponto.

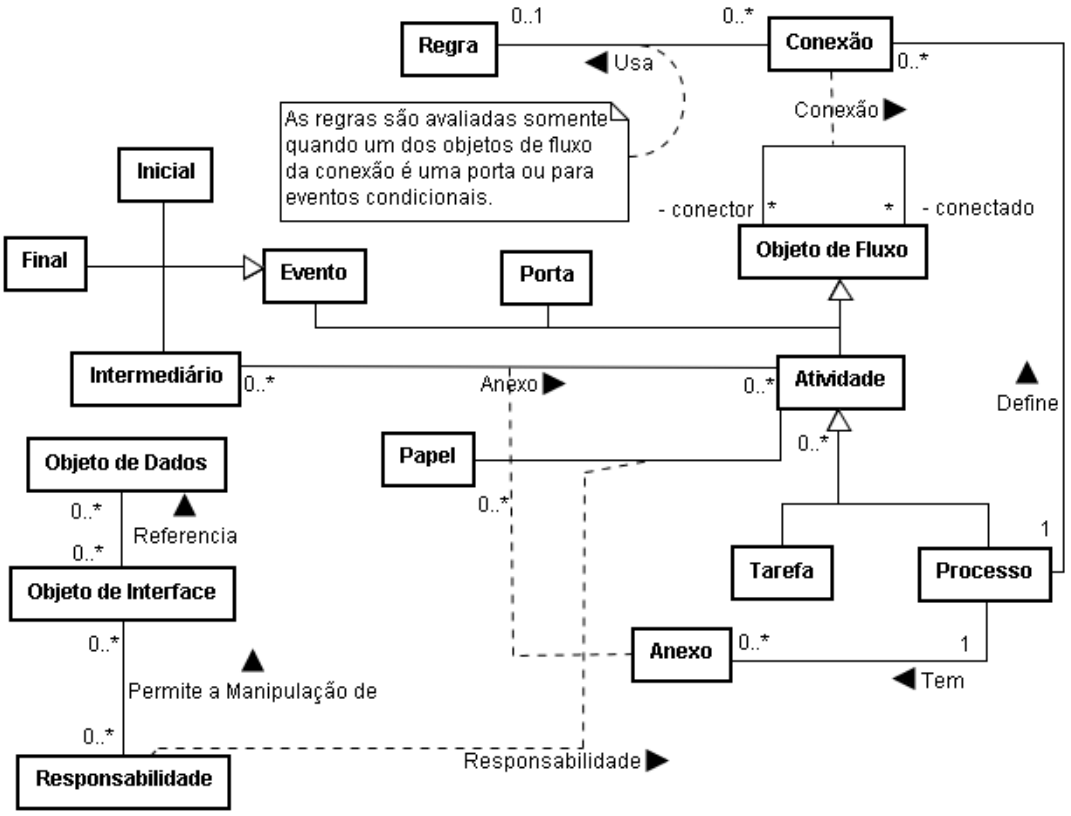


Figura 3.1: *O Metamodelo de Processos de Negócio*

Os Objetos de Fluxo do tipo **Atividade** determinam o que é feito no processo. Atividades podem ser atômicas (**Tarefas**) ou não atômicas (**Processos**). Para cada atividade são definidos um ou mais **Papéis** que determinam quais participantes estão aptos a serem alocados para a realização da tarefa. A associação entre atividade e papel gera uma **Responsabilidade** que determina a maneira do participante interagir com a atividade.

Para cada responsabilidade atribuída, **Objetos de Interface** podem ser associados. Estes objeto representam os serviços (disponibilizados em *webservices*, por exemplo) responsáveis por fornecerem as interfaces com as quais os participantes alocados interagem para completar a tarefa.

Os dados relacionados ao domínio do negócio são representados através dos **Objetos de Dados**. Esses objetos são referenciados nos **Objetos de Interface** e definem os dados do negócio que podem ser referenciados na execução do processo. Estes dados

podem influenciar diretamente no fluxo do processo. Eles podem ser usados, por exemplo, para verificação de **Regras** e para processamento de eventos.

Uma **Regra** é um elemento do processo que define uma expressão condicional que pode ser usada nas **Conexões**. As portas utilizam as conexões ligadas às regras com a finalidade de controlar o fluxo de cada processo. Por exemplo, quando o fluxo de execução chega a uma porta, as conexões que tiverem suas regras avaliadas como verdadeiras determinam o caminho que será seguido pelo fluxo de execução do processo.

3.1.1 Atividades, Tarefas e Processos

As atividades definem o que é realizado durante o fluxo do processo. Cada atividade representa uma ação, detalhada ou não, que pode ser executada por um ou mais participantes do sistema, ou pelo próprio sistema, caso seja uma atividade que não envolva participantes. Toda atividade possui um nome e uma descrição, além de ser categorizada em dois tipos distintos: processo e tarefa. Ademais, uma atividade pode ser definida como *múltipla-instância*, indicando que ela pode ser repetida múltiplas vezes assim que iniciar sua execução.

O nome da atividade é define sua função e a descrição discorre sobre o que é feito para realizá-la. Por exemplo, considerando o contexto de desenvolvimento de *software*, o nome de uma atividade poderia ser “Eliciação de Requisitos” e sua descrição poderia ser: “Identificar os requisitos do *software* a partir de interações com os *stakeholders*”.

Caso o criador (modelador) da atividade considere que apenas estas informações (nome e descrição) sejam suficientes para que a ação seja compreendida e realizada por seus participantes, então esta atividade é uma tarefa atômica.

Entretanto, se o modelador entende que esta descrição não é suficiente para que os participantes a executem, então ele poder detalhar esta atividade como um processo.

O detalhamento é feito através da adição de Objetos de Fluxo ao contexto da atividade. No exemplo da atividade de “Eliciação de Requisitos”, o modelador criaria outras atividades, como identificação de fonte de requisitos, classificação de requisitos, resolução de conflitos, definição das prioridades e validação de requisitos. Além disso, poderia ser elaborada uma sequência de fluxo de execução para essas atividades.

Neste trabalho, uma atividade detalhada em outras atividades é considerada um Processo. No Processo são definidas Conexões que ligam Objetos de Fluxo, determinando a sequência de execução do processo. Um processo deve indicar, ainda, o papel que qualifica o responsável pelo processo.

Um processo é caracterizado como *ad hoc* se possuir atividades bem definidas e fluxo desconhecido [54]. Dessa maneira, em um processo *ad hoc* são informadas apenas as atividades que o compõem, mas a sequência de execução do processo é determinada

durante a sua execução. No processo não *ad hoc* (também conhecido como processo **definido**) tanto as atividades quanto seu fluxo estão bem determinados em tempo de definição do processo. O fluxo deste tipo de processo é executado com base na sequência pré-estabelecida na definição do processo.

Vale ressaltar que um Objeto de Fluxo (OF) pode ser um processo, de forma que é possível reutilizar processos já definidos em outros fluxos de processo. Até mesmo o próprio processo pode ser utilizado em seu fluxo, criando o conceito de *recursividade*.

A Tarefa é a menor unidade executável de um processo. Para se criar uma tarefa é necessário definir seu tipo e os dados necessários para que a máquina de execução aloque corretamente os participantes aptos a realizarem tal tarefa.

O tipo da tarefa determina a forma de executá-la. Estão previstos no modelo cinco formas de realizar uma tarefa que são organizadas nas seguintes categorias de tarefa:

- As tarefas da categoria Usuário são executadas através da interação entre o(s) participante(s) responsável(is) pela tarefa com as interfaces correspondentes aos objetos de interface. Um exemplo deste tipo de tarefa é o preenchimento de uma solicitação eletrônica de férias, em um sistema de administração de recursos humanos
- As tarefas da categoria Manual são executadas sem o apoio de um sistema de gerência de processos. Por exemplo, levar um documento a um departamento, ou assinar um documento em papel, são tarefas manuais. Portanto, para realizar esta tarefa o responsável simplesmente indica que a tarefa foi concluída e adiciona uma descrição do que foi feito para executá-la.
- Tarefas da categoria Script não necessitam de participantes e são realizadas pelo próprio sistema de gerência de processos. Para efetuar este tipo de tarefa o sistema executa um script codificado pelo modelador que cria a tarefa. Exemplos deste tipo de tarefa incluem executar um backup dos dados, ou ajustar hora do servidor.
- As tarefas da categoria Envio de mensagem envolvem a interação do usuário com algum tipo de software de envio de mensagem; Igualmente as tarefas do tipo Recebimento de mensagem aguardam a chegada de alguma mensagem.

Certas atividades precisam ser executadas múltiplas vezes. Um exemplo é uma tarefa de aplicação de questionários, na qual vários participantes precisam preencher o mesmo formulário. Visando prover suporte a este tipo de situação, uma atividade pode ser definida como múltipla-instância.

Para executar atividades múltipla-instância é preciso conhecer o número de vezes que elas devem ser executadas. Este número pode ser definido antes ou durante o fluxo do processo e, algumas vezes, ele pode ser redefinido durante a execução da própria atividade. Sendo assim, para criar este tipo de atividade é preciso informar a forma como

as instâncias da atividade serão criadas. Além disso, deve ser definido se as instâncias podem ocorrer de forma paralela ou sequencial. Quando sua forma de criação é em paralelo, todas as instâncias da atividade são criadas ao mesmo tempo, enquanto se a forma de criação for sequencial então uma nova instância da atividade só poderá ser criada caso a instância em execução seja finalizada.

A representação gráfica para diversos tipos de Processos e Tarefas é apresentada na Figura 3.2.

Figura 3.2: *Tipos de Atividade: Processos e Tarefas*

O símbolo de um quadrado com uma cruz no centro determina que a atividade é um processo, já os três traços na vertical aparecem quando a atividade é definida como multipla-instância e finalmente o símbolo “~” caracteriza a atividade como *adhoc*.

3.1.2 Eventos

Um evento é um fenômeno que ocorre enquanto um processo está em execução. Eventos têm o poder de influenciar o fluxo de execução do processo, alterando-o, atrasando-o, ou causando outros tipos de interferência, tal como o envio de um sinal. Os eventos são divididos em três tipos: Inicial, Intermediário e Final. Cada evento possui um gatilho, que determina a maneira como o evento é acionado. Todo evento deve ser categorizado como receptor ou lançador [54].

Quando um **evento receptor** é acionado, ele fica aguardando até que algo aconteça para que o fluxo do processo continue. Por exemplo, um Evento Inicial de Tempo fica aguardando até que seu ciclo de tempo seja concluído para dar continuidade ao fluxo do processo.

Já os **eventos lançadores** são responsáveis por disparar eventos receptores ou executarem ações. Um exemplo é o Evento de Sinal que, quando acionado, dispara seu código para todos os Eventos de Sinal receptores. Assim todos os receptores que tiverem o mesmo código do evento lançador serão disparados.

Eventos Iniciais

Para que um processo tenha início, é necessário que ele tenha pelo menos um Evento Inícial, a menos que o processo seja do tipo *ad hoc*. Portanto, os Eventos Iniciais são responsáveis por começarem o fluxo do processo. Isso implica que não haverá qualquer fluxo saindo de um OF e chegando neste tipo de evento.

Os Eventos Iniciais possuem vários tipos de gatilhos que podem ser usados para iniciar um processo. Estes gatilhos funcionam como receptores, aguardando que algo aconteça para que sejam disparados. O metamodelo define cinco maneiras distintas para acionar um evento inicial.

- O Evento Padrão é disparado assim que o processo começa a ser executado.
- O Evento de Tempo é acionado com base em um dado de tempo definido pelo usuário. Este tempo pode ser uma data, (por exemplo, dia 25/08/2009) ou um período, como (por exemplo, todos os dias ou semana às 12:00 horas ou às 17:00 horas).
- O Evento de Mensagem é disparado quando determinada mensagem chega a um processo. Esta mensagem pode ser enviada através de uma tarefa do tipo envio de mensagem ou de um evento de envio de mensagem.
- O Evento Condicional é disparado assim que um conjunto de condições seja satisfeito. Alguns exemplos de condições que podem ser especificadas são: assim que o estoque atingir o nível mínimo, quando determinado documento for alterado, assim que determinada instância do processo for concluída. Portanto, quando a condição do evento for verdadeira, o processo terá início.
- O Evento de Sinal inicial funciona como um observador. Quando um Evento de Sinal Lançador é acionado ele emite um aviso, contendo o código do sinal, para todos os processos que estiverem sendo executados. Todos os eventos de sinal receptores interceptam o código lançado e verificam se possuem o mesmo código. Caso o evento possua o mesmo código, então ele é acionado.

A figura 3.3(a) ilustra graficamente os tipos de eventos iniciais.

Eventos Intermediários

Os Eventos Intermediários acontecem após o início e antes do final do processo. Estes eventos podem funcionar como lançadores ou como receptores. Como lançadores os Eventos Intermediários são:

- O Evento Intermediário de Mensagem lança uma mensagem para um ou mais participantes do sistema.

- O Evento Intermediário do tipo Sinal envia um sinal com seu código para todas as instâncias de processo em execução e qualquer evento receptor do tipo sinal que tenha o mesmo código será acionado.

Como receptores, os Eventos Intermediários são:

- Evento Intermediário de Mensagem: assim que o fluxo do processo chega a ele, o processo fica em estado de espera até que um participante envie uma mensagem para o evento. Quando esta mensagem chega ao evento, o fluxo do processo continua.
- Evento de Tempo: causa um atraso na execução do processo. Quando o fluxo atinge um Evento Intermediário de Tempo, ele fica aguardando até que o tempo estabelecido neste evento seja esgotado. Este tempo, assim como ocorre no evento Inicial, pode ser um ciclo ou uma data.
- Evento Condicional: assim que o fluxo chega a ele, o processo fica aguardando até que a expressão do evento seja avaliada como verdadeira. Somente depois que a expressão se torna verdadeira o fluxo do processo continua.
- O Evento Intermediário de Sinal, assim como o Evento Inicial de Sinal, fica aguardando que outro evento de sinal do tipo lançador mande um aviso com o mesmo código para que o evento seja acionado.
- O Evento Intermediário de Erro fica observando o processo para verificar se algum evento de erro foi lançado. Caso o evento de erro seja lançado, então o evento intermediário anexado ao processo o intercepta, o processo é cancelado e seu fluxo é desviado para o OF que está ligado ao evento intermediário.

Durante a execução dos processos de negócio podem ocorrer exceções no fluxo do processo, como erros lançados por eventos de erro. Este tipo de exceção deve ser tratado no momento em que acontece, pois pode exigir o desvio do fluxo do processo. Para isto foi criado o conceito de Anexo, que é um evento intermediário que fica agregado ao processo. Este evento é ligado a outro OF através de uma conexão. O evento Intermediário de erro funciona apenas quando anexado a um processo.

A figura 3.3(b) mostra a representação gráfica de eventos intermediários.

Eventos Finais

Os Eventos Finais acontecem no final de cada fluxo pertencente ao processo. Este tipo de evento funciona apenas como lançador, ou seja, eles podem ativar outros eventos como receptores do tipo Inicial e Intermediário. Foram determinados seis tipos de eventos finais:

- Evento Final Padrão: termina o processo silenciosamente e não ativa qualquer outro tipo de evento.

- Evento Final de Mensagem: finaliza o processo e envia uma mensagem para um ou mais participantes ou para outras instâncias de processo que estão em execução sobre a conclusão do processo.
- Evento Final de Erro: indica que ocorreu um erro no processo. Assim que este evento é disparado, um código de erro é lançado pelo processo. Este código pode ser interceptado e tratado por um Evento Intermediário de recepção anexado ao processo que gerou o erro.
- Evento Final de Sinal: lança seu código no sistema e pode ser interceptado por um ou mais eventos intermediários e iniciais. Assim que estes eventos capturam o sinal emitido, eles são disparados, caso possuam o mesmo código do evento que foi lançado.
- O Evento Final Terminal finaliza todos os fluxos do processo que estiverem em execução. Assim que um fluxo do processo atinge este evento, todos os seus fluxos, inclusive o fluxo de seus processos internos, são finalizados.

A Figura 3.3(c) apresenta a representação gráfica dos tipos de eventos finais.

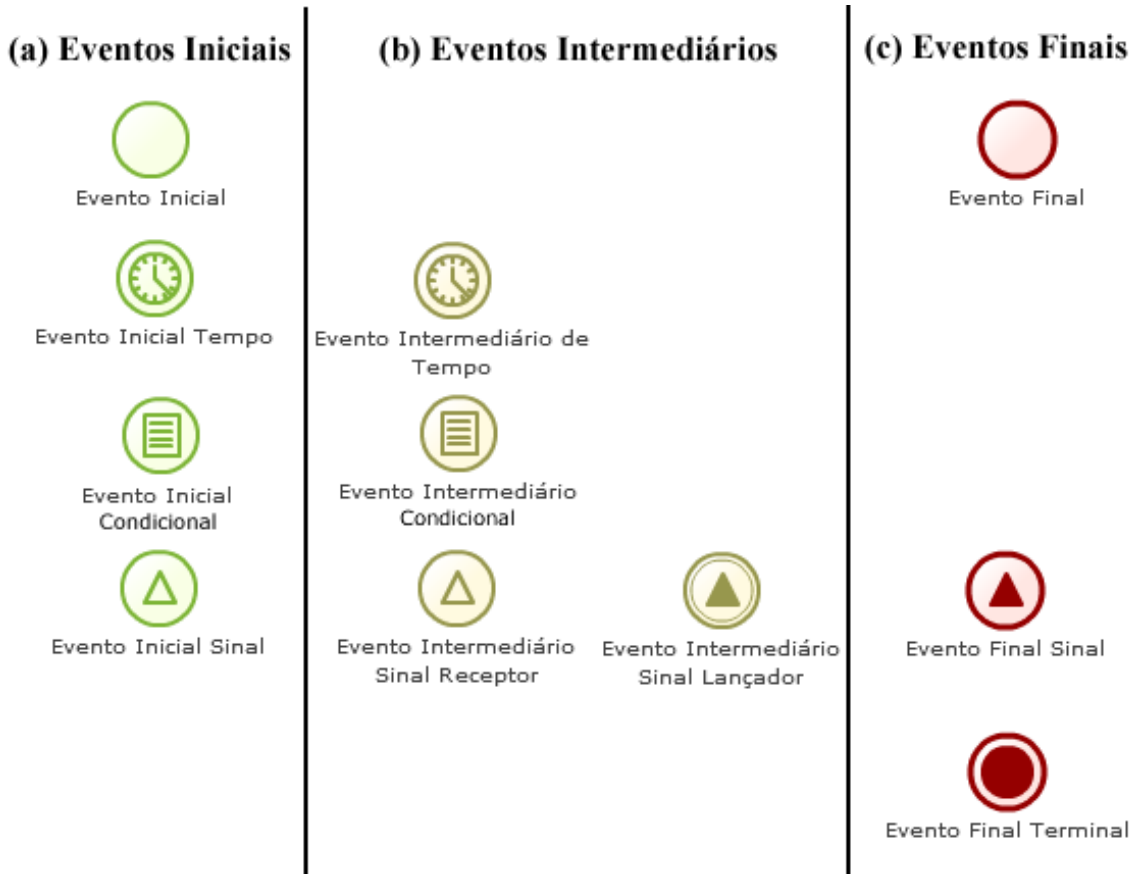


Figura 3.3: *Tipos de Eventos*

3.1.3 Portas

Muitas vezes, enquanto um processo é executado, faz-se necessário desviar seu fluxo em função dos dados informados pelo usuário. Da mesma forma, é importante poder executar tarefas em paralelo e sincronizar o fluxo do processo. As portas são os Objetos de Fluxo utilizados para viabilizar estes tipos de comportamento. Elas são responsáveis por controlar a sequência do fluxo, convergindo e desviando o fluxo.

O metamodelo provê suporte a cinco tipos de porta: Exclusiva, Inclusiva, Complexa, Paralela e de Evento. O tipo da porta determina a maneira como o fluxo é tratado e possui uma representação diferenciada na notação gráfica utilizada neste trabalho, como ilustra a Figura 3.4.

Figura 3.4: *Tipos de Porta*

A **Porta Exclusiva** é utilizada para desviar o fluxo do processo com base nos dados do próprio processo, ou nos dados inseridos através de objetos de interface. Portanto, toda vez que um processo tiver mais de um caminho possível e somente um dos caminhos for passível de escolha, então uma porta exclusiva deve ser utilizada.

Para prover este tipo de comportamento, toda vez que o fluxo do processo chega a uma Porta Exclusiva, as regras das conexões nas quais a porta faz o papel de conector são avaliadas até que se encontre uma conexão que retorne valor verdadeiro. Então esta conexão é selecionada para continuar o fluxo do processo. As demais conexões que ainda não foram avaliadas são ignoradas.

Existe também a possibilidade de nenhuma conexão ter sua regra avaliada como verdadeira. Para estes casos o modelador do processo pode criar uma Conexão Padrão que garante que, caso nenhuma regra de conexão seja verdadeira, o fluxo seguirá através da conexão marcada como padrão. É importante salientar que toda porta deverá possuir pelo menos uma conexão padrão. Isso evita que o fluxo do processo fique retido em uma porta exclusiva.

A **Porta Inclusiva** também toma a decisão de qual fluxo seguir com base nos dados inseridos através dos objetos de interface ou nos dados do próprio domínio do sistema. Porém, diferentemente da Porta Exclusiva, a Porta Inclusiva permite que mais de uma conexão seja selecionada para continuar o fluxo. Outra diferença é que a Porta Inclusiva aguarda até que todos os fluxos que chegam até ela sejam executados para analisar suas conexões.

Portanto, somente após todos os fluxos que chegam a uma Porta Inclusiva serem concluídos as regras associadas às conexões que saem da porta são analisadas. Cada conexão cuja regra retornar o valor verdadeiro será selecionada para dar continuidade ao fluxo do processo. Caso nenhuma regra seja verdadeira, a Conexão Padrão será selecionada para continuar o andamento do processo.

Vale ressaltar que este tipo de porta pode ser utilizada para sincronização dos fluxos do processo. Isso pode ser feito ligando todos os fluxos que precisam ser sincronizados a uma Porta Inclusiva e definindo apenas uma conexão padrão de saída da porta. Assim, somente quando todos os fluxos chegarem a esta porta, o fluxo de saída da porta será percorrido.

A **Porta Complexa** funciona de maneira customizável, diferentemente das outras portas. Assim como a Porta Inclusiva, ela aguarda uma determina quantidade de fluxos para dar continuidade ao andamento do processo. Entretanto, em vez de aguardar todos os fluxos em execução, a porta complexa espera somente uma quantidade de fluxos determinada pelo usuário. Este número de fluxos pode ser definido até mesmo em tempo de execução, com base nos dados do processo.

A quantidade de fluxos que saem deste tipo de porta também pode ser configurada. Em princípio, a cada vez que a porta sincroniza os fluxos entrantes, apenas um fluxo sai. Porém, ela poderá ser reorganizada para que, a cada sincronismo, mais de um fluxo seja executado.

A **Porta de Eventos**, assim como as Portas Inclusiva e Exclusiva, serve para tomada de decisão de qual caminho deve ser seguido. Porém, esta porta não se baseia nos dados do sistema ou nos dados inseridos através dos objetos de interface. Para determinar o fluxo que será seguido, esta porta avalia os eventos que estão ligados a ela. Estes eventos são verificados periodicamente e, assim que o primeiro evento for acionado, o fluxo ligado a ele será executado e os demais fluxos serão ignorados.

A **Porta Paralela** é diferente das demais portas, pois não toma decisões baseadas em dados. Ela apenas faz ramificação e sincronismo do fluxo do processo. Sendo assim, todos os fluxos que saem da Porta Paralela são necessariamente percorridos independentemente de qualquer estado do processo ou de seus dados. Isso possibilita a execução de fluxos em paralelo.

Outra funcionalidade da Porta Paralela é o sincronismo do fluxo. Para que os fluxos de saída sejam executados, ela aguarda até que todos os fluxos que chegam até ela sejam concluídos pelo menos uma vez. Ou seja, vários fluxos que chegam à porta podem ser executados várias vezes, entrentanto ela aguarda que todos sejam executados. Só depois da conclusão de cada fluxo a porta continua a execução do processo. Este comportamento pode ser utilizado para conversão de fluxos criados por outras portas paralelas.

3.1.4 Conexões e Regras

Para determinar o comportamento de um processo é necessário associar os Objetos de Fluxo formando uma sequência de fluxo que será seguida durante a execução do processo. Para ligar um OF a outro é utilizado o conceito de Conexão. Um dos OF exerce o papel de Conector e o outro faz o papel de Conectado em um determinado contexto de processo.

Portanto, uma conexão é formada por dois Objetos de Fluxo (conector e conectado). O Conector é o elemento do qual o fluxo sai enquanto o Conectado é o objeto no qual o fluxo chega. Assim, o fluxo segue do conector para o conectado. Além disso, toda conexão pertence a um único Processo, ou seja, a conexão é determinada no contexto do processo.

Embora a Conexão ligue dois Objetos de Fluxo, nem todos os objetos de fluxo podem conectar-se entre si. Um exemplo disso é o evento final; como ele termina o fluxo do processo, ele não pode fazer parte de uma conexão exercendo o papel de conector, ou seja ele é o último elemento do fluxo do processo.

A Figura 3.5 ilustra os fluxos possíveis entre Objetos de Fluxo. Vale notar que apenas portas permitem saídas múltiplas de fluxo (múltiplas conexões), ou seja, as portas são os únicos Objetos de Fluxo que permitem a ramificação do fluxo do processo.

Outra característica das conexões é que elas podem ser associadas a Regras. Estas Regras determinam se o fluxo do processo passa por aquela conexão ou não. Uma Regra é definida como um elemento independente de processo que possui um nome identificador, uma descrição e uma expressão condicional. As Regras são usadas em duas situações distintas: para verificar se um fluxo pode passar por uma conexão na qual seu conector é uma Porta Inclusiva ou Exclusiva; e para validar a condição de eventos do tipo condicional.

O conceito de Regra é importante na modelagem de processos, já que é essencial determinar o fluxo do processo com base em dados inseridos durante sua execução. O conceito de regra, associado a conexões relacionadas a portas de inclusão e exclusão, torna isso possível.

A validação da regra é feita antes que o fluxo passe pela conexão. Caso a regra seja avaliada como verdadeira, então o fluxo do processo irá passar pela conexão; caso contrário, o fluxo é retido. Vale ressaltar que as regras somente são avaliadas caso a conexão possua como conector um OF do tipo Porta Exclusiva ou Porta Inclusiva. Caso contrário, a regra não é avaliada e o fluxo segue pela conexão.

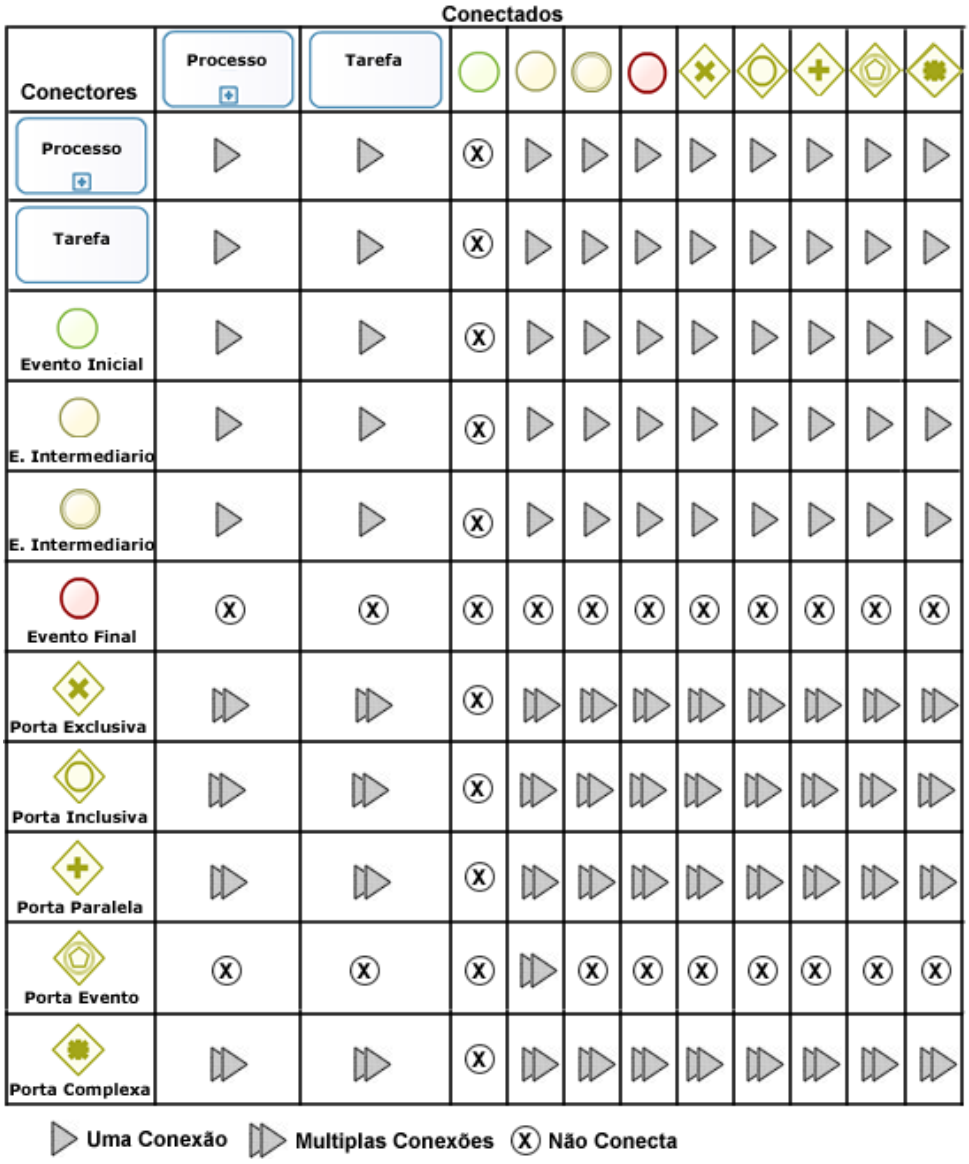


Figura 3.5: *Conectividade entre Objetos de Fluxo*

3.1.5 Objetos de Interface e Objetos de Dados

Para realizar tarefas da categoria Usuário, os envolvidos devem interagir com o sistema gerenciador de processos e, possivelmente, com outros sistemas. O **Objeto de Interface** é um elemento do processo que define as configurações necessárias para identificar e invocar a aplicação que deve ser apresentada ao usuário. O **Objeto de Dados** representa os dados manipulados por esta interface.

Para configurar o acesso às aplicações externas, o Objeto de Interface deve detalhar as formas de acessar a aplicação, de se autenticar nesta aplicação, e de identificar os dados do processo e do usuário que são intercambiados com essas aplicações.

O acesso a uma aplicação externa é dependente da tecnologia utilizada na criação da aplicação. Por exemplo, caso a aplicação seja desenvolvida na plataforma web então

uma provável forma de acesso é através de uma URL. Entretanto, se a aplicação funcionar localmente, sua chamada pode ser através da execução de um aplicativo. Portanto, é necessário que a aplicação externa informe o modo de acessá-la. Este modo deve ser definido pelo modelador responsável por configurar o Objeto de Interface.

Além disso, para que o usuário obtenha os devidos privilégios na aplicação externa, é preciso passar pela sua política de acesso, caso ela implemente uma. Portanto, o usuário deve ser autenticado na aplicação invocada, caso seja necessário. Logo, o Objeto de Interface deve possuir os dados para autenticação na aplicação externa.

Outro ponto importante é o envio dos dados do processo e do usuário à aplicação externa. Por exemplo, algumas vezes a aplicação invocada precisa averiguar qual processo está sendo executado ou qual o cargo de determinado usuário para montar a tela de interação. Consequentemente, é importante que o Objeto de Interface envie os dados do usuário e do processo à aplicação externa.

A aplicação externa pode exigir que outros dados específicos lhe sejam enviados. Logo, é necessário que o Objeto de Interface seja capaz de enviar estas informações adicionais à aplicação invocada. Por exemplo, a aplicação pode exigir que sejam enviados dados relacionados à forma da apresentação de suas telas.

Não menos importante que os dados do processo e do usuário são os dados relacionados aos objetos que devem ser manipulados pela aplicação invocada. Estes dados são controlados através dos Objetos de Dados, que referenciam o domínio da aplicação externa. O Objeto de Interface é responsável por agrupar estes dados e enviá-los à aplicação invocada. Estes dados podem ser utilizados pela aplicação chamada e, após seu processamento, devem ser preenchidos e reenviados por ela. Por exemplo, supondo que os dados esperados pelo software gerenciador de processo seja um nome e um telefone, então os metadados destes dados são enviados para a aplicação invocada. Assim que esta aplicação finalizar sua interação com o usuário, ou seja, assim que o nome e o telefone forem preenchidos pelo usuário, os dados informados são reenviados ao sistema gerenciador de processos.

3.1.6 Papéis e Responsabilidades

Para executar uma instância de processo, em geral, é necessário que os participantes da instância interajam com o sistema para executar suas tarefas e gerenciar os processos em execução. Entretanto, durante a modelagem do processo é inviável informar os participantes da tarefa ou os gerenciadores do processo, pois esses dados dependem da instância do processo e, ao modelar um processo, não se sabe quais serão os colaboradores disponíveis para participarem da execução da instância do processo.

Porém, durante o andamento do processo é importante que a máquina de execução consiga alocar os participantes para executarem as tarefas. Sendo assim, faz-se necessário determinar uma maneira de viabilizar a alocação de participantes em tempo de execução. Isto é feito através do conceito de Papel.

O Papel é um tipo de rótulo: ele pode ser um cargo (como analista de sistemas), um título (como o de mestre ou doutor), um tipo de função administrativa (como supervisor adjunto), entre outros. Durante o cadastramento dos usuários é possível determinar quais papéis eles podem assumir. Cada participante pode estar qualificado para um ou mais papéis.

Responsabilidades nas Tarefas

Para que tarefas dependentes de pessoas sejam realizadas é necessário determinar quais tipos de papéis são responsáveis por executá-las. Para isso foi empregada a metodologia de atribuição de responsabilidades proposta no COBIT (*Control Objectives for Information and related Technology*) [27], baseada nas melhores práticas de governança de processos na alocação de colaboradores para execução de tarefas.

O COBIT propõe uma matriz de responsabilidades que define os participantes e sua responsabilidade na execução da tarefa. As responsabilidades são vinculadas diretamente aos papéis e determinam a maneira como o participante interage com a tarefa.

Um participante pode se envolver com as tarefas de quatro formas diferentes: *responsável, aprovador, consultado e interessado*. Cada tipo de responsabilidade define uma maneira diferente do participante interagir com a tarefa.

Os responsáveis são incumbidos de executar a tarefa, realizando o que é proposto pelo tipo da tarefa, como por exemplo, interagir com outros sistemas (caso seja uma tarefa da categoria Usuário), realizar uma tarefa manualmente, enviar ou receber uma mensagem.

Os aprovadores verificam se a tarefa executada pelo responsável foi corretamente cumprida. Portanto, eles avaliam a tarefa realizada pelo responsável.

Os interessados recebem informações sobre o andamento da tarefa como, por exemplo, horário de início, tempo de duração, quem executou a tarefa, qual foi o parecer final dos aprovadores, entre outras informações.

Os consultados podem ajudar diretamente os responsáveis, esclarecendo dúvidas, ou fornecendo os dados necessários para execução e aprovação da tarefa.

Um papel pode estar relacionado com mais de uma responsabilidade. Isto é interessante, por exemplo, nos casos em que um papel define o executor e o mesmo papel determina o avaliador ou os consultados. Um exemplo disso pode ocorrer em uma tarefa de desenvolvimento de classes na qual um programador é responsável por escrever determinada classe e pode consultar outros programadores para ajudá-lo.

Responsabilidades nos Processos

A cadeia de responsabilidades também é utilizada na gerência de processos. Porém, as funções de cada responsabilidade na gerência de processos são diferentes das funções das responsabilidades empregadas na gerência de tarefas.

Na gerência de processos, os responsáveis são os administradores do processo. Eles podem interferir em seu fluxo, durante a execução, podem realocar tarefas para outros participantes, e devem resolver as pendências que apareçam.

Os aprovadores verificam se o processo foi executado corretamente, ou seja, se o objetivo do processo foi atingido.

Os consultados podem ser acionados pelos responsáveis pelo processo, com o objetivo de ajudar a atingir o objetivo do processo.

Os interessados recebem mensagens toda vez que há uma ocorrência significativa no processo, como, por exemplo, uma determinada tarefa é iniciada e finalizada, ou quando o processo fica pendente.

3.2 Comparação do Metamodelo Proposto com BPMN

Para expressar um metamodelo de processo foi necessário decidir qual notação seria utilizada. As notações BPMN e UML foram consideradas, e BPMN foi escolhida por ser considerada como um padrão para modelagem de processos [35].

Porém a BPMN é uma notação abrangente que visa modelar processos privados, públicos e colaborativos, enquanto que o objetivo da proposta desta dissertação é voltada para processos que possam ser executados por uma máquina de orquestração.

Consequentemente, BPMN foi tomada como base para a criação de um metamodelo capaz de expressar um modelo de processo. Este modelo deve conter apenas as características necessárias para sua execução, além de possuir um alto poder de expressão em relação a processos executáveis.

Para garantir que o metamodelo fosse dotado de um poder de expressão tão grande quanto o da notação que o originou, tomou-se como base para o desenvolvimento do metamodelo os vinte primeiros padrões propostos em [73].

Assim, o metamodelo proposto foi criado a partir de uma adaptação da notação BPMN. Este metamodelo, apresenta um conjunto reduzido, mas não necessariamente mínimo, de construções da BPMN capaz de representar processos de negócio em sistemas de informação. O metamodelo diminui a redundância, elimina algumas ambiguidades e apresenta outras mudanças em relação à notação utilizada como base. Estas modificações, que geraram este novo metamodelo, foram baseadas nos trabalhos de [50, 80, 81, 58, 18, 26].

Alguns exemplos de mudanças propostas pelo metamodelo de processo proposto são: a supressão de conceitos (por exemplo, Raias, Baias, Subprocesso, fluxo de mensagem, artefatos e eventos de *link*); a alteração na atribuição de responsabilidades nas tarefas; a eliminação de construções redundantes, tal como a indicação de laço na própria atividade; e a mudança no modo como os OFs se conectam.

A pesquisa descrita em [80] utiliza técnicas estatísticas para identificar os grupos de elementos de BPMN mais utilizados. Os pesquisadores coletaram diagramas que utilizavam a notação BPMN de sites de empresas, de fóruns práticos e similares, além de modelos utilizados em projetos de organizações e de seminários de ensino de BPMN. Estes diagramas foram analisados e os resultados identificaram que menos de 20% do vocabulário disponibilizado pela BPMN é utilizado nos modelos pesquisados. Ou seja, realmente há um excesso de elementos na notação, desnecessários para modelagem de processos executáveis.

Em [81] é apresentada uma análise da notação BPMN, baseada em ontologia de processo, com o foco voltado ao entendimento da notação por quem a utiliza. As conclusões desta análise destacam que, apesar da notação possuir várias vantagens, a ampliação do número de estruturas da BPMN tem prejudicado seu entendimento, devido ao aumento de sua complexidade. Os autores concluem que, embora a capacidade de expressão da notação seja maior, sua clareza é severamente prejudicada.

Com base nas conclusões dos trabalhos [80, 81], para processos executáveis o benefício do alto poder de expressão da BPMN não supera o custo do aumento da complexidade. Por essa razão conceitos como artefatos, evento de link, subprocesso, fluxo de mensagem e associação, presentes na BPMN, não foram incorporados ao metamodelo, por questões de simplicidade, clareza, objetividade e diminuição de redundância.

O conceito de laço, definido na própria atividade, também foi removido, haja vista que o mesmo laço pode ser feito utilizando conexões com portas de exclusão.

O Objeto de Dados foi mantido, porém seu objetivo foi redefinido. No metamodelo proposto ele não é utilizado na modelagem gráfica e tem a responsabilidade de representar o domínio dos sistemas que se comunicam com o processo.

Outra peculiaridade do metamodelo proposto neste trabalho é a forma de criação dos elementos que formam o processo. Em BPMN [54] os elementos que determinam o comportamento do processo são dependentes do contexto do processo no qual são criados. Porém, no metamodelo proposto os Objetos de Fluxo são independentes. Eles são criados sem vínculo algum com o contexto do processo. A principal vantagem deste método de criação é o compartilhamento e a reutilização dos Objetos de Fluxo.

Como cada Objeto de Fluxo possui independência dentro do metamodelo, é possível reutilizá-lo no contexto de vários processos diferentes. Isso diminui a redundância dos elementos do processo, elimina o OF de *SubProcesso* (pois qualquer processo pode

ser reutilizado em outros fluxos de processo) e torna o conceito de recursividade entre processos viável.

Além de reutilizar os Objetos de Fluxo em qualquer processo, o modelador pode compartilhá-los com outros colaboradores. Isso proporciona uma modelagem colaborativa na qual os usuários do sistema podem fazer uso de Objetos de Fluxo criados não só por eles como por outros participantes do sistema.

Outras pesquisas que motivaram diretamente a criação de um novo metamodelo [18, 26, 58] indicam potenciais falhas da BPMN. Entre as deficiências apresentadas na notação destacam-se a dificuldade de representação das regras de negócio, a falta de claridade nos conceitos de Baias e Raias, e a ambiguidade dessas construções.

Como vários elementos foram suprimidos do novo metamodelo, fez-se necessário analisar o novo metamodelo em relação ao seu poder de expressão. Para fazer esta análise utilizou-se os padrões de processos propostos por [73]. A partir destes padrões foram realizados experimentos que avaliam o poder de expressão deste metamodelo, conforme discute o Capítulo 6.

O próximo capítulo apresenta a máquina de orquestração responsável por executar qualquer processo representado através do metamodelo de processo proposto neste trabalho.

CAPÍTULO 4

Máquina de Execução de Processos

Este capítulo apresenta o projeto da máquina de execução responsável por executar as instâncias dos processos especificados através do metamodelo definido no Capítulo 3. Esta máquina tem como característica principal a não utilização de uma linguaguem intermediária para executar o processo.

Devido às fortes críticas em relação à perda de informação durante a conversão entre a notação gráfica e a linguagem de execução, resolveu-se criar uma máquina de execução não convencional. A máquina de execução proposta, ao contrário das outras máquinas que leem a linguagem de execução de processo como um bloco, simulando a execução de um algoritmo, lê e executa o grafo que representa o processo. Isso porque a notação gráfica expressada pelo metamodelo pode ser vista como um grafo constituído por um conjunto de nós e arestas. Os nós representam os elementos do processo e as arestas as conexões entre eles.

A Seção 4.1 apresenta o modelo da máquina de execução de processos e explica seus elementos. A Seção 4.2 descreve o funcionamento da máquina de execução de processos.

4.1 Modelo da Máquina de Execução

A máquina de execução proposta neste trabalho é responsável por instanciar processos definidos com base no metamodelo descrito no Capítulo 3 e controlar sua execução. Para isso, ela aloca os participantes às tarefas, avisa sobre pendências ocorridas na execução do processo e mantém os dados inseridos durante a execução dos processos.

O modelo da máquina de execução é ilustrado na Figura 4.1, onde retângulos de cor branca representam os conceitos do metamodelo de processo, enquanto os em cinza representam os conceitos da máquina de execução.

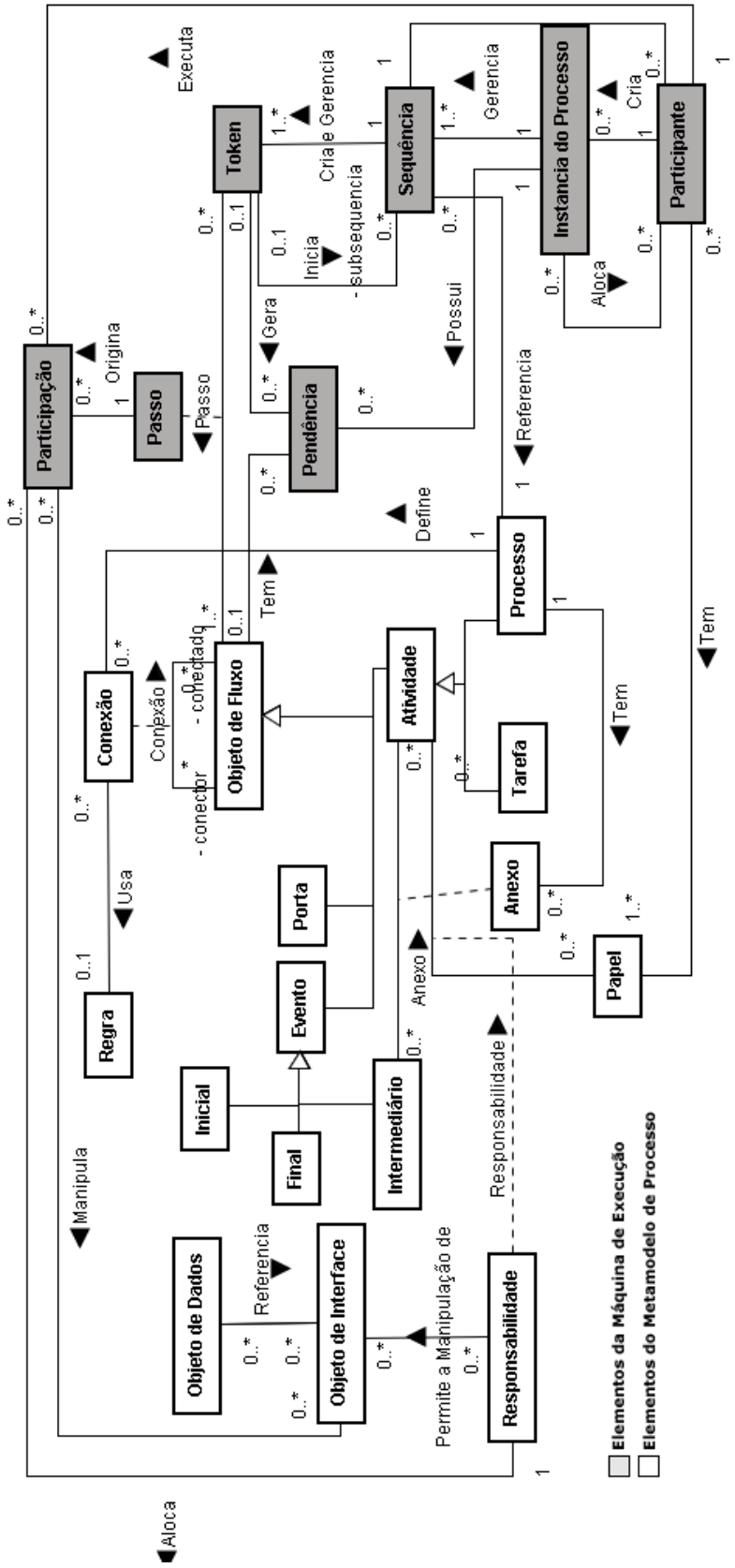


Figura 4.1: *Metamodelo de Processo e sua relação com a Máquina de Execução*

Após definir um processo através do metamodelo, é possível criar uma Instância de Processo, que permite sua execução pela máquina. Ao criar uma instância, é necessário definir os participantes e seus respectivos papéis. Os participantes do processo são alocados para execução das tarefas de acordo com estes papéis.

Para executar a instância do processo a máquina de execução deve criar uma **Sequência** de execução. Esta sequência é o conceito que coordena a execução de qualquer processo. A instância do processo inicial é chamada de processo principal (PP) e dá origem à sequência principal. Para cada processo interno (PI), ou seja, processos que pertençam ao fluxo do PP em execução, é originada uma nova sequência de execução, chamada sequência interna.

A máquina de execução gerencia estas sequências com a finalidade de executar a instância do processo. Para cada instância de processo, pode haver várias sequências.

As sequências são responsáveis por criar e manipular **Tokens**. Um Token é usado para marcar o OF que está sendo executado em um determinado momento. Para cada OF ao qual o token chega é criado um **Passo**. Ele é responsável por efetuar registros temporais correspondentes à chegada e à saída do Token de algum OF.

Assim que um token chega a um OF uma ação é executada. Quando o objeto em questão é uma **Tarefa**, então é criada uma **Participação**. Através dela os participantes são alocados para cumprirem a tarefa. Caso seja um **Processo**, uma nova sequência é criada, um administrador é definido e o token aguarda até que ela seja finalizada. Se o objeto for uma **Porta**, o token pode ser convergido, ou juntado a outro token, ou bifurcado, ou ainda ramificado em outros tokens, dependendo do tipo de porta. Sendo o objeto um **Evento**, sua ação dependerá de seu tipo e de seu gatilho.

Depois de executar a ação o token segue para o próximo OF até que chegue a um Evento Final. Quando isso acontece, o token é consumido e o seu fluxo chega ao fim.

Uma sequência termina apenas quando todos os tokens criados por ela forem consumidos, assim como a instância do processo é finalizada apenas quando todas as suas sequências terminarem.

4.1.1 Instância de Processo

O processo modelado é apenas uma representação abstrata do processo real. Para que um processo se torne executável, é necessário instanciá-lo. Assim, uma instância do processo é uma concretização do seu modelo abstrato. Para criar uma instância de processo é necessário informar: o processo a ser instanciado; o nome e uma descrição para a instância; e os participantes envolvidos em sua execução. Além disso, caso o criador da instância deseje, poderá definir explicitamente seu administrador.

O processo informado no momento de criação da instância é interpretado e executado pela máquina de execução. Portanto, a máquina analisa as informações pertencentes a este processo, como suas Conexões, os OF envolvidos nestas conexões e seu tipo para tomar decisões sobre a execução do processo.

O nome da instância é usado para identificá-la entre as demais instâncias do mesmo processo. Por exemplo: um processo chamado de "Desenvolvimento de Software", pode ser usado para implementar vários softwares diferentes. Assim, para desenvolver um programa de controle de patrimônio, por exemplo, a instância do processo seria chamada de "Controle de Patrimônio", enquanto, que outra instância deste mesmo processo poderia ser usada para construir um software de compras, e esta instância de processo seria chamada de "Compras". As duas instâncias seriam do mesmo processo, entretanto o produto de cada instância seria diferente.

A descrição da instância relata o que deve ser efetuado no decorrer do processo, e o que se espera como produto da instância criada. Esta descrição é usada pelos participantes com a finalidade de esclarecer o propósito da instância.

Os participantes do processo são definidos no momento da criação de sua instância e podem sofrer alterações durante a sua execução. Os participantes são associados às instâncias através de papéis que cada participante pode assumir no decorrer do processo. Um participante pode ter mais de um papel associado a ele, ou seja, pode assumir mais de um papel no decorrer do processo. Assim, o participante pode ser associado a tarefas exercendo diferentes papéis.

Existem seis tipos de estados possíveis para uma instância de processo: Aguardando Execução, Em Execução, Com Pendência, Terminado, Abortado e Finalizado. A Figura 4.2 mostra as transições possíveis entre esses estados.

O estado "Aguardando Execução" é o primeiro estado para o qual a instância transita. Neste estado, o processo foi instanciado, porém seu criador ainda não o executou. A instância não mudará de estado até o momento de iniciar sua execução.

Assim que o processo inicia sua execução ele passa ao estado "Em Execução". Neste estado, o processo está em andamento, ou seja seus fluxos estão sendo percorridos.

Durante o andamento do processo é possível que haja várias pendências que impeçam a continuação de seu fluxo, como por exemplo a falta de um OF que possa iniciar o processo. Caso uma pendência ocorra, a instância transitará para o estado "Com Pendência".

O processo vai para o estado "Terminado" apenas quando ele recebe um Evento Final de Término. Assim que este evento é lançado ele interrompe a execução do processo. Quando o próprio criador (ou administrador) do processo o aborta, sua execução passa para o estado "Abortado".

Se o andamento da instância ocorre sem problemas e todos os seus fluxos chegam

a um Evento Final, com exceção do Evento Final de Término ou de Erro, a instância passa para o estado “Finalizado”. A diferença entre os estados “Terminado” e “Finalizado” esta na conclusão do processo. O estado “Terminado” mostra que o processo foi interrompido enquanto estava sendo executado. Já o estado “Finalizado” indica que o processo foi concluído normalmente.

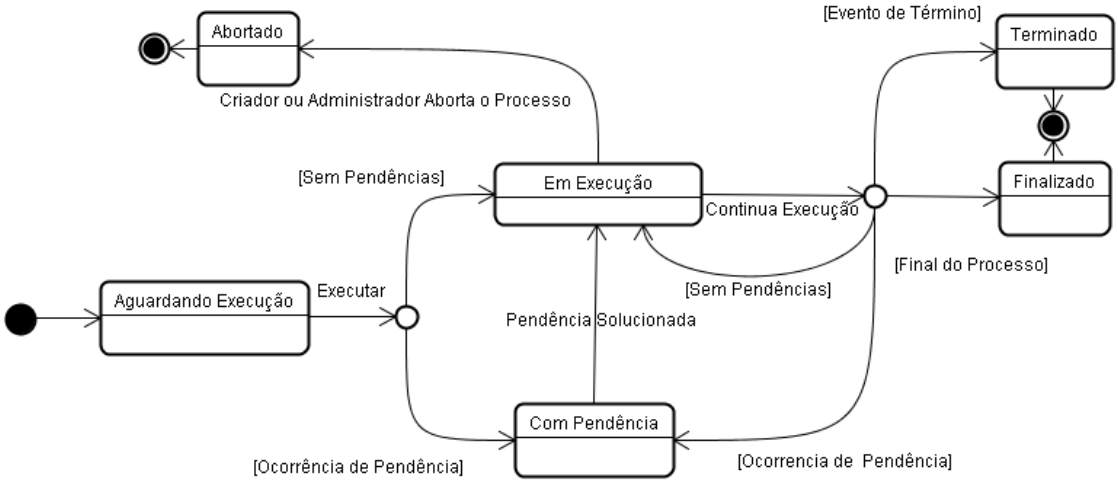


Figura 4.2: *Transição entre os Estados Possíveis para Instância do Processo*

4.1.2 Sequência

Para controlar o andamento de qualquer processo é necessário que a máquina de execução gerencie a transição entre os elementos do processo. É importante que o mecanismo responsável por realizar a execução do processo inicie e finalize a execução de seus fluxos. Ele também deve conhecer os elementos do processo que podem iniciar a execução de um fluxo; os fluxos que estão sendo percorridos e aqueles que foram finalizados; bem como o estado atual do processo. A **Sequência** é o elemento da máquina de execução responsável por lidar com todas estas questões.

Assim que a instância do processo é iniciada, a máquina de execução cria uma Sequência. Esta Sequência é associada ao processo e à sua instância. Ela é responsável por administrar o andamento do processo a ela associado, controlando a execução de seus fluxos.

Quando a Sequência é criada, ela analisa as Conexões pertencentes ao processo com a finalidade de encontrar dois tipos de OF: um Evento Inicial; ou uma Porta de Evento que não possua Conexões nas quais ela faz papel de “conectada”. Estes dois Objetos de Fluxo são os possíveis pontos de partida do processo. Após encontrá-los é iniciada a execução da instância do processo. Assim, para cada OF encontrado, é iniciada a execução do fluxo a ele associado.

Se o OF encontrado for um Evento Inicial, então o andamento do processo aguarda até que o gatilho do evento seja disparado. Por exemplo: se o gatilho do evento encontrado for padrão, a execução do fluxo relacionado a ele é iniciada imediatamente; caso seja de sinal, a execução do fluxo aguarda até que um Sinal de mesmo código seja acionado e dispare o evento; se o gatilho for de tempo, então a execução do fluxo aguarda até o final do tempo.

Entretanto se o OF for uma Porta de Evento, o fluxo aguarda até que um dos eventos relacionados com a porta seja acionado para prosseguir. Por exemplo: caso a porta esteja relacionada com um evento intermediário de tempo e outro de condição, a execução do fluxo aguarda até que a condição seja satisfeita ou até que o tempo seja atingido. Assim que um dos dois eventos é acionado, o outro evento é ignorado.

Durante o andamento de um processo é possível que outros processos que estejam contidos no fluxo daquele processo, os PI, necessitem ser iniciados. Como a Sequência principal controla o andamento de um PP, e qualquer processo pode referenciar outros em seu fluxo, várias sequências internas poderão ser originadas durante o andamento de um PP. Consequentemente, quando um PI é executado, uma sequência interna é criada, objetivando gerenciar o andamento deste processo.

Embora a sequência gerada seja vinculada ao PI encontrado no fluxo do processo principal, a instância referenciada pela nova sequência é a mesma do PP. Logo, apesar de existir uma nova sequência, não será originada uma nova instância para o PI. Isso acontece porque o PI que acabou de ser iniciado faz parte da execução do PP instanciado, causando uma relação de hierarquia. A Figura 4.3 ilustra esta relação.

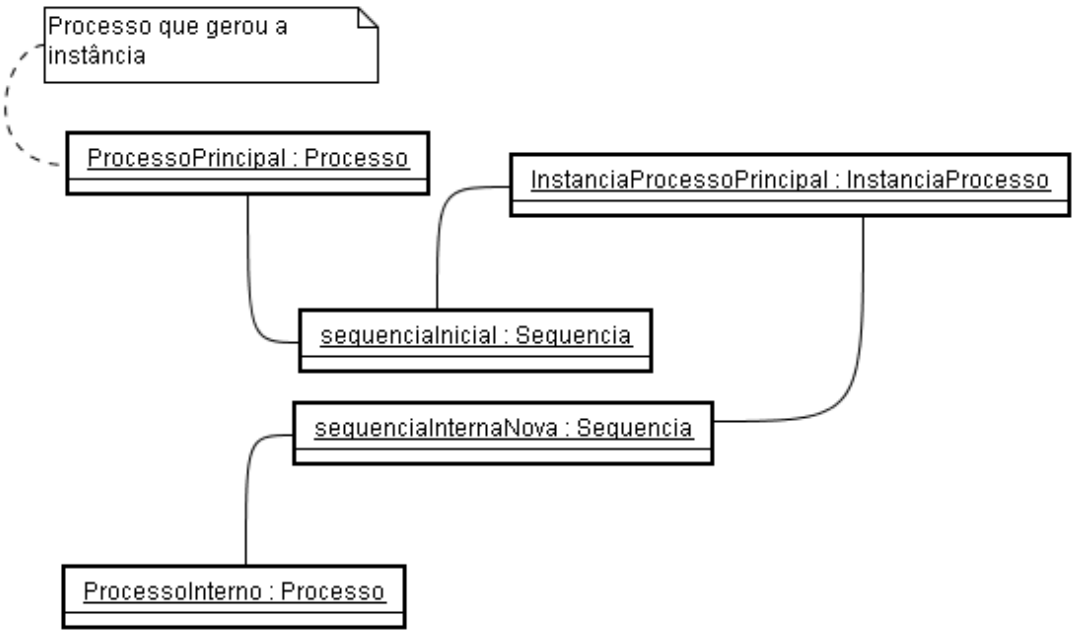


Figura 4.3: *Relação entre o Processo Principal e o Processo Interno*

O processo que inicia a execução de outro processo é chamado de Processo Pai (PPa) e o processo que é iniciado é denominado Processo Filho (PF). Um PPa pode ser um PI que possua outros processos vinculados a ele. Apesar da Sequência controlar apenas o fluxo do processo a ela relacionado, se o PPa for finalizado, todos os PF também serão. Isso pode acontecer caso o PPa seja abortado, ou se algum fluxo executar um Evento Final de Erro ou Término.

Porém, se o fluxo de algum PF atingir um Evento de Erro, ou se for abortado, ele não finaliza a execução do PPa. O fluxo do PPa somente será finalizado pelo PF, caso um de seus fluxos execute um Evento de Término.

Quando uma nova sequência é criada, o fluxo do PPa aguarda até que ela seja finalizada para que ele continue. Uma sequência pode ser finalizada de três maneiras distintas: 1) se todos os fluxos atingirem um Evento de Final que não seja do gatilho Término ou Erro; 2) se um dos fluxos atingir um Evento Final de Término ou de Erro; 3) e se os administradores do processo (ou seu criador) decidir abortar a execução do processo explicitamente.

Se todos os fluxos da sequência chegarem a um Evento Final que não seja de Término ou de Erro, a sequência será terminada, a instância do processo ganhará o estado de "Finalizada" e o gatilho do evento será disparado. Por exemplo: caso o gatilho seja de Sinal, então um sinal será lançado para todas as instâncias em execução.

Caso um Evento de Término seja atingido por algum fluxo, então todos os fluxos deste processo e de seus processos filhos e pais serão finalizados.

Se o fluxo atingir um Evento de Erro, então um erro será lançado para o processo da sequência. Este erro pode ser tratado por um evento Intermediário de Erro que deve estar anexado ao processo que o lançou. Além disso, este processo deve ser um processo interno. O tratamento deste erro pode mudar o andamento do fluxo do processo pai. Entretanto, se o erro foi lançado pelo processo relacionado à instância em execução, então todas suas sequências serão finalizadas.

Se os administradores do processo, ou seu criador, abortar sua execução, então todas as suas sequências serão finalizadas.

4.1.3 Token e Passo

Os Tokens têm a finalidade de registrar dados sobre os elementos percorridos durante a execução do processo. Eles são usados para determinar qual OF está em execução e quais já foram percorridos. Os Tokens são originados através de três tipos de OF: Eventos, quando forem do tipo inicial; Portas, do tipo Inclusiva ou Paralela; e Atividades, do tipo múltipla-instância.

Os Eventos Iniciais originam Tokens com o intuito de iniciar a execução do processo. Conforme estes eventos são acionados, os Tokens são criados e o processo é iniciado. O tipo do gatilho do evento determina o comportamento do Token. Por exemplo: caso o evento possua o gatilho padrão, então o Token é criado e, no mesmo momento, passa para o próximo OF relacionado com o evento. Se o evento tiver o gatilho de tempo, então o Token é criado e aguarda até que o tempo, definido pelo evento seja concluído. Depois da conclusão desse tempo, o Token passa para o próximo OF da sequência de execução.

As Portas criam Tokens com a finalidade de ramificar o fluxo do processo. Se o Token chega a uma Porta Inclusiva, ela analisa suas Conexões de saída para determinar quais fluxos podem ser percorridos. Para cada fluxo selecionado continuar o andamento do processo, um token é criado. Já as Portas Paralelas, como não fazem nenhuma análise das Conexões de saída, criam um token para cada uma delas. Cada token criado percorre uma das saídas da Porta Paralela.

Além de originarem Tokens, as portas também podem finalizá-los. Isso ocorre durante a ramificação ou sincronismo do fluxo. Ao ramificar o fluxo, a Porta finaliza o Token que chega e cria outros Tokens para continuar o fluxo do processo. Já no sincronismo, todos os tokens que estão na porta são finalizados e novos tokens são criados para dar continuidade ao andamento do processo.

As Atividades do tipo múltipla-instância também podem criar Tokens. Assim que um Token chega a uma Atividade deste tipo, a máquina de execução analisa a expressão que determina o número de Tokens a serem criados pela atividade. Este número pode sofrer alterações durante a execução de cada atividade. Estas alterações podem criar mais Tokens e, assim, adicionar mais instâncias da mesma atividade à execução do processo.

Outra possibilidade de criação de tokens em atividades múltipla-instância é quando o administrador do processo determina a criação de mais uma instância da atividade. Esta ação pode ser tomada pelo administrador em qualquer momento do fluxo do processo, caso a atividade permita.

Após ser criado, um Token deve percorrer o fluxo do processo de acordo com suas Conexões. Isso é feito através da passagem de um OF para outro. Assim, o Token desloca do OF que faz o papel de Conector e segue para o OF que exerce o papel de Conectado. Para cada movimentação realizada pelo Token, a máquina de execução cria um Passo. Esta movimentação é ilustrada pela Figura 4.4.

Passo é o elemento que marca os Objetos de Fluxo pelos quais o Token passou. A máquina de execução utiliza este conceito para determinar em qual OF o Token está. Antes que o Token seja movimentado e o passo seja criado, é necessário avaliar se a ação do OF Conector foi realizada. Esta ação a ser executada pela máquina é determinada pelo tipo do OF que está em execução.

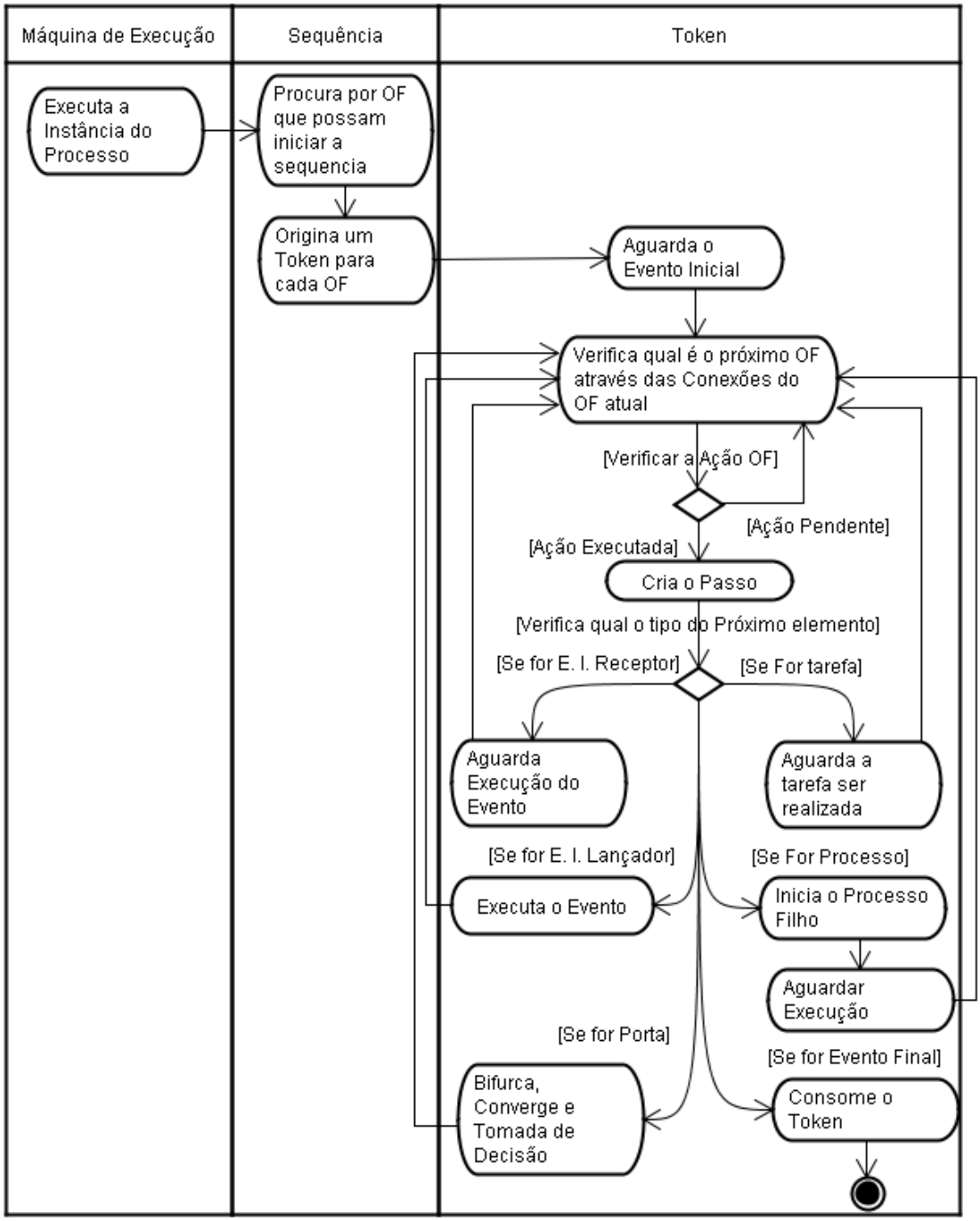


Figura 4.4: *Movimentação do Token pelo Processo*

Se o Conector for uma tarefa, então esta deve ser cumprida. Porém, se o OF for um processo, então seu andamento deve ser iniciado e concluído para que o Token siga para o próximo OF. Quando o Token entra em contato com um Evento do Tipo Intermediário e lançador, ele executa a ação do evento e passa para o próximo OF. Caso o evento intermediário seja receptor, ele aguarda até que o evento seja disparado. Finalmente, se o OF for uma Porta ele executa o comportamento determinado pelo tipo da porta e continua o fluxo.

Após percorrer o fluxo do processo o Token pode ser finalizado por apenas um tipo de OF: o Evento Final. Ao atingir um Evento Final, o Token é finalizado e o gatilho do evento é disparado. Caso seja um Evento Final de Término, todos os outros tokens deverão ser finalizados, independentemente de sua posição no fluxo do processo. Entretanto, se for o gatilho padrão, o token é simplesmente finalizado pelo OF.

Assim, a máquina avalia qual a situação atual do Token e se a ação a ser executada pelo OF atual já foi realizada. Caso a ação tenha sido realizada, então o Token passa para o próximo OF da sequência de execução; caso contrário o Token aguarda até que a ação do OF seja realizada.

Esta abordagem baseada em tokens permite a modificação do processo durante sua execução. Isso é possível porque o processo não é completamente instanciado pois, na realidade, a máquina de execução lê o modelo do processo passo por passo.

Portanto, conforme o processo é executado, a máquina de execução analisa o token e o passo corrente para decidir sobre o próximo passo. Caso o Fluxo do processo em andamento seja alterado, a máquina de execução analisa o Token de acordo com a nova modelagem do processo e o desloca seguindo a ordem estabelecida pelo novo fluxo. Assim, modificações no processo são instantaneamente refletidas em seu fluxo, permitindo a modelagem de processos empíricos.

4.1.4 Participante e Participação

Para executar uma tarefa é necessário alocar participantes capazes de colaborarem em prol de sua execução. A alocação destes participantes deve ser realizada obedecendo as restrições impostas pela tarefa, como a quantidade de participantes por papel e responsabilidade. Outra questão importante é o controle dos dados que podem ser inseridos por estes participantes durante a execução da tarefa.

O conceito de Participação controla a alocação de participantes, o produto de saída e a realização das tarefas. Ele é responsável por guardar os participantes que estão executando cada tarefa, além das responsabilidades exercidas por eles e a data em que a tarefa foi vinculada ao participante. Ademais, a Participação informa se o envolvido alocado para determinada tarefa já realizou sua parte na execução. Além disso, o objeto contendo os dados retornados pelos sistemas gerenciadores de interface são guardados pela Participação.

Quando um Token chega a um OF do tipo tarefa ele verifica se os participantes já foram definidos para aquela tarefa. Caso os participantes estejam definidos, o Token verifica se a tarefa foi concluída. Se os participantes ainda não tiverem sido definidos, então eles deve ser alocados.

Para determinar os participantes da tarefa, primeiramente é verificado se o administrador do processo já os definiu. Caso eles não estejam definidos, a máquina de execução os aloca dinamicamente. Por exemplo: caso a tarefa exija dois participantes que tenham papel de Desenvolvedor e outros dois com papel de Analista de Sistemas, a máquina irá selecionar os participantes envolvidos na instância do processo que possam exercer tais papeis e alocá-los com a finalidade de cumprirem a tarefa.

Porém, se os participantes já tiverem sido definidos, a máquina de execução alocará somente os participantes necessários para completar o grupo de execução da tarefa. Por exemplo, supondo que sejam necessários dois Analistas de Negócio e um Técnico, e que o administrador do processo já determinou um Analista de Negócio e um Técnico para o processo, a máquina de execução alocará somente o Analista de Negócio que está faltando.

A alocação dos colaboradores é aleatória, ou seja, a máquina seleciona todos os participantes qualificados para realizarem aquele tipo de tarefa que estejam associados à instância do processo e os aloca sem obedecer nenhum critério. Porém, um participante não pode ser alocado para a mesma tarefa, exercendo o mesmo papel e a mesma responsabilidade, pois neste caso estaria realizando a mesma tarefa duas vezes.

No momento em que as tarefas são atribuídas aos participantes, são também criadas as Participações. Estas fazem a relação entre a tarefa, o participante, sua responsabilidade e seu papel na execução da tarefa, definindo, ainda, com qual objeto de interface o participante irá interagir, se for o caso. Assim que esta relação é estabelecida, a participação criada fica pendente para o participante resolver. O participante alocado para a tarefa deve realizar a ação definida para ele através da participação.

Caso a tarefa seja do tipo usuário, então o participante entra em contato com o objeto de interface indicado na participaçao que lhe fornece um meio de entrada de dados. Os dados preenchidos pelo participante devem ser guardados pela Participação.

Estes dados poderão ser referenciados em diversos locais, tais como nas expressões que compõem as regras, na determinação do número de instâncias a serem criadas de uma atividade múltipla-instância, nas expressões que determinam as condições de sincronismo de atividades de múltipla-instância e nas expressões de condição de eventos condicionais.

A alocação de administradores para o processo se dá da mesma maneira que os participantes são alocados para tarefas. Entretanto, os participantes alocados para o processo não necessitam executá-lo, mas sim gerenciá-lo.

Os administradores são responsáveis por: definir o fluxo das tarefas no processo *adhoc*; criar novas instâncias de tarefas, quando estas permitirem a criação de novas instâncias em tempo de execução do processo; determinar os participantes de cada tarefa, e o tipo de sua interação e resolver as pendências que aparecerem.

4.1.5 Pendência

São várias as pendências que podem ocorrer durante a execução de um processo. Alguns exemplos são: usuário tenta instanciar um processo não *adhoc* que não possui um OF que possa iniciá-lo; o número de participantes necessários para efetuar determinada tarefa é insuficiente ou não existe nenhum participante no processo que possa assumir um determinado papel no processo. Nessas situações o processo fica no estado "Com Pendência". Este estado só será modificado caso a pendência seja resolvida.

Para solucionar uma pendência é necessário que o criador da instância do processo e/ou o administrador sejam informados da atual situação do processo. De posse desta informação, é possível corrigir o problema e o processo poderá continuar sua execução. Para prover suporte a este tipo de controle de exceção foi elaborado o tratamento de Pendência.

Este conceito foi criado para informar acontecimentos inesperados durante o andamento do processo aos seus administradores, com o objetivo de que eles solucionem a situação inesperada.

Quando uma pendência é lançada, a execução do processo aguarda até que a pendência seja resolvida. Toda pendência possui um motivo e uma solução. O motivo explica porque a pendência ocorreu. Por exemplo: o processo exige um participante que possua um certo papel e não há participantes com esse papel alocados à instância do processo.

A solução informa o que deve ser feito para solucionar a pendência, neste caso: "Adicione participantes com papel <tipo de papel> a esta instância de processo". Caso os responsáveis pela instância do processo, criador e/ou administrador, adicionem participantes qualificados para tal papel, o processo voltará ao estado "Em Execução".

A máquina de execução é capaz de indicar uma solução para a pendência porque os possíveis erros que podem ocorrer durante a execução de um processo foram mapeados durante o desenvolvimento dessa máquina. Estes erros e suas respectivas soluções são apresentadas na Tabela 4.1. Assim, quando acontece uma pendência, a máquina de execução tem o conhecimento necessário para indicar uma solução.

Existem dois tipos de pendências: as que são criadas antes da execução do processo e as que são originadas durante sua execução. Como antes da execução do processo não há nenhum Token para percorrer seus fluxos, as pendências criadas antes do início do processo não são vinculadas a nenhum fluxo. Geralmente estas pendências são relacionadas com a instância do processo e ocorrem antes mesmo que o processo comece sua execução. Para resolver este tipo de pendência o usuário deve interagir com o processo ou com sua instância.

As pendências criadas após o início da execução da instância do processo são associadas aos Tokens e aos Objetos de Fluxo que as ocasionaram. Isso garante um maior

Pendência	**Solução**
Nenhum objeto inicial foi definido para o processo <*nome do processo*>.	Defina pelo menos um evento de início ou porta de evento para iniciar o processo!
Não há participante com papel necessário para administrar o processo <*nome do processo*>.	Adicione participantes com papel <nome do papel> a esta instância do processo <*nome do processo*>!
Não existem participantes definidos para a tarefa <*nome da tarefa*>.	Adicione participantes à tarefa <*nome da tarefa*>!
Não existem participantes definidos para a instância do processo <*nome do processo*>.	Adicione participantes ao processo <*nome do processo*>!
Não existem participantes definidos para a instância de processo <*nome da instância do processo*>.	Adicione participantes à instância de processo <*nome da instância do processo*>!
Não existem participantes suficientes para executar a tarefa <*nome da tarefa*>.	Adicione mais <*quantidade*> participantes com o papel <*nome do papel*> à instância de processo <nome da instância do processo>!
Processo é *adhoc*, mas a próxima atividade deste fluxo não foi definida.	Defina a atividade a ser executada depois de <*nome da atividade*>!
Processo não é adhoc e não possui continuação da sequência. .	Adicione um fluxo ao objeto <*nome do objeto de fluxo*>!
Erro na execução da regra <*nome da regra*>.	Arrume a regra <*nome da regra*>!

Tabela 4.1: *Mapeamento das Pendências*

nível de detalhe no momento de informar ao usuário qual foi o problema ocorrido. Além disso, a solução para este tipo de problema pode ser mais específica, já que se sabe qual o fluxo e OF geraram tal condição.

4.2 Execução de Processos

A máquina de execução proposta pode gerenciar o andamento de dois tipos de processo: definidos e *adhoc*.

Os processos definidos são utilizados para representar fluxos de trabalho já consolidados. Tal situação é mais comum em organizações maduras, cujos processos estão institucionalizados.

Os processos *adhoc* representam processos que não possuem uma sequência lógica definida, logo este tipo de processo é utilizando quando a instituição não sabe como as atividades devem ser encadeadas. Geralmente esta falta de padronização de processos ocorre em empresas que não conseguem definir seus fluxos internos ou quando o processo requer flexibilidade.

4.2.1 Execução de Processos Definidos

Para executar um processo definido, primeiramente, é necessário modelá-lo. Todo processo definido deve ter pelo menos um Evento Inicial e um Final. Além disso, todas as suas Conexões devem possuir Conector e Conectado.

Após modelar o processo, seu criador pode iniciar sua execução. Quando um processo definido é executado, a máquina verifica se entre seus participantes existe algum que possa gerenciar o processo. Primeiramente é analisado se o criador da instância do processo já definiu explicitamente seu administrador. Caso ele não tenha sido determinado, a máquina de execução analisa entre os alocados para execução da instância do processo se existe algum qualificado para administrar o processo.

Se nenhum participante for qualificado para exercer tal papel, a máquina lançará uma Pendência. A Pendência lançada será "Não há participante com papel necessário para ser Administrador do Processo" e a solução "Adicione participantes com papel <Papel Necessário > a esta instância de processo".

Porém, se for definido um administrador para o processo, então a máquina de execução cria uma Sequência para a instância do processo. A Sequência verifica se existe um OF que possa iniciar o andamento do processo.

Caso ela não encontre um OF que seja apto a iniciar o fluxo do processo, então será lançada outra pendência. A Pendência lançada será "Nenhum objeto inicial foi definido para o processo <Nome da instância do processo>". Sua solução será "Defina pelo menos um evento de início ou porta de evento para iniciar o processo".

Esta pendência também poderia ser tratada por um mecanismo de validação do processo em tempo de modelagem. Porém, considerando que o modelo do processo pode ser alterado a qualquer momento e que processos incompletos podem ser executados, este tipo de abordagem seria inviável.

Se a Sequência encontrar um ou mais Objetos de Fluxo que possam iniciar a instância do processo, então é criado um Token para cada um deles. Todos os Tokens criados recebem uma data e hora de criação e executam a ação dos Objetos de Fluxo em que eles foram criados. Para cada OF qualificado para iniciar o processo é executada uma ação diferente:

- **Evento Inicial Padrão**: não há ação a ser executada. O Token simplesmente é passado para o próximo elemento da lista.
- **Evento Inicial de Tempo**: o Token deve aguardar o tempo definido por este evento. Caso seja um período de tempo (como, por exemplo, uma hora ou um dia), o Token aguarda este prazo, que será contado a partir do momento em que o Token é criado. Entretanto, se for uma data, então o Token só irá para o próximo elemento na data especificada.

- **Evento Inicial de Condição**: a máquina de execução avalia a condição definida pelo evento a cada *n* segundos, onde *n* é um número definido pelo modelador do processo. Quando a condição se tornar verdadeira, ela passa o Token para o próximo elemento.
- **Evento Inicial de Sinal**: o Token aguarda até que um evento de Sinal Lançador acione este evento. Assim que o evento é disparado ele passa o Token para o próximo OF.
- **Porta Evento**: todos os eventos ligados à porta são testados em intervalos de *n* segundos, onde *n* é um número definido pelo modelador do processo. Assim que um dos eventos é disparado o Token segue o fluxo.

Após a execução da ação do OF que iniciou o processo, a máquina de execução passa o Token para o próximo OF. Caso este objeto de fluxo não tenha sido definido, a máquina de execução lançará outra pendência: “Processo não é *adhoc* e não possui continuação da sequência.”. A solução será "Adicione um fluxo ao objeto de fluxo <Nome do Objeto> do processo <Nome do Processo>! ".

Se o próximo objeto a ser pecorrido já está definido, então o Token é passado para ele, criando seu primeiro Passo e registrando a data e hora de criação. Os Objetos de Fluxo que podem ser percorridos durante o andamento do processo são:

- **Tarefa Usuário**: primeiramente, os participantes são alocados. A máquina de execução aguarda até que todos os envolvidos na tarefa façam sua parte. Depois que todas as participações são concluídas, o Token é passado para frente.
- **Tarefa Script**: um script é executado pela máquina e o Token é passado para frente.
- **Tarefa Manual**: após alocar os participantes, a máquina aguarda todas as interações com a tarefa. Em seguida, o Token segue o fluxo do processo.
- **Tarefa Envio de Mensagem**: é apresentada para o participante uma interface de envio de mensagem. Após o envio desta mensagem, o Token segue para o próximo elemento.
- **Tarefa Recebimento de Mensagem**: a máquina de execução aguarda que o responsável informe que recebeu a mensagem esperada para prosseguir o andamento do processo.
- **Processo**: assim que o Token chega a um processo interno, a máquina de execução: aloca um participante para ser o gerente da execução do processo e cria uma sequência para tal processo. Caso o participante responsável por administrar o processo não tenha sido declarado explicitamente a máquina o alocará. A sequência cuida da execução do processo. O Token só prossegue para o próximo elemento do fluxo após a conclusão deste processo.
- **Evento Intermediário de Tempo**: a máquina comporta-se similarmente ao Evento de Tempo Inicial. A única diferença é que se o tempo definido pelo processo for um

período de tempo, então este tempo será contado a partir da chegada do Token ao evento.

- **Evento Intermediário de Condição**: a máquina de execução realiza o mesmo comportamento do Evento Inicial de Condição.
- **Evento Intermediário Sinal Lançador**: a máquina de execução verifica entre todas as instâncias dos processos que estão em andamento quais eventos de Sinal estão aguardando. Ela analisa se o evento que lançou o sinal tem o mesmo código do evento que está aguardando e, neste caso, dispara este evento. Logo depois, o Token prossegue para o próximo OF.
- **Evento Intermediário Sinal Receptor**: a máquina de execução realiza o mesmo procedimento que no Evento Inicial de Sinal.
- **Porta Evento**: todos os eventos ligados à porta são testados em intervalos de *n* segundos, no qual *n* é um número definido pelo modelador do processo. Assim que um dos eventos, for disparado o Token segue o fluxo.
- **Porta Exclusiva**: para cada Token que chega à porta, a máquina de execução analisa todas as regras das Conexões na qual a porta é o Conector. O Token é direcionado para a primeira Conexão cuja regra retorna verdadeiro. Caso nenhuma regra retorne verdadeiro o Token segue pela Conexão Padrão.
- **Porta Inclusiva**: a máquina de execução analisa se existe algum Token que poderá chegar a esta porta. Esta análise é feita verificando se alguns dos caminhos possíveis até a Porta Inclusiva possui algum Token. Caso algum caminho possua Tokens, então a máquina aguarda. Entretanto, se todos os Tokens que puderem chegar até a porta já estiverem nela, a porta finaliza todos os Tokens e analisa as conexões de saída. Para cada conexão cuja regra seja verdadeira, a máquina cria um novo Token que dá continuidade ao fluxo. Caso nenhuma regra retorne verdadeiro, a porta criará um Token e passará para a Conexão padrão.
- **Porta Paralela**: para cada Token que chega a este tipo de porta, a máquina de execução avalia se todas as Conexões nas quais a porta faz o papel de Conectado foram executadas. Caso ainda não tenham sido, a máquina de execução aguarda até que sejam executadas. Assim que todas as Conexões forem executadas, a máquina finaliza todos os Tokens que estiverem na porta e, para cada Conexão na qual a porta faz o papel de Conector, é criado um novo Token que dá continuidade ao fluxo.
- **Porta Complexa**: a cada Token que chega a uma Porta Complexa, a máquina de execução compara a quantidade de Tokens que estão parados com o número retornado pela regra de sincronismo da porta. Se o número de Tokens for igual ao número retornado pela regra, então a máquina dá continuidade ao fluxo. Caso contrário, a máquina aguarda a chegada de mais tokens. Para continuar o andamento do processo, a máquina verifica o número retornado pela regra de saída. Para cada

Conexão na qual a porta faz o papel de Conector, será criada a quantidade de Tokens equivalente a esse número.

A qualquer momento um processo pode ser alterado. As alterações são efetivadas automaticamente em tempo de execução de qualquer instância do processo que estiver em execução.

Depois de percorrer todo o fluxo do processo, o Token chega a um OF que possa finalizá-lo. Então, o token será consumido e receberá uma data de desativação. Cada evento possui seu comportamento específico:

- **Evento Final Padrão**: assim que o Token chega a este evento, ele é consumido e recebe uma data de desativação.
- **Evento Final de Sinal**: a máquina de execução se comporta do mesmo modo que no Evento Intermediário de Sinal Lançador.
- **Evento Final Terminal**: ao atingir este evento todas as sequências relacionadas com a instância do processo são finalizadas. Consequentemente, todos os tokens também não finalizados, tanto dos processos pais como dos processos filhos.
- **Evento Final de Erro**: assim que o Token chega a este evento, a sequência a ele relacionada e todas as sequências de seus processos filhos são canceladas. Caso o processo seja um processo interno, a máquina de execução verifica se o processo relacionado à sequência possui um evento que possa tratar o erro. Este evento intermediário deve ter o mesmo código de erro do evento final, ou não possuir nenhum código de erro. Se o evento intermediário tratar o erro, o fluxo do processo que deu origem ao processo responsável por lançar o erro é desviado para o OF conectado a ele.

Assim que o Token é consumido, ele recebe uma data de desativação e não percorre mais os fluxos do processo. Quando todos os Tokens pertencentes a uma Sequência são finalizados, ela também é. Ela recebe, assim como os Tokens, uma data de término. Finalmente, quando todas as sequências de uma instância de processo são finalizadas, a instância do processo ganha o estado de "Finalizada" e termina.

4.2.2 Execução de Processos *Adhoc*

A execução de uma instância de um processo *adhoc* é similar à execução de um processo definido. Todavia, o fluxo do processo *adhoc* deve ser determinado no momento de sua execução. O fluxo do processo é especificado de duas maneiras diferentes: pelo administrador do processo ou pelos participantes envolvidos em execução de tarefas.

Como processos *adhoc* podem não possuir Evento Inicial, assim que é criada uma instância do processo, é necessário determinar qual a tarefa ou processo de início.

Depois que seu criador informa a atividade que inicia o andamento do processo, a máquina de execução cria um Token com origem nela. Entretanto, se o processo *adhoc* possuir um evento inicial, então este é usado para começar o processo.

Quando uma atividade é concluída, a máquina de execução busca todas as Conexões desta atividade que não possuem um OF exercendo papel de Conectado. Todas as atividades que fizerem papel de conector nestas conexões são listadas para o administrador do processo. Para cada atividade apresentada, o administrador pode especificar qual será a próxima tarefa ou processo a ser executado.

Outra maneira de definir a atividade seguinte é através da colaboração dos participantes envolvidos com a tarefa em execução. Cada participante pode dar uma sugestão de qual a próxima atividade a ser realizada. Assim, quando a atividade é concluída, a máquina avalia as sugestões de todos os participantes. Esta análise consiste em determinar qual a atividade mais recomendada para continuar o andamento do processo. Se houver um empate entre as atividades avaliadas, a máquina escolherá a primeira atividade especificada por um participante que tenha responsabilidade de aprovador.

Uma vez que a atividade seguinte é definida, a máquina de execução faz com que o Token siga por esta atividade, a não ser que ela seja redefinida durante o andamento desta instância do processo. Assim que um Token conclui uma atividade, a máquina de execução verifica as Conexões do OF. Caso a tarefa ou processo possua alguma Conexão na qual exista um OF exercendo o papel de Conectado, então o Token seguirá para este objeto. Entretanto, se não existir tal elemento, a máquina buscará o próximo OF definido para continuar a execução do fluxo.

Porém, se o administrador ou os participantes da atividade em execução não tiverem definido o próximo objeto a ser executado, a máquina lança uma Pendência: “Não há próximo elemento especificado para a atividade <Nome da Atividade>”. Sua solução será “Especifique o próximo elemento a ser executado após o término da atividade <Nome da Atividade>”. Contudo, se o próximo elemento já tiver sido especificado, a máquina de execução passará o Token para este OF e continuará o andamento do processo.

Uma instância de processo *adhoc* pode ser finalizada de duas maneiras: da maneira convencional ou de maneira explícita. A maneira convencional funciona como em processos definidos: todos os Tokens do processo instanciado chegam a um evento final e são consumidos, finalizando assim suas sequências e, por consequência, a instância do processo.

A maneira explícita de finalizar um processo *adhoc* ocorre por iniciativa de seu administrador. Basta que ele determine que o processo seja finalizado para que todos os seus Tokens sejam desabilitados e suas sequências finalizadas.

CAPÍTULO 5

A Arquitetura da Ferramenta Sinfonia

Este capítulo apresenta o design da ferramenta Sinfonia, um BPMS baseado no metamodelo de processo e na máquina de execução de processos propostos neste trabalho. Este software, desenvolvido para a plataforma web, provê interface gráficas dinâmicas que auxiliam os usuários na modelagem e execução de processos.

A codificação da ferramenta foi feita na linguagem *Ruby* [47] em conjunto com o framework *Rails* [23]. A interação com o usuário é baseada nos frameworks *ExtJs* [64] e também utiliza componentes feitos em *Flex* [4], codificados na linguagem *Actionscript* [5]. A tecnologia *Flex* proveu suporte à criação de um componente gráfico capaz de modelar os processos visualmente.

5.1 Visão Geral da Arquitetura

Para apresentar a arquitetura deste software, foi utilizada uma adaptação da abordagem proposta por Kruchten [37], que expressa a arquitetura do software em visões distintas. A arquitetura do software Sinfonia é demonstrada utilizando a UML [52].

A separação dos diferentes aspectos do sistema em visões distintas permite gerenciar a complexidade, reduzir a quantidade de informações expressas em cada modelo e tratar cada aspecto arquitetural em diferentes visões.

A Figura 5.1 ilustra a visão física do sistema, refletindo a distribuição do software entre diferentes máquinas. O navegador do cliente comunica-se com o servidor de aplicação no qual está hospedado o *software* Sinfonia. O servidor de aplicação recebe a requisição do cliente e repassa para a aplicação web Sinfonia. Ela, por sua vez, interpreta a solicitação e, caso seja necessário algum dado persistente para montar a resposta da aplicação, ela comunica-se com o SGBD. Além disso, Sinfonia oferece uma maneira de se comunicar com outras aplicações web descrita na Seção 5.6.

A arquitetura de Sinfonia foi desenvolvida utilizando o padrão arquitetural Modelo Visão Controle (*Model View Controler* - MVC). Este padrão divide uma aplicação interativa respeitando três perspectivas: modelo, visão e controlador [12]. O “modelo” contém o núcleo funcional, as informações sobre o domínio e os dados da aplicação. A

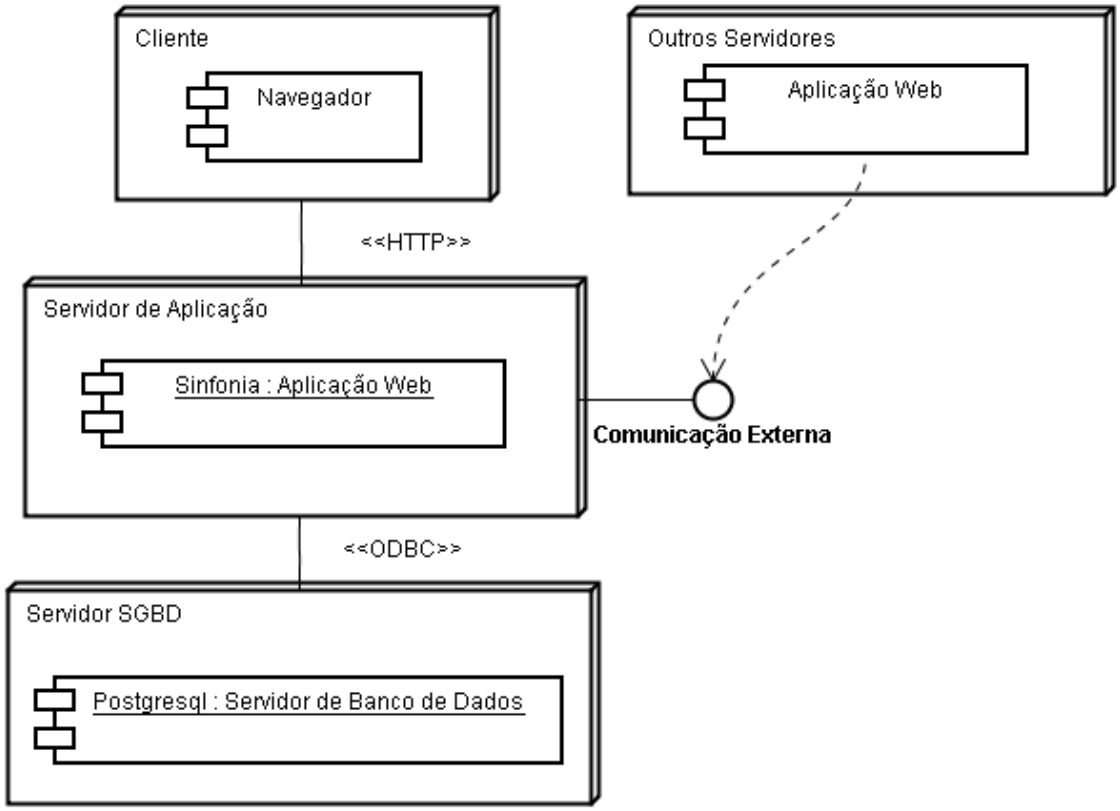


Figura 5.1: *Visão Física do Sistema Sinfonia*

"visão" é responsável por apresentar as informações ao usuário. O "controlador" cuida das solicitações feitas pelo usuário e manipula a visão e o modelo com a finalidade de mostrar os dados do sistema.

A Figura 5.2 ilustra o funcionamento do padrão MVC na arquitetura de Sinfonia. Uma requisição HTTP chega de um navegador cliente ao servidor web, que trata a solicitação e a direciona para o controlador correto. O controlador lê os dados da solicitação HTTP e recupera ou instancia objetos do modelo, manipulando estes objetos de acordo com o comportamento da ação requisitada. Baseado nos dados obtidos através da lógica executada, o controlador decide a visão que será apresentada. Se necessário, ele altera os dados da solicitação HTTP, de acordo com a visão, ou seja, se a camada de visão necessitar de um dado que não esteja na solicitação HTTP o controlador pode adicionar esta informação específica à solicitação.

A visão selecionada, por sua vez, lê os dados dos objetos de domínio manipulados pelo controlador e gera uma resposta. Esta resposta é enviada para o servidor web que a reenvia ao navegador que fez a solicitação.

A distribuição dos módulos na arquitetura MVC é apresentada através do diagrama de pacotes do sistema, exibido na Figura 5.3. O pacote de modelo é uma extensão do pacote Registro Ativo contido dentro do *framework* Rails [23]. Esta especialização foi realizada com a finalidade de implementar uma camada de persistência para a aplicação.

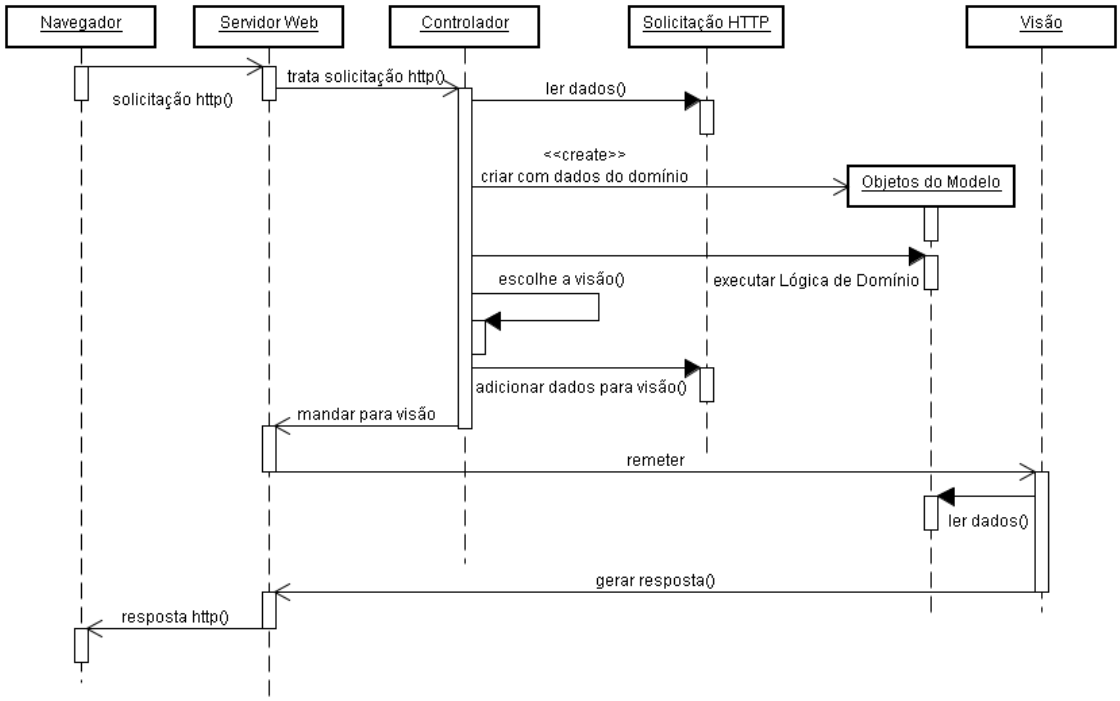


Figura 5.2: *Comunicação entre Componentes do MVC*

Os pacotes que foram especializados são três: Metamodelo de Processo, Gerência do Sistema e Máquina de Execução. O primeiro pacote é responsável por manipular o metamodelo de processo, enquanto que o pacote Máquina de Execução contém os detalhes da máquina de execução de processo. Já o pacote Gerência de Sistema administra os participantes, faz o controle de acesso e gerencia o compartilhamento dos OF entre os usuários.

Para a grande maioria das classes disponíveis no pacote de modelo, existe uma estrutura de classes e arquivos formada pelos pacotes de controle e visão. No caso da visão, esta estrutura provê suporte à interação do usuário através da utilização de uma biblioteca que baseada no *framework* Extjs [64] para criação de interfaces e do componente de modelagem gráfica.

A biblioteca é responsável por criar os formulários de dados para que o usuário possa entrar com as informações relativas à criação e execução dos processos, além da gerência dos usuários do sistema. O modelador gráfico é um subsistema utilizado para a modelagem dos processos de negócio.

Já o pacote controlador cuida para que as ações solicitadas através do navegador sejam corretamente executadas. Além disso, ele seleciona a visão correta para apresentar ao usuário. Este pacote é uma especialização do pacote Controle de Aplicação do Rails. Esta especialização é realizada com o objetivo de utilizar os padrões de comunicação propostos pelo *framework* Rails.

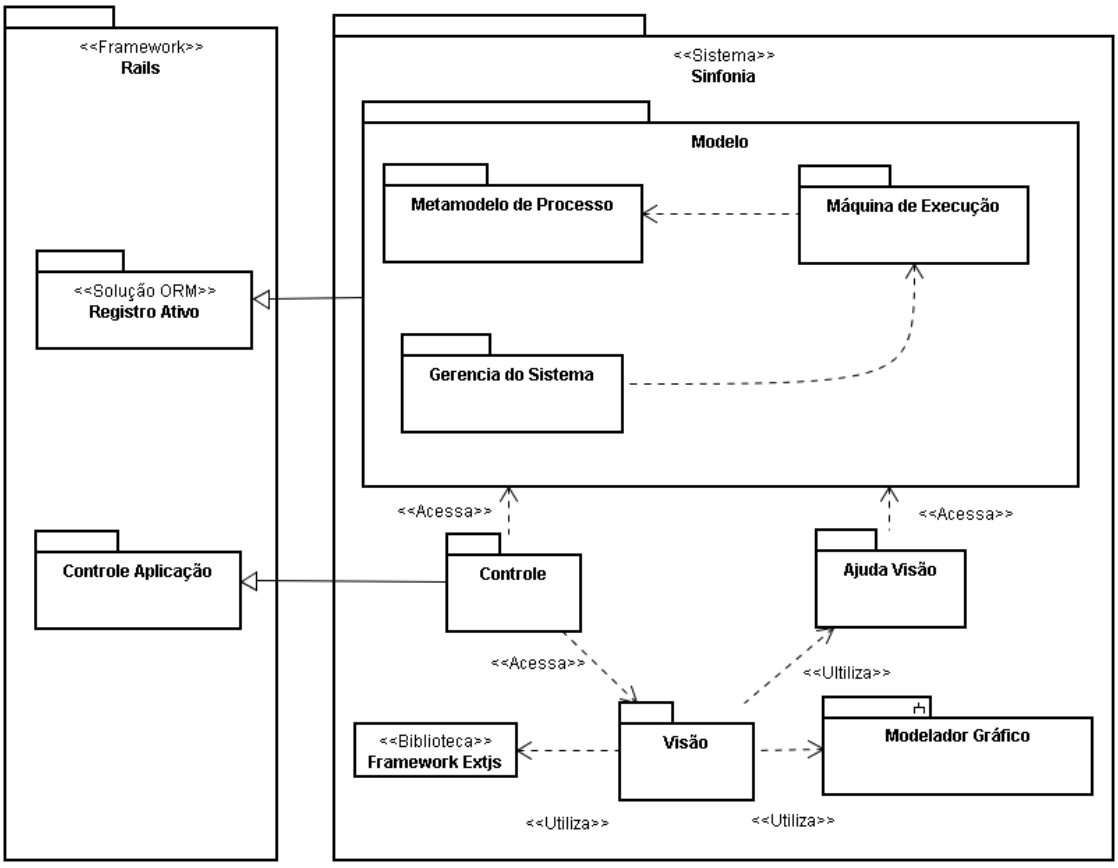


Figura 5.3: *Modelo de Pacotes de Sinfonia*

5.2 O Pacote Modelo

O pacote de Modelo foi desenvolvido com base em dois padrões arquiteturais Modelo de Domínio e Registro Ativo [17]. Um Modelo de Domínio é uma rede de objetos interconectados em que cada objeto representa algum conceito significativo, que pode ser tão grande quanto uma corporação, ou tão pequeno quanto uma linha em um formulário de pedidos [17]. Assim a lógica de um sistema complexo fica distribuída entre seus objetos, granularizando os problemas da lógica de domínio.

O padrão arquitetural Registro Ativo coloca a lógica de acesso aos dados no objeto do domínio. Dessa maneira todos os objetos sabem como ler e gravar seus dados no banco de dados [17]. Neste caso, utilizou-se a solução ORM (*Object Relational Mapping*) disponível pelo framework *Rails*, chamada de *Active Record*, que implementa a ideia do padrão Registro Ativo.

5.2.1 Pacote Metamodelo de Processo

A estrutura do pacote Metamodelo de Processo é ilustrada no diagrama de classes da Figura 5.4. Além de mostrar as associações entre classes, são apresentados apenas os

métodos mais importantes para o funcionamento da máquina de execução.

Todo elemento que pertence ao fluxo do processo herda da classe **ObjetoFluxo**. Logo, esta classe agrupa características comuns a todos os Objetos de Fluxo. Além disso, ela é uma classe abstrata que contém dois métodos: **executarAcao** e **acaoExecutada**. O primeiro método representa a ação que deve ser executada pelo OF e o segundo verifica se esta ação já foi concluída. Esta classe possibilita guardar um conjunto de operações e atributos comuns a todos os Objetos de Fluxo e obriga os elementos do processo a implementarem suas próprias ações.

Este tipo de abordagem auxilia na manutenção do metamodelo, facilitando a adição de novos tipos de objetos de fluxo. Caso seja necessário criar um novo tipo de OF, basta herdar da classe OF e implementar o comportamento deste novo objeto.

Entre as responsabilidades do **ObjetoFluxo** estão auxiliar a máquina de execução quanto à gerência do token e manipular as propriedades dinâmicas dos objetos. Os Objetos de Fluxo colaboram com a gerência da máquina de execução fornecendo informações sobre o token a eles relacionados. Algumas dessas informações incluem: se o OF faz papel de conector ou conectado na conexão em que o token passou; quais tokens já passaram por determinado OF; e se existe algum token no estado de espera no objeto.

Existem três classes que herdam de **ObjetoFluxo**: **Atividade**, **Porta** e **Evento**. A classe Atividade contempla funcionalidades que existem tanto no Processo quanto na Tarefa. Um exemplo disso é a classe **Anexo**, que permite que um Evento Intermediário seja relacionado tanto a Tarefas quanto a Processos. Como Atividade é uma classe abstrata, a responsabilidade de implementar os métodos do OF é repassada para suas classes filhas **Processo** e **Tarefa**.

A classe **Processo** implementa o método **executarAcao** fazendo com que a máquina de execução dê andamento ao fluxo do processo. A classe **Conexão** relaciona um OF (conector) a outro OF (conectado) em determinado processo. O atributo *padrão* desta classe determina se a conexão dá continuidade ao fluxo, caso nenhuma regra relacionada à conexão retorne resultado verdadeiro.

A classe **Regra**, relacionada à conexão, é utilizada para tomada de decisão nas portas Inclusiva e Exclusiva, guiando a execução do fluxo.

A classe **Tarefa** implementa o método **executarAção** de acordo com seu tipo. Por exemplo, tarefas do tipo usuário apresentam uma interface de interação com o usuário determinada pela classe **Interface**; já tarefas do tipo Script executam um script predefinido. Além disso, esta classe tem a responsabilidade de alocar os participantes de uma tarefa. A classe **AtividadePapel** é responsável por armazenar as responsabilidades, as respectivas quantidades necessárias para executá-las, além de indicar a interface com a qual o participante interage.

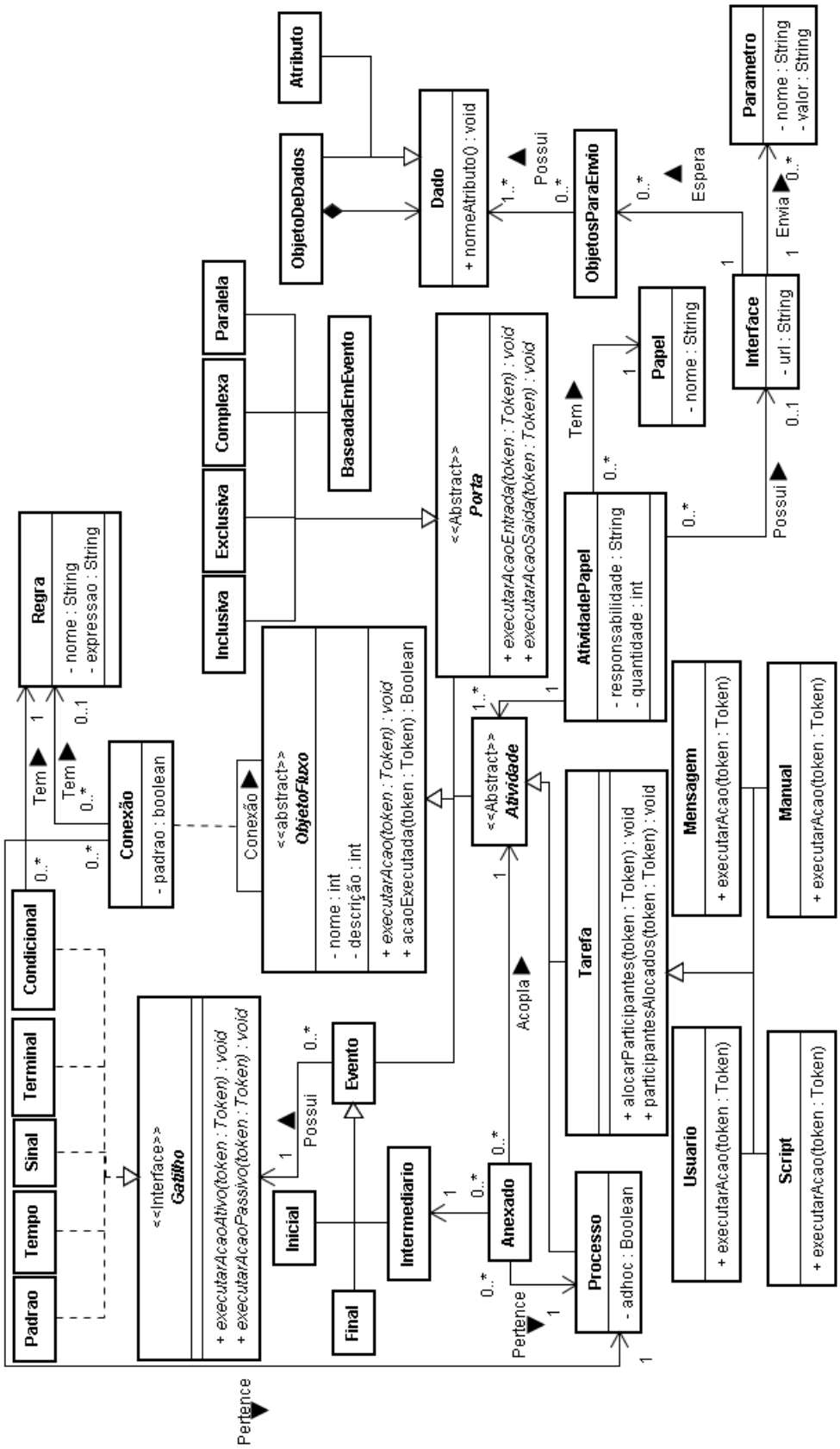


Figura 5.4: *Visão Lógica do Metamodelo de Processo*

Interface é uma classe que faz a comunicação externa com outros sistemas. O atributo URL indica o caminho para o serviço que deve ser acessado. Este serviço pode estar em qualquer servidor de aplicação. A classe **Parametro** é utilizada para passar os dados do domínio de Sinfonia para um servidor externo. O atributo *nome* representa o identificador do parâmetro esperado pelo servidor. O atributo *valor* é um script que, no momento do acesso, é executado por Sinfonia e retorna um valor, geralmente relacionado a seu domínio. Este valor é passado para o servidor externo.

Além de possiblitar a passagem de parâmetros pertencentes ao domínio do software Sinfonia aos servidores externos, a Interface permite enviar um conjunto de dados que determinam as informações que o sistema espera como retorno do servidor externo. Estes dados são representados pela classe **Dado**. Relacionados a ela estão as classes **ObjetoDeDados** e **Atributo** formando o padrão Compositor [20]. Através destas três classes é possível expressar um domínio de dados compostos.

Com o objetivo de facilitar a manutenção do metamodelo, a classe **Porta** foi implementada como abstrata. A **Porta** possui dois métodos abstratos, **executarAcaoEntrada** e **executarAcaoSaida**, que determinam seu comportamento quando o token chega e quando o token sai da porta, respectivamente.

Para incluir um tipo novo de porta no modelo, basta herdar da classe **Porta** e implementar os métodos que determinam seu comportamento. As classes filhas de **Porta** são: **Inclusiva**, **Exclusiva**,**Complexa**, **Paralela** e **BaseadaEmEvento**. Todas estas classes implementam estes métodos de acordo com seu comportamento.

O comportamento da classe **Evento** é determinado pelo tipo do seu gatilho. A interface **Gatilho** possui os métodos **executarAcaoAtivo** e **executarAcaoPassivo**. O primeiro método é utilizando quando o evento é do tipo lançador, enquanto o outro método é disparado quando o evento é do tipo receptor.

Se for necessário adicionar outros tipos de gatilho, basta implementar a interface gatilho. Os tipos do evento determinam qual método chamar. A classe **Inicial** chama apenas o método **executarAcaoPassivo**, pois é do tipo receptor, enquanto a classe **Final** invoca os métodos ativos, pois o evento Final é do tipo lançador. Finalmente, a classe **Intermediária** dispara os dois métodos, tanto ativo quanto passivo, de acordo com o tipo do evento Intermediário.

Exemplo de Representação de Processo pelo Metamodelo

A Figura 5.5 descreve um processo abstrato para demonstrar a instanciação de alguns conceitos do metamodelo.

O processo inicia com um Evento Inicial que segue para Tarefa 1. Esta tarefa é conectada com uma porta exclusiva que muda o fluxo de acordo com as regras de conexão. A conexão que utiliza a regra “Sim” leva à Tarefa 2, enquanto que a conexão que utiliza a

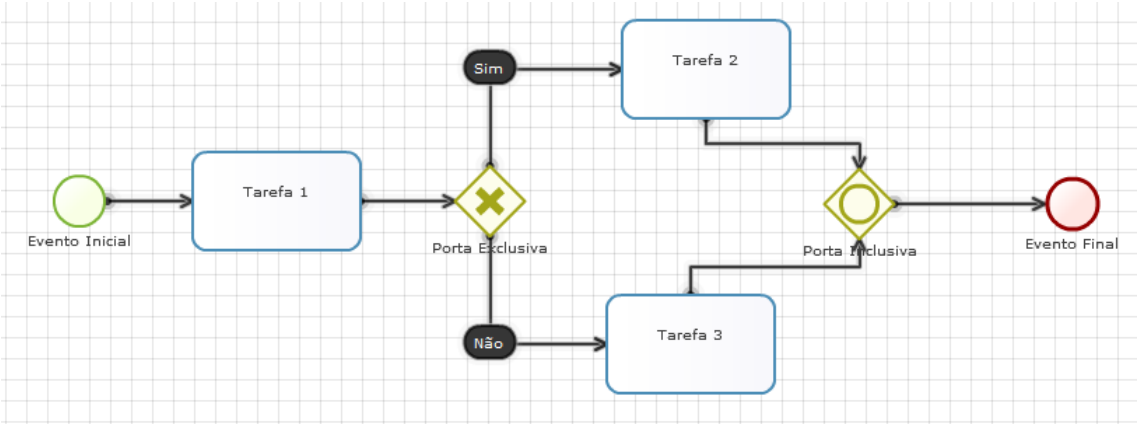


Figura 5.5: *Exemplo de Processo Definido*

regra "Não" segue para Tarefa 3. Ambas tarefas são contectadas com uma porta inclusiva. Esta porta sincroniza os fluxos e envia o Token para o Evento Final, onde ele é consumido.

A Figura 5.6 apresenta um diagrama de objetos que ilustra este processo simples utilizando o metamodelo de processo.

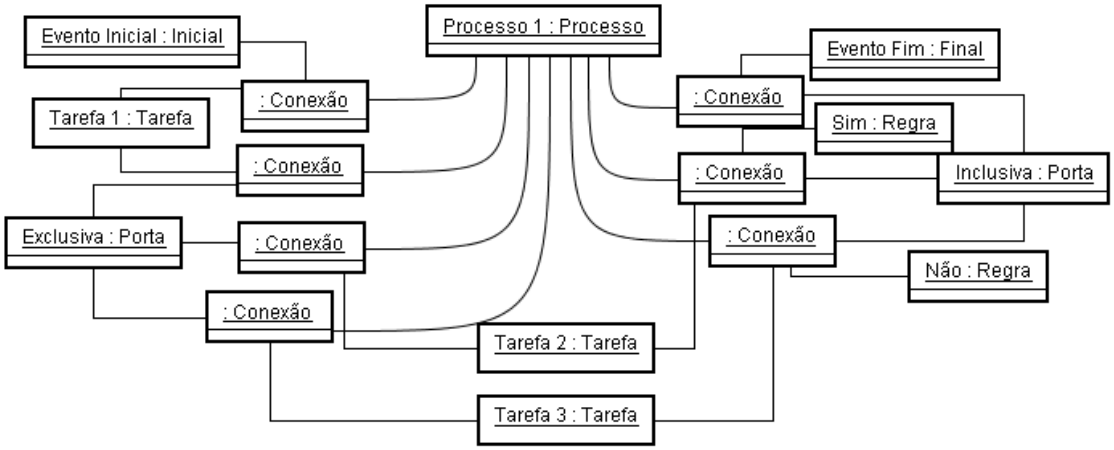


Figura 5.6: *Diagrama de Objetos do Processo Abstrato*

Todos os OF são ligados por objetos de Conexão. As regras ("Sim" e "Não") também são associadas a conexões. Isso significa que o token passará por elas somente se suas regras retornarem verdadeiro quando avaliadas.

5.2.2 Pacote Máquina de Execução

Este pacote tem o objetivo de controlar as características e comportamento da máquina de execução. A Figura 5.7 exibe as classes que pertencem ao modelo da máquina de execução (em cinza), além de algumas classes que pertencem ao metamodelo de processos (em branco). Por clareza apenas as classes do metamodelo de processo que se associam diretamente às classes da máquina de execução são apresentadas neste modelo.

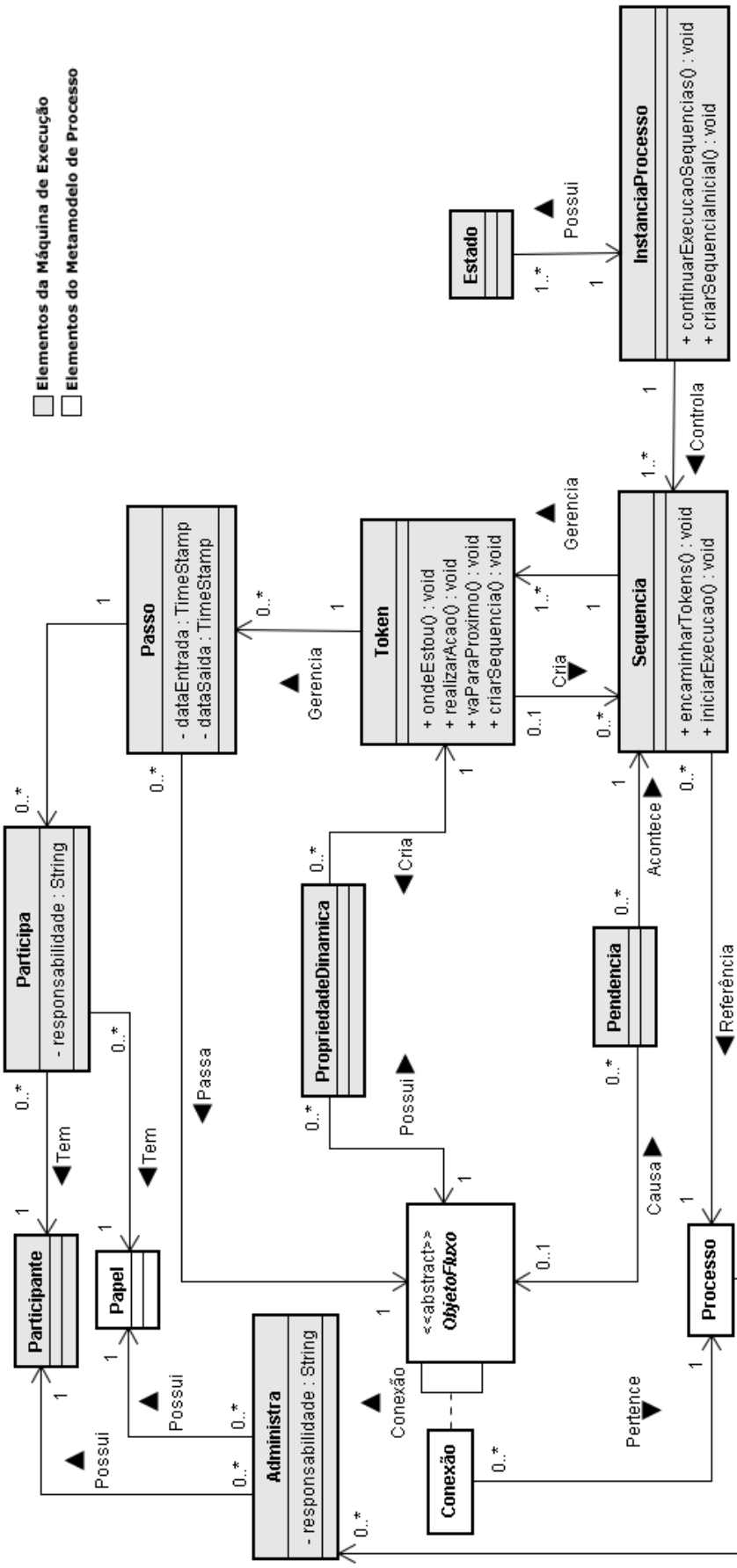


Figura 5.7: *Visão Lógica do Modelo de Máquina de Execução*

A classe **InstanciaProcesso** define o processo em execução e controla seu andamento. Durante a execução do processo, a instância passa por vários estados, conforme discutido no Capítulo 4. Os dados relacionados à situação na qual encontra-se a instância são gerenciados pela classe Estado.

A classe **InstanciaProcesso** não possui associação direta com o processo; este relacionamento é estabelecido através da classe **Sequencia**. Assim que a instância do processo é criada ela origina uma sequência, através do método **criarSequenciaInicial**. Esta sequência dá início ao andamento do processo.

InstanciaProcesso administra todas as outras sequências derivadas da sequência inicial. Estas sequências são provenientes dos processos internos pertencentes a um processo pai.

Uma sequência pertence a uma única instância de processo e referencia somente um processo, o que determina qual processo está em andamento.

A maior responsabilidade da classe Sequência é criar e gerenciar tokens. Uma sequência pode originar vários tokens. Há casos em que o token gerado por uma sequência cria outra sequência. Isto acontece quando o token chega a um processo interno.

A classe **Token** é responsável por rastrerar o fluxo entre os OF do processo, criar as propriedades dinâmicas pertencentes ao OF; lançar as pendências; criar as sequências dos processos internos; e executar a ação do OF.

Quando o Token transita pelos OF do processo, ele cria e administra instâncias da classe **Passo**. A responsabilidade desta classe é marcar a transição que o token faz entre os OF. Para isso ela guarda as datas e horas de entrada e saída das movimentações realizadas pelo Token. Para saber em qual OF o token está em uma dado momento, basta verificar o último passo do Token.

Quando o token executa a ação de uma tarefa que envolve participantes, várias participações (instâncias da classe Participa) podem ser criadas. A classe **Participa** associa os participantes às tarefas que devem ser executadas, mantendo a data de vínculo entre o participante e a tarefa e o status da participação do envolvido na tarefa. Assim como a classe **Participa** relaciona os participantes à execução da tarefa, a classe **Administra** os associa à gerência do processo.

Para executar a ação dos objetos de fluxo, foi utilizado o padrão Estratégia, que muda o comportamento do objeto alterando o objeto para o qual ele delega as requisições. Portanto, o Token executa seu método **realizarAcao** e delega o comportamento deste método para o OF atual. Isso é feito através da invocação do método **executarAcao** do OF. Cada OF implementa este método de acordo com seu comportamento.

Quando é necessário manter um dado relacionado a algum OF e ao Token durante a execução do processo, o Token cria uma propriedade dinâmica. Como várias instâncias de processo podem usar o mesmo OF, este tipo de atributo não pode ser modelado através

de um atributo estático. Estas propriedades são tratadas utilizando o padrão de Reflexão [11] que provê um mecanismo para mudanças estruturais e de comportamento do sistema dinamicamente.

Para lidar com estes atributos, foi criado o conceito de propriedade dinâmica. As propriedades dinâmicas são atributos relacionados ao Token e a determinados OF criados em tempo de execução da instância do processo.

A classe **PropriedadeDinamica** representa os atributos relacionados ao OF e ao Token criados em tempo de execução do processo. Sua responsabilidade é armazenar o nome e o atributo do OF que deve ser armazenado de forma persistente. Estes atributos são guardados e recuperados do banco de dados. Eles são serializados para armazenamento, e na recuperação eles são deserializados e acoplados ao OF, alterando sua estrutura original.

Vale ressaltar que estas modificações são feitas em relação ao objeto, sendo assim, elas não modificam todas as classes **ObjetoFluxo** do sistema. Alguns exemplos de propriedades dinâmicas são: a quantidade de instâncias criadas por uma tarefa múltipla-instância em determinado fluxo ou a próxima atividade a ser executada em um processo *adhoc*.

As exceções que podem acontecer durante o andamento do processo foram tratadas para causarem o lançamento de **Pendencia**. Portanto, assim que acontece algum problema no andamento do processo, uma pendência é acionada. A pendência coloca a instância do processo no estado "Com Pendência", paraliza sua execução e informa ao usuário o problema ocorrido e sua provável solução.

Exemplo de Execução de Instância de Processo

O funcionamento da máquina de execução pode ser ilustrado com o mesmo o processo utilizado no exemplo da seção anterior. A Figura 5.8 exibe o início da execução da instância do processo.

Quando o usuário executa a instância do processo, ela dispara o método **executarSequencia**, percorrendo todas as sequências e fazendo com que elas dêem continuidade à execução do fluxo, caso ele esteja parado. Como a **sequenciaInicial** ainda não foi executada a **instanciaProcesso1** chama o método **iniciarExecucao** na classe **Sequencia**. Este método procura por elementos que possam iniciar o processo. Neste caso o objeto encontrado para iniciar o processo é o **eventoInicial**.

Depois de descobrir um elemento para iniciar o fluxo, a **sequenciaInicial** cria um token com origem no **eventoInicial** chamado **token1**. Logo depois de posicionar o token1, a sequência dispara o método realizarAcao da classe Token. Então, o token1 invoca o método **executarAcao** do eventoInicial com o intuito de realizar o comportamento do evento.

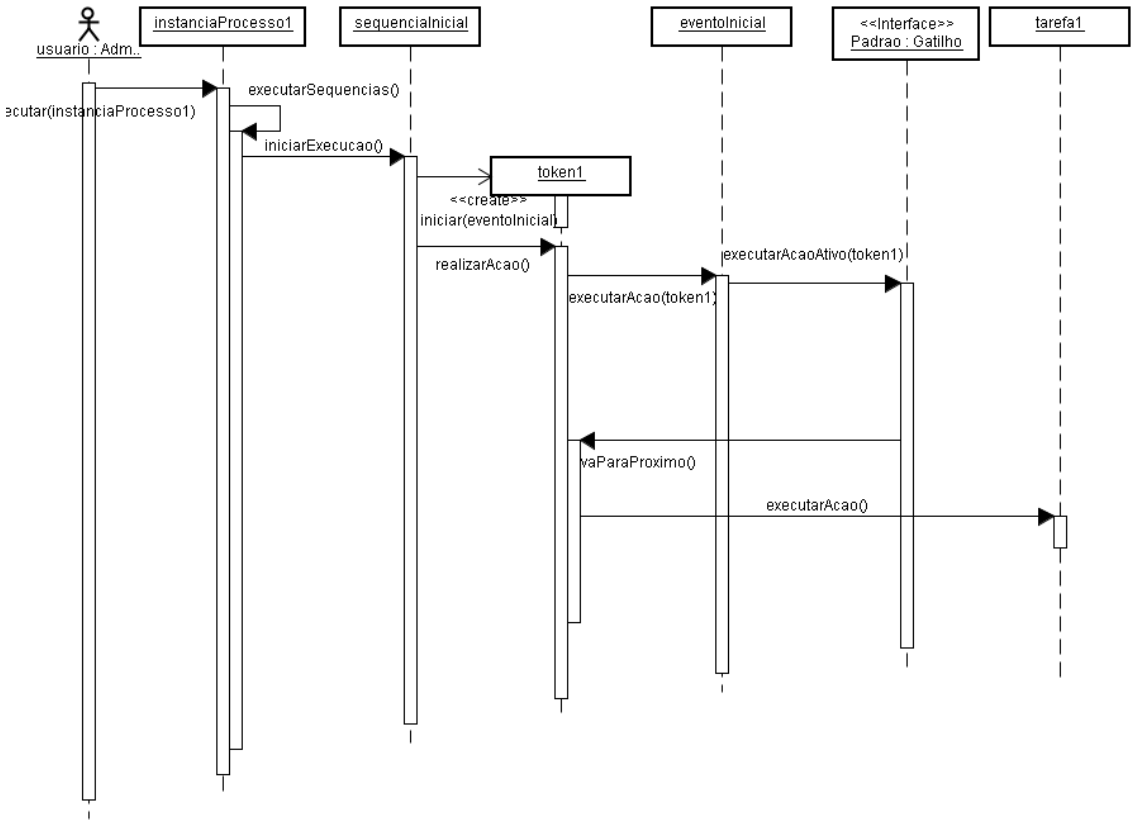


Figura 5.8: *Comunicação no Início da Execução do Processo*

O **eventoInicial** dispara o método **executarAcaoAtivo** do gatilho que está nele. Como seu gatilho é padrão, ele simplesmente chama o método **vaParaProximo** na classe token. Este método analisa o processo no qual o token está, qual seu OF atual e localiza o próximo OF do processo. Então, o token é direcionado para o OF posterior.

O próximo elemento do fluxo é a **tarefa1**. A **tarefa1** pode se comportar de várias formas diferentes, desde alocar participantes para sua realização até executar um simples script. Por este motivo, a execução deste método não está detalhada neste diagrama, já que os diagramas aqui descritos focam na execução do fluxo. A Figura 5.10 dá continuidade à execução do processo.

Supondo que **tarefa1** seja do tipo usuário e que vários participantes são alocados para cumprirem-na, quando o último envolvido na tarefa cumprir sua parte na execução, a **instânciaProcesso1** invoca o método **executarSequencia**. Assim, é dada continuidade a todas as sequências do processo.

Consequentemente, para cada instância de **Sequencia** pertencente a **instancia-Processo1** é disparado o método **encaminharTokens**. Este método faz com que todos os tokens da sequência sejam encaminhados para o próximo elemento do processo, caso a ação do elemento atual tenha sido concluída. Com a conclusão da **tarefa1**, o **token1** transita para o próximo elemento da sequência, neste caso, a porta exclusiva.

Quando o **token1** chega à porta exclusiva ele dispara o método **executarAcao**.

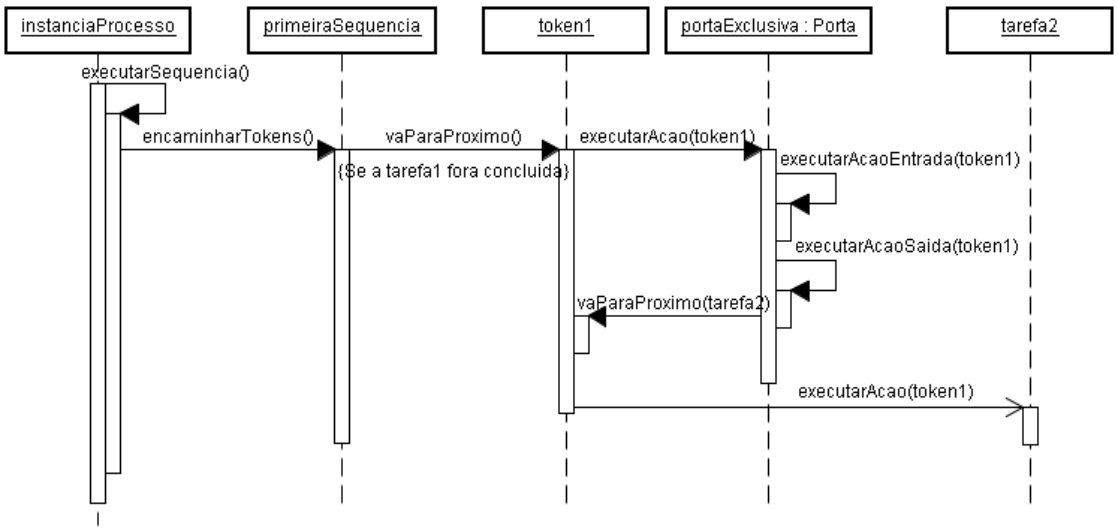


Figura 5.9: *Comunicação Durante a Execução do Processo*

A **portaExclusiva**, por sua vez, executa duas ações: uma ação assim que o token chega nela e outra quando o token a deixa. O método **executarAcaoSaida** analisa as conexões da **portaExclusiva** e, baseado nos dados inseridos pela **tarefa1**, toma a decisão de qual fluxo seguir.

Neste caso, o fluxo escolhido para dar continuidade ao andamento do processo foi o caminho que leva à **tarefa2**. A continuação do fluxo é exibida pela Figura 5.10. Após a execução da **tarefa2**, novamente a **instanciaProcesso** aciona o método executarSequencias que, por sua vez, dispara o método **encaminharTokens** do objeto **primeiraSequencia**. Ao invocar este método, o **token1** é encaminhado para **portaInclusiva** e invoca seu método **executarAcao**.

O método **executarAcao** dispara **executarAcaoEntrada**. Como é uma porta inclusiva, o método verifica se deve aguardar a chegada de mais algum token para prosseguir com o fluxo. Neste caso, nenhum token é aguardado, portanto o **token1** segue para o **eventoFinal**.

Já no **eventoFinal**, o token invoca a ação **executarAcao** do **eventoFinal**. Esta ação consome o **token1** finalizando-o. Como a **primeiraSequencia** não possui mais tokens ativos, ela é finalizada. Assim como a sequência, a **instanciaProcesso1** não possui mais sequências ativas, por isso, ela também é finalizada. Com isso, a execução da **instanciaProcesso1** é concluída.

5.2.3 Pacote Gerência do Sistema

Este módulo é responsável por gerenciar os participantes, controlar suas permissões de acesso e compartilhar OF. Os conceitos deste pacote são apresentados pela Figura 5.11.

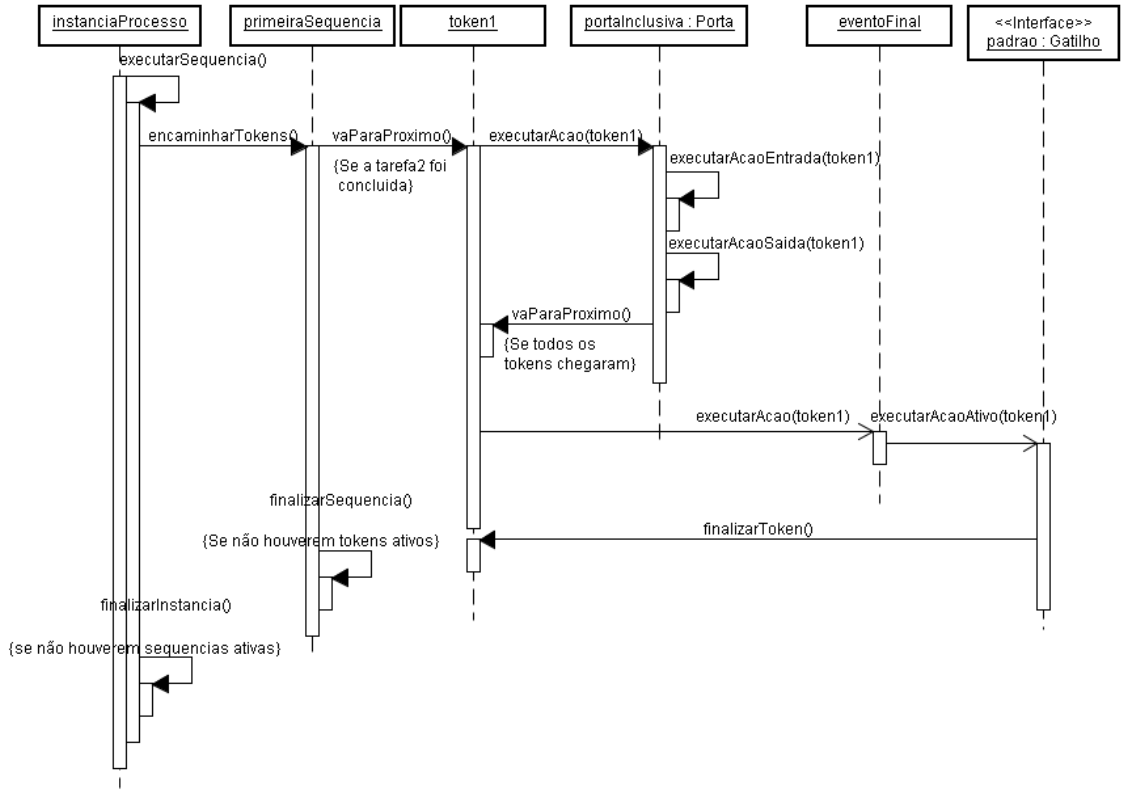


Figura 5.10: *Comunicação no Fim da Execução do Processo*

A gerência dos participantes compreende as ações básicas de inserir, alterar e desativar um participante. Todo participante precisa informar pelo menos um nome, email e uma senha. O usuário entra no sistema informando seu email e sua senha. A senha é criptografada por motivos de segurança. Ao cadastrar um participante, é possível definir quais papéis ele está apto a assumir. Um participante pode assumir vários papéis diferentes.

Existem dois conceitos responsáveis por controlar o acesso do usuário ao sistema, são eles: ação do controlador e controlador. Durante a implementação eles foram convertidos em duas classes distintas: Controlador e Ação.

As classes do pacote Controlador são responsáveis por determinar qual ação é executada de acordo com o URL e o verbo do cabeçalho HTTP. Esta ação deve ser codificada na própria classe através de um método para que seja acionada.

Por exemplo, o controlador responsável por administrar as ações relacionadas a tarefa, **TarefaControlador**, possui um método denominado listar_tarefas_pendentes. Este método retorna ao solicitante todas as suas tarefas pendentes. Esta ação seria acionada por um URL igual a “sinfonia_tarefas_pendentes” utilizando o verbo GET no cabeçalho HTTP.

A metaclasse **Controlador** guarda o nome da classe do controlador, um nome que a identifica e sua descrição geral. Para cada método criado no controlador de tarefa,

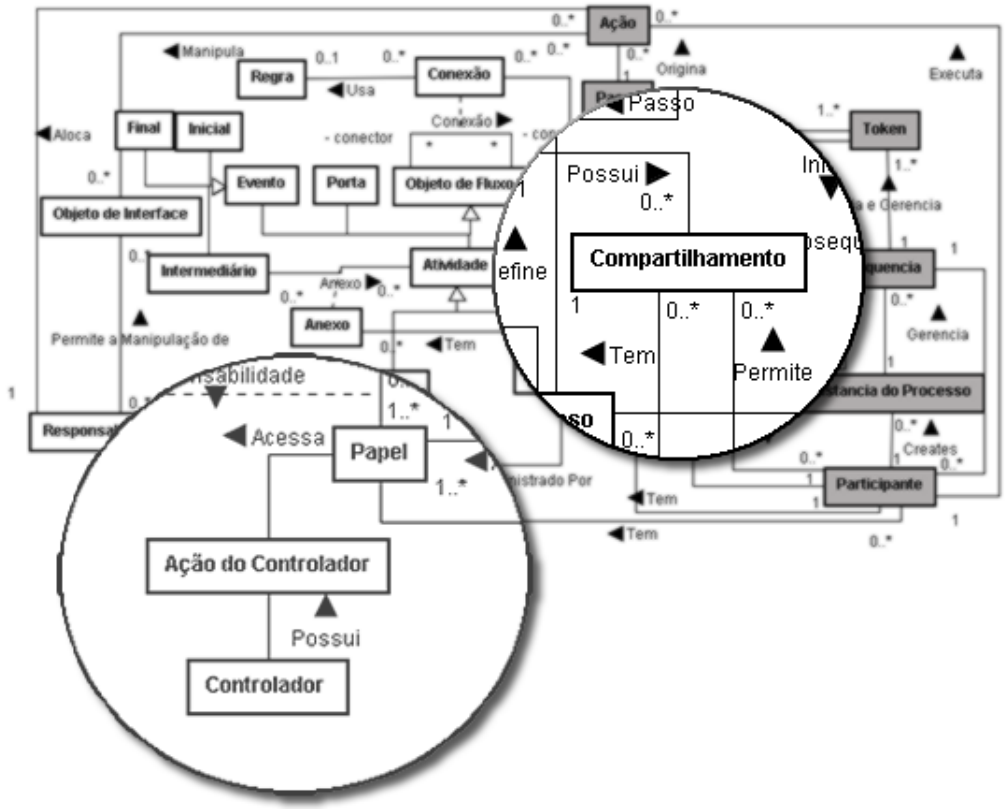


Figura 5.11: *Conceitos da Administração*

ou em qualquer outro, uma instância de **Acao** deve ser relacionada à metaclasse **Controlador**. Portanto, a classe **Acao** fica responsável por armazenar o nome do método do controlador, um nome que o identifique e um texto que descreva a lógica de domínio executada pelo método. Utilizando esta estrutura, é possível mapear todos os controladores e seus métodos de forma persistente em um banco de dados relacional.

Sinfonia permite fazer a varredura de todos os controladores do módulo **Controlador** e relacioná-los a seus métodos automaticamente. Ao executar esta funcionalidade, o sistema busca no pacote os nomes de todos os **controladores** e, para cada um, é criada uma instância da metaclasse **Controlador**. Depois, todos os seus métodos são lidos, e para cada um, é criada uma instância de **Acao**. Todas estas instâncias são relacionadas a seus repectivos controladores.

Após relacioná-los, os métodos podem ser associados aos papeis. Estes são os mesmo papeis que podem ser assumidos pelos participantes. Portanto, através deles é possível determinar o nível de acesso de cada usuário do sistema. Este tipo de controle foi embasado em [21].

Para permitir ou restringir o acesso dos usuários, assim que um usuário entra no Sinfonia, o sistema busca todas as ações que estão relacionadas aos papeis que ele pode assumir. Assim, quando o usuário faz uma solicitação ao servidor para executar alguma ação, antes de executá-la o sistema verifica se o usuário possui acesso a ela. Caso ele

possua, a ação é realizada, se não um aviso é enviado a ele e a solicitada não é executada.

O conceito de Compartilhamento permite compartilhar Objetos de Fluxo entre usuários, possibilitando a modelagem colaborativa. Apenas os usuários que criaram os OF podem compartilhá-los. Existem três maneiras de compartilhamento: Alteração, Utilização e Visualização.

O compartilhamento do tipo **Alteração** permite que o usuário altere o objeto compartilhado, ou seja, possibilita visualização e alteração dos valores dos atributos dos OF e modificações na modelagem dos processos. Apenas o usuário que criou o OF pode removê-lo.

O compartilhamento de **Utilização** permite que o usuário use determinado Objeto de Fluxo em seus processos. O usuário que possui um OF com este nível de compartilhamento pode apenas visualizar seus dados e utilizar o OF compartilhado em seus processos.

O compartilhamento de **Visualização** autoriza apenas que o usuário veja o OF, ou seja, ele pode ver os dados do OF, mas não pode utilizá-lo em seus processos, nem alterá-lo. Entretanto, se um processo for compartilhado neste nível o usuário fica autorizado a visualizar seu fluxo.

Caso um elemento seja compartilhado no modo Alteração ou Utilização e, posteriormente, seja descompartilhado, o usuário com quem este objeto foi compartilhado não poderá mais vê-lo, alterá-lo ou utilizá-lo. Entretanto, isso não acarretará mudanças nos processos em que tal objeto foi utilizado, ou seja, o OF não é retirado dos processos que os utilizam.

5.3 O Pacote Controlador

Todas as solicitações que chegam através de requisições web são tratadas pelo pacote Controlador, que implementa o padrão Controlador de Aplicação [17].

Este padrão é um ponto centralizado para manipular navegação de tela e fluxo de uma aplicação. Sua responsabilidade é decidir qual lógica de domínio executar e a visão para exibir a resposta. Portanto, no pacote controlador existem vários controladores e cada um possui várias ações.

Para determinar qual ação executar, o controlador faz uso da técnica de comunicação denominada Transferência de Estado Representacional (REST) [16].

Existem várias maneiras de informar ao servidor o que fazer com os dados enviados a ele. Segundo Richardson e Ruby [59], uma maneira de enviar as informações do método em um serviço web é colocá-las no método do HTTP. Existem vários métodos HTTP, tais como: GET, PUT, DELETE e POST. Uma vantagem da utilização dos métodos HTTP é a padronização. Outra vantagem é a separação entre a lógica de domínio e a visão

exibida. Esta funcionalidade permite adaptar o software para se comunicar com outros sistemas. Como o controlador pode se basear no URL para fornecer a resposta, é possível trocá-lo, dependendo do formato do arquivo solicitado. Por exemplo, caso uma solicitação para inserir um usuário chegue ao servidor com final ".xml" o controlador pode exibir uma mensagem em arquivo xml, sem alterar a lógica de domínio.

Além disso, o padrão Controlador de Aplicação provê a base para que suas alterações se reflitam em todos os outros controladores. Esta característica é importante no controle de acesso do sistema, pois apenas alterando esta classe é possível aplicar o controle de permissão a todos os outros controladores.

Outra característica desta arquitetura é que a maioria das classes do pacote Controlador possui uma classe correspondente no pacote Modelo. O diagrama exibido pela Figura 5.12 apresenta a associação entre os dois pacotes, que promove a divisão de responsabilidades entre as classes.

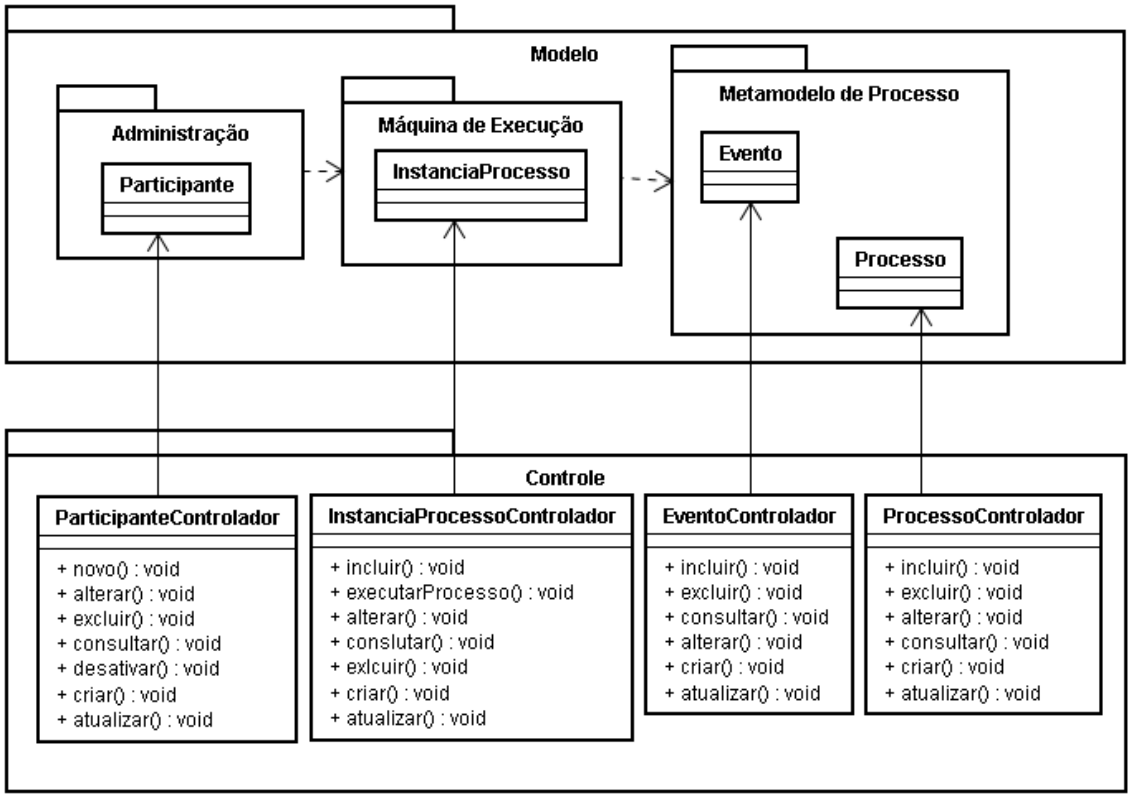


Figura 5.12: *Relação entre Modelo e Controle*

Cada método do controlador corresponde a uma ação. A maioria dos controladores possui métodos inserir, alterar, consultar e excluir. Os métodos atualizar e criar são responsáveis por apresentar a visão de alteração e inclusão. A partir destas visões os usuários podem entrar com os dados e inseri-los ou alterá-los no sistema.

5.4 O Pacote Visão

O Pacote Visão, diferentemente dos outros, não possui classes, mas sim arquivos de *script* que contêm a lógica da interface. A associação entre Visão e os demais pacotes é ilustrada pela Figura 5.13.

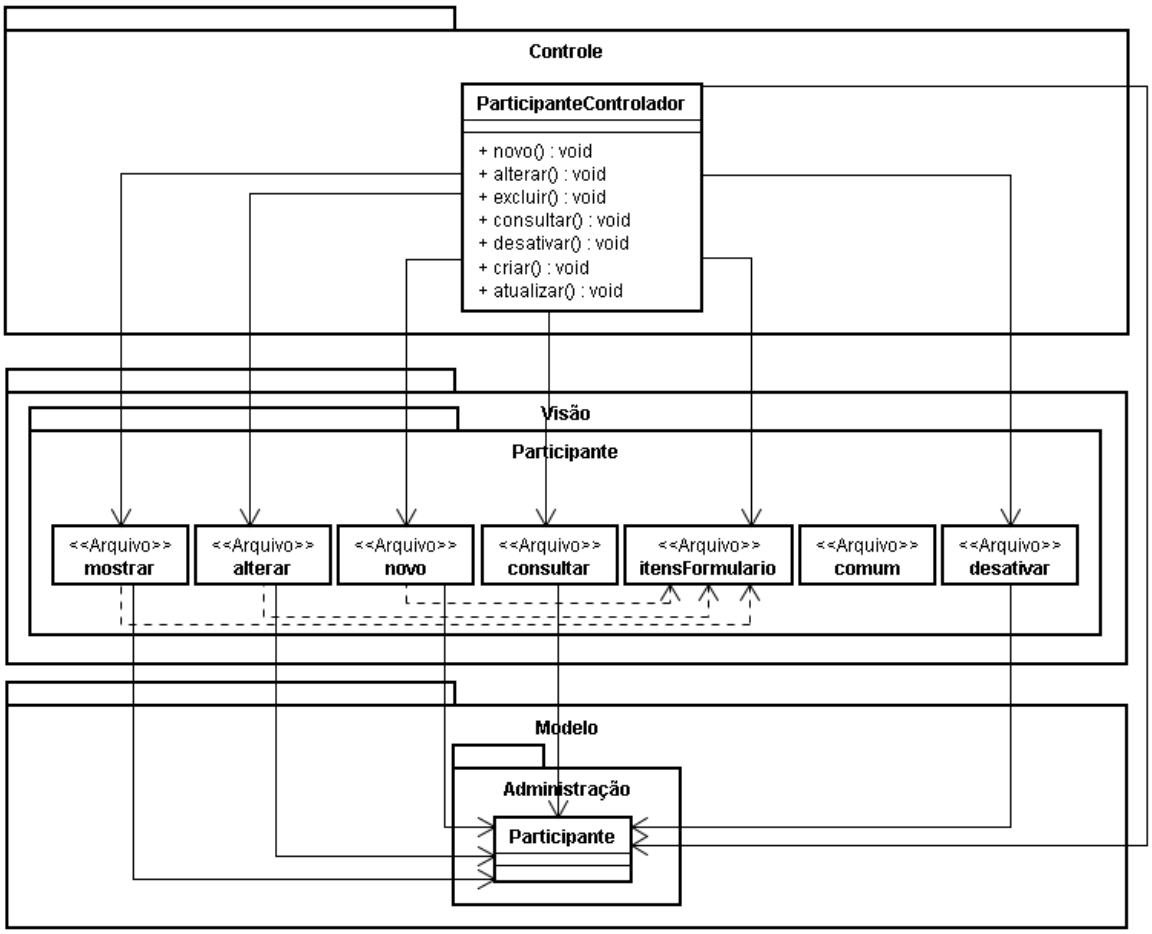


Figura 5.13: *Relação entre Visão, Modelo e Controlador*

Os métodos do controlador podem ou não utilizar uma visão. Nota-se que para alguns métodos do controlador (como novo, alterar e mostrar), existe um arquivo correspondente na visão. Este arquivo é utilizado para formatar a visão que é apresentada ao usuário, contendo os comandos que irão originar a interface gráfica.

Para os métodos que não possuem arquivos análogos na camada de visão, a resposta do controlador é apresentada como uma interface gráfica. Nestes casos, o controlador se comunica utilizando uma estrutura de dados *JavaScript Object Notation* (JSON) [14].

Por exemplo, quando o usuário necessita inserir um novo participante, primeiramente ele dispara o método *novo* da classe ParticipanteControlador. O controlador mostra a interface codificada no arquivo novo. Após a interação do usuário com a interface, este

submete os dados novamente ao controlador. Esta chamada é feita enviando os dados e passando o método POST do HTTP. Então, o controlador invoca o método *criar*, executa a lógica de criação do elemento, e responde com os dados formatados em JSON.

Nos dados formatados utilizando JSON, consta se a lógica de domínio foi executada corretamente e uma mensagem de sucesso ou falha. A interface é responsável por ler estes dados (a partir de JSON) e intepretá-los; então, ela apresenta a mensagem enviada pelo servidor ao usuário.

A Visão apresenta os dados do modelo através de uma forma adequada ao usuário. Para conseguir estes dados, muitas vezes ela precisa interagir com o Modelo. Para apresentar estes dados, são utilizadas duas tecnologias: o framework ExtJS [64] e Componentes em Flex [4].

Sinfonia apresenta duas maneiras de interação com o usuário: através de formulários eletrônicos e pela utilização da ferramenta de modelagem de processos (modelador gráfico).

Os formulários controlam os dados que preenchem o metamodelo criando, listando e alterando Objetos de Fluxo, e gerenciando objetos de interface, regras e conexões. Ademais, eles são utilizados para gerenciar a máquina de execução de processos, provendo interfaces para: instanciar e executar os processos; interagir com os processos em execução; gerenciar tarefas; e verificar processos com pendências. O controle de usuários, a gerência de permissão e o compartilhamento de Objetos de Fluxo também são realizados através dos formulários eletrônicos.

O modelador gráfico permite compor visualmente o fluxo do processo. Utilizando este componente, é possível montar um processo ligando graficamente seus elementos. Além disso, ele também permite acompanhar a execução do processo, visualizando seu andamento de forma gráfica, o que facilita o entendimento do usuário em relação ao cumprimento do fluxo do processo.

A Figura 5.14 ilustra a tela de Sinfonia. No canto superior esquerdo da interface constam as funcionalidades relacionadas à modelagem e execução de processos, e à participantes e permissões. Do lado esquerdo da tela é possível visualizar os elementos disponíveis para navegação e, no canto inferior esquerdo, um registro de *log* de acesso e interações com o usuário.

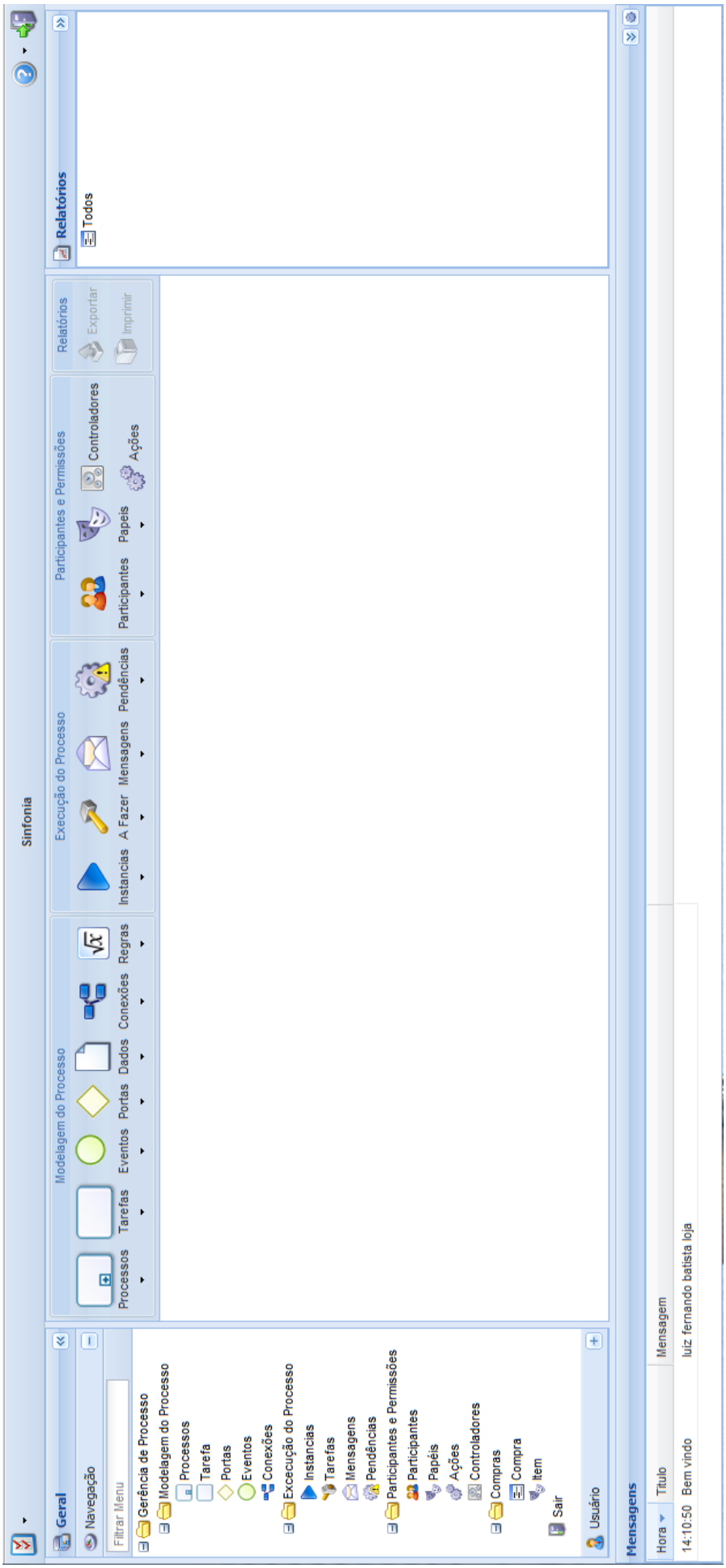


Figura 5.14: *Tela Inicial de Sinfonia*

O conjunto de telas que fazem inserção, alteração, remoção e consulta dos dados é padronizado, ou seja, as interfaces de todos os objetos que possuem estas funcionalidades possuem comportamento e forma consistentes [9].

5.4.1 Interface de Formulários

Formulários são interfaces que permitem instanciar o metamodelo de processo e gerenciar a máquina de execução. A Figura 5.15 ilustra a tela de consulta do OF Processo, que apresenta as seguintes partes principais:

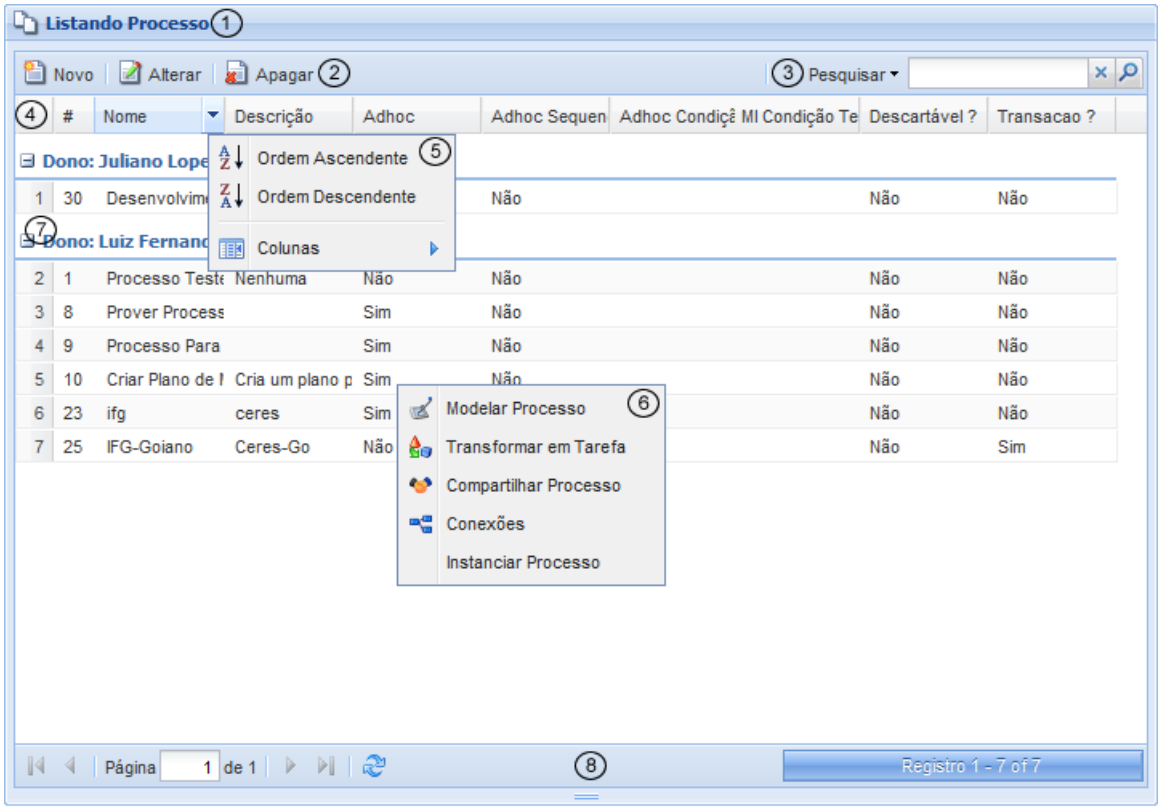


Figura 5.15: *Tela de Consulta de Processo*

1. **Título:** No topo de toda consulta aparece um título que detalha o OF que está sendo aprsentado. Neste caso, o OF consultado é Processo, portanto o título da consulta é "Listando Processo".
2. **Ações:** Na barra de tarefas da consulta são apresentadas as opções de incluir, alterar e remover. Para incluir um novo registro basta que o usuário clique em incluir. Já para alterar ou remover o registro do OF, o usuário deve selecionar o OF na lista e clicar no botão da ação desejada.
3. **Pesquisa:** Para encontrar os objetos desejados, o usuário conta com um campo de pesquisa. Ao digitar um texto na pesquisa e pressionar "Enter" ou clicar na imagem da lupa o sistema procura o texto digitado em todos os campos do objeto. Caso

o nome digitado seja correspondente ao valor (completo ou parcial) de qualquer campo, o registro é retornado através da grade de dados.

4. **Atributos:** Apresenta as definições de atributos de cada objeto. Neste caso, os atributos do processo são: nome, descrição, Adhoc, Adhoc Sequência e assim por diante.
5. **Ordenação:** Toda coluna pode ser ordenada na ordem ascendente e descendente, assim como pode ser suprimida da visualização da consulta.
6. **Menu de Contexto:** Para cada objeto consultado é apresentado um menu de contexto específico. Cada objeto possui opções distintas que são executadas no registro selecionado. As ações que podem ser executadas no processo são: acionar o modelador gráfico para aquele processo; transformar o processo em tarefa, removendo todas as suas conexões; compartilhar o processo para que outros usuários tenham acesso a ele; listar somente suas conexões; e instanciar o processo.
7. **Agrupamento:** Algumas consultas podem ser apresentadas tendo um atributo agrupador. Os OF são agrupados pelo nome de seus criadores. Como pode ser notado na Figura 5.15 existem dois donos, Luiz Fernando Batista Loja e Juliano Lopes de Oliveira. Ambos possuem processos que são listados abaixo de seus respectivos nomes.
8. **Paginação:** Todas as consultas são paginadas, ou seja, seus registros não são apresentados em sua totalidade. Eles são mostrados em conjuntos de 20 registros. O usuário pode usar a barra de navegação inferior para navegar entre os objetos apresentados.

Para todos os objetos do sistema é utilizado o mesmo padrão de tela de consulta. Algumas modificações (como título, atributos e apresentação da barra superior) são realizadas de acordo com o objeto. Por exemplo: caso seja consultado o OF Porta, então o título da consulta mudará para “Listando Porta”, os atributos da porta são listados e as opções do menu de contexto também são alteradas. Outra mudança comum é a supressão dos botões de adição, alteração e remoção, pois alguns objetos não podem ser manipulados pelo usuário (como, por exemplo, as “Pendências”).

A Figura 5.16 ilustra a tela de inclusão de Processo que é composta da seguinte forma:

1. **Título:** No topo da tela aparece o título, utilizado para contextualizar o usuário sobre o objeto que está sendo manipulado na tela.
2. **Abas:** Com o objetivo de diminuir a quantidade de campos apresentados na tela e , consequentemente, minimizar a sobrecarga visual do formulário, algumas telas usam o recurso de aba. Este recurso é utilizado agrupando campos que possuem informações relacionadas em abas. Os títulos das abas devem descrever a qual contexto o campo listado está associado.

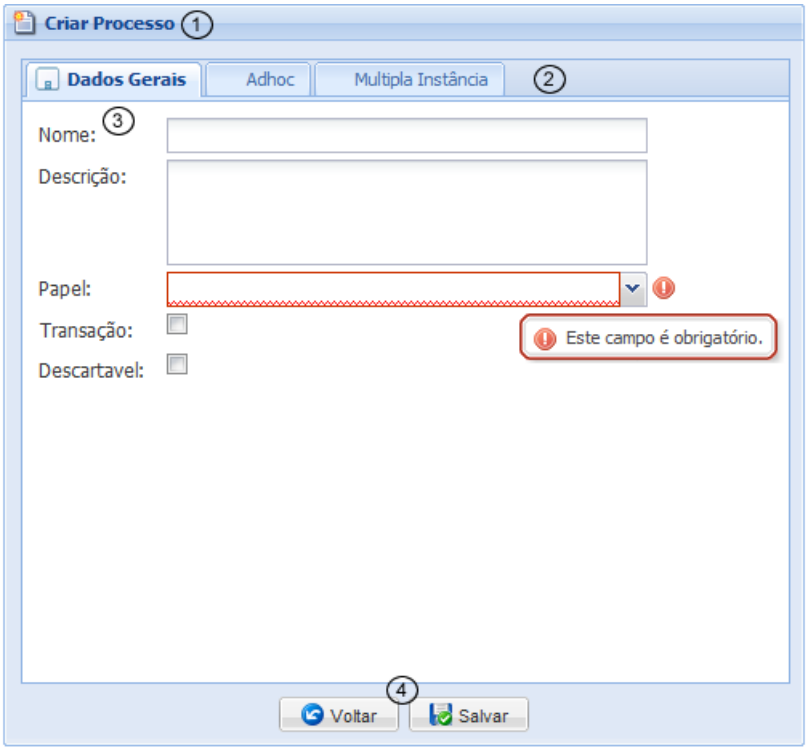


Figura 5.16: *Tela de Inclusão de Processo*

3. **Corpo do Formulário:** No corpo do formulário, são apresentados os campos que podem ser preenchidos. Os campos com um ícone de exclamação à sua direita indicam que houve alguma exceção no momento de preenchê-lo.
4. **Botões:** Na inclusão, são apresentados apenas dois botões, "Salvar" e "Voltar". O botão salvar grava as informações no sistema, enquanto que o botão "Voltar" retorna à tela anterior.

A única diferença entre alterar e incluir um objeto é que os dados do objeto já vêm preenchidos na alteração. Outra funcionalidade do sistema é a edição direta dos dados do objeto na própria consulta. A Figura 5.17 apresenta o modo de edição direta.

Diferentemente da inclusão e da alteração, a remoção de objetos não apresenta interface de preenchimento de dados. Ao definir que um objeto será removido, na tela de consulta, apenas uma caixa de dialogo de confirmação é apresentada.

Além de incluir, alterar, consultar e listar o usuário também pode analisar todo o objeto. Uma tela semelhante à tela de alteração é apresentada para o usuário com todos os dados do objeto, porém os campos são desabilitados para alteração.

As mensagens do sistema são apresentadas na parte superior central do navegador e gravadas no log do sistema, localizado na parte inferior da tela inicial. Algumas mensagens de log são interativas; por exemplo: quando um campo está incorreto a mensagem da exceção é apresentada no log; quando o usuário interage com esta mensagem ela o

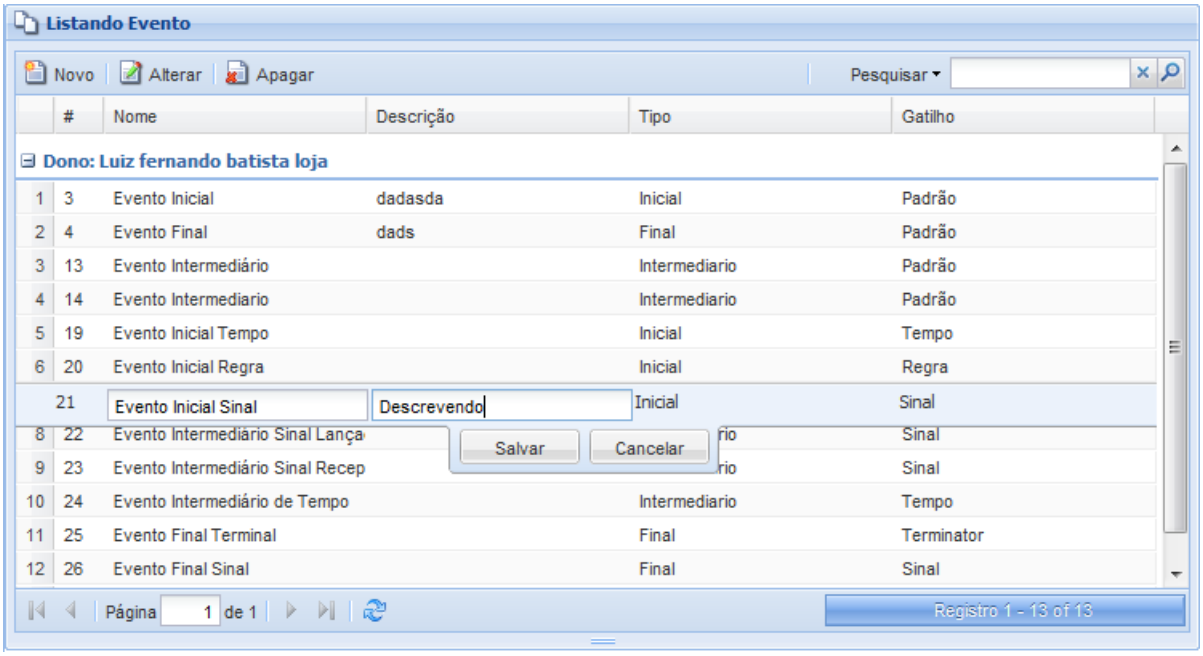


Figura 5.17: *Tela de Edição Direta dos Dados*

direciona para o campo que gerou a inconsistência. A Figura 5.18 mostra uma mensagem sendo exibida e seu log.

Com a finalidade de melhorar a usabilidade, algumas visões utilizam um padrão diferente do convencional. Um exemplo disso é a tela de compartilhamento de OF, ilustrada pela Figura 5.19. Esta tela é composta da seguinte forma:

1. **Analistas Disponíveis:** Neste contêiner são listados todos os usuários do sistema que não estão na lista de compartilhamento.
2. **Analistas com Compartilhamento:** Neste contêiner são listados todos os usuários que possuem algum tipo de acesso aos OFs que estão sendo compartilhados.
3. **Filtrar Analistas Disponíveis:** Ao digitar o nome do usuário ou parte dele, o sistema filtra entre os analistas que não possuem este tipo de compartilhamento.
4. **Filtrar Analistas com Compartilhamento:** Ao digitar o nome do usuário ou parte dele o sistema filtra entre os analistas que fazem parte deste compartilhamento.
5. **Listar Todos os Analistas:** Lista todos os usuários que não fazem parte do compartilhamento em forma de árvore.
6. **Alteração:** Lista os usuários que têm permissão de alteração no OF.
7. **Utilização:** Lista os usuários que podem utilizar o OF.
8. **Visualização:** Lista os usuários que apenas visualizam o OF.

Esta tela possibilita compartilhar objetos de fluxo com os usuários do sistema. A interação com ela é feita através da manipulação dos usuários, arrastando e soltando os usuários no tipo de compartilhamento desejado.

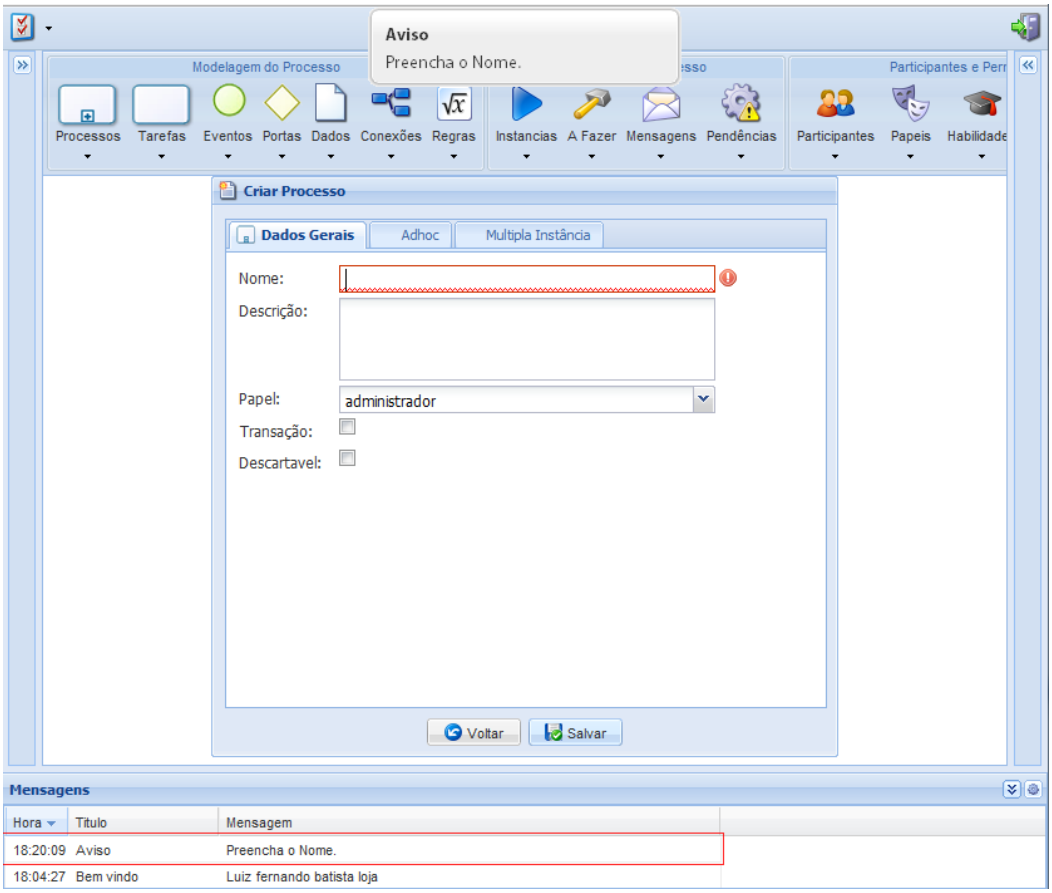


Figura 5.18: *Mensagem de Aviso e Log de Mensagens*

5.5 Modelagem Gráfica de Processos

A ferramenta de modelagem gráfica facilita a criação do fluxo do processo provendo suporte a funcionalidades de arrastar e soltar componentes, criar conexões através de uma interface visual de desenho e adicionar regras a estas conexões.

O componente foi desenvolvido como um subsistema de Sinfonia, por isso apresenta uma arquitetura própria. A Figura 5.20 ilustra os pacotes deste subsistema ferramenta.

O pacote **Modelo** é responsável por guardar os dados relacionados a objetos da máquina de execução e dos metadados de processos. Todas as classes deste pacote possuem uma classe correspondente no pacote Modelo utilizado por Sinfonia.

O pacote **ElementoGráfico** guarda toda a lógica de domínio envolvida no modelador gráfico. Ou seja, este pacote possui as classes responsáveis por desenhar os objetos e fazer as conexões gráficas entre eles.

O pacote **Visão** guarda componentes que são apresentados na estrutura inicial do diagrama, como a lista de objetos, lista de processos e menu superior.

O pacote **Comunicação** realiza o intercâmbio de dados entre o modelador gráfico e o software Sinfonia. Ele é responsável por fazer as chamadas ao servidor de

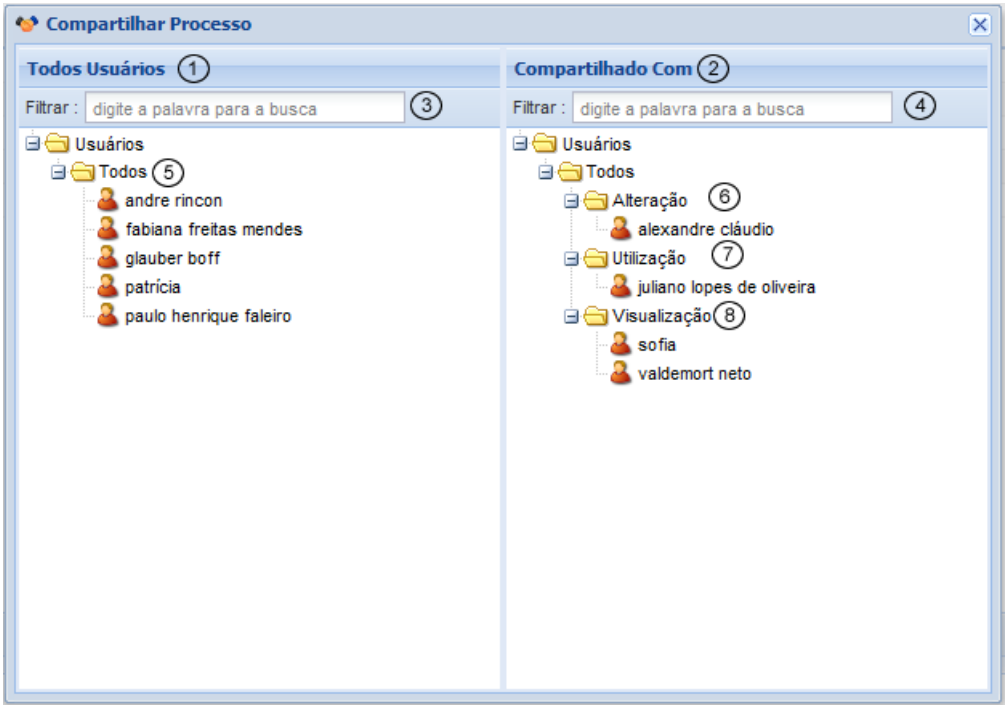


Figura 5.19: *Compartilhamento de Objeto de Fluxo*

aplicação.

A Figura 5.21 ilustra a interface do modelador gráfico que é composto pelos seguintes elementos:

1. **Barra Superior:** contêm opções para salvar as alterações feitas na modelagem do processo.
2. **Área de Desenho:** espaço reservado para desenhar o fluxo do processo.
3. **Barra Inferior:** lista todos os Objetos de Fluxo aos quais o usuário tem permissão de alteração e utilização. Estes Objetos de Fluxo obedecem as regras de compartilhamento definidas no sistema e podem ser Processos e Tarefas (3.1), Eventos (3.2) ou Portas (3.3) .

O pacote Modelo segue a mesma estrutura de seu pacote equivalente na arquitetura. Da mesma forma, o pacote Visão é constituído de componentes que formam a interface gráfica principal do modelador gráfico.

Para permitir o acompanhamento da execução do processo, a ferramenta gráfica possibilitando ao usuário visualizar o andamento dos tokens pelas conexões do processo. Cada token é representado por uma esfera acima do Objeto de Fluxo. Quando o objeto está em execução, o token fica acima deste objeto (Figura 5.22). Ao transitar pelos Objetos de Fluxo o token se movimenta seguindo a linha da conexão (Figura 5.23). Assim que o token é finalizado ele é pintado com a cor vermelha.

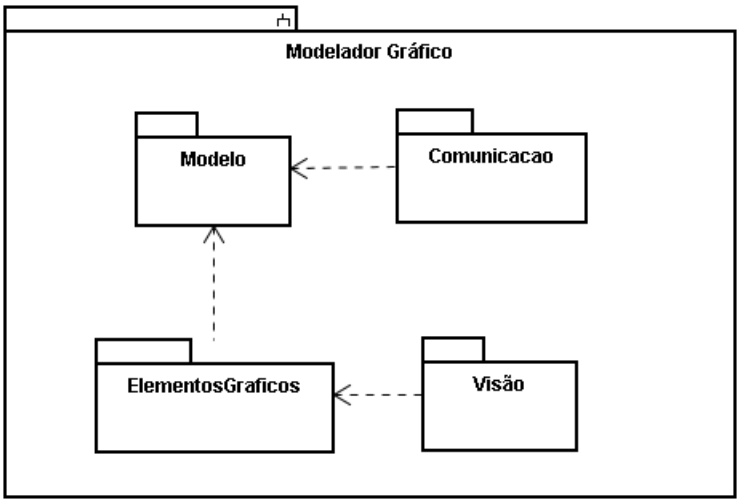


Figura 5.20: *Diagrama de Pacotes do Modelador Gráfico*

5.5.1 Elemento Gráfico

O pacote de Elementos Gráficos é composto pelos conceitos que podem ser apresentados visualmente pelo modelador. A Figura 5.24 apresenta a estrutura deste pacote.

A classe **Diagrama** representa o processo que está sendo modelado. Ela é responsável por carregar os componentes do processo, como objetos de fluxo e conexões, e apresentá-los de maneira visual.

A classe **NoBase** contém a lógica de movimentação e posicionamento dos elementos incluindo, a maneira como os elementos se comportam no diagrama, como eles se posicionam, em qual lugar do elemento as linhas das conexões devem ser apresentadas, e entre outros aspectos. Entretanto, **NoBase** não tem a responsabilidade de desenhar o elemento.

A classe **ObjetoFluxo**, que herda de **NoBase**, possui dois métodos abstratos: **desenharObjetoFluxo** e **elementosValidosParaConexao**. O primeiro método determina como o objeto de fluxo deve ser representado no diagrama e o segundo define os objetos com o quais o OF pode se conectar.

As classes **Atividade**, **Porta**, **Evento** herdam da classe **ObjetoFluxo** e implementam os dois métodos. Como processos e tarefas são representados graficamente de maneiras equivalentes não foi necessário criar uma classe para cada conceito.

A **Porta** implementa o método **desenharObjetoFluxo** e se baseia em seu tipo para determinar sua representação gráfica no diagrama. Assim como a **Porta**, a classe **Evento** também se baseia em seu tipo para desenhar sua representação visual. Entretanto, a responsabilidade do desenho do gatilho é passada para a classe **Gatilho**. Esta classe possui o método **desenharGatilho** que tem como parâmetro o tipo do gatilho (lançador ou receptor). Este método contém a lógica para desenhar graficamente o símbolo do gatilho.

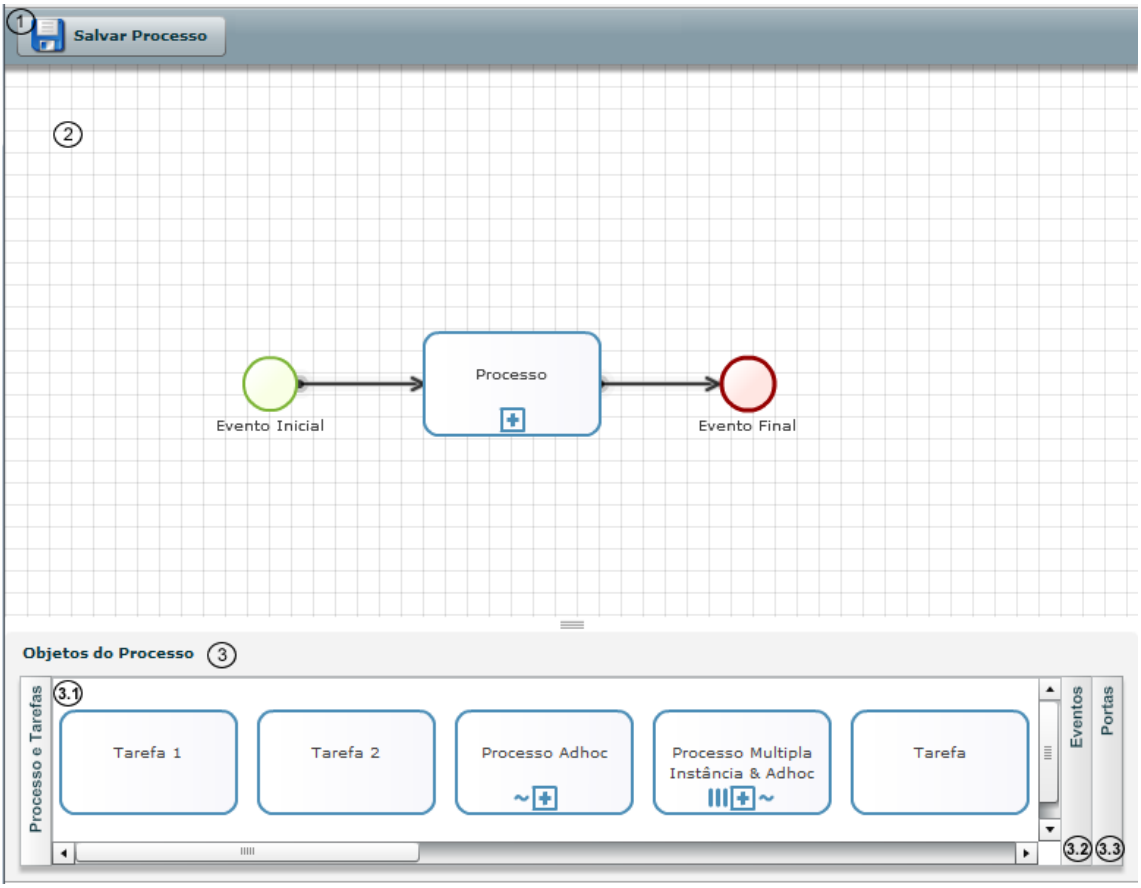


Figura 5.21: *Tela do Modelador Gráfico*

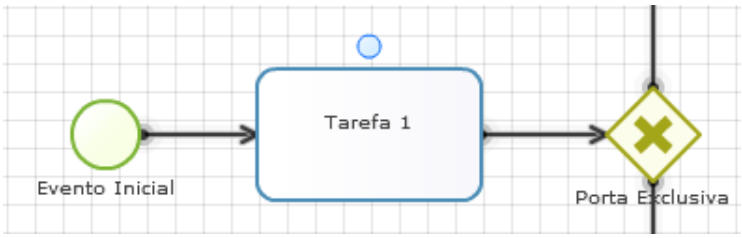


Figura 5.22: *O Token Acima da Tarefa 1 Indica sua Execução*

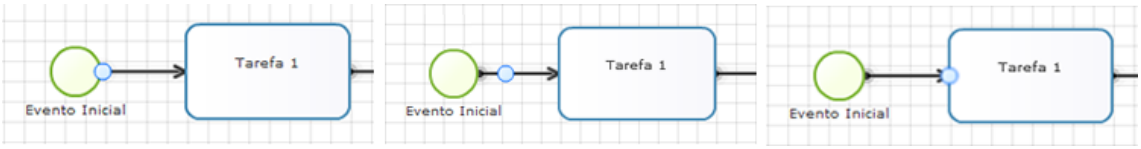


Figura 5.23: *Movimentação do Token do Evento Inicial para Tarefa 1*

A classe **Conexao** determina o comportamento das linhas existentes entre os OF. Nela, é definida quando a linha é direcionada e como a linha da conexão é representada visualmente.

A classe **Token** é utilizada para mostrar o andamento do token no fluxo do processo. A classe **tokenContainer** auxilia na visualização do andamento do token. Ela é um elemento gráfico que mostra os tokens agrupados acima do objeto de fluxo e contém

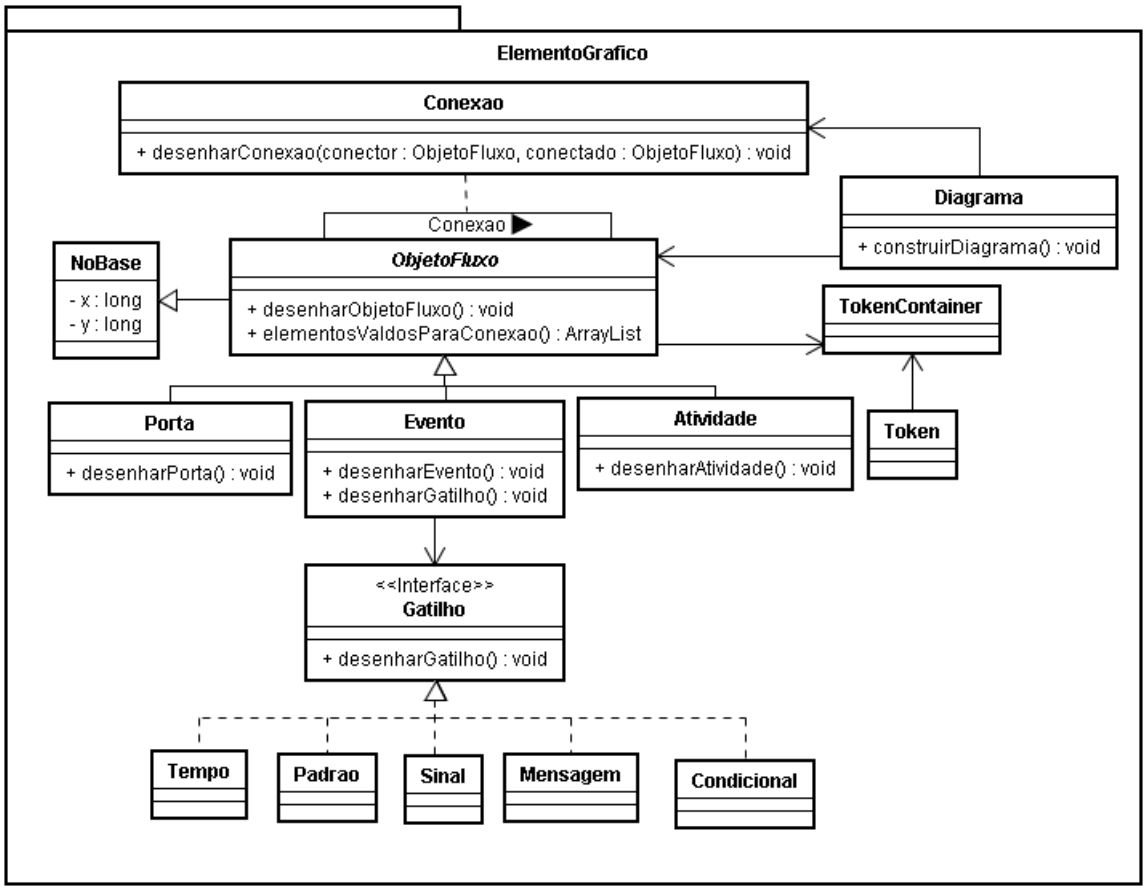


Figura 5.24: *Pacote de Elementos Gráficos*

a lógica da representação gráfica da entrada e saída do token de um OF.

5.5.2 Comunicação entre Modelador Gráfico e Sinfonia

O modelador gráfico troca mensagens com Sinfonia através de um protocolo de comunicação específico, denominado *Action Message Format* (AMF) [3], cuja finalidade é trocar dados entre aplicativos feitos em *Flex* ou *Flash* e servidores remotos.

A vantagem do protocolo AMF é que os componentes são serializados na aplicação web e deserializados pelo componente Flex e vice versa. Portanto, se um conceito for utilizado tanto na aplicação web quanto no modelador gráfico ele poderá ser mapeado no modelador. Logo, toda a comunicação entre Sinfonia e o Modelador Gráfico é feita através de objetos serializados. Sinfonia envia objetos serializados em ActionScript para o modelador gráfico, enquanto o modelador envia objetos serializados em Ruby para Sinfonia. Isso evita o grande custo de desenvolvimento para ler os dados enviados pela aplicação e convertê-los em objetos.

O pacote **Comunicacao** foi criado para auxiliar na troca de objetos entre Sinfonia e o Modelador Gráfico. **InterfaceComunicacao** é a classe responsável por fazer as

chamadas para o servidor web e receber os dados solicitados. Para desenvolver esta classe foi utilizado o padrão Fachada [20]. O modelo de classes deste pacote é apresentado na Figura 5.25.

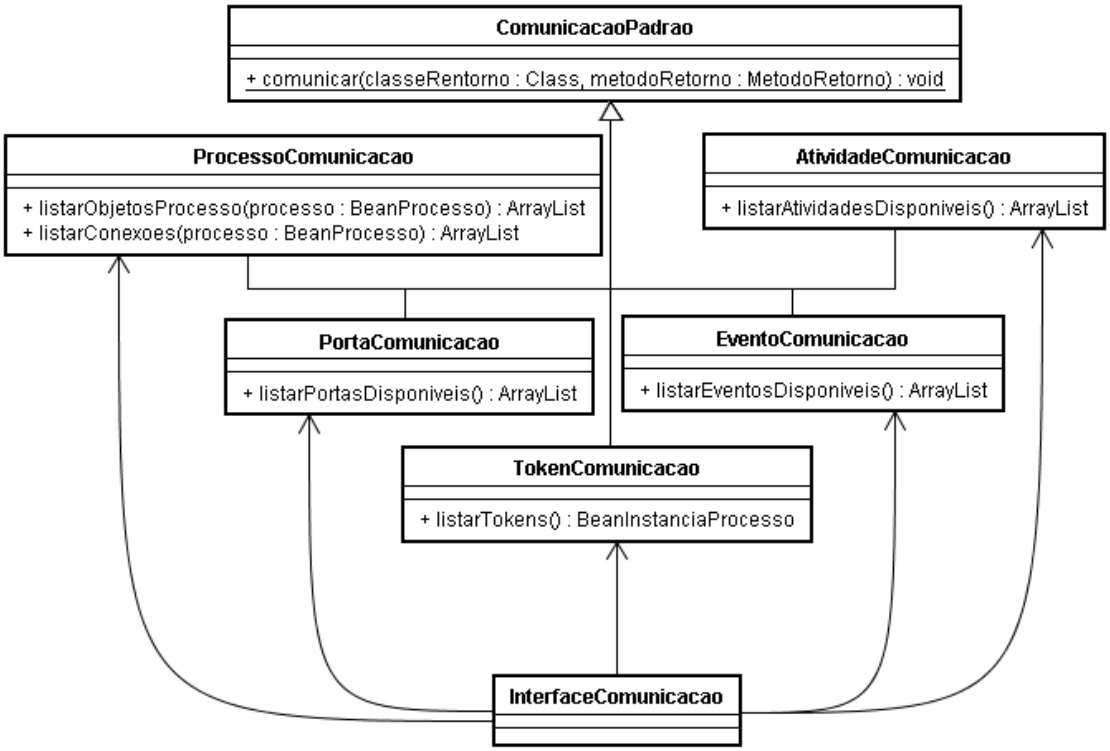


Figura 5.25: *Diagrama de Classes do Pacote Comunicação*

A classe **ComunicacaoPadrão** oferece um método para executar a troca de mensagens entre o software Sinfonia e o Componente. Os parâmetros deste método são: a classe do objeto que é retornado por Sinfonia; o objeto **MetodoRetorno** que contém o nome da ação que é invocada no servidor e os parâmetros que são passados para ele.

As classes **ProcessoComunicacao**, **PortaComunicacao**, **EventoComunicacao**, **AtividadeComunicacao** e **TokenComunicacao** possuem métodos de comunicação que retornam dados relativos a processo, porta, evento, atividade e token, respectivamente.

A **InterfaceComunicacao** utiliza cada uma destas classes para fazer a comunicação com Sinfonia. Um exemplo desta troca de mensagens é apresentado pelo diagrama exibido na Figura 5.26.

Após o usuário acionar o botão salvar, o modelador gráfico invoca o método da **InterfaceComunicacao** para salvar o processo. Este método dispara outro método na classe **ProcessoComunicacao** que envia o processo serializado para o software Sinfonia. O processo chega ao controlador de destino e dispara a ação **alterarProcesso** na classe **ProcessoControlador**. A lógica de negócio para modificar o processo é executada e o controlador responde se a operação foi bem sucedida ou não.

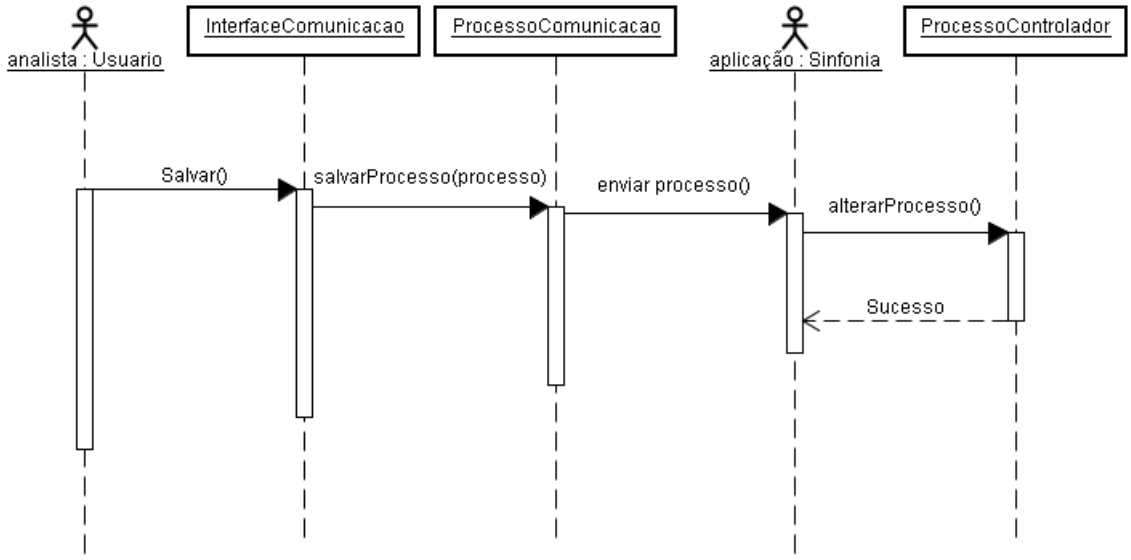


Figura 5.26: *Comunicação entre o Sinfonia e Modelador Gráfico*

5.6 Comunicação Externa

A comunicação externa feita por Sinfonia tem o objetivo de organizar o intercâmbio de dados com outros sistemas de acordo com o andamento do processo. Esta comunicação é realizada durante a execução das tarefas. Quando o usuário cumpre sua parte na tarefa e esta possui uma interface, então Sinfonia estabelece uma comunicação externa. O diagrama de atividades da Figura 5.27 ilustra esta comunicação.

Primeiramente, Sinfonia faz a autenticação no sistema com o qual se estabelecerá a comunicação. Para isso o usuário que executa a tarefa deve possuir permissão de acesso (senha e um nome de usuário, por exemplo) no sistema externo. As informações de autenticação são guardadas em Sinfonia e naturalmente devem ser válidos no outro sistema. Se a autenticação falhar, Sinfonia lança uma pendência. Caso contrário Sinfonia solicita ao Sistema Externo a visão com a qual o usuário deve interagir.

A solicitação da visão é enviada de acordo com a configuração do objeto **Interface** ligado à **Participacao**. Além disso, são enviados como parâmetro o identificador do usuário que está Sinfonia, o identificador da participação que originou a chamada externa, e os dados esperados como resposta.

Estes dados de resposta são baseados no objeto de dados associado à Interface. Eles servirão de auxílio para que o sitema externo saiba o que o software Sinfonia espera que ele retorne como resposta à solicitação.

A visão solicitada é apresentada para o usuário que deve preenchê-la. Após seu preenchimento os dados são enviados à aplicação que gerou a visão. Esta aplicação irá receber e processar os dados e, em seguida, enviá-los para Sinfonia com o identificador da participação que gerou a chamada. Caso aconteça alguma exceção na hora do proces-

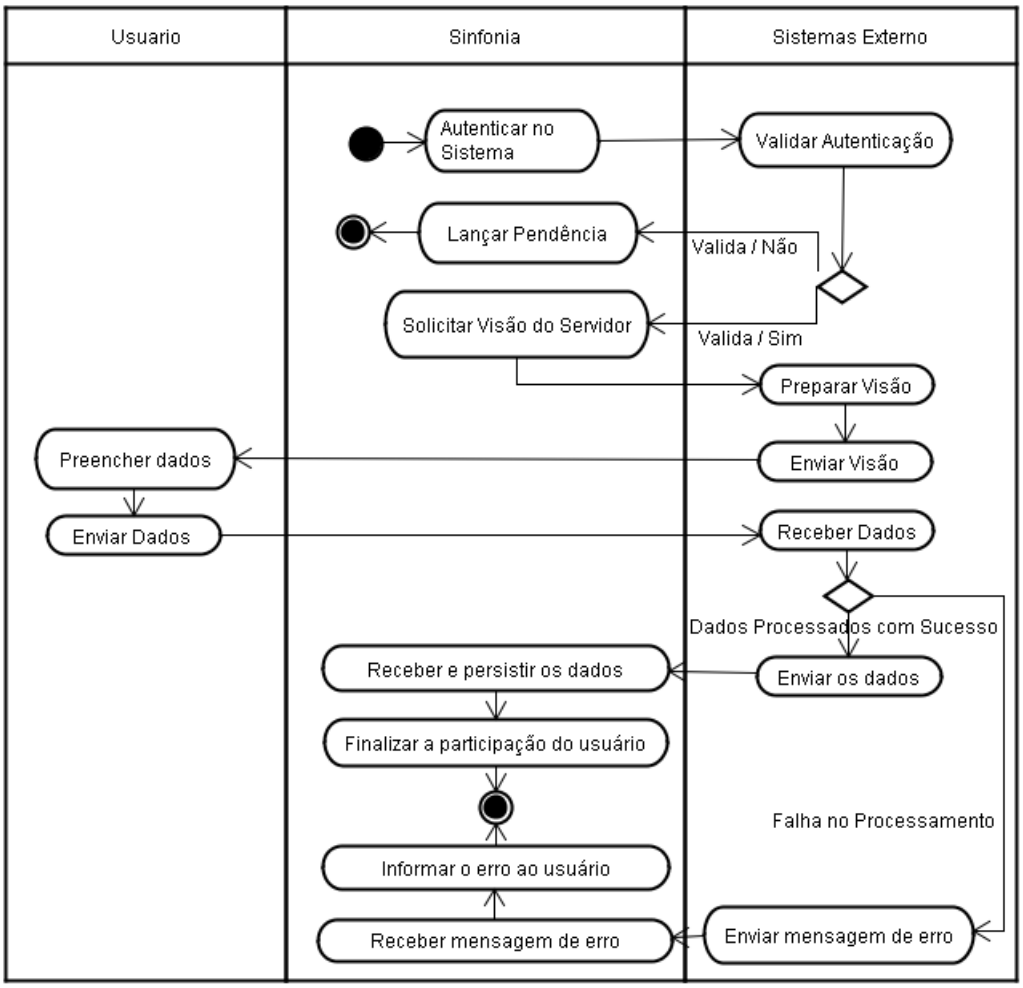


Figura 5.27: *Comunicação entre Sinfonia e outras Aplicações*

samento dos dados pelo sistema externo, ele enviará o erro ao Sinfonia que o informará ao usuário e não finalizará a tarefa.

Sinfonia recebe, serializa e persiste os dados junto à entidade de participação responsável por criar a chamada externa. Posteriormente estes dados podem ser recuperados para tomada de decisão no processo executado dentro de Sinfonia.

A classe **ObjetoContexto** pode recuperar os dados efetuando buscas por todo o fluxo da instância do processo baseando-se em parâmetros como: a responsabilidade, o papel, o nome da tarefa que gerou a participação ou o número da participação. Todos estes critérios podem ser combinados para recuperar a participação na qual os dados originados no sistema externo foram persistidos.

Para exemplificar a recuperação dos dados, considere a situação exibida pela Figura 5.28. Nesta situação, é necessário verificar se o usuário (um desenvolvedor) respondeu “Sim” ou “Não” a uma questão do sistema. Esta questão foi apresentada na última tarefa efetuada e será avaliada para tomada de decisão da porta. A busca pode ser feita como se segue:

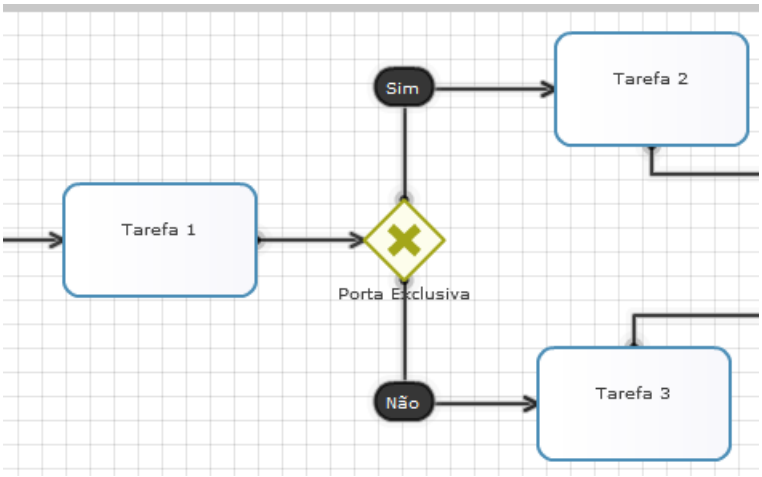


Figura 5.28: *Situação de Decisão*

Código 5.1 Filtrando Participações utilizando o Objeto de Contexto

```
participacao = contexto.participacoes(TipoResponsabilidade.RESPONSAVEL,
          TipoPapel.DESENVOLVEDOR,ULTIMA_PASSAGEM)

return participacao.dado.respostaDaPergunta == "Sim"
```

Toda vez que uma expressão (como scripts, regras, condições de sincronismo de atividade múltipla instância) é analisada, um objeto **contexto** é criado. Este objeto é uma instância da classe **ObjetoContexto**. O método **participacoes** faz buscas nas entidades **Participacao** geradas pelos passos. Neste caso, buscou-se a última participação na qual o papel envolvido foi o de desenvolvedor e o participante alocado tenha responsabilidade de executor (responsável pela execução).

Ao recuperar os dados da participação, a expressão pode referênciar qualquer um de seus atributos. Neste caso, verificou-se se a resposta para a pergunta teve o valor igual a "Sim". Caso a resposta tenha valor igual a "Sim", então "true" será retornado; caso contrário, o resultado será "false". Esta expressão está ligada à regra associada à conexão que possui rótulo "Sim". Portanto, se o resultado for verdadeiro o fluxo seguirá por esta conexão.

CAPÍTULO 6

Experimentando a Sinfonia

Com o intuito de validar as propostas deste trabalho, foi conduzido um experimento com objetivo de responder as seguintes questões:

1. A ferramenta cumpre os objetivos propostos?
2. A ferramenta tem poder de expressão suficiente para as necessidades de processos de negócio?

A primeira questão verifica se as funcionalidades apresentadas no Capítulo 1 , Seção 1.2 foram realmente implementadas. Já a segunda verifica se a ferramenta implementa os padrões de processo apresentados no Apêndice B.

Segundo [79] existem quatro formas relevantes de se conduzir um experimento na linha de Engenharia de Software: experimental, de engenharia, científico e analítico. Este trabalho adotou o método experimental para realização deste processo, tanto para a avaliação das funcionalidades quanto para a avaliação do poder de expressão da ferramenta. Sendo assim, foram definidos procedimentos de coleta de dados e o modo como os dados obtidos deveriam ser analisados.

6.1 Avaliação das Funcionalidades de Sinfonia

Este experimento avalia se Sinfonia implementa as funcionalidades propostas por este trabalho. Estas funcionalidades proporcionam a ferramenta caracteristicas flexíveis, em relação a modelagem e execução de processos, e colaborativas no tocante a interação entre usuários, por exemplo, o compartilhamento de elementos do processo.

6.1.1 Planejamento do Experimento

Para este experimento foi planejada a avaliação de cinco características, definidas no Capítulo 1, relacionadas aos diferenciais da ferramenta Sinfonia. São elas:

- Alterações no modelo do processo durante a execução

- Execução de processos empíricos
- Execução de processos *adhoc*
- Distribuição de responsabilidades
- Compartilhamento de Objetos de Fluxo

Para a coleta dos dados foi utilizado o questionário apresentado na Seção A.2 do Apêndice A. O relacionamento entre as perguntas do questionário e as características avaliadas é apresentado na Tabela 6.1.

Característica	**Pergunta**
Alterações no modelo do processo durante a execução	1 e 2
Execução de processos empíricos	3
Execução de processos *adhoc*	4
Distribuição de responsabilidades	5 e 6
Compartilhamento de Objetos de Fluxo	7 e 8

Tabela 6.1: *Relacionamento entre perguntas do questionário e características avaliadas no experimento*

A intrepretação dos resultados obtidos das resposta deveria ser baseada na Tabela 6.2 proposta por este trabalho.

Porcentagem de "Sim" (p)	**Conclusão**
$\geq 80\%$	Implementa sem erros ou com tratamento que guia o usuário à correção
$\geq 40\%$ e $< 80\%$	Implementa com erros
$< 40\%$	Não implementa

Tabela 6.2: *Modo de interpretação das respostas do questionário*

A execução da primeira parte do experimento foi planejada para ser aplicada com pelo menos dez pessoas: de um conjunto de profissionais de uma empresa seriam selecionados aqueles que se dispusessem a realizá-lo. Os procedimentos seriam executados individualmente com cada um dos participantes, tendo a presença do pesquisador durante a execução.

Para nivelar o conhecimento dos participantes, deveria ser ministrado um curso sobre gerência e modelagem de processos, e sobre a utilização do software desenvolvido neste projeto de pesquisa, o software Sinfonia. Os *slides* deste curso constam no Apêndice A.

O conhecimento sobre a utilização do software Sinfonia também poderia ser obtido na própria ferramenta por meio de vídeos que instruiam sobre como utilizar a ferramenta para modelar, executar, compartilhar e criar processos e todos os tipos de objetos de fluxo.

O objetivo do experimento não é verificar se a ferramenta possui características de usabilidade que possibilitem a execução dos procedimentos com rapidez. Assim, tanto os vídeos quanto o pesquisador poderiam ser consultados a qualquer momento.

Para poupar tempo dos participantes foi feita uma pré-configuração do ambiente que envolveu:

- Cadastrar todas as interfaces (responsáveis pela comunicação externa) e todas as regras (incumbidas de fazer o controle condicional do fluxo) necessárias para execução dos procedimentos;
- Cadastrar os participantes do experimento no sistema com o perfil adequado para a execução das tarefas a ele atribuídas.

Característica 1: Alterações no processo durante a execução

A finalidade deste procedimento é avaliar a capacidade da ferramenta de alterar os processos enquanto estão sendo executados pela máquina de orquestração. Esta funcionalidade é importante visto que muitos processos possuem uma natureza altamente mutável [65].

Os procedimentos de coleta de dados são apresentados a seguir, na ordem em que eles deveriam ser executados, pelo responsável pela execução do procedimento:

1. Pesquisador disponibiliza texto com a descrição de um processo de venda de produtos. O texto utilizado está presente no Apêndice A;
2. Participante modela o processo descrito no texto;
3. Participante instancia e executa o processo:
   (a) Escolhe como forma de pagamento “dinheiro” durante a tarefa de determinação do tipo de pagamento;
   (b) Executa a tarefa relacionada ao pagamento da compra;
   (c) Durante a execução desta tarefa, cria uma nova tarefa nomeando-a como “Devolver troco”;
   (d) Continua a execução do processo até a conclusão.
4. Participante responde as questões 1 e 2 do questionário;
5. Após a finalização do experimento com todos os participantes, pesquisador soma a quantidade de respostas “sim”, “não” e “não sei informar” dadas pelos participantes para as questões 1 e 2 do questionário.
6. Pesquisador conclui a análise da implementação dessa característica em Sinfonia, por meio da verificação da porcentagem de respostas “sim” das perguntas, considerando a Tabela 6.2.

Característica 2: Execução de processos Empíricos

O intuito deste experimento é avaliar a capacidade de Sinfonia apoiar a execução de processos empíricos. Este tipo de processo é importante, pois permite que processos não institucionalizados sejam modelados em tempo de execução.

Os procedimentos de coleta de dados são apresentados a seguir, na ordem em que eles deveriam ser executados pelo responsável pela execução do procedimento:

1. Após a execução do procedimento anterior, participante cria um novo processo de vendas, nomeando-o como "Vendas Empírico", e modela o processo de vendas empírico como ele achar mais adequado
   (a) Modelar uma atividade;
   (b) Salvar o processo;
   (c) Executar o processo;
   (d) Voltar ao passo (a) até que todas as atividades do processo sejam modeladas.
2. Participante responde a questão 3 do questionário;
3. Após a finalização do experimento com todos os participantes, pesquisador soma a quantidade "sim", "não" e "não sei informar" dadas pelos participantes para a questão 3 do questionário.
4. Pesquisador conclui a análise da implementação dessa característica em Sinfonia, por meio da verificação da porcentagem de respostas "sim" das perguntas, considerando a Tabela 6.2.

Característica 3: Execução de processos *ad hoc*

A finalidade deste procedimento é verificar a viabilidade de execução de processos *adhoc* pela ferramenta Sinfonia. Este tipo de processo é importante, pois geralmente processos não institucionalizados não possuem uma ordem de execução em suas tarefas, apesar de muitas vezes possuírem tais tarefas bem definidas.

Os procedimentos de coleta de dados são apresentados a seguir, na ordem em que eles deveriam ser executados pelo responsável pela execução do procedimento:

1. Após a execução do procedimento anterior, participante cria um novo processo, nomeando-o como "Vendas Adhoc", e modela este processo usando apenas atividades, ou seja, as atividades adicionadas ao processo não devem possuir qualquer conexão entre elas;
2. Participante instancia e executa o processo "Vendas Adhoc";
3. Participante responde a questão 4 do questionário;

4. Após a finalização do experimento com todos os participantes, pesquisador soma a quantidade “sim”, “não” e “não sei informar” dadas pelos participantes para a questão 4 do questionário.
5. Pesquisador conclui a análise da implementação dessa característica em Sinfonia, por meio da verificação da porcentagem de respostas “sim” das perguntas, considerando a Tabela 6.2.

Característica 4: Distribuição de responsabilidades

Este experimento tem a finalidade de verificar a distribuição de responsabilidades entre os participantes do processo.

Os procedimentos de coleta de dados são apresentados a seguir, na ordem em que eles deveriam ser executados pelo responsável pela execução do procedimento:

1. Para cada uma das tarefas criadas, participante adiciona uma nova responsabilidade. Esta responsabilidade deve envolver a função de aprovador da tarefa;
2. Participante instancia e executa qualquer um dos processos criados;
3. Participante responde as questões 5 e 6 do questionário;
4. Após a finalização do experimento com todos os participantes, pesquisador soma a quantidade “sim”, “não” e “não sei informar” dadas pelos participantes para as questões 5 e 6 do questionário.
5. Pesquisador conclui a análise da implementação da característica na ferramenta, por meio da verificação da porcentagem de respostas “sim” das perguntas, considerando a Tabela 6.2.

Característica 5: Compartilhamento de elementos do processo

O intuito deste experimento é testar o compartilhamento de processos e elementos do processo entre os usuários do sistema. Esta funcionalidade permite a modelagem colaborativa dos processos.

Os procedimentos de coleta de dados são apresentados a seguir na ordem em que eles deveriam ser executados pelo responsável pela execução do procedimento:

1. Após a execução do procedimento anterior, pesquisador apresenta ao participante um vídeo mostrando como compartilhar objetos de fluxo entre os usuários do sistema;
2. Participante compartilha qualquer objeto de fluxo com outro usuário do sistema;
3. Pesquisador realiza novo procedimento de autenticação no sistema como o usuário com o qual o objeto de fluxo foi compartilhado;
4. Participante certifica o acesso ao objeto compartilhado;

5. Participante responde as questões 7 e 8 do questionário;
6. Após a finalização do experimento com todos os participantes, pesquisador soma a quantidade “sim”, “não” e “não sei informar” dadas pelos participantes para as questões 7 e 8 do questionário.
7. Pesquisador conclui a análise da implementação dessa característica na ferramenta, por meio da verificação da porcentagem de respostas “sim” das perguntas, considerando a Tabela 6.2.

6.1.2 Execução e Resultados da Avaliação das Funcionalidades

Antes de conduzir o experimento, foi realizada uma validação do protocolo apresentado na Seção 6.1 com a finalidade de encontrar deficiências que poderiam ser visualizadas durante a execução.

A avaliação foi feita por dois alunos de mestrado do Instituto de Informática da Universidade Federal de Goiás que possuem experiência no desenvolvimento de aplicações empresariais há mais de cinco anos. Eles encontraram deficiências nos slides e conteúdo passado e, portanto, o treinamento foi revisto. Além disso, foi sugerida a criação de tutoriais em vídeo, com o objetivo de esclarecer as dúvidas dos usuários relacionadas à utilização da ferramenta.

Depois da implementação das melhorias, o experimento foi executado com dois outros participantes. Eles observaram que algumas funcionalidades não estavam corretamente implementadas ou apresentavam mensagens de erros não intuitivas. Também foi sugerida a reelaboração do texto do processo que seria apresentado aos participantes do experimento.

Após a correção destas deficiências, a validação do experimento foi realizada com uma pessoa com o perfil de administrador de empresas. Esta pessoa trabalha no ramo de administração de empresas a quatro anos e é formada nesta área de conhecimento. Entretanto, ela conhece a modelagem de processos apenas superficialmente e não possui conhecimento na área de Tecnologia da Informação. Por isso, considerou-se que o resultado positivo desta validação representaria uma evidência de viabilidade da aplicação do protocolo de experimento. Assim, como ele não teve qualquer dúvida em relação aos procedimentos, conclui-se que o experimento poderia ser realizado com outros participantes sem grandes riscos de insucesso.

Após a validação procedeu-se com a execução do experimento. Para a avaliação das características, foram selecionados quatorze usuários que possuem conhecimentos básicos sobre processo e modelagem de processo. Além disso, todos eles são profissionais de TI, com formação superior e experiência de mais de dois anos em desenvolvimento de software e/ou gerência de pessoas e projetos. Destas quatorze pessoas, quatro partici-

param recentemente de um projeto de implantação da norma ISO 9001 no setor de atendimento da instituição onde o experimento foi aplicado. Os usuários selecionados atuam em diferentes papéis nessa instituição: analistas de sistemas (S) , desenvolvedores (D) e analistas de negócio (N).

Os usuários com perfil S trabalham exclusivamente com análise de sistemas e possuem experiência na utilização de notações de processos, tais como a UML e BPMN.

Os usuários com perfil D trabalham na codificação de sistemas, possuem boa experiência em programação e estão aptos a interpretar os diagramas da UML.

Os usuários com perfil N são os que participaram da implantação da norma ISO 9001. Eles possuem conhecimento em modelagem de processos utilizando a notação BPMN.

Os procedimentos foram realizados individualmente com dois usuários do perfil S, sete do perfil D e cinco do perfil N. O curso foi ministrado dividindo os participantes em três turmas diferentes. Isso foi necessário devido à disponibilidade de horários de cada participante.

Foram necessários três dias para realizar os procedimentos com todos os participantes. O experimento durou, em média, uma hora por participante, portanto foram gastas cerca de 14 horas na realização dos procedimentos, além de mais 3 horas com a realização dos cursos de nivelamento, totalizando 17 horas de experimento.

A Tabela 6.3 apresenta as respostas de todos os participantes do experimento. A primeira coluna indica o número do usuário; a segunda representa o perfil do participante (analista de sistemas (S) , desenvolvedor (D) ou analista de negócio (N)); e o restante das colunas representam as respostas.

Nº	**Perfil**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
1	S1	Sim	Sim	Sim	Sim	Sim	**Não**	Sim	Sim
2	S2	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
3	D1	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
4	D2	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
5	D3	Sim	Sim	Sim	**Não**	Sim	Sim	Sim	Sim
6	D4	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
7	D5	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
8	D6	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
9	D7	Sim	Sim	Sim	Sim	**Não**	Sim	Sim	Sim
10	N1	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
11	N2	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
12	N3	Sim	**Não**	Sim	Sim	Sim	Sim	Sim	Sim
13	N4	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
14	N5	Sim	Sim	Sim	-	-	-	-	-

Tabela 6.3: *Respostas dos Participantes do Experimento*

O participante 14 não finalizou o protocolo de experimento, por isso as respostas das perguntas **P4**, **P5**, **P6**, **P7** e **P8** foram marcadas com o carácter "-". Para calcular os percentuais estes valores foram considerados como se o participante tivesse respondido "Não sei informar".

A compilação dos resultados apresentados na Tabela 6.3 gerou a Tabela 6.4. As porcentagens obtidas foram geradas através da soma do número de "Sim" dividido pelo número total de respostas esperadas de cada característica. Para as características que possuíam mais de uma pergunta, o número total foi dividido pela quantidade de perguntas.

Característica	**% de "Sim"**
Alterações no modelo do processo durante a execução	96,42%
Execução de processos empíricos	100%
Execução de processos *adhoc*	85,71%
Distribuição de responsabilidades	85,71%
Compartilhamento de Objetos de Fluxo	92,85%

Tabela 6.4: *Respostas por característica avaliada*

Através da análise da Tabela 6.4, é possível verificar que as porcentagens de respostas "Sim" foram superiores a 80% em todas as características. Com base na análise de dados proposta pela Tabela 6.2, conclui-se que as funcionalidades propostas por este trabalho foram, de fato, implementadas. Ou seja, Sinfonia provê suporte a: alteração do modelo do processo durante sua execução; execução de processos empíricos e *adhoc*; além de distribuição de responsabilidades sobre a tarefa de acordo com o papel do participante; e ao compartilhamento de objetos de fluxo entre os usuários do sistema.

Os participantes 1 e 5 responderam "Não" às perguntas P6 e P4, respectivamente. Ao investigar a razão destas respostas respostas, notou-se que eles não haviam entendido corretamente as perguntas. Já os participantes 9 e 12 responderam "Não" às perguntas P5 e P2 respectivamente, pois encontraram uma falha no sistema que os impossibilitou de executar a operação proposta.

A principal ameaça à validade deste experimento vem do fato de que o pesquisador acompanhou a execução do protocolo de experimento com todos os participantes. Para minimizar esta ameaça ao experimento, algumas medidas foram tomadas para amenizar a interferência, do pesquisador. Por exemplo, foi feita a criação de tutoriais em vídeo que esclareciam dúvidas sobre o funcionamento de Sinfonia.

6.2 Avaliação do Poder de Expressão de Sinfonia

Este experimento avalia o poder de expressão do metamodelo de processo e da máquina de execução de processos. Para isso são utilizados os vinte primeiros padrões de processo propostos por [73] e descritos no Apêndice B. Estes padrões foram escolhidos

por serem representativos com relação à realidade da maior parte das necessidades de modelagem de processos relatadas na literatura.

6.2.1 Planejamento do Experimento

O experimento aqui especificado foi executado pelo próprio pesquisador e, portanto, não foi necessário qualquer tipo de treinamento.

O modo como a coleta de dados deveria ser realizada foi inspirado nos trabalhos de [51, 76, 31, 78, 62] . Entretanto, os experimentos descritos nestes trabalhos verificam apenas a linguagem de modelagem do processo ou a linguagem de execução. O experimento aqui proposto verifica tanto a maneira como o processo é definido, como a capacidade da máquina de execução de executar as situações propostas pelos padrões de processo. A seguir são apresentados os procedimentos de coleta de dados, na ordem em que eles deveriam ser executados.

1. Para cada situação, gerar um modelo de processo utilizando o metamodelo de processo;
2. Executar o modelo gerado usando a máquina de execução;
3. Verificar o comportamento da máquina de execução;
4. Comparar o comportamendo da máquina com o comportamento esperado descrito em [73].

Para a análise dos resultados foi considerado que, se o comportamento da máquina é igual ao descrito por [73], a máquina de execução e/ou metamodelo apoia o padrão analisado e, caso contrário, que Sinfonia não é capaz de modelar e/ou executar o padrão descrito. Naturalmente, quanto maior o número de padrões de processos apoiados pela máquina de execução e/ou metamodelo, maior é o poder de expressão de Sinfonia.

6.2.2 Execução e Resultados da Avaliação do Poder de Expressão

Para realizar este experimento, todos os padrões de processos propostos por [73] foram modelados no software Sinfonia utilizando-se o metamodelo de processos descrito neste trabalho.

Cada padrão de processo modelado foi executado pela máquina de execução de Sinfonia e o resultado foi analisado e comparado com o comportamento descrito em [73]. Dos vinte padrões propostos para este experimento, apenas o padrão de Marco não foi implementado devido à sua complexidade.

A implementação de dezenove dos vinte padrões descritos em [73] garante que o metamodelo proposto é capaz de expressar uma grande variedade de situações pertencentes ao fluxo de processos. Além disso, a máquina de execução proposta está apta

a executá-los, uma vez que o comportamento da máquina é idêntico ao comportamento esperado de cada padrão.

CAPÍTULO 7

Considerações Finais

Este trabalho apresentou a construção de Sinfonia, um sistema de gerenciamento de processos (BPMS) capaz de oferecer um ambiente colaborativo para modelagem e execução de processos. O sistema oferece suporte para execução de processos empíricos e *adhoc*, implementando os principais padrões de processo identificados na literatura [73].

A arquitetura de Sinfonia é composta por três componentes básicos: o metamodelo de processos, a máquina de execução de processos e as interfaces que proporcionam a interação com o usuário. O metamodelo de processos permite a reutilização e o compartilhamento dos Objetos de Fluxo dos processos. Ademais, o metamodelo apoia a modelagem de processos *adhoc* e empíricos.

A máquina de execução é capaz de gerenciar a realização de qualquer tipo de processo representado através do metamodelo de processos. Além disso, o mecanismo de execução apoia a interação de mais de um participante por atividade, permitindo até quatro tipos de interações diferentes, com base na definição de responsabilidades proposta no modelo COBIT.

Outro diferencial da máquina de execução descrita neste trabalho é a capacidade de execução de processos empíricos e *adhoc*. Portanto, todos os processos representados através do metamodelo podem ser executados pela máquina de execução.

O software foi desenvolvido na plataforma web o que facilita seu acesso , além de proporcionar independência de sistema operacional para o cliente do software. As interfaces de interação com o usuário foram implementadas para serem interpretadas por alguns dos principais navegadores de internet, tais como Mozilla FireFox, Chrome e Internet Explorer 8.

As propostas deste trabalho foram avaliadas através de um experimento envolvendo quatorze usuários que utilizaram a ferramenta Sinfonia para modelar e executar diversos processos.

7.1 Trabalhos Correlatos

Existem várias ferramentas BPMS disponíveis no mercado, entretanto, foram escolhidas duas ferramentas para serem comparadas a Sinfonia: Bizagi [44] e Bonita [10]. Bizagi foi escolhida por ser uma ferramenta com reconhecimento internacional, enquanto Bonita foi selecionada por ser uma ferramenta livre de código aberto. Essas duas ferramentas apresentam características de otimização de processos, funcionalidade que Sinfonia ainda não possui. No entanto, elas não implementam as cinco funcionalidades motivadoras deste trabalho (alteração de processos em tempo de execução, execução de processos *adhoc* e empíricos, compartilhamento de artefatos do processo e apoio à distribuição de responsabilidades na execução das atividades).

Bonita é a ferramenta que mais se assemelha a Sinfonia pelo fato de possuir código livre e utilizar apenas um subconjunto da notação BPMN para modelagem de processos. Entretanto, ela não permite a alteração de processos durante sua execução. Assim que um processo é implantado no servidor, ele não pode mais sofrer alterações. Portanto, se houver alguma modificação no modelo do processo, as mudanças não serão efetivadas com a sua implantação. Na realidade, cada alteração no modelo do processo dará origem a um processo diferente.

Outra funcionalidade não implementada por Bonita é o apoio à execução de processos *adhoc* e empíricos, em razão da impossibilidade de alteração de processos em execução.

Bonita também não permite a distribuição de responsabilidades durante a execução de atividades. Portanto, os únicos a interagirem com as atividades são seu executores. Isso dificulta a modelagem das cadeias de reponsabilidades que são empregadas no controle de processos das organizações modernas.

Bonita permite o compartilhamento de processos, entretanto a forma de compartilhamento é limitada. Bonita só permite que um processo seja compartilhado através de sua exportação. Assim, o dono do modelo do processo deve exportá-lo e enviar o arquivo de exportação para o usuário com o qual ele deseja compartilhá-lo. Então, o usuário deve importar o processo para seu ambiente de modelagem.

Este tipo de abordagem não permite que o usuário, com o qual o processo foi compartilhado, tenha acesso a futuras alterações no processo. Além disso, este usuário não pode colaborar com a modelagem deste processo, pois ele só possui uma cópia do processo.

Bizagi é uma ferramenta BPMS proprietária que permite a modelagem e execução de processos através de uma interface gráfica. Esta ferramenta utiliza a notação BPMN em toda sua amplitude para modelar processos de negócio.

Segundo a documentação do produto, Bizagi não apoia a execução de processos

adhoc, apesar de permitir sua modelagem. Em relação à alteração dos processos, uma vez que o processo é implantado, o modelo do processo é congelado de maneira que não é possível alterar o modelo do processo. Portanto, não é possível alterar o modelo do processo adicionando novas atividades ou alterando o fluxo do processo. Para alterar o processo, é necessário criar uma nova versão dele. Logo, um processo não pode ter seu fluxo alterado durante sua execução. Naturalmente isso dificulta a execução de processos empíricos.

A distribuição de responsabilidades durante a execução de atividades também não é apoiada por Bizagi. A distribuição das tarefas pode ser feita de acordo com o papel dos envolvidos na instância do processo, porém estes só podem realizar a responsabilidade de executor na atividade.

Bizagi permite o compartilhamento de processos de maneira que o modelo do processo compartilhado possa ser alterado e visualizado pelos usuários. Porém, o comparitlhamento proposto não permite que apenas um único Objeto de Fluxo seja compartilhado com outros usuários. Sendo assim, o compartilhamento de processos proposto por Bizagi também é limitado.

Alguns padrões de processos citados por [60] não são implementados por essas ferramentas, tais como o padrão de recursividade e a execução de processos *adhoc*. Mais ainda, os usuários não podem fazer uso de elementos já definidos, como eventos, portas e processos, o que dificulta a modelagem colaborativa. Por fim, cada tarefa pode ser executada por apenas um único usuário.

Existem trabalhos que propõem outras abordagens de metamodelo partindo da BPMN, como [70], [8] e [34].

Em [70] é descrita uma notação denominada BPMN* que é uma extensão de BPMN com algumas restrições. Estas restrições tem o objetivo de solucionar problemas de ambiguidade de BPMN. Entretanto, BPMN* não implementa alguns padrões de processo e tampouco apoia a reutilização de elementos do processo e a definição de responsabilidades.

Ahmed [8] propõe um metamodelo de processo com foco em papéis e deveres. Este metamodelo define como integrar OCL (*Object Constraint Language*) à modelagem de processo com a finalidade de determinar as responsabilidades dos participantes. Entretanto, este metamodelo não soluciona outros problemas que existem na notação BPMN, como a ambiguidade de estruturas.

Birgit [34] cria uma extenção da BPMN focando em problemas de medição. Logo, BPMN foi ultilizada sem qualquer tipo de restrição, o que significa que o metamodelo proposto possui as mesmas deficiências de BPMN.

7.2 Contribuições

A principal contribuição deste trabalho de pesquisa é a abordagem utilizada para a criação do sistema Sinfonia, composta por um metamodelo de processo, uma máquina de execução de processo e a interface com o usuário, que incluem modelador gráfico de processos.

Este conjunto de itens apoia parcialmente o ciclo de vida da BPM e provê funcionalidades que outras ferramentas não implementam como: alteração de processos em tempo de execução, execução de processos *adhoc* e empíricos, compartilhamento de artefatos do processo e apoio à distribuição de responsabilidades na execução das atividades.

Outras contribuições importantes envolvem o desenvolvimento de um metamodelo de processos com foco apenas na execução de processos. Isso diminui a complexidade da notação utilizada para expressar o metamodelo, facilitando a compreensão do usuário em relação à representação de processos.

Além do metamodelo de processos, a máquina de execução descrita nesta dissertação também é uma contribuição relevante, pois, junto com o metamodelo de processos, é capaz de realizar grande parte dos padrões de processos propostos na literatura [73, 60].

Uma inovação importante na arquitetura de Sinfonia é a abordagem utilizada para invocar aplicações externas ao sistema gerenciador de processos de negócio. Da maneira como a arquitetura foi projetada, todo o controle dos dados é passado para as aplicações que entram em contato com Sinfonia. Sendo assim, Sinfonia fica responsável apenas pela execução do processo, deixando para as aplicações específicas o gerenciamento dos dados envolvidos nesta execução.

7.3 Trabalhos Futuros

Melhorias e extensões podem ser realizadas sobre as propostas deste trabalho. Elas estão relacionadas ao metamodelo de processos, à máquina de execução e à ferramenta Sinfonia.

O metamodelo proposto trabalho apoia uma grande variedade de padrões de processo. Outros padrões precisam ser implementados. Estas implementações tendem a aumentar o poder de expressão do metamodelo, embora possam tornar seus conceitos mais complexos.

Apesar de permitir a alteração dos modelos dos processos enquanto são executados, se um modelo for alterado todas as instâncias que estão utilizando aquele modelo também o serão. Um aprimoramento da ferramenta seria alterar o modelo do processo individualmente para cada instância.

Além disso, não há restrições para alteração do modelo do processo. No trabalho de Kowalkiewicz et al. [36] é apresentada uma adaptação para linguagens de expressão de processo com a finalidade de prover suporte a restrição de alteração dos modelos do processo durante a execução. Utilizando essas ideias, seria possível criar um modelo de processo que fosse modificável apenas parcialmente. Por exemplo: supondo que um processo seja definido e institucionalizado e possa sofrer uma alteração em apenas um único ponto, devido à hierarquia organizacional da empresa. Adicionando este recurso, seria possível modelar este tipo de processo.

Também é importante que seja feito um controle de versão das alterações no processo, guardando um histórico de alterações, para permitir que o modelo do processo atual seja comparado a modelos anteriores.

Além dos gatilhos dos eventos implementados pelo metamodelo e a máquina de execução, outros eventos poderiam ser implementados. Trabalhos como [19] apresentam eventos que poderiam ser adicionados a Sinfonia.

Outra extensão relacionada ao metamodelo é a implementação dos padrões de processo relacionados a recursos [61]. A maneira como os participantes são alocados é limitada e, com a implementação destes padrões, a interação de participantes com o processo ficaria mais dinâmica. Ou seja, haveria outras formas de alocar os participantes como, por exemplo, baseado em suas habilidades.

Para que o mecanismo de modelagem de processos seja mais eficiente, é preciso implementar funcionalidades como a modelagem simultânea por diverssos modeladores do processo. Além disso, este componente ainda apresenta algumas deficiências que precisam ser corrigidas, como a estruturação das linhas que representam as conexões.

Uma alteração importante na ferramenta tange à área de recuperação de informações a partir do objeto de dados. Para recuperar dados em Sinfonia, é necessário conhecimentos de programação para lidar com os objetos de contexto. Assim, apenas usuários com experiência em desenvolvimento estariam aptos a criarem regras. Seria interessante desenvolver uma forma menos complexa de elaborar as regras. Uma solução viável seria criar um componente gráfico que facilitasse a criação destas regras.

A ferramenta proposta neste trabalho provê suporte ao ciclo de vida da abordagem BPM descrita na Seção 2.1. Na fase de definição de processos, Sinfonia possibilita o desenho do processo através de uma ferramenta gráfica de modelagem. Na fase de configuração, Sinfonia permite a comunicação com outros aplicativos e a definição de papéis para alocação de participantes. Na terceira fase Sinfonia executa o processo definido através do modelador gráfico. Na fase de diagnóstico, a ferramenta permite visualizar o andamento do processo através da interface gráfica.

Para tornar Sinfonia uma ferramenta BPMS completa é necessário implementar um módulo de otimização de processos. Este módulo seria responsável por apontar os

pontos deficientes no modelo do processo, fazer simulações de execução de processo e indicar pontos que podem ser otimizados. Com a implementação deste módulo, Sinfonia seria capaz de apoiar todo o ciclo de BPM.

Referências Bibliográficas

[1] AALSTW. M. P. VAN DER AALST. **Business process management demystified: A tutorial on models, systems and standards for workflow management**. In: Desel, J.; Reisig, W.; Rozenberg, G., editors, *Lectures on Concurrency and Petri Nets*, p. 1–65. Springer, 2003.

[2] ACHILLEOS, A.; GEORGALAS, N.; YANG, K. **An open source domain-specific tools framework to support model driven development of oss**. *Model Driven Architecture- Foundations and Applications*, p. 1–16, 2007.

[3] ADOBE SYSTEMS INCORPORATED. **Action message format – amf 3**. Disponível em: http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf, 2010.

[4] ADOBE SYSTEMS INCORPORATED. **Home - adobe flex.org**. Disponível em: http://flex.org, 2010.

[5] ADOBE SYSTEMS INCORPORATED. **Learn actionscript adobe developer connection**. Disponível em: http://www.adobe.com/devnet/actionscript.html, 2010.

[6] ALMEIDA, A. C.; BOFF, G.; OLIVEIRA, J. L. **A framework for modeling, building and maintaining enterprise information systems software**. *XXIII Brazilian Symposium on Software Engineering*, 2009.

[7] AUER, D.; GEIST, V.; DRAHEIM, D. **Extending bpmn with submit/response-style user interaction modeling**. In: *Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing*, p. 368–374, Washington, DC, USA, 2009. IEEE Computer Society.

[8] AWAD, A.; GROSSKOPF, A.; MEYER, A.; WESKE, M. **Enabling resource assignment constraints in BPMN**. Technical report, Business Process Technology - Hasso Plattner Institute, 2009.

[9] BASTIEN, J. C.; SCAPIN, D. L. **Ergonomic criteria for the evaluation of human-computer interfaces**. Technical report, 1993.

[10] BONITASOFT COMPANY. **Open source bpm community**. Disponível em: http://www.bonitasoft.org, 2010.

[11] BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. **Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing**. Wiley, Chichester, UK, 2007.

[12] BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. **Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages**. Wiley, June 2007.

[13] COSTA, S.; NETO, V.; LOJA, L.; OLIVEIRA, J. **A metamodel for automatic generation of enterprise information systems**. In: *I Workshop de Desenvolvimento Dirigido a Modelo nos Anais do I Congresso Brasileiro de Software: Teoria e Prática*, volume 8, p. 45–52, Setembro 2010.

[14] CROCKFORD, D. **The application/json media type for javascript object notation (json)**. Disponível em: http://tools.ietf.org/html/rfc4627, 2010.

[15] DUMAS, MARLON.; VAN DER AALST, WIL M..; TER HOFSTEDE, ARTHUR H.. **Process Aware Information Systems: Bridging People and Software Through Process Technology**. Wiley-Interscience, September 2005.

[16] FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. PhD thesis, University of California, Irvine, 2000.

[17] FOWLER, M. **Patterns of Enterprise Application Architecture**. Addison-Wesley Professional, November 2002.

[18] G. KOLIADIS, A. VRANESEVIC, M. B. A. K.; GHOSE, A. K. **A combined approach for supporting the business process model lifecycle**. In: *Asia-Pacific Conference on Information Systems*, 2006.

[19] GAGNÉ, D.; TRUDEL, A. **Time-bpmn.** In: Hofreiter, B.; Werthner, H., editors, *CEC*, p. 361–367. IEEE Computer Society, 2009.

[20] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns: elements of reusable object-oriented software**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[21] GOH, C.; BALDWIN, A. **Towards a more complete model of role**. In: *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, p. 55–62, New York, NY, USA, 1998. ACM.

[22] HAMMER, MICHAEL.; CHAMPY, JAMES. **Reengineering the Corporation: A Manifesto for Business Revolution**. HarperBusiness, April 1994.

[23] HANSSON, D. H. **Ruby on rails**. Disponível em: http://rubyonrails.org, 2010.

[24] HAVEY, M. **Essential Business Process Modeling**. O'Reilly Media, Inc., 2005.

[25] HOLLINGSWORTH, D. **The workflow reference model: 10 years on**. In: *Technical Committee Chair of WfMC*, p. 295–312, 2004.

[26] HOLMES, T.; VASKO, M.; DUSTDAR, S. **Viebop: Extending bpel engines with bpel4people**. In: *16TH EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING*, p. 547–555. IEEE Computer Society, 2008.

[27] ITGI. **Cobit: Control objective management guidelines maturity model 4.1**. http://www.itgi.org, 2009. IT Governance Institute.

[28] JACOBSON, I.; ERICSSON, M.; JACOBSON, A. **The object advantage : business process reengineering with object technology / [Ivar Jacobson, Maria Ericsson, Agneta Jacobson]**. Addison-Wesley, Wokingham, England ; Reading, Mass. :, 1995.

[29] JANSEN-VULLERS, M. H.; NETJES, M. **Business process simulation – a tool survey**. In: *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*, 2006.

[30] JURIC, M. B. **Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition**. Packt Publishing, 2006.

[31] KABILAN, V. **Contract Workflow Model Patterns Using BPMN**. In: Halpin, T.; Siau, K.; Krogstie, J., editors, *Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'05), held in conjunctiun with the 17th Conference on Advanced Information Systems (CAiSE'05), Porto, Portugal, EU*, p. 557–568. FEUP, Porto, Portugal, EU, 2005.

[32] KO, R. K. L. **A computer scientist's introductory guide to business process management (bpm)**. *Crossroads*, 15(4):11–18, 2009.

[33] KO, R. K.; LEE, S. S.; LEE, E. W. **Business process management (bpm) standards: A survey**. volume 15, 2009.

[34] KORHERR, B.; LIST, B. **Extending the EPC and the BPMN with business process goals and performance measures**. In: *Proc. of 9th International Conference on Enterprise Information Systems*, 2006.

[35] KOSKELA, M.; HAAJANEN, J. **Business process modeling and execution: Tools and technologies report for soames project**. Technical report, 2007.

[36] KOWALKIEWICZ, M.; LU, R.; BAEUERLE, S.; KRUEMPELMANN, M.; LIPPE, S. **Weak dependencies in business process models**. p. 177–188, 2008.

[37] KRUCHTEN, P. **Architecture blueprints—the "4+1" view model of software architecture**. In: *TRI-Ada '95: Tutorial proceedings on TRI-Ada '91*, p. 540–555, New York, NY, USA, 1995. ACM.

[38] LEYMANN, F.; ROLLER, D. **Workflow-based applications**. *IBM Syst. J.*, 36(1):102–123, 1997.

[39] LINDSAY, A.; DOWNS, D.; LUNN, K. **Business processes—attempts to find a definition**. volume 45, p. 1015–1019. Elsevier, 2003.

[40] LIST, B.; KORHERR, B. **An evaluation of conceptual business process modelling languages**. In: *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, p. 1532–1539, New York, NY, USA, 2006. ACM.

[41] LOJA, L. F. B.; DA COSTA, S. L.; GRACIANO NETO, V. V.; DE OLIVEIRA, J. L. **A business process metamodel for enterprise information systems automatic generation**. In: *Proc. of First Brazilian Workshop on Model Driven Development (BW-MDD)*, 2010.

[42] LOJA, L. F. B.; DE OLIVEIRA, J. L. **Modelo de colaboração para excecução e gerência de processos de negócio em organizações**. In: *Workshop de Sistemas Colaborativos nos Anais do VII Simpósio Brasileiro de Sistemas Colaborativos*, Outubro 2010. Belo Horizonte, MG.

[43] LOJA, L. F. B.; NETO, V. V. G.; DA COSTA, S. L.; DE OLIVEIRA, J. L. **A business process metamodel for enterprise information systems automatic generation**. In: *I Workshop de Desenvolvimento Dirigido a Modelo nos Anais do I Congresso Brasileiro de Software: Teoria e Prática*, volume 8, p. 37–44, Setembro 2010.

[44] LTDA, B. **Business agility**. Disponível em: http://www.bizagi.com, 2010.

[45] LU, R.; SADIQ, S. **A survey of comparative business process modeling approaches**. In: Abramowicz, W., editor, *Business Information Systems*, volume 4439 de **Lecture Notes in Computer Science**, p. 82–94. Springer Berlin / Heidelberg, 2007.

[46] MADISON, D. **Process Mapping, Process Improvement and Process Management: a practical guide to enhancing work and information workflow**. Paton Press LCC, 2005.

[47] MATSUMOTO, Y. **Ruby programming language**. Disponível em: http://www.ruby-lang.org/en, 2010.

[48] MUEHLEN, M. Z. **Workflow-based Process Controlling**. Logos Verlag Berlin, 2004.

[49] NEZHAD, H. R. M.; BENATALLAH, B.; CASATI, F. **Process spaces management systems**. 2009.

[50] NUFFEL, D. V.; MULDER, H.; KERVEL, S. V. **Enhancing the formal foundations of bpmn by enterprise ontology**. *Lecture Notes in Business Information Processing - 1865-1348 (Print) - Volume 34 - Advances in Enterprise Engineering III*, 2009.

[51] NYSETVOLD, A. G.; KROGSTIE, J. **Pattern-based analysis of bpmn - an extensive evaluation of the control-flow, the data and the resource perspectives**. *Via Nova Architectura*, 2005.

[52] OMG. **Unified modeling language (omg uml), superstructure, v1.5**. March 2003.

[53] OMG. **Meta object facility (mof) core specification version 2.0**. Technical report, January 2006.

[54] OMG. **Bpmn information home**. Disponível em: http://www.bpmn.org, 2009.

[55] OMG. **Business process management initiative**. Disponível em: http://www.bpmi.org/, 2009.

[56] OMG. **Object management group**. Disponível em: http://www.omg.org/, 2010.

[57] PETRI, C. A. **Kommunikation mit Automaten**. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[58] RECKER, J.; MENDLING, J. **On the translation between bpmn and bpel**. In: Krogstie, J.; Halpin, T.; Proper, H. E., editors, *Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06), held in conjunctiun with the 18th Conference on Advanced Information Systems (CAiSE'06), Luxembourg, Luxembourg, EU*, p. 521–532. Namur University Press, Namur, Belgium, EU, 2006.

[59] RICHARDSON, L.; RUBY, S. **RESTful Web Services**. O'Reilly, Beijing, 2007.

[60] RUSSELL, N.; ARTHUR.; VAN DER AALST, W. M. P.; MULYAR, N. **Workflow Control-Flow Patterns: A Revised View**. Technical report, BPMcenter.org, 2006.

[61] RUSSELL, N.; HOFSTEDE, A. H. M. T.; EDMOND, D. **Workflow resource patterns**. Technical report, In the 17th Conference on Advanced Information Systems Egineering (CAISE'05, 2004.

[62] RUSSELL, N.; VAN DER AALST, W. M.; TER HOFSTEDE, A. H.; WOHED, P. **On the suitability of uml 2.0 activity diagrams for business process modelling**. In: *In Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), volume 53 of CRPIT*, p. 95–104. ACS, 2006.

[63] SCHWABER, K.; BEEDLE, M. **Agile Software Development with SCRUM**. Prentice Hall, February 2002.

[64] SENCHA LABS. **Sencha - mobile javascript - ext js, ext gwt and sencha touch**. Disponível em: http://www.sencha.com, 2010.

[65] SMITH, H.; FINGAR, P. **Business Process Management: The Third Wave**. Meghan-Kiffer Press, 2003.

[66] TAYLOR, C. I.; PROBST, C. **Business process reference model languages: Experiences from bpi projects**. In: *GI Jahrestagung (1)*, p. 259–263, 2003.

[67] THIAGARAJAN, R. K.; SRIVASTAVA, A. K.; PUJARI, A. K.; BULUSU, V. K. **Bpml: A process modeling language for dynamic business models**. In: *Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, WECWIS '02, p. 239–, Washington, DC, USA, 2002. IEEE Computer Society.

[68] THOM, L. H.; IOCHPE, C.; REICHERT, M. U. **Workflow Patterns for Business Process Modeling**. In: *Proceedings of Workshops and Doctoral Consortium of the 19th Int'l Conference on Advanced Information Systems Engineering (CAiSE 2007)*, volume I, p. 349–358, Trondheim, Norway, 2007. Tapir Academic Press.

[69] THOM, L. H.; LAU, J. M.; IOCHPE, C.; MENDLING, J. **Extending business process modeling tools with workflow pattern reuse**. In: *8º International Conference on Enterprise Information Systems*, p. 447–452, 2007.

[70] TSAI, C.-H.; LUO, H.-J.; WANG, F.-J. **Constructing a BPM environment with BPMN***. *Proceedings of the 11th IEEE International Workshop*, 2007.

[71] VAN DER AALST, W.; HOFSTEDE, A. T.; WESKE, M. **Business process management: A survey**. *Business Process Management*, p. 1019, 2003.

[72] VAN DER AALST, W. **Business process management: A personal view**. *Business Process Management Journal*, 10(2), 2004.

[73] VAN DER AALST, W.; TER HOFSTEDE, A.; KIEPUSZEWSKI, B.; BARROS, A. **Workflow patterns**. *Distributed and Parallel Databases*, 14(1):5–51, jul 2003.

[74] WESKE, M.; VAN DER AALST, W. M. P.; VERBEEK, H. M. W. **Advances in business process management**. *Data Knowl. Eng.*, 50(1):1–8, 2004.

[75] WESKE, MATHIAS. **Business Process Management: Concepts, Languages, Architectures**. Springer, 1 edition, November 2007.

[76] WHITE, S. A. **Process Modeling Notations and Workflow Patterns**. On BPMN website, 2004.

[77] WOHED, P.; DUMAS, M.; HOFSTEDE, A. H. M. T.; RUSSELL, N. **On the suitability of bpmn for business process modelling**. In: *In Proceedings 4th International Conference on Business Process Management (BPM 2006), LNCS*, p. 161–176. Springer Verlag, 2006.

[78] WOHED, P.; VAN DER AALST, W. M.; DUMAS, M.; HOFSTEDE, A. T.; RUSSELL, N. **Pattern-based analysis of BPMN – an extensive evaluation of the control-flow, the data and the resource perspectives (revised version)**. Technical Report BPM-06-17, BPMcenter.org, 2006.

[79] WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering: an Introduction**. Kluver Academic Publishers, 2000.

[80] ZUR MUEHLEN, M.; RECKER, J. **How much language is enough? theoretical and practical use of business process modeling notation**. 2008.

[81] ZUR MUEHLEN, M.; RECKER, J.; INDULSKA, M. **Sometimes less is more: Are process modeling languages overly complex?** *Enterprise Distributed Object Computing Workshops, International Conference on*, 0:197–204, 2007.

APÊNDICE A

Experimento da Ferramenta Sinfonia

A.1 Texto do Processo de Compras

O processo de compras abaixo deve ser modelado no software Sinfonia. Somente as tarefas relativas a atendente devem ser associadas ao modelo do processo.

- Cliente chega no caixa com os itens que deseja comprar.
- Atendente registra os itens escolhidos pelo cliente.
- Atendente informa qual a forma de pagamento: dinheiro, cartão ou cheque.
- Se a forma de pagamento for cartão, Atendente entra com o número do cartão.
- Se a forma de pagamento for dinheiro, Atendente entra com a quantia que lhe foi passada em dinheiro.
- Se a forma de pagamento for cheque a atendente entra com o nome do cliente e um telefone para contato.
- O proceso termina após a Cliente efetuar o pagamento.

A.2 Formulário de Avaliação

Este formulário foi utilizado no experimento com a finalidade de avaliar as funcionalidades propostas neste trabalho:

- Alterações no Processo Durante a Execução
- Execução de Processos Empíricos
- Execução de processos *ad hoc*
- Distribuição de responsabilidades
- Compartilhamento de Objetos de Fluxo

Pergunta 1. Foi possível alterar o processo de vendas durante sua execução adicionando uma nova atividade?

(a) Sim

(b) Não sei informar
(c) Não

Pergunta 2. Ao concluir a atividade de “Pagamento”, a próxima tarefa executada foi “Devolver Troco” ?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 3. Foi possível executar todo o processo, mesmo que alterando-o constantemente?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 4. Foi possível executar todo o processo *adhoc*?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 5. A ferramenta alocou os participantes corretamente para o cumprimento da tarefa, de acordo com seus papeis?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 6. Cada papel recebeu sua devida responsablidade na execução das tarefas?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 7. Foi possível compartilhar os Objetos de Fluxo entre os usuários do sistema?

(a) Sim
(b) Não sei informar
(c) Não

Pergunta 8. Foi possível acessar os Objetos de Fluxo compartilhados?

(a) Sim
(b) Não sei informar
(c) Não

A.3 Slides do Curso de BPM

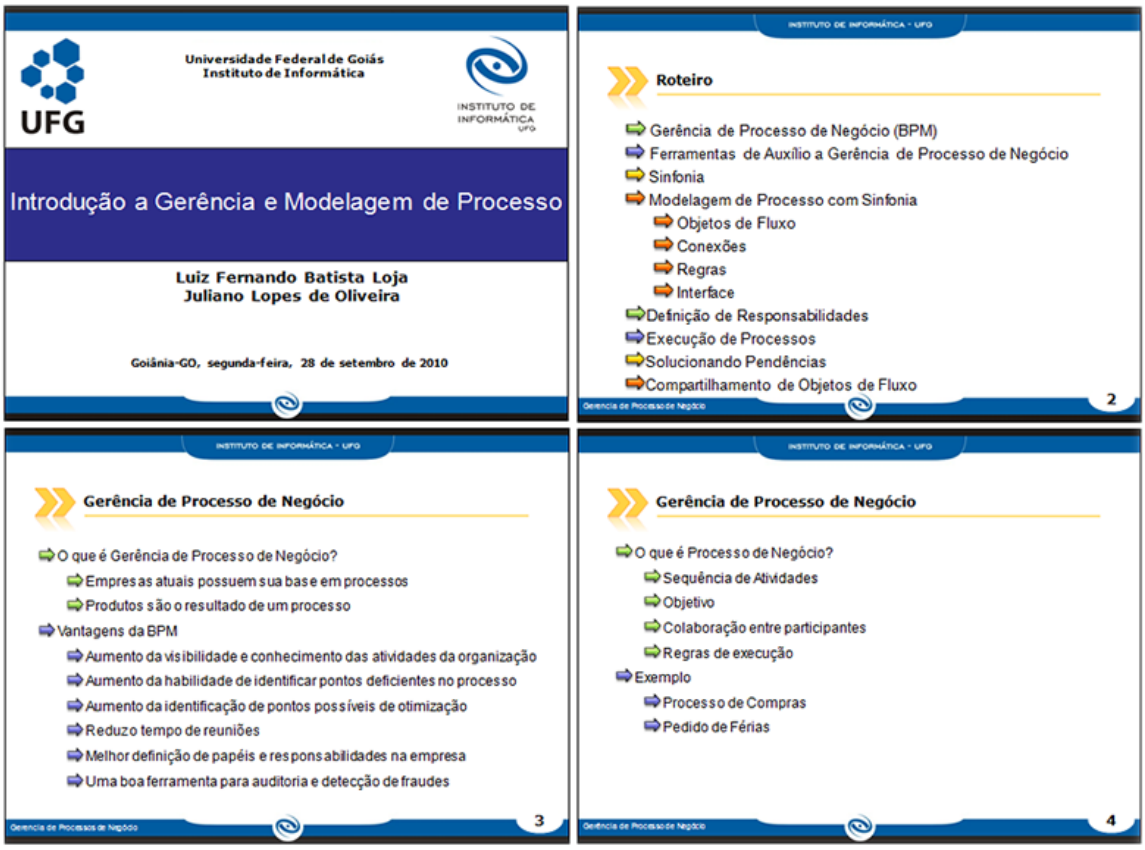


Figura A.1: *Curso de BPM Slides de 1 a 4*

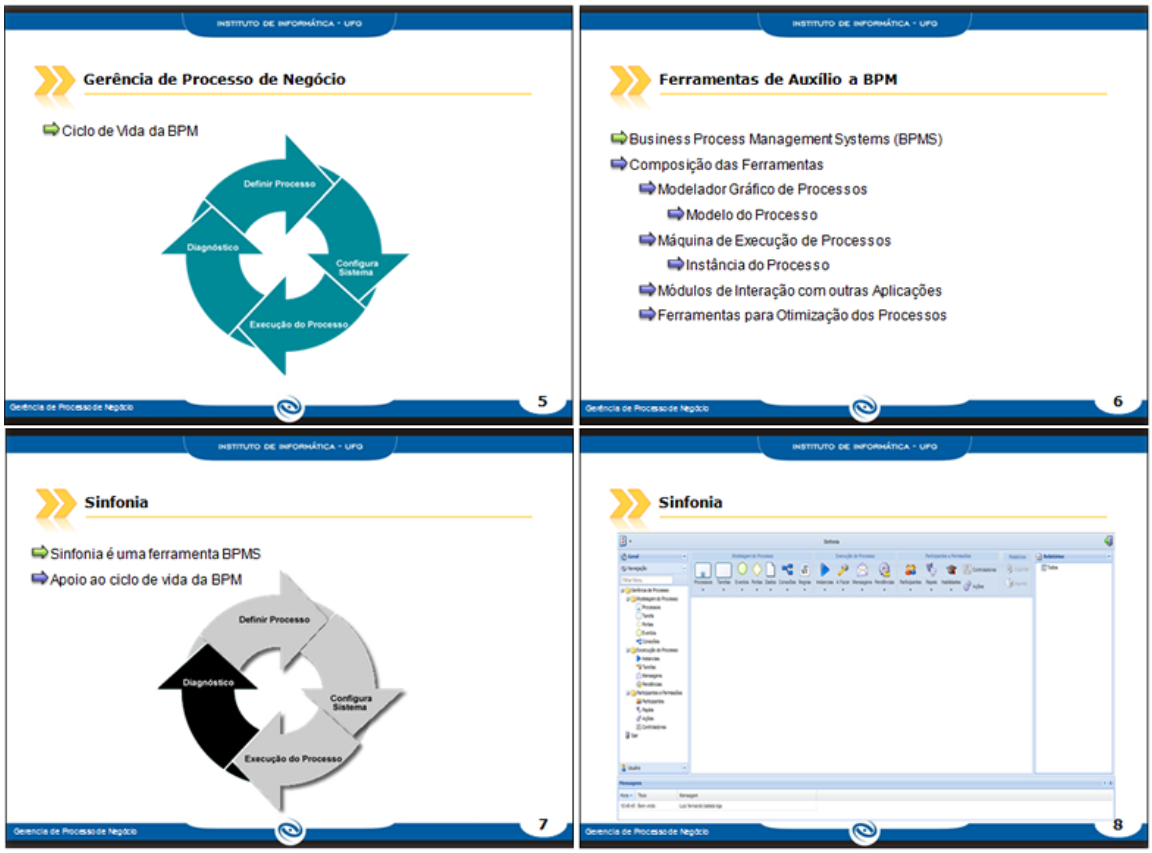


Figura A.2: *Curso de BPM Slides de 5 a 8*

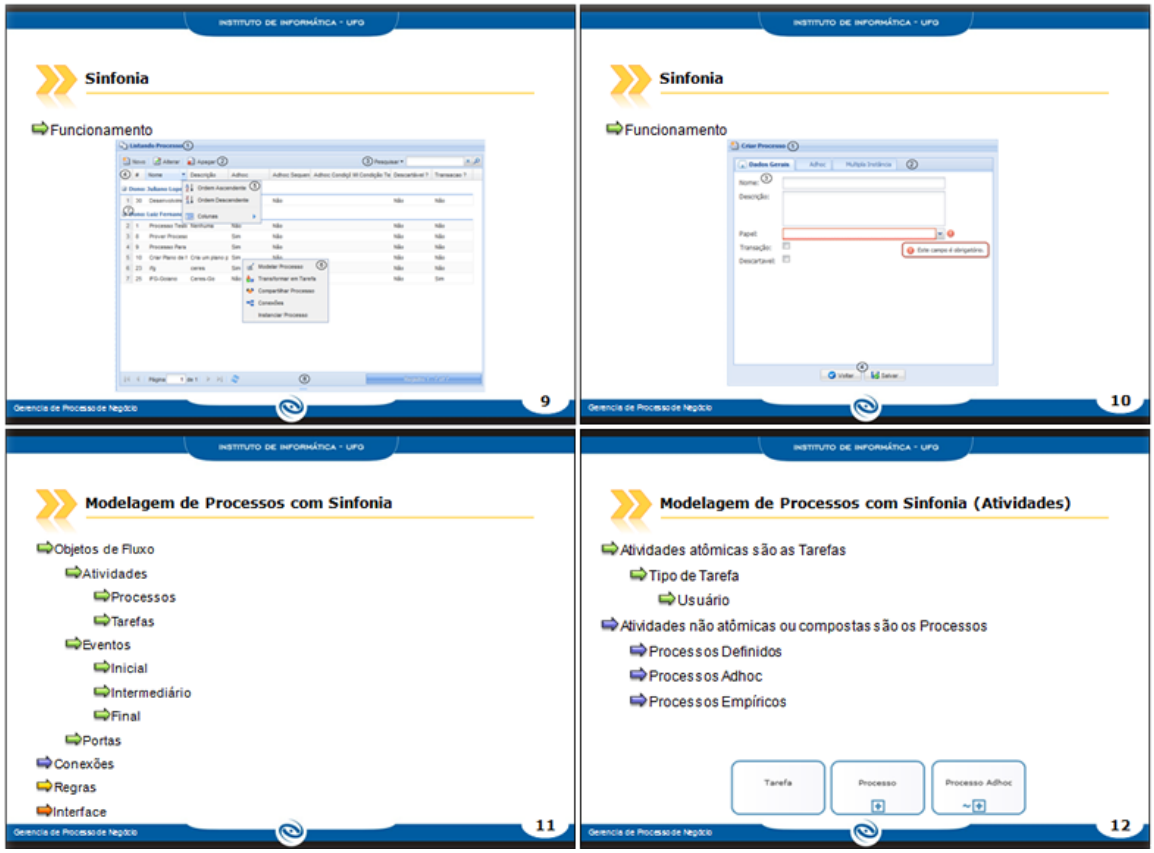


Figura A.3: *Curso de BPM Slides de 9 a 12*

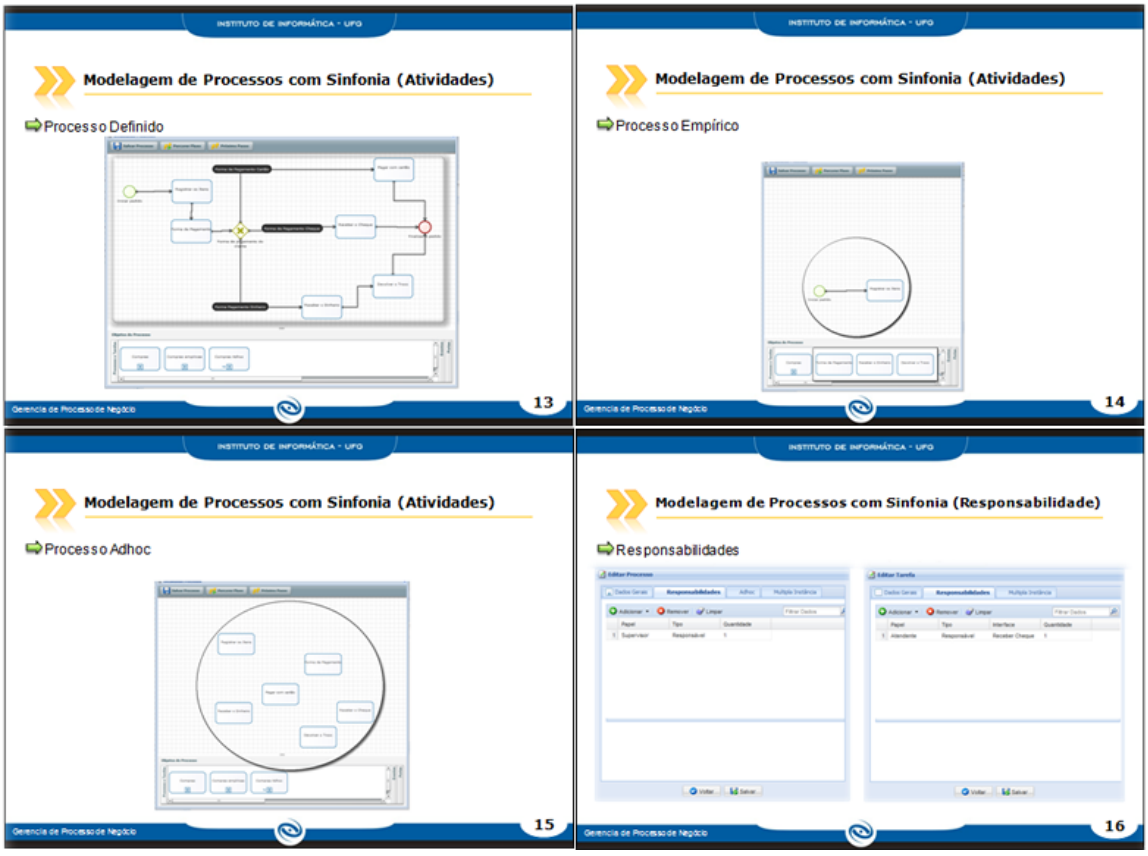


Figura A.4: *Curso de BPM Slides de 13 a 16*

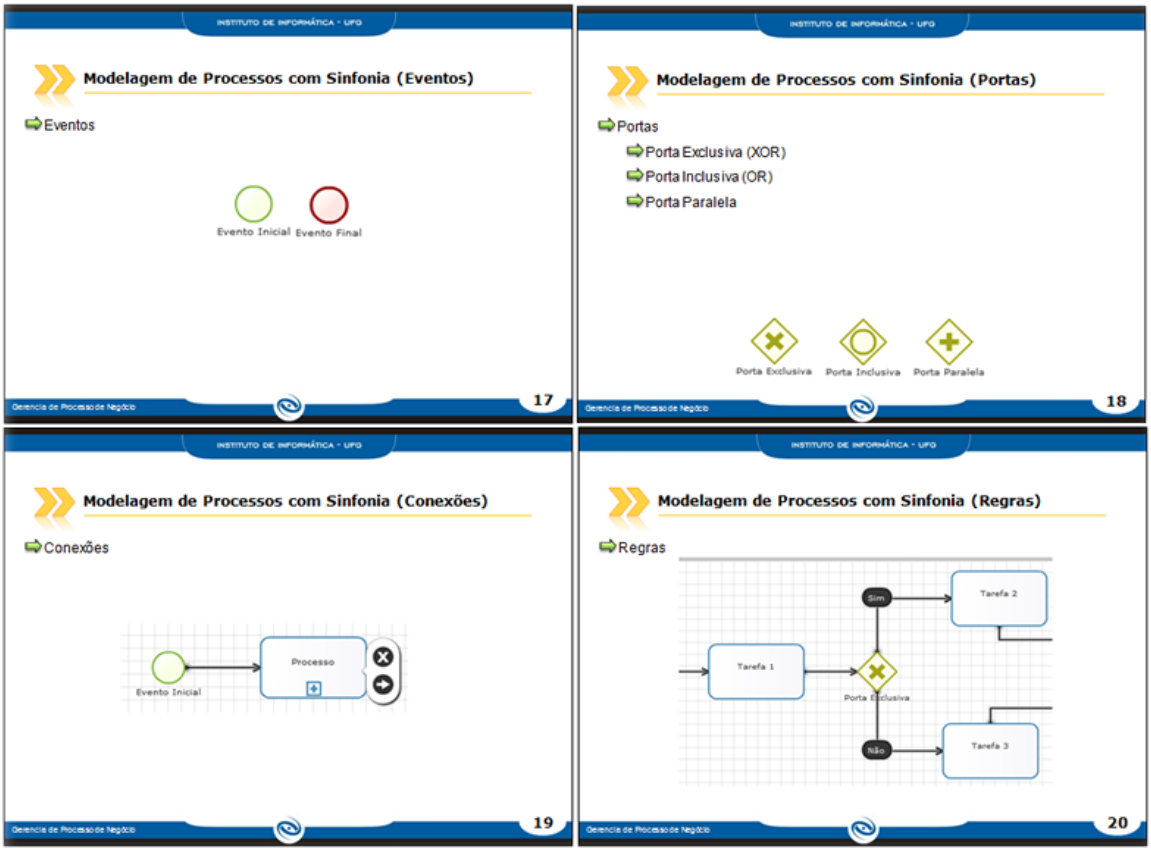


Figura A.5: *Curso de BPM Slides de 17 a 20*

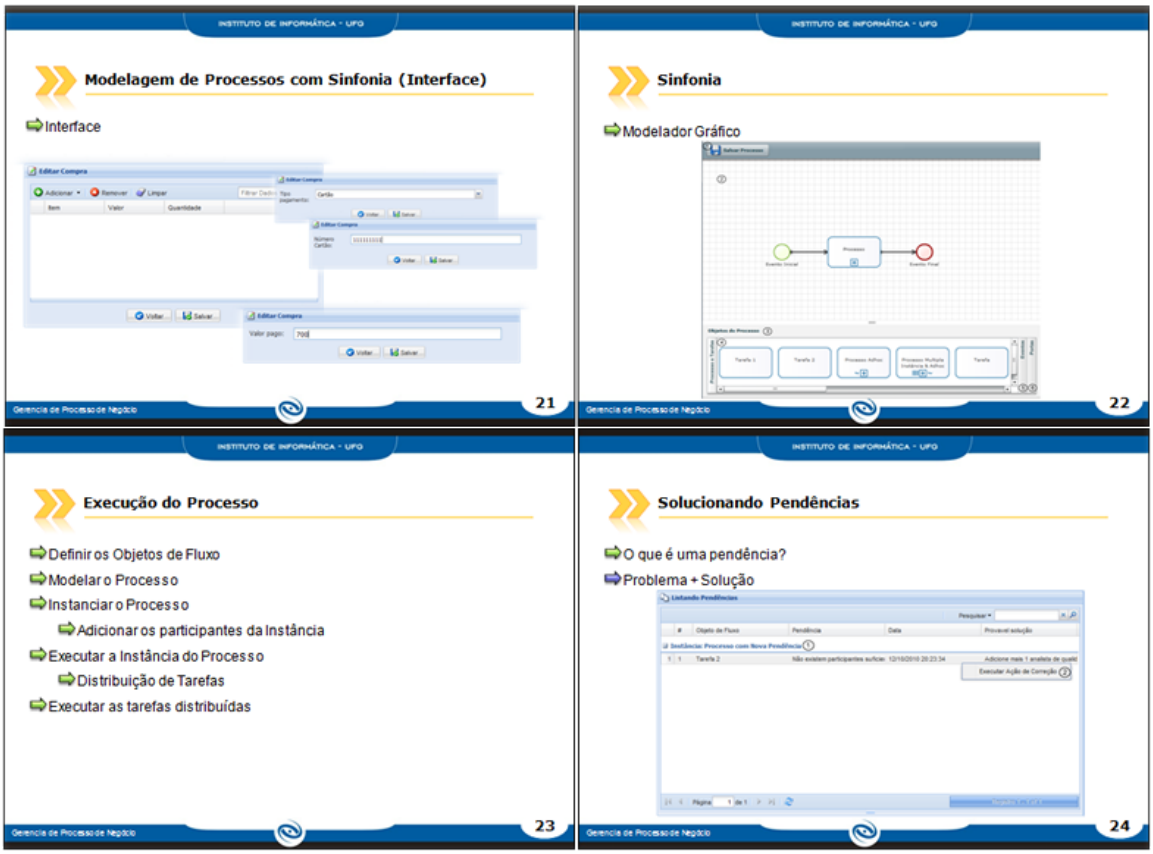


Figura A.6: *Curso de BPM Slides de 21 a 24*

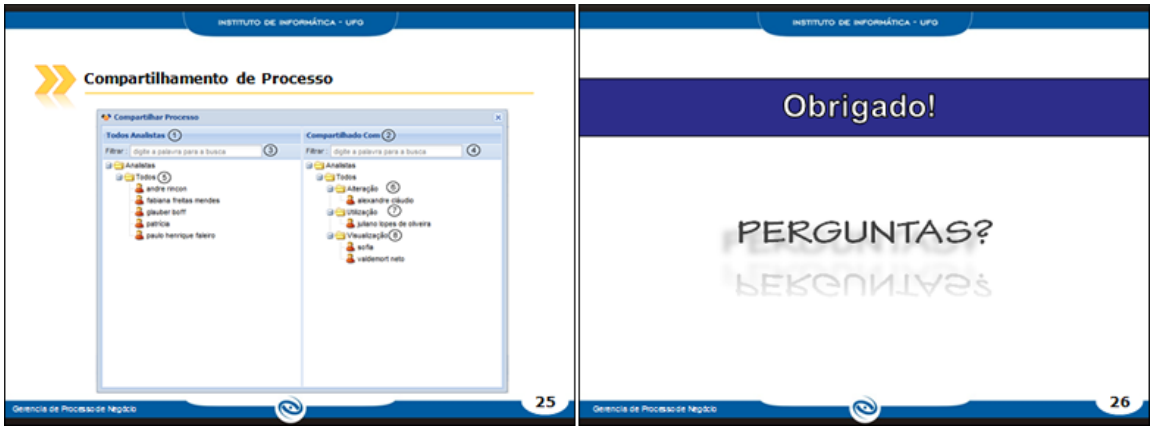


Figura A.7: *Curso de BPM Slides de 25 a 26*

APÊNDICE B

Padrões de Processo

B.1 Padrões de Controle de Fluxo

Os padrões de controle de fluxo determinam aspectos elementares do controle do processo. São eles:

- **Sequência:** Este padrão permite representar uma sequência de atividades.
  Exemplo: A verificação de uma conta é executada depois que os detalhes do cartão de crédito são capturados.
- **Divisão em paralelo:** Este padrão permite dividir um único fluxo de controle em múltiplos fluxos que podem ser executados em paralelo.
  Exemplo: Depois que o cliente pagou pela mercadoria, os produtos devem ser embalados enquanto o recibo é emitido.
- **Sincronização:** Após os fluxos serem paralelizados, este padrão permite sincronizar os fluxos novamente, convergindo múltiplos fluxos de controle em um único fluxo.
  Exemplo: O despache dos produtos deve ocorrer imediatamente após o fechamento da fatura e a verificação do valor pago pelos produtos.
- **Escolha Exclusiva:** Este padrão permite decidir o fluxo a ser seguido entre duas ou mais ações de fluxo a serem seguidos. Este padrão expressa a capacidade de representar um ponto de decisão em um processo no qual um dos vários fluxos é escolhido para dar continuidade ao processo.
  Exemplo: Em determinado ponto de uma compra a atendente pergunta se o cliente vai pagar em cheque, dinheiro ou cartão.
- **Fusão Simples:** Este padrão permite unir fluxos em execução sem sincronizá-los. Este padrão viabiliza retratar um ponto no processo no qual dois ou mais fluxos se unem sem sincronização.
  Exemplo: Após o pagamento em dinheiro ou pagamento em cartão, emitir o recibo.

B.2 Padrões de Sincronização e Ramificação Avançados

Este grupo de padrões expressa junções e ramificações de fluxo mais complexas que podem ocorrer em um processo de negócio. São eles:

- **Múltipla Escolha:** Este padrão permite dividir o fluxo de controle em vários fluxos em paralelo, de um modo seletivo. Logo, vários fluxos podem dar continuidade ao andamento do processo. Esta seleção de fluxos dependerá do mecanismo interno de decisão da máquina de execução.
  Exemplo: Em alguns estabelecimentos o cliente pode escolher mais de uma forma de pagamento. Portanto, é possível que ele pague com dinheiro e cheque.
- **Sincronização Direta:** Este padrão permite a conversão de múltiplos fluxos de controle sincronizadamente. Após a utilização do padrão Múltipla Escolha deve ser possível sincronizar todos os fluxos criados em um só fluxo de controle.
  Exemplo: Supondo que o cliente pagou sua compra com dinheiro e cheque, ao entregar o cheque e o dinheiro o atendente deve emitir um único recibo.
- **Fusão Múltipla:** Este padrão permite convergir, de forma não sincronizada, dois ou mais fluxos distintos. Consequentemente, cada fluxo deve originar a execução de uma instância da próxima atividade.
  Exemplo: Supondo que o cliente pagou em cartão e dinheiro. Após o pagamento é necessário emitir dois recibos um para o cartão e outro para o dinheiro.
- **Discriminador:** Este padrão permite a convergência de dois ou mais fluxos, tais que a primeira ativação do fluxo de entrada resulte na atividade posterior. Os demais fluxos que chegarem devem ser discriminados, ou seja, ignorados.
  Exemplo: Ao lidar com uma parada cardíaca, as atividades de verificação de respiração e de verificação do pulso acontecem em paralelo. Uma vez que uma destas atividades for concluída, a atividade de triagem é iniciada. A conclusão seguida da atividade é ignorada e não resulta em uma segunda execução da atividade de triagem.
- **Discriminador de Bloco:** Este padrão permite a fusão entre dois ou mais fluxos em um único fluxo de controle de acordo com as divergências antecedentes. O fluxo de saída do Discriminador de Bloco é ativado assim que o primeiro fluxo é habilitado, independentemente do estado dos outros fluxos. Este padrão se reinicia assim que todos os fluxos de entrada são habilitados. Portanto, o fluxo só poderá ser continuado quando todas as entradas forem habilitadas.
  Exemplo: o departamento de marketing tem apenas um espaço publicitário disponível em um evento. O departamento tenta encontrar um cliente para este espaço via internet e jornal. Após a reserva do espaço o publicitário do departamento de

marketing deve esperar até que o cliente use as duas vias tanto a internet como o jornal. Para que uma nova oferta de espaço seja feita.

- **Discriminador de Cancelamento:** Este padrão permite a junção de dois ou mais fluxos em um único fluxo, de acordo com as divergências antecedentes. O fluxo de controle é passado para o fluxo de saída assim que o primeiro fluxo é habilitado. Todos os outros fluxos são cancelados, independentemente de seu estado.
  Exemplo: o departamento de marketing tem apenas um espaço publicitário disponível em um evento. O departamento tenta encontrar um cliente para este espaço via internet e jornal. Assim que a reserva for feita por uma das duas vias a outra é cancelada, assim não será possível reservar o espaço do anúncio pela outra via.
- **Junção Parcial Estruturada:** Este padrão permite a conversão de dois ou mais fluxos em um único fluxo correspondente a uma divisão que tenha ocorrido anteriormente. O processo continua seu fluxo após um número *m* de fluxos terem sido executados. Os outros fluxos entrantes são ignorados.
  Exemplo: o departamento de marketing, tem apenas um espaço publicitário disponível em um evento. O departamento tenta encontrar um cliente para este espaço via internet, jornal e telefone. Assim que a reserva for feita por duas das três vias eles poderão decidir qual cliente escolher. A terceira via é ignorada.
- **Junção Parcial Bloqueada:** Este padrão permite a fusão de dois ou mais fluxos (supondo *m* fluxos) em um único fluxo, de acordo com a divisão em algum lugar anterior do processo. É dada continuidade aos fluxos apenas quando *n* dos *m* fluxos tenham sido habilitados. A junção se reinicia assim que todos os fluxos entrantes tenham sido habilitados pelo menos uma vez. Todos os fluxos são bloqueados até seu reinício.
  Exemplo: o departamento de marketing tem apenas um espaço publicitário disponível em um evento. Ele tenta encontrar um cliente para este espaço via internet, jornal e telefone. Logo que dois ou mais clientes aceitam anunciar, as próximas atividades são executadas. As outras atividades que ainda não foram terminadas continuam em execução, entretanto quando são finalizadas elas não iniciam a próxima atividade.
- **Junção Parcial Cancelada:** Este padrão é semelhante ao Padrão Junção Parcial Bloqueada. A diferença é que após dar continuidade ao fluxo os outros fluxos em andamento são cancelados.
  Exemplo: o departamento de marketing tem apenas um espaço publicitário disponível em um evento. Ele tenta encontrar um cliente para este espaço via internet, jornal e telefone. Logo que dois ou mais clientes aceitam anunciar, as próximas atividades são executadas e a via que não fidelizou o cliente é cancelada.
- **Junção Generalizada:** Este padrão permite juntar dois ou mais fluxos em apenas um único fluxo. O fluxo de controle é continuado assim que todos fluxos entrantes

são habilitados.

Exemplo: Assim que todas as assinaturas de determinado documento sejam obtidas, conclua o contrato.

- **Conversão Síncrona Acíclica:** Este padrão permite convergir dois ou mais fluxos divididos anteriormente no processo em um único fluxo de saída. O fluxo de controle é continuado assim que cada fluxo tenha sido habilitado. O número de fluxos requeridos para sincronização é determinado com base nas informações do próprio processo.

  Exemplo: o comite de uma olimpíada de matemática deseja fazer uma prova onde um dos critérios de avaliação é a velocidade com os alunos finalizam o teste. Este teste será aplicada a um determinado número de alunos desconhecido. O comite não sabe quantos alunos virão fazer a prova, mas desejam categoriza-los em três grupos. Cada categoria de aluno terá um professor diferente para corrigir sua prova. As provas dos primeiros alunos serão corrigidas pelo professor mais "amigo", já o segundo professor é imparcial, enquanto que o terceiro irá corrigir a prova com toda cautela. Sendo assim, a cada um terço dos alunos que terminarem a prova o comitê irá iniciar o processo de correção. Quanto mais demorarem para entregarem as provas mais severa será a correção.
- **Conversão Síncrona Generalizada:** Este padrão permite convergir dois ou mais fluxos divididos anteriormente no processo em um único fluxo de saída. O fluxo deve ser continuado assim que todos os fluxos são completados ou se não houver a possibilidade de qualquer fluxo ser concluído.

  Exemplo: o departamento de marketing tem três espaços de anúncios disponíveis para seus clientes. A promoção destes três espaços é feita através do telefone, da internet e do jornal. Se o departamento de marketing decidir promover os espaços via internet ou telefone, ambos precisam ser sincronizados. Se o departamento decidir usar o jornal como canal de divulgação, eles podem decidir mais tarde se irão sincronizar os canais de divulgação ou não. Assim, é possível que não haja ofertas válidas através do jornal.
- **Junção de *Threads*:** Este padrão permite representar um determinado ponto no processo no qual um número específico de fluxos é sincronizado em um único fluxo.

  Exemplo: A atividade de registro de cada veículo é executada independentemente das outras no processo de inquérito. Quando dez registros de veículos são concluídos, a atividade de registo de lote deve ser executada, uma vez, para finalizar o sistema de registo de veículos e atualizar os registros.
- **Divisão de *Threads*:** Este padrão permite representar um determinado ponto no processo no qual um número distinto de fluxos podem ser iniciados.

  Exemplo: após a confirmação do recebimento de um artigo é iniciada a execução de

três atividades de revisão do trabalho submetido.

B.3 Padrões de Finalização

Estes padrões determinam situações em que o processo deve ser finalizado. São eles:

- **Término Implícito:** Este padrão permite descrever que um processo deve terminar quando não houver mais atividades para serem executadas.
  Exemplo: Em uma situação de compra de produtos em um supermecado, assim que a atendente entregar o recibo ao cliente o processo deve terminar.
- **Término Explícito:** Este padrão permite terminar um processo ao atingir determinado estado. Assim que este estado for atingido o processo inteiro deve ser finalizado.
  Exemplo: Duas equipes estão analisando um mesmo defeito. Assim que uma equipe tenha encontrado e resolvido o problema, todo o processo é terminado, independentemente do andamento da outra equipe.

B.4 Padrões Estruturais

Padrões Estruturais identificam se o formalismo da modelagem possui alguma restrição em relação à estruturação do processo, particularmente em termos de laços (repetições).

- **Ciclo Arbitrário:** Este padrão permite representar laços em um processo que possui múltiplos pontos de entrada e saída.
  Exemplo: Um cliente pode comprar vários itens. O atendente deve registra item a item até que todos os itens sejam cadastrados na compra.
- **Ciclo Estruturado:** Este padrão permite executar atividades repetidamente. O laço pode ter uma pré ou pós condição de término. Portanto, ele pode ser avaliado no início ou no final de sua execução.
  Exemplo: Enquanto a máquina ainda tem combustível, continue com o processo de produção, a menos que o número de produtos chegue a trinta itens.
- **Recursividade:** Este padrão permite uma atividade invocar a si mesma durante sua execução ou a um processo que lhe deu origem.
  Exemplo: Uma atividade de solucionar defeito é iniciada para cada problema mecânico identificado na planta de uma construção. Durante a execução desta atividade, se uma nova falha mecânica é identificada, uma outra atividade *solucionar defeito* é criada.

B.5 Padrões de Múltipla Instância (MI)

Este padrões descrevem atividades que podem possuir múltiplas execuções.

- **MI sem Sincronização:** Este padrão permite iniciar várias instâncias de uma atividade dentro de uma instância de um determinado processo.
  Exemplo: Uma lista de infrações de trânsito é recebida pela AMT (Agência Municipal de Transporte). Para cada infração na lista, uma atividade de criação *cadastro de multa* é criada. Estas atividades acontecem em paralelo e não causam quaisquer atividades subsequentes. Elas não precisam ser sincronizadas assim que forem concluídas.
- **MI com Número Definido no Momento da Modelagem:** Este padrão permite iniciar múltiplas instâncias de uma atividade. O número de atividades é conhecido em tempo de modelagem. Depois de ter completado todas as instâncias, uma atividade posterior é iniciada.
  Exemplo: Um relatório anual que precisa ser assinado por todos os seis diretores de uma determinada empresa antes que possa ser emitido.
- **MI com Número Definido no Momento da Execução:** Este padrão permite iniciar múltiplas instâncias de uma atividade. O número de atividades varia, mas é conhecido em tempo de execução. Uma vez que todas as instâncias tenham sido concluídas, uma atividade posterior é iniciada.
  Exemplo: Ao diagnosticar uma falha em um motor, várias mensagens são enviadas simultaneamente pelos sensores de detecção de falha. Somente quando todas as mensagens tiverem sido processados pelo mecanismo de controle de erros, a tarefa de correção de falhas será iniciada.
- **MI sem Número de Instâncias Definido:** Este padrão permite iniciar múltiplas instâncias de uma atividade. O número de atividades varia, e não é conhecido no momento da modelagem ou em tempo de execução. Uma vez que todas as instâncias tenham sido concluídas, a próxima atividade é iniciada. Outras atividades podem ser criadas, mesmo quando outras instâncias estão em execução ou já foram concluídas.
  Exemplo: Uma empresa que será inaugurada em breve deseja contratar funcionários. Entretanto não se sabe quantos funcionários serão necessários para realizarem o trabalho. Logo esta empresa envia uma oportunidade de emprego para várias pessoas cadastradas em um banco de dados. Novas pessoas podem ser cadastradas nesta base de dados. A empresa só irá parar de enviar esta mensagem quando todos os funcionários forem contratados.
- **Junção Parcial Estática para MI:** Este padrão permite que *m* tarefas concorrentes sejam iniciadas. Uma vez que *n* tarefas tenham sido concluídas (onde *n* é menor

do que *m*), a tarefa seguinte é disparada. As tarefas que restaram não iniciam a tarefa seguinte, porém todas as tarefas que tiveram início devem ser finalizadas para reiniciar o padrão.
Exemplo: Procurar o defeito em dez amostras da linha de produção e, executar a próxima atividade quando sete dos exames forem concluídos.

- **Cancelando Junção Parcial para Multipla Instância:** Este padrão é semelhante ao Junção Parcial Estática para Multipla Instância. A diferença é que assim que as *n* atividades forem concluídas as outras atividades em execução são canceladas.
Exemplo: Inicie quinhentas ocorrências da tarefa de teste da proteína com amostras distintas. Depois de completar quatrocentos exames, cancele as instâncias restantes e inicie a próxima atividade.
- **Junção Parcial Dinâmica para MI** Este padrão permite que, durante a execução da instância de um processo, múltiplas instâncias de uma atividade sejam criadas. O número requisitado de instâncias é dependente dos dados de tempo de execução do processo. Enquanto as instâncias estiverem sendo executadas é possível adicionar novas instâncias da mesma atividade. Uma condição de término é definida e avaliada a cada atividade concluída. Assim que esta condição for satisfeita o andamento do processo é continuado. As atividades que já foram criadas continuam sua execução até sua conclusão.
Exemplo: o departamento de desenvolvimento tem um problema e agrupa um número dinâmico de desenvolvedores para resolver este problema. Se um dos desenvolvedores encontra uma solução para o problema ele submete a solução ao líder da equipe. O líder da equipe decide se a solução é satisfatória ou não. Enquanto não existe uma solução que satisfaça o líder da equipe todos os outros desenvolvedores podem apresentar suas soluções, e outros desenvolvedores podem ser contratados. No entanto, a administração pode determinar uma paralisação nas contratações, resultando em um número constante de desenvolvedores. Os desenvolvedores que já foram contratados continuam trabalhando para encontrar uma solução para o problema.

B.6 Padrões Baseados Em Estado

Estes padrões são caracterizados por cenários nos quais uma execução é determinada pelo estado da instância do processo.

- **Escolha Diferenciada:** Este padrão permite representar um ponto de decisão em processo no qual várias ramificações devem ser ativadas. A decisão sobre quais fluxos devem ser seguidos é feita pela análise do ambiente e deve ser postergada o

máximo de tempo possível.

Exemplo: Um processo pode ser iniciado e entretanto ele só continuará seu andamento caso algum gerente de departamento envie uma mensagem para que ele continue. Outra maneira de continuar seu andamento seria através de uma condição qualquer que fosse verificada periodicamente.

- **Roteamento Paralelo Intercalado:** Este padrão permite descrever um conjunto de atividades que podem ser executadas em ordem arbitrária.
  Exemplo: Ao despachar uma encomenda, as atividades de escolha de mercadoria, empacotamento dos produtos, e emissão da fatura devem estar concluídas. A atividade escolha de mercadoria deve ser realizada antes da atividade de empacotamento do produto. A atividade de emissão de fatura pode ocorrer a qualquer momento. Apenas uma dessas tarefas pode ser feita a qualquer momento por uma determinada ordem.
- **Marco:** Este padrão permite representar que uma atividade específica não pode ser iniciada até algum estado indicado.
  Exemplo: O estudante pode se matricular apenas quando as inscrições estão sendo aceitas. Ou seja, assim que o período de inscrições for finalizado, não será mais possível que o aluno se inscreva.
- **Seção Crítica:** Este padrão descreve duas ou mais atividades identificadas como seções críticas. Enquanto uma atividade pertencente a uma seção crítica estiver em execução as outras atividades não poderão ser iniciadas.
  Exemplo: A aceitação de depósitos e o pagamento de seguro no processo de reserva de férias exigem o uso exclusivo da máquina de cartão de crédito. Sendo assim, apenas um deles pode ser executado a qualquer momento.
- **Rotina Interrelevante:** Este padrão permite que duas ou mais atividades devem ser executadas pelo menos uma vez, em qualquer ordem, porém não devem ser executadas simultaneamente. Uma vez que as duas tarefas forem completadas, a próxima tarefa é iniciada.
  Exemplo: As atividades de verificação do óleo, teste de alimentação, análise da unidade principal e revisão da garantia precisam ser realizados como parte do processo de manutenção de máquinas. Apenas uma atividade pode ser realizada ao mesmo tempo. Porém elas podem ser executadas em qualquer ordem.

B.7 Padrões de Cancelamento

Estes padrões caracterizam a capacidade do formalismo da modelagem de processos de representar a terminação de atividades e processos em determinadas circunstâncias.

- **Cancelar Atividade:** Este padrão permite desativar uma atividade habilitada em determinada circunstância ou momento.
  Exemplo: O comprador pode cancelar seu pedido de compras a qualquer momento durante a execução da atividade de pedido de compra.
- **Caso Cancelar:** Este padrão permite cancelar uma instância do processo (ou seja, todas as atividades relacionadas com a instância do processo) em alguma circunstância.
  Exemplo: Durante uma aplicação de hipoteca, o comprador pode decidir não continuar com a compra de casa e cancelar o pedido.
- **Cancelamento de Região:** Este padrão permite cancelar um grupo de atividades. Se qualquer tarefa foi habilitada ela deverá ser cancelada.
  Exemplo: Cancelar todas as atividades do processo de reserva após a atividade de apresentação de mercadorias.
- **Cancelamento da Atividade de Múltipla-Instância:** Este padrão determina o cancelamento de todas as atividades criadas pela atividade de múltipla-instância que não foram concluídas.
  Exemplo: executar quinhentas ocorrências da tarefa de teste da proteína com amostras distintas. Caso não tenha completado em uma hora após o início, cancelar o procedimento.
- **Completar Atividade de Multipla-Instância:** Este padrão permite forçar a conclusão das instâncias que ainda não foram concluídas.
  Exemplo: Executar quinhentas ocorrências da tarefa de teste da proteína com amostras distintas. Uma hora após o início, finalizar todas as atividades que não foram concluídas e iniciar a próxima atividade.

B.8 Padrões de Gatilho

Os Padrões de Processo associados a mecanismos de gatilho, que são responsáveis por disparar eventos durante a execução do processo.

- **Gatilho Transitório:** Este padrão descreve que uma atividade pode ser acionada por um sinal vindo de um processo externo ou de outras partes do mesmo processo. Este gatilho é transitório, pois é desabilitado caso não receba o sinal assim que é acionado.
  Exemplo: se possível, iniciar a atividade de verificação dos sensores a cada vez que um sinal de alarme é recebido.
- **Gatilho Persistente:** Este padrão descreve que uma atividade pode ser acionada por um sinal vindo de um processo externo ou de outras partes do mesmo processo.

Este gatilho é persistente, tendo em vista que quando o fluxo do processo chega a ele, o gatilho fica aguardando a chegada do sinal.
Exemplo: inicie uma nova instância da tarefa de inspecionar veículos para cada sinal do serviço em atraso, que é recebido.