Universidade Federal de Pernambuco

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BRUNO LOUREIRO DE ABREU

UMA LINGUAGEM PARA MODELAGEM DE PROCESSOS BASEADA EM SEMÂNTICA DE AÇÕES

ORIENTADOR: Hermano Perrelli de Moura
CO-ORIENTADOR: Luiz Carlos Menezes

RECIFE, DEZEMBRO / 2005.

BRUNO LOUREIRO DE ABREU

UMA LINGUAGEM PARA MODELAGEM DE PROCESSOS BASEADA EM SEMÂNTICA DE AÇÕES

DISSERTAÇÃO APRESENTADA À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Hermano Perrelli de Moura
CO-ORIENTADOR: Luiz Carlos Menezes

RECIFE, DEZEMBRO / 2005.

BRUNO LOUREIRO DE ABREU

UMA LINGUAGEM PARA MODELAGEM DE PROCESSOS BASEADA EM SEMÂNTICA DE AÇÕES

DISSERTAÇÃO APRESENTADA À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada em ____de_____de _____.

Banca examinadora:

_____

Profª. Dr. Hermano Perrelli Moura – Orientador
Universidade Federal de Pernambuco

_____

Profª. Dr. André Santos
Universidade Federal de Pernambuco

_____

Profª. Dr. Ricardo Massa
Universidade de Pernambuco

Dedico esta dissertação a minha família, bela e incansável companheira na principal missão humana: amar a vida e vê-la sempre com esperanças...

AGRADECIMENTOS

Agora, com nossos melhores sentimentos, dedicamos nosso tempo para agradecimentos, sem dúvidas tão merecedores.

A Deus pela graça da vida e pelas condições de realizações tão valiosas.

A minha família, em especial a minha tia, Iolanda Abreu, meu tio Odálio Xavier (em memória), por tornarem possível esta realização, sempre acreditando em mim, e a minha mãe, Giovana Abreu (em memória), que mesmo fora deste ambiente material sempre me deu força e inspiração para atingir meus objetivos.

A Alexsandra Santos, minha esposa e companheira, pela intensa felicidade que me proporciona.

Ao doutor, amigo e orientador, Hermano Perrelli de Moura, pelo enorme estímulo e apoio para que mais este passo fosse vitorioso em minha vida profissional.

A Luiz Carlos Menezes, meu co-orientador, que sempre se mostrou disposto a tirar as dúvidas que surgiram no decorrer da realização deste trabalho.

A UFPE que desde a formação da graduação em Ciência da Computação, tenho enorme respeito e admiração, aproveito para externar os inesquecíveis agradecimentos.

A amiga Angélica, pela ajuda na revisão do abstract.

A todos aqueles que com palavras ou ações contribuíram para realização deste trabalho.

RESUMO

Modelos de processos de negócio são criados para descrever, em uma linguagem de fácil acesso, como os processos de uma organização são realizados. Para isso, o processo é visto como uma coleção de atividades que interagem para a realização do objetivo final do processo.

Para facilitar o entendimento e a análise de processos deste tipo, a modelagem visual é a mais adequada. Nesta modelagem, as atividades são representadas por unidades de execução e o fluxo de execução do processo é descrito através de um fluxograma, que define a seqüência de execução destas atividades.

Os modelos de processos são utilizados como fonte de informação para o diagnóstico de pontos de gargalo que levem à perda de eficiência da organização na realização de suas tarefas. A análise de processos é um procedimento que tem como objetivo realizar ajustes no processo ou até mesmo refazê-lo por completo, visando sempre o aumento de produtividade da organização. A utilização de modelos visuais como ponto de entrada para a análise torna a realização desta tarefa muito subjetiva, uma vez que o impacto de mudanças no processo é apenas sentido quando o mesmo é colocado em prática. Dessa forma, torna-se necessária à utilização de um mecanismo mais formal para a representação do processo que venha possibilitar a análise e a simulação mecanizada destes processos.

Neste contexto, este trabalho define a ANBPL (Action Notation Business Process Language), que viabiliza a especificação de processos formalmente. Na definição da semântica desta linguagem foi utilizado o *framework* de Semântica de Ações, utilizado para a especificação de linguagens de programação.

Apesar do caráter textual utilizado para descrever processos, a ANBPL possui uma representação simples, o que a torna de fácil entendimento. Na ANBPL, um processo de negócio é representado como uma coleção de conexões entre atividades, onde operadores são especificados para descrever os diferentes tipos de conexões. Cada operador define uma regra particular para o controle do fluxo de execução das atividades interligadas pelo mesmo.

Palavras-chave: Linguagem para Execução de Processos, Modelagem de Processos de Negócio, Semântica de Ações.

ABSTRACT

Business process models are created in order to describe, in a common language, how the organization processes are accomplished. The process is perceived as a compilation of activities that interact among them to accomplish the main goal of the process. In order to ease both understanding and process analysis of this kind, visual modeling is the best choice. On this modeling, the activities are represented by execution units and the process execution flows are described through a flowchart, which defines the order of execution of these activities.

The process models are used as an information source to the diagnosis of points that impliys the loss of organization efficiency in the tasks accomplishment. The process analysis is a procedure that point out how to make adjustments to the process or even refector it as a whole, always aiming towards productiveness increase. The use of visual models a gateways to the analysis make the task accomplishment too subjective, because the change impact on the process is only perceived when it comes to execution. This way, it's necessary the use of a formal mechanism into the process representation that might enable analysis and mechanized simulation of those processes.

In this context, this work defines the ANBPL (Action Notation Business Process Language), which enables formal process specifications. At the semantic definition of this language it was used Action Semantic, a common framework to specify programming languages.

Besides the textual aspect used to describe process, an ANBPL have a very simple representation that made it easy to understand. In the ANBPL environment, a business process is represented as a collection of links between tasks that have operators to describe different kinds of connections. Each operator defines a particular rule to the execution flow control of the interlinked activities owned by it.

Key-words: Business Process Modeling, Business Process Execution Language, Action Semantics.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

A modelagem de processos de negócio consiste em uma área de contínuo desenvolvimento e amadurecimento, onde diversas metodologias foram e continuam sendo desenvolvidas com o objetivo de oferecer mecanismos simples e práticos para a representação de processos de negócio.

Como a audiência dos envolvidos na análise destes tipos de processos dentro de uma organização é bem ampla, faz-se necessário à adoção de modelos de processos de negócios que sejam de fácil entendimento por todos os envolvidos, facilitando a comunicação entre as partes interessadas na análise e entendimento dos processos. Com base neste princípio, as metodologias para modelagem de processos existentes procuram se basear em notações que utilizam componentes visuais para a criação de modelos de processos de negócio.

Um crescente número de ferramentas para gerenciamento de processos está oferecendo a funcionalidade de simulação para estender as funções de modelagem. Simulação é colocada como um facilitador para avaliar o impacto de mudanças em processos existentes ou na utilização de novos processos em uma organização. A simulação viabiliza a avaliação do impacto, em um ambiente real, decorrente de mudanças nos processos de uma organização, possibilitando tomadas de decisões mais seguras na adoção de novos processos ou mudanças nos processos de uma organização. Outro ponto de destaque no uso da simulação de processos corresponde na identificação de pontos falhos ou de gargalo em processos de negócio [3].

1.1 Motivação

Apesar da existência de inúmeras técnicas de modelagem de processos [6] [9] [14] [16] [20] [34] e do amadurecimento deste mercado, existe uma lacuna que vem demandando muito estudo e pesquisa. Esta lacuna está na falta de linguagens onde processos de negócio possam ser escritos de acordo com uma sintaxe e semântica bem definidas, para

que possam ser executados, permitindo a simulação de situações organizacionais em um ambiente de execução computacional.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é de definir uma linguagem textual e formal, onde processos de negócio possam ser escritos da forma mais natural possível. Esta linguagem deve possuir operadores que representem as diversas maneiras possíveis de controle de fluxo entre as atividades pertencentes a um processo de negócio.

A linguagem deve ter sua semântica definida de maneira formal, onde deve ser utilizado um *framework* formal, para especificação de linguagens, capaz de dar significado aos operadores presentes na linguagem.

1.2.2 Objetivos Específicos

As atividades necessárias para a realização do objetivo geral são:

- entender o significado e todos os conceitos que norteiam a modelagem de processos de negócio;
- pesquisar as diversas técnicas ou notações para modelagem de processos existentes no mercado, possibilitando o entendimento dos requisitos mínimos que uma linguagem de modelagem de processos deve ser dotada;
- eleger a técnica ou a notação gráfica utilizada para modelagem de processos de negócio, que sirva como ponto de inspiração para o projeto da linguagem de processos;
- estudos sobre o *framework* formal utilizado para projeto de linguagens de programação, denominado de Semântica de Ações [26];
- projetar uma linguagem formal para representação de processos de negócios, definindo sua sintaxe e sua semântica utilizando Semântica de Ações;
- validar a completude da linguagem, utilizando-a para representar processos de negócios definidos graficamente;
- sugerir novas linhas de pesquisas que possam ser derivadas deste trabalho.

1.3 Metodologia

O desenvolvimento deste trabalho foi direcionado por etapas bem definidas, com o objetivo de adquirir o conhecimento necessário para a definição de uma linguagem capaz de representar processos de negócios. Em um primeiro momento ocorreu a realização de estudos sobre a área de modelagem de processos de negócio, onde boa parte da bibliografia foi levantada e os conceitos que norteiam este tema foram estudados.

Em seguida, a pesquisa foi direcionada ao estudo das técnicas e notações existentes para modelagem de processos de negócio, este estudo contribuiu para o entendimento de aspectos importantes a respeito da modelagem de processos. Nessa etapa foi eleita uma notação que servisse como alicerce para a definição da linguagem proposta. Como resultado, a notação de maior destaque na literatura para a modelagem de processos de negócio, BPMN [8], foi a selecionada.

A próxima etapa deste trabalho correspondeu ao entendimento de um *framework* formal para descrição de uma linguagem de programação.

Com base nos conhecimentos adquiridos nas etapas anteriores, foi realizado o projeto de uma linguagem para representação de processos de negócios.

1.4 Estrutura do Trabalho

O Capítulo 2 deste trabalho dá uma visão geral sobre a modelagem de processos de negócio, onde são tratados aspectos importantes como: benefícios trazidos pela adoção da modelagem de processos de negócios, técnicas de modelagem e a atual situação que se encontram os projetos existentes de linguagens para execução de processos de negócio.

No Capítulo 3 será apresentada uma notação para modelagem de processo de negócio, conhecida como BPMN. Em um mundo marcado pela heterogeneidade de notações e técnicas para modelagem de processos, levando a conseguinte carência de uma notação padrão para este fim, a BPMN surge com muita importância e vem sendo considerada uma notação padrão para modelem de processos.

O Capítulo 4 apresenta o *framework* de Semântica de Ações utilizado no projeto linguagens de programação, sendo de extrema importância para o entendimento do projeto da linguagem, objetivo deste trabalho de pesquisa.

No Capítulo 5, onde o produto final deste trabalho de pesquisa é apresentado, é detalhada uma linguagem para representação de processos de negócio, denominada de ANBPL (Action Notation Business Process Language). Onde sua sintaxe é definida, inspirando-se nos operadores existentes na BPMN, apresentada no Capítulo 3, e sua semântica é descrita com a utilização do *framework* de Semântica de Ações. Ainda neste capítulo, são apresentados exemplos de processos de negócio escritos na BPMN e sua transformação para os elementos existentes na ANBPL.

As conclusões e sugestões para trabalhos futuros são detalhadas no Capítulo 6.

Capítulo 2

Modelagem de Processos de Negócio

A modelagem de processos de negócio tem como objetivo principal descrever as atividades que fazem parte de um processo de negócio e como elas se relacionam para atingir seu objetivo final. As atividades são tarefas que tanto podem ser realizadas manualmente como mecanicamente. É importante não confundir a modelagem com a engenharia de processos de negócio, uma vez que a modelagem não tem o objetivo de indicar como um processo de negócio deve funcionar, sendo apenas uma técnica utilizada para visualizar e documentar o funcionamento de um processo de negócio.

O modelo de um Processo de Negócio é desenvolvido através da aplicação de técnicas especializadas. Ele pode ser construído com diferentes objetivos, dependendo das necessidades da organização. Na Tabela 2.1 são apresentados quatro propósitos que motivam a criação de um modelo de processo de negócios, sendo estes: funcional, informativo, organizacional ou comportamental [20].

Tabela 2.1 – Motivações para a criação de um modelo de processo de negócio.

Tipo	Objetivo
Funcional	Definir que atividades são realizadas e quais os dados que fluem entre elas. Esclarecendo o papel de cada atividade, de acordo com as transformações que são realizadas sobre os dados manipulados.
Informativo	Levantar as entidades (documentos, dados, artefatos, produtos) produzidas ou manipuladas no decorrer de um processo, incluído seus relacionamentos e sua estrutura.
Organizacional	Esclarecer o motivo pelo qual as atividades são realizadas e onde elas acontecem. Definir os mecanismos de comunicação física e armazenagem das informações pertencentes ao processo.
Comportamental	Representar o momento e a seqüência que as atividades são realizadas, como elas interagem e suas condições de execução.

Uma atividade que faz parte de um processo de negócio é tida como um componente que realiza uma função específica. Atividades complexas são compostas por outras atividades, que direcionam sua execução. Um processo é tido como uma composição de atividades, podendo também ser visto como uma atividade dentro de um processo de maior magnitude [4].

Existem três tipos de processos: core, suporte e processo de gerência [18] [24]. Os processos classificados como core são aqueles que geram produtos para clientes externos da organização, sendo considerados processos comercialmente importantes, podendo se tomar como exemplo: processos de venda e processos de atendimento ao cliente. Processos de suporte são aqueles que geram resultados para a própria organização, sendo fundamentais para o funcionamento do negócio. Enquanto que os processos de gerenciamento descrevem como os outros processos devem ser gerenciados.

Nos próximos sub-tópicos serão tratados aspectos relevantes aos benefícios conquistados quando organizações adotam a modelagem de processos, técnicas utilizadas para modelagem e linguagens para execução de processos.

2.1 Benefícios da Modelagem de Processos de Negócio

Como as complexidades de uma organização são oriundas do seu comportamento, a modelagem dos processos de negócios existentes em uma organização surge como a melhor estratégia para entender a organização. São grandes as vantagens existentes ao se modelar um processo de negócio, uma vez que viabiliza um melhor entendimento do funcionamento do mesmo, permitindo sua documentação e facilitando uma análise mais detalhada, a partir do levantamento de informações a respeito de como o processo é realizado, que recursos são necessários para seu funcionamento e quais os seus responsáveis.

A modelagem de processo de negócios é utilizada em uma organização para o entendimento, a representação, e, quando necessário, serve como meio para re-projetar um processo [20]. A especificação correta dos processos é essencial para qualquer empresa, uma vez que processos de negócios bem projetados são necessários para que, nos momentos corretos, sejam tomadas as decisões corretas dentro de uma organização.

Com o aumento da competitividade, empresas dos mais diversos setores buscam cada vez mais aprimorar seus processos, para que possam atender da melhor forma possível seus clientes, reduzir custos e aumentar sua produtividade. Neste contexto, a modelagem de processo tem um papel primordial no entendimento do funcionamento dos processos existentes nas empresas e na detecção de pontos crítico ao funcionamento do mesmo.

Na contínua procura por obter processos de negócio mais eficiente, muitas empresas concentram seus esforços no aperfeiçoamento de seus processos ou tomam medidas mais radicais, conhecidas como reengenharia de processos, fazendo com que o processo sofra reestruturações mais profundas [34]. A opção pela reengenharia de processos, em algumas situações, traz ganhos de maior magnitude com relação à primeira, porém consiste em um caminho mais demorado, custoso e de maior risco à organização.

2.2 Técnicas de Modelagem de Processos de Negócios

Esta Seção tem o propósito de descrever algumas técnicas utilizadas para modelagem de processos de negócios.

Como a complexidade das organizações torna-se cada vez mais crescente, é de extrema importância o uso de boas técnicas de modelagem de processos. Os modelos de processos permitem diferentes pessoas envolvidas com os processos organizacionais observarem estes de uma forma padronizada, possibilitando uma melhor comunicação entre elas.

O entendimento de como um processo funciona é o ponto de partida para a criação do modelo deste processo. Porém, a identificação um processo nem sempre é uma tarefa fácil de ser realizada. Como pode ser observado em [30], após contatos com pessoas responsáveis por definir o funcionamento dos processo de negócios em suas empresas, geralmente com a missão de dar mais produtividade a organização, elas acabam concordando sobre a grande dificuldade da realização desta tarefa.

A primeira etapa que uma organização deve realizar para criar modelos de seus processos de negócios é eleger qual a técnica de modelagem de processos melhor se adequa as suas necessidades. Esta definição não é uma tarefa fácil de ser realizada, visto que, com a crescente popularidade da modelagem de processos de negócios,

muitas técnicas surgiram com o passar do tempo, tornado essa escolha uma tarefa dispendiosa. Nas seguintes subseções serão descritas algumas técnicas que podem ser adotadas para a realização da modelagem de processos.

2.2.1 ASME – American Society of Mechanical Engineers

A ASME, fundada em 1880, desenvolveu um padrão para mapeamento de processos que é amplamente utilizado em fábricas e cada vez mais popular em escritórios e ambientes que disponibilizam serviços [34].

Este método tem a grande vantagem de produzir um modelo de processo onde são identificadas claramente quais atividades trazem benefícios ao objetivo final do processo. Durante a elaboração do modelo deve ser especificado para cada etapa ou atividade pertencente ao processo, se a mesma acrescenta benefícios ao objetivo final do processo. A construção de um modelo de processo baseado nesta técnica facilita a realização de uma análise mais direta sobre o funcionamento do processo, facilitando a identificação de quais atividades, pertencentes ao processo, não acrescentam benefícios direto ao objetivo final do processo, tornando atividades deste gênero sujeitas a serem excluídas do processo em uma tomada de decisão da equipe de análise dos processos da organização.

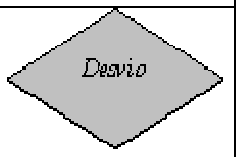
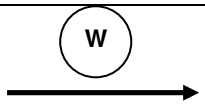
2.2.2 EPC – Event Process Chain

A EPC consiste em uma técnica de modelagem de processos que, apesar de sua simplicidade, possui uma boa expressividade na modelagem de processos de negócios [20]. Esta técnica tem foco no cliente beneficiado pelo processo, em vez de ser orientada aos serviços da organização [21], tornando-a uma ferramenta atraente para a realização de reengenharia de processos de negócios. O modelo adotado por EPC facilita a identificação de gargalos no funcionamento do processo, que podem torná-lo lento e ineficiente, levando subseqüentemente à insatisfação dos clientes da organização.

A notação utilizada por EPC, descrita pela

Tabela 2.2, é formada por cinco elementos: Evento; Processo; Desvio; Fluxo e; Espera.

Tabela 2.2 – **Notação Gráfica Utilizada em EPC**

Elemento	Descrição	Notação
Evento	Uma mudança de estado em algum determinado momento que tenha alguma relevância ao cliente (qualquer pessoa ou organização, interna ou externa à organização). Um evento deve ser representado através de um círculo com um nome em seu interior, que o identifica.	
Processo	Atividade ou uma série de atividades que ocorrem durante um intervalo de tempo. Para a notação EPC apenas as atividades que possuem envolvimento dos clientes são consideradas. Os elementos do diagrama que são processos são representados em forma de retângulos, possuindo no seu interior o nome que o identifica.	
Desvio	O Desvio representa um ponto de bifurcação condicional no fluxo de um evento ou processo, podendo levar o fluxo corrente de execução a um outro sub-fluxo, dependendo do estado no qual o fluxo corrente se encontre.	
Fluxo	Representa o movimento de objetos do cliente entre eventos e processos. Os objetos do cliente representam entidades de interesse a organização, que fluem através de processos ou eventos internos a organização.	
Espera	Tempo médio de atraso para o início de um evento ou processo. Sendo representado por uma seta abaixo de um círculo com a letra **W** (*Waiting Time*), no seu interior.	

Na Figura 2.1 é exibido o exemplo de um processo de consulta médica modelado com a notação existente na EPC. Este cenário retrata todos os passos a serem realizados por um paciente (cliente da organização), ao realizar uma consulta em um hospital (a organização em questão), desde sua chegada até o momento de sua saída [21].

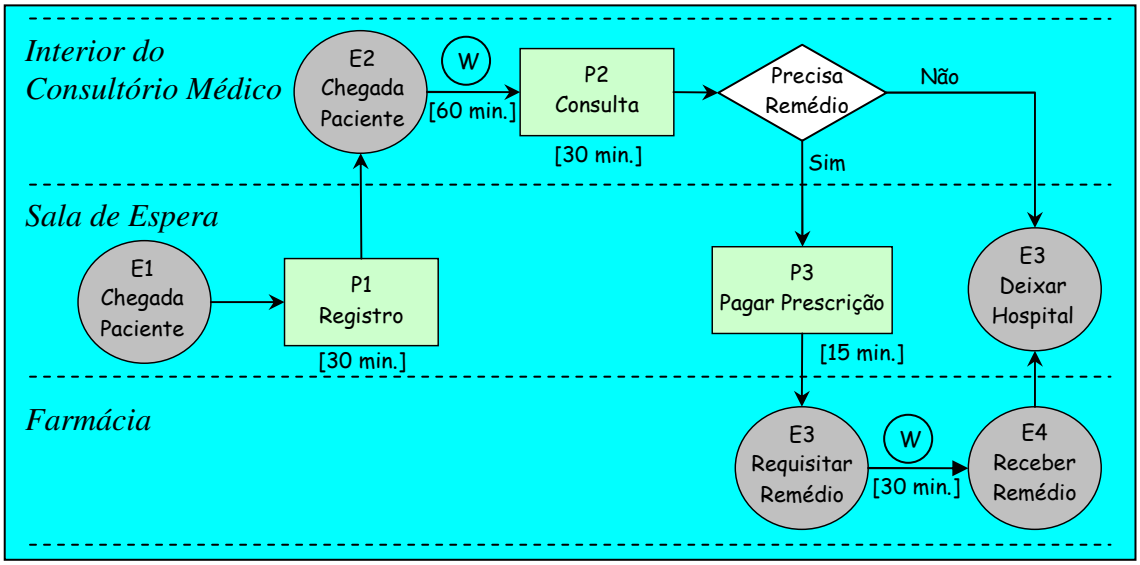


Figura 2.1 – Modelo de Processo utilizando técnica EPC.

Como pode ser observado no processo modelado acima, das 2:45 (duas horas e quarenta e cinco minutos) gastos em todo o processo apenas 30 minutos foram gastos no processo de consulta, que era o real objetivo do paciente, ficando clara a necessidade de uma melhoria nesse processo, para que tempo gasto em outras atividades e eventos caia o máximo possível, tornando o tempo de todo o processo o mais próximo possível do tempo gasto no processo consulta.

2.2.3 IDEF3 – Integration Definition For Function Modeling 3

IDEF3 é um mecanismo criado com o objetivo principal de coletar e documentar um processo de negócio. Esta técnica possui mecanismos capazes de expressar claramente relacionamentos de precedência e casualidades entre os eventos pertencentes a um processo de forma natural, devido ao seu método estruturado para expressar conhecimentos a respeito de um sistema, processo ou organização [1].

IDEF3 possui o poder de descrever aspectos comportamentais do funcionamento de um processo, servindo também como uma ótima ferramenta para descrever um sistema existente ou um sistema proposto. O fato de IDEF3 descrever o processo dentro do contexto de um cenário o torna um ótimo dispositivo para aquisição de conhecimento do funcionamento do processo.

Uma descrição de processo IDEF3 é desenvolvida usando-se dois tipos de estratégias para a aquisição de conhecimento. Uma direcionada à descrição do processo e outra direcionada aos objetos que fazem parte do processo. A estratégia direcionada a processos organiza o conhecimento do processo focalizando no processo e seus relacionamentos temporal, casual e lógico dentro de um cenário. Enquanto que na estratégia focalizada em objetos são observados os estados de transação dos objetos que fazem parte do processo. Esta observação pode ser realizada através de um ou mais cenários. Nesta seção será apenas apresentada a estratégia de modelagem baseada em processos.

O modelo IDEF3 baseado na modelagem de processos captura o conhecimento de quais são os processo existentes em uma organização, para que determinado produto seja desenvolvido ou serviço seja realizado. Esta descrição de processo também leva em consideração toda rede de relacionamentos entre os processos existentes dentro do contexto peculiar ao cenário no qual os processos ocorrem.

O principal objetivo deste tipo descrição de processo consiste em mostrar como as atividades acontecem em uma organização, responsável pela realização de uma determinada atividade ou solução de um problema. O desenvolvimento deste tipo de modelo consiste em expressar como as ações são realizadas, cujas informações são extraídas de especialistas ligados à área em questão [17].

O seguinte exemplo ilustra como os elementos pertencentes ao método IDEF3 podem ser utilizados para descrever um cenário comum a um ambiente de produção. O cenário a ser modelado corresponde ao processo de pintura de peças e inspeção que ocorrem durante a realização da pintura de peças, que irão fazer parte de uma máquina de maior porte. Tal processo ocorre de acordo com o texto relatado, logo abaixo, pelo supervisor da loja de pintura quando é pedido para que o mesmo descreva o que ocorre durante tal processo:

> *"As peças, prontas para receberem a primeira camada de tinta, entram na loja onde o processo de pintura será realizado. Nós aplicamos uma primeira camada de tinta a uma temperatura elevada. A tinta é colocada para secar em um forno. Após um certo período de tempo a peça pintada parte para o processo de inspeção da pintura. Caso a peça não passe pelo processo de inspeção, verificando que a mesma não recebeu tinta suficiente, fazendo com que a peça receba uma nova camada de tinta. Caso contrário, o processo de pintura é considerado como finalizado."*

O cenário descrito acima é mapeado para um modelo descritivo de processo IDEF3 de acordo com a Figura 2.2. Podemos observar que as caixas rotuladas descrevem claramente as atividades existentes no cenário apresentado.

Erro! Indicador não definido.

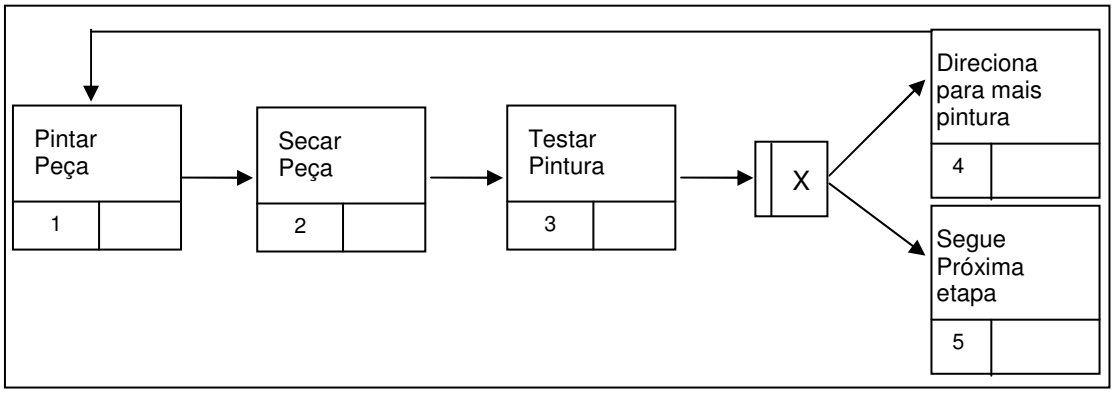
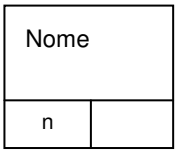
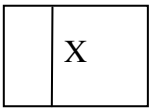


Figura 2.2 – Processo de Pintura Modelado em IDEF3

O gráfico acima é composto por vários tipos de elementos, os quais, são apresentados a seguir, na Tabela 2.3.

Tabela 2.3 – Notação Utilizada pela técnica IDEF3.

Elemento	Descrição	Notação
Unidades Comportamentais (UC)	Representam tipos de acontecimentos como eventos, ações ou processos. Na figura além de ser destacado o nome do acontecimento, também é exibido um número, servindo como um identificador único que também indica sua seqüência no fluxo da execução do processo.	
Fluxo	As setas que ligam as UC's, definem o fluxo de execução. Ou seja, uma UC, cuja seta esteja apontando, só dará início a sua execução até que a UC, de onde a seta partiu, termine sua execução.	
Ponto de Junção	As caixas menores definem pontos de junção no fluxo do processo, onde vários caminhos podem convergir para um único, ou um caminho pode divergir para outros possíveis [36].	

No modelo IDEF3, cada UC pode ser decomposta em outro gráfico IDEF3, com o objetivo de aumentar o grau de detalhamento da UC, além da possibilidade de existir várias decomposições diferentes para uma mesma UC, cada qual expressando um ponto de vista do funcionamento do processo.

2.2.4 IDEF∅ – Integration Definition For Function Modeling ∅

IDEF∅ é um método projetado para modelar decisões, ações e atividades de uma organização ou sistema. Esse método foi oriundo de uma linguagem gráfica bem estabelecida, a SADT (Structured Analysis e Design Technique), a qual foi projetada à partir da necessidade da força aérea americana de possuir um mecanismo para realização de análise de sistemas sobre uma visão funcional.

A criação de modelos, utilizando IDEF∅, além de possibilitar a análise do sistema, facilita o entendimento de suas funcionalidades, tanto por parte do analista quanto pelo cliente. IDEF∅ tem grande utilidade no estabelecimento do escopo de uma análise.

Como uma ferramenta de comunicação, IDEF∅ amplia o domínio do sistema, proporcionando maior consenso nas tomadas de decisão, devido à simplicidade de seus dispositivos gráficos. Como ferramenta de análise, IDEF∅ auxilia o modelador do sistema a identificar que funções são realizadas, quais os recursos alocados à execução destas funções, e o que um determinado sistema faz de correto e de errado [14].

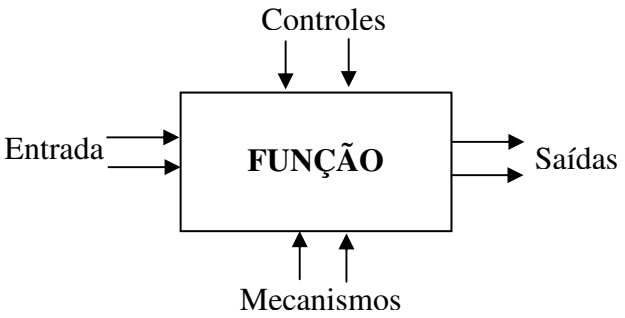


Figura 2.3 – Notação IDEF∅

O modelo IDEF∅ é representado graficamente através de caixas e setas como pode ser observado na Figura 2.3, acima. As caixas representam funções, que são identificadas pelo nome existente em seu interior. As setas identificam os seguintes tipos de elementos: interfaces de entrada e saída, elementos de controle e mecanismos das atividades. As setas que representam as interfaces de entrada e saída são orientadas da esquerda para direita, onde as que entram representam as entradas, enquanto que as que saem representam as saídas. Os elementos de controle e mecanismos são modelados por meio de setas orientadas verticalmente, cruzando a atividade [25]. Os mecanismos representam os meios ou ferramentas utilizadas para a realização de uma atividade e os controles ditam as condições necessárias para a produção correta da saída.

Atividades podem se comunicar e operarem em seqüência, bastando estarem interligadas via as interfaces de entrada e saída, onde a interface de saída de uma atividade serve como interface de entrada para a outra. Essas interfaces são munidas de regras que fazem com que uma função ative ou não outra. A composição de funções auxilia a captura e o entendimento das funcionalidades existentes em um processo modelado.

Para exemplificar a utilização de IDEF∅, o exemplo do processo de pintura, usado para demonstrar a técnica de modelagem IDEF3, apresentada na seção anterior, será reutilizado na criação do modelo de processo presente na Figura 2.4, porém, fazendo uso da notação definida por IDEF∅.

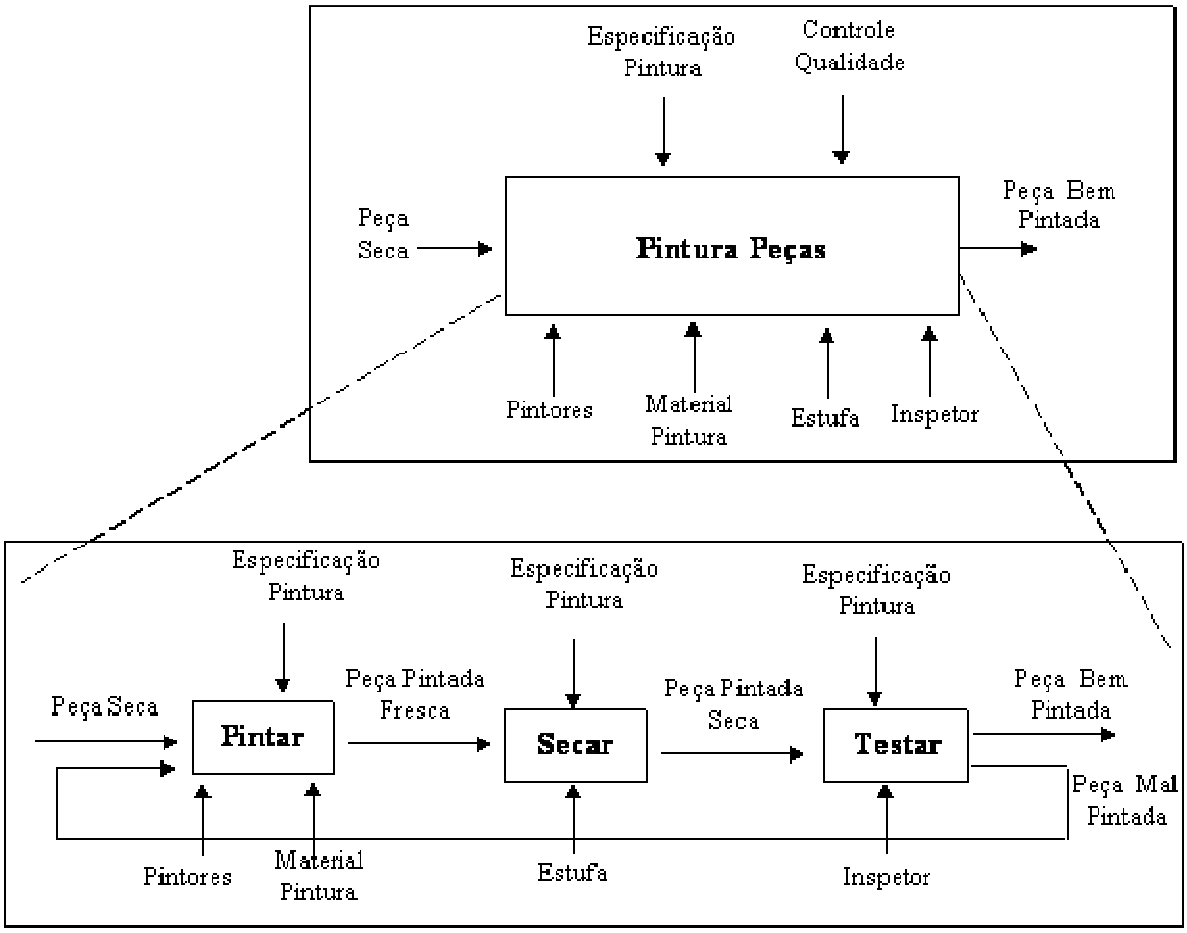


Figura 2.4 – Processo de Pintura modelado em IDEF∅

Primeiramente o processo de pintura, apresentado acima, é mostrado em uma visão macro, onde apenas a atividade "Pintar Peça" é apresentada, com suas entradas; saídas; controles e; mecanismos. Em seguida, abaixo da primeira versão, o mesmo processo é apresentado de forma mais detalhada, onde a atividade "Pintar Peça" é representada pelas atividades Pintar, Secar e Testar. Nesse segundo cenário as atividades se interligam através das entradas e saídas e os controles e mecanismos são distribuídos entre elas.

2.2.5 UML – Unified Modeling Language

A linguagem UML vem cada vez mais se consolidando na modelagem de sistemas dentro do paradigma Orientado a Objetos [23]. Também é tida como uma linguagem de propósito geral para a modelagem de software, possuindo habilidades para especificar, visualizar, construir, e documentar os artefatos de um sistema de software, podendo também ser aproveitada para modelagem de negócios e outras modalidades de sistemas. Esta definição é baseada em princípios e conceitos que a enquadram em qualquer tipo de sistema, seja software ou não [31].

UML se originou a partir da união da *Rational Software Corporation* [16] com três grandes metodologistas em sistemas de informação e indústria tecnológica, Grady Booch, James Rumbaugh, e Ivar Jacobson. Passando a ser utilizada nos mais diversos tipos de organizações, para posteriormente, em 17 de novembro de 1997, ser adotada pela OMG como um padrão [37].

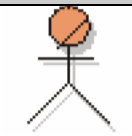
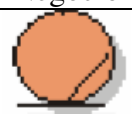
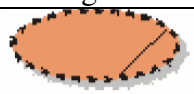
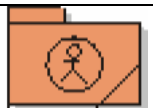
UML possui um vasto conjunto de diagramas, sendo útil tanto na especificação estrutural quanto comportamental de um sistema. Cada um desses diagramas é utilizado em diferentes fases do ciclo de vida de desenvolvimento de uma aplicação, que vão desde a fase de especificação até a fase de implementação. Com um total de nove diagramas definidos, UML é capaz de representar várias perspectivas, inclusive a representação comportamental.

Durante muitos anos UML vem sendo usada na modelagem de uma variedade de tipos de sistemas, que vão de sistemas de software, sistemas de hardware, bancos de dados, sistemas de tempo real e organizações. O suporte ferramental oferecido pela *Rational* para modelagem de negócio com UML possibilita a modelagem dos negócios das organizações e seus processos. O uso compartilhado de uma ferramenta simples para modelagem de sistemas e negócio, facilita o entendimento das necessidades organizacionais.

A seguir, na Tabela 2.4, apresentamos os elementos que fazem parte da notação utilizada por UML para modelagem de negócio. Os ícones específicos nesse tipo de modelagem foram criados graças ao poder de extensibilidade de UML. Tal notação

traduz os elementos de negócio existentes no ambiente de organizacional em modelos visuais [9].

Tabela 2.4 – Notação Utilizada Por UML

Ícone	Definição
 Ator de Negócio	O Ator de Negócio é considerado alguém ou algo fora da organização que interage com a mesma.
 Trabalhador de Negócio	O Trabalhador de Negócio representa um Papel ou conjunto de papéis dentro da organização. Ele interage com outros Trabalhadores de Negócio, manipulando Entidades de Negócio.
 Entidades de Negócio	As Entidades de Negócio são elementos manipulados ou utilizados por Trabalhadores de Negócio. Esses elementos podem sofrer transformações no decorrer do processo.
 Casos de Uso de Negócio	Uma seqüência de ações organizacionais (processo de negócio) que produza como resultado algum valor para um Ator de Negócio.
 Realização de Casos de Uso de Negócio	A Realização de Casos de Uso de Negócio é representada por uma coleção de diagramas para demonstrar como os elementos organizacionais são organizados para dar suporte aos processos de negócio.
 Unidade Organizacional	Uma Unidade Organizacional é composta por uma coleção de: trabalhadores de negócio, entidades de negócio, relacionamentos, realizações de casos de uso, diagramas, e outras unidades organizacionais. Usada para estruturar o modelo de negócio pela divisão em partes menores.

Nas subseções seguintes serão descritos os diagramas de UML que dão suporte a modelagem de negócio. Cada qual provendo um diferente ponto de vista do negócio na organização.

2.2.5.1 Diagramas de Casos de Uso

Do ponto de vista do uso de UML no desenvolvimento de sistemas, um caso de uso consiste em uma seqüência de ações que um ator (sendo um usuário do sistema, ou outro sistema externo) realiza sobre um sistema, para atingir um determinado objetivo. Tendo o papel de descrever um aspecto de utilização do sistema [10].

Do ponto de vista de modelagem de negócio, o diagrama de caso de uso possui o papel de representar as interações entre as entidades externas à organização (fornecedores, clientes, etc.), e os processos de negócio existentes na organização. Um diagrama de casos de uso de negócio deve representar visualmente as interações existentes entre os serviços oferecidos pela organização e aqueles que fazem uso destes serviços [9].

Atores e metas de processos negócios são elementos definidos no *profile* de UML para modelagem de processos de negócios, que são tratados pelo diagrama de casos de uso. Os Atores são claramente explicitados; as metas definem o objetivo de um processo de negócio. Para demonstrar a meta de um processo de negócio, pode-se utilizar um ou mais casos de uso. Por exemplo, na Figura 2.5 é utilizado apenas o caso de uso Realizar Venda, enquanto que na Figura 2.6 a meta do mesmo processo é alcançada pela realização dos casos de uso: Realizar Venda, Checar Estoque e Checar Perfil Cliente, aumentando o detalhamento das ações que precisam acontecer para atingir os objetivos do processo de negócio.

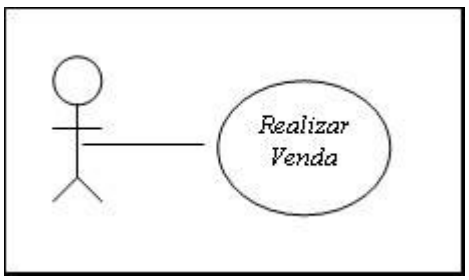


Figura 2.5 – Diagrama de caso de uso Realizar Venda

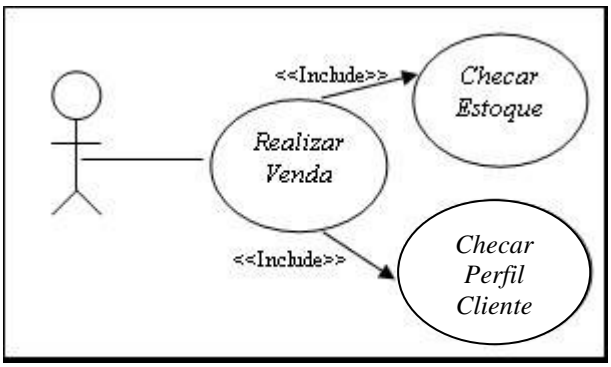


Figura 2.6 – Diagrama de caso de uso Realizar Venda Detalhado

Associar casos de uso a processos de negócio é um mecanismo útil para identificar os casos de usos que são realizados na perspectivas dos usuários com a realização de processo de negócio. Porém, casos de uso apenas refletem funcionalidades de alto nível, na perspectiva de um ou mais atores, não sendo capazes de representar o fluxo das atividades que ocorrem dentro de uma organização para atingir tais objetivos [10].

2.2.5.2 Diagramas de atividades

Na modelagem de um sistema, um diagrama de atividades de UML é definido como forma de expressar o dinamismo do sistema, através do seu fluxo de execução.

O diagrama de atividades utilizado na modelagem de negócio provê uma forma gráfica de documentar o fluxo das atividades existentes em uma organização. Possibilitando o entendimento de como ocorre o fluxo de execução das atividades pertencentes a um processo, quais atividades podem ser executadas em paralelo e a verificação de fluxos alternativos de execução do processo [9].

Além de modelar o fluxo de atividades, estes diagramas possuem a capacidade de modelar a realização de tarefas por múltiplos atores em paralelo. Para expressar isto graficamente, o diagrama é dividido em colunas, denominadas de *swimlanes*. Os *swimlanes* são bastante úteis para mostrar como os atores de negócio participam da realização do fluxo de execução das atividades. Além dos *swimlanes*, o diagrama de atividades possui elementos importantes como *barra de sincronização*, indicando pontos de execução onde o fluxo de execução pode ser bifurcado para execução paralela de atividades ou servindo como um ponto de convergência de várias atividades. Outro elemento que faz parte dos diagramas de atividades são os *pontos de decisão*,

representados por losangos, indicando caminhos condicionais através do processo de negócio.

Para modelagem de negócios os Diagramas de Atividades sofrem uma configuração especial, fazendo uso de objetos denominados de Entidades de Negócio, que podem ser criados, atualizados ou utilizados no decorrer do processamento das atividades. Se a modelagem de processos de negócio é feita com o objetivo de esclarecer os requisitos de um sistema, algumas entidades de negócios poderão vir a se tornar classes de análise durante projeto de um sistema orientado a objetos para dar suporte a tal processo.Na Figura 2.7 podemos observar o exemplo de um processo de negócio modelado em um diagrama de atividades.

Erro! Indicador não definido.

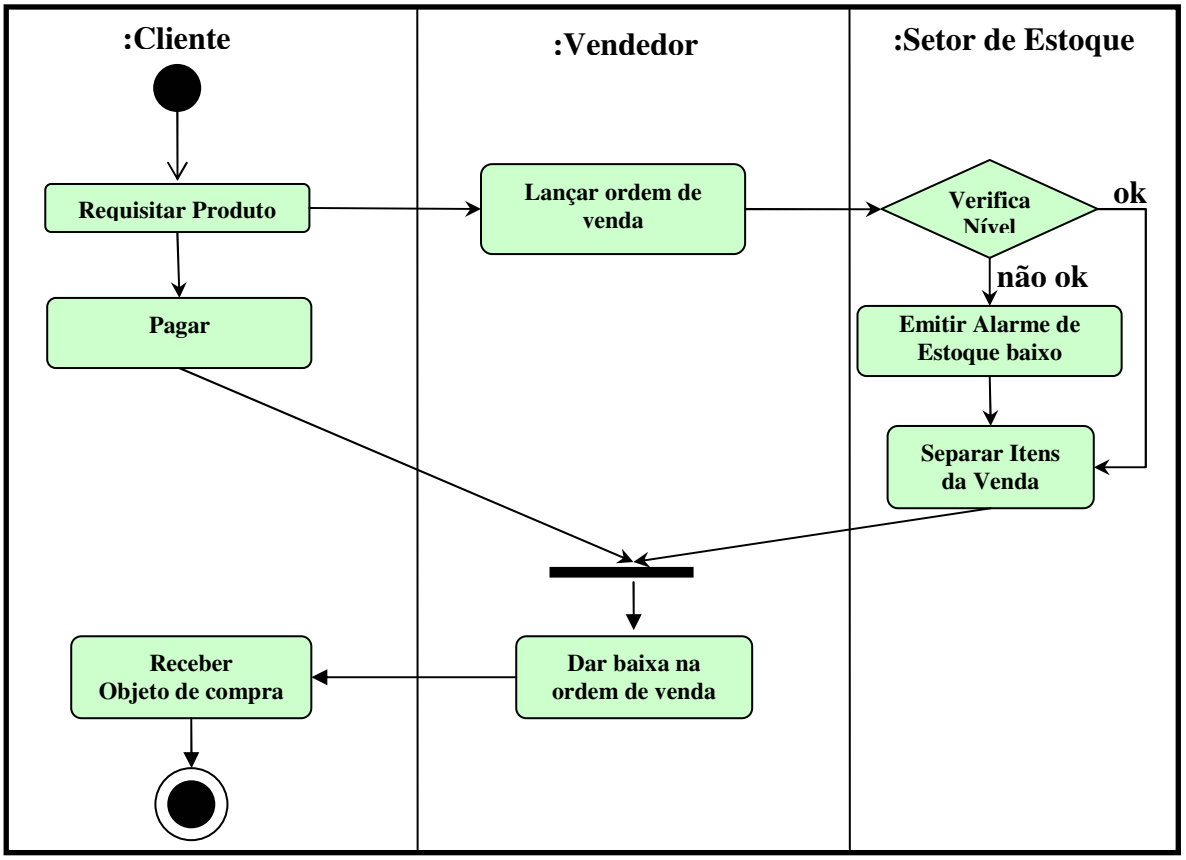


Figura 2.7 – Processo de Venda modelado em diagrama de atividades UML

Uma maneira complementar para documentar as atividades utiliza a narrativa como um instrumento importante para melhor esclarecer o modelo. A Tabela 2.5 apresenta um exemplo de como documentar os passos do processo referente ao exemplo citado acima, utilizando a técnica de narrativa. [37]

Tabela 2.5 – Processo de Venda descrito por narrativa

Passo	Cliente	Vendedor	Setor de Estoque
1	Requisitar produto		
2	Pagar pelo produto	Lançar ordem de venda.	
3			Verificar estoque do objeto da compra.
4			Separar Itens da venda.
5		Dar baixa em ordem de venda, liberando mercadoria mediante verificação de pagamento integral.	
6	Tomar posse de produto comprado.		

2.2.5.3 Diagrama de Classes

O diagrama de classes de negócio tem o objetivo de modelar a estrutura interna do negócio. Neste diagrama os elementos trabalhador de negócio e entidade de negócio são modelados como sendo classes, possibilitando a documentação dos seus relacionamentos, provendo uma forma fácil de visualizar como se dá à interação entre esses elementos. Com o diagrama de classes de negócio, é possível esclarecer como as entidades e trabalhadores de negócio interagem para a realização de um determinado processo de negócio. Esse tipo de diagrama é um dispositivo de bastante utilidade na modelagem da estrutura estática e relacionamentos das entidades de negócio [9].

2.2.5.4 Diagramas de Interação

Os diagramas de interação podem assumir dois formatos: colaboração ou; seqüência. Os diagramas de interação usados na modelagem de sistemas orientados a objetos modelam os tipos de interações que ocorrem entre os objetos que fazem parte do ambiente de execução do sistema, mais precisamente, quando objetos se comunicam com outros, através do envio de mensagens, onde os objetos emissores das mensagens realizam chamadas as operações que são executadas pelos objetos receptores das mensagens. A realização de um caso de uso corresponde a um conjunto de interações que são representadas dentro de um diagrama de interações.

Com foco na modelagem de negócios, os diagramas de interação, no formato de colaboração, documentam a realização das interações entre os trabalhadores de negócio e as entidades de negócio, necessárias para a execução das funções referentes aos negócios da organização. As interações entre as entidades de negócio e trabalhadores de

negócio são representadas via de trocas de mensagens. Análises podem ser realizadas sobre o diagrama de colaboração no sentido de identificar trabalhadores de negócio sobrecarregados com um grande número de tarefas, possibilitando tomadas de decisões que visem um aumento de produtividade na realização das funções de negócio organizacionais [9].

Apesar das informações colocadas nos diagramas de seqüência serem as mesmas informações tratadas pelo diagrama colaboração, elas são colocadas em um formato diferente, dando maior ênfase à seqüência das interações entre os elementos de negócio, provendo uma melhor visualização da execução de um caso de uso de negócio [37].

Uma limitação destacada no uso de diagrama de seqüência para modelagem de processos de negócio, consiste na falta de um padrão que possibilite a referência entre o diagrama de atividades em questão com outros diagramas de atividades, sendo esta uma característica fundamental para modelagem de complexos processos de negócios, onde vários níveis de refinamentos são necessários a fim de gerenciar a complexidade existente [10].

2.2.6 QPL – Quality Process Language

A QPL é definida como sendo uma notação para descrever os processos existentes em uma organização, juntamente com um framework para gerenciamento de qualidade [6]. O método da QPL considera alguns elementos básicos para a modelagem de processos como: as informações manuseadas pelos processos; proprietários de processos; autoridade; controle e; *links* entre as atividades que fazem parte do processo. Tais elementos são descritos na Tabela 2.6.

Tabela 2.6 – Elementos de diagrama QPL.

Informações	As informações descrevem os dados manipulados pelo processo e como eles estão organizados. Como por exemplo: documentos impressos; comunicação verbal; bancos de dados de computadores e; descrições de objetos materiais. Na QPL, os recursos consumidos pelo processo em execução são identificados através da verificação de que informações são alteradas ou modificadas durante a realização de um processo [6].
Processos	Opera sobre as informações e realiza transformações sobre a mesma. Processos incluem todas as regras de negócio existentes dentro da organização.
Proprietário do processo	Responsável pela execução do processo, podendo ser uma pessoa, um grupo de pessoas (como um departamento) ou uma máquina.
Autoridade	A especificação, descrição ou justificativa de um processo.
Controle	As condições ou restrições para a execução de um processo
Links	Descreve como as informações fluem entre os processos.

Na Figura 2.8 podemos observar como os elementos pertencentes à notação de QPL são representados.

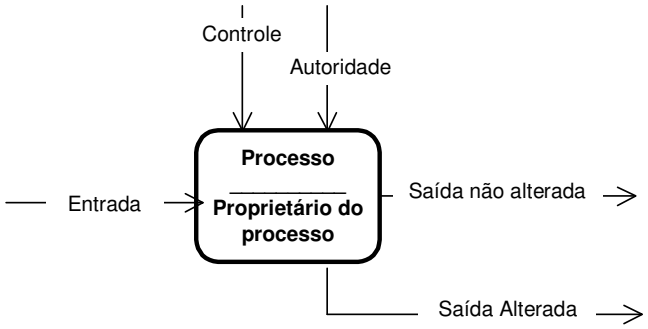


Figura 2.8 – Notação usada pela QPL.

2.3 Linguagens de Execução de Processos

As linguagens de execução de processos de negócio possuem o papel de representar processos em uma notação que possibilite sua execução. Elas não possuem utilidade na fase de análise e modelagem de processos. As linguagens de processos definidas até então possuem sua sintaxe expressa em XML, cada qual possuindo o seu formato nativo para representação de processos. Por isso, nenhuma dessas linguagens possui uma notação gráfica padrão para modelagem de processos.

Essas linguagens podem ser utilizadas por ferramentas de modelagem de processos de negócio, para dar suporte à simulação da execução de processos.

Foi com este objetivo que a WFMC (*Workflow Management Coalition*) desenvolveu a primeira linguagem com este fim. Uma nova versão XML da linguagem da WFMC foi lançada em 2002, com o nome de XPDL [42].

Outra iniciativa para concepção de uma linguagem para execução de processos partiu da BPMI (*Business Process Manager Initiative group*), lançada em 2001, denominada de BPL (*Business Process Language*) [2][7]. Tal acontecimento aqueceu os estudos voltados para esta área e deu muitas contribuições a sua sucessora, BPEL.

A BPEL (*Business Process Execution Language*) [7][4] foi uma iniciativa da Microssoft e IBM em resposta a tentativa da BPMI em colocar a BPML como a linguagem padrão para execução de processos de negócio. Porém, desta vez, a linguagem recebeu um foco de atenção bem maior, a ponto de ser suportada pelas maiores empresas do mercado na área, incluindo a (antes concorrente) BPMI, que não deu mais continuidade a BPML. BPEL, de fato, tornou-se o padrão adotado para execução de processos, sendo adotado por inúmeras ferramentas de modelagem. Desde 2003 a organização OASIS vem trabalhando na evolução continua da linguagem BPEL, por um grupo de engenheiros especialistas na área.

A BPMI, apesar de ter deixado de lado suas ambições com relação a BPML, deu continuidade ao seu projeto de uma notação visual para modelagem de processos, denominada BPMN (Business Process Modeling Notation). Tal notação será

apresentada em maior detalhe no próximo capítulo, visto que a mesma serviu como base para a definição da linguagem proposta neste trabalho.

Apesar da existência de poucos projetos voltados à construção de linguagens de execução de processos, esses projetos são de grande magnitude e importância, visto que envolvem uma quantidade grande de organizações renomadas, como por exemplo, o projeto da BPEL, do qual faz parte a IBM e Microssoft, entre outras organizações.

Capítulo 3

BPMN – Business Process Modeling Notation

A BPMN (Business Process Modeling Notation), desenvolvida pela BPMI [7], provê um diagrama para representação de processos de negócio, o qual foi projetado para ser utilizada tanto pelas pessoas que especificam como as que gerenciam processos dessa natureza.

A corporação sem fins lucrativos, BPMI, tem o papel de capacitar empresas de todos os tamanhos a desenvolver e operar processos de negócios. Sua missão principal é prover o desenvolvimento e uso da gerência de processos de negócio através do estabelecimento de padrões que possam ser utilizados, nas fases de projeto, execução, acompanhamento e otimização processos. Ela desenvolve especificações abertas, que venham facilitar o gerenciamento de processos de negócio.

Pessoas que lidam com os negócios da organização estão habituadas a visualizar os processos em representações visuais, baseando-se em simples fluxogramas. Neste contexto, com o passar do tempo diversas técnicas e vários tipos de diagramas apareceram, cada qual com suas vantagens e desvantagens. Com a carência de uma notação padronizada para modelagem processos de negócio, surge a BPMI, intencionada em criar uma notação universal para fechar esta lacuna. Para alcançar tal objetivo, essa notação deve ser simples e de fácil alcance a todas as pessoas que fazem parte ou possuem algum tipo de interesse nos processos da organização.

A notação gráfica de BPMN surgiu com o objetivo de oferecer a organizações a possibilidade de comunicar seus processos entre seus indivíduos de uma forma uniforme. Atualmente existem diversas ferramentas e metodologias para modelagem de processos de negócio. Dado que estes indivíduos possam migrar entre organizações e o fato das organizações normalmente lidarem com diferentes notações para a representação de seus processos, é necessário que os analistas estejam sempre aptos a utilizarem as diversas notações. Essa heterogeneidade de notações também pode ocorrer

quando se trabalha com o mesmo processo de negócio na mesma organização, a partir do momento que são utilizadas diferentes notações para representação de um processo em diferentes etapas do seu ciclo de vida, como: desenvolvimento; implementação; execução; monitoramento e; análise.

Uma notação padronizada facilita o entendimento da realização das colaborações e transações de negócio dentro e entre as organizações. Assegurando o entendimento interno da realização do processo bem como a comunicação entre diversas entidades que cooperam para a realização do mesmo, dando maior agilidade às organizações para se ajustarem às novas circunstâncias e cenários, tanto internamente quanto entre as organizações com quem se relaciona.

BPMN foi projetada no intuído de ser uma notação extensível por modeladores e ferramentas de modelagem. Possibilitando a adição de elementos ou artefatos, que não fazem parte da notação padrão, para satisfazer alguma necessidade de modelagem não suportada pela notação padrão. Apesar desta facilidade, deve-se ter sempre em mente o requisito da facilidade de compreensão, uma vez que, qualquer diagrama projetado, utilizando-se algum elemento adicionado a notação da BPMN deva ser facilmente entendido.

3.1 Sub modelos

Os elementos estruturais da BPMN permitem que o leitor do processo tenha a capacidade de facilmente identificar as diferentes seções de um diagrama BPMN. Existem três tipos básicos de sub modelos dentro de um modelo completo especificado em BPMN. Sendo estes, descritos nas próximas sub-seções.

3.1.1 Processos Privados

Os processos de negócio privados, ou internos, são aqueles que ocorrem dentro da organização, correspondendo à representação das atividades realizadas internamente e como elas interagem entre si. A Figura 3.1 ilustra um processo privado de venda, onde apenas as atividades que estão no âmbito interno da organização são retratadas no processo.

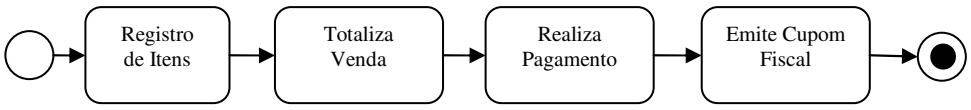


Figura 3.1 – Processo de Venda Privado.

3.1.2 Processos Abstratos

Os processos de negócio abstratos, conhecido também como processos públicos, retratam as interações das atividades pertencentes a um processo privado com outra entidade de negócio (outro processo ou um participante) externa ao processo privado. Nesses tipos de processos devem aparecer apenas as atividades que realizam algum tipo de comunicação com a entidade de negócio externa ao processo privado, além das conexões entre as atividades, que ditam o fluxo de execução do processo. As demais atividades, que não interagem diretamente com a entidade externa, não devem ser retratadas no modelo do processo. O processo abstrato de atendimento médico, ilustrado pela Figura 3.2, por exemplo, retrata apenas as atividades que têm interação direta com o paciente.

Erro! Indicador não definido.

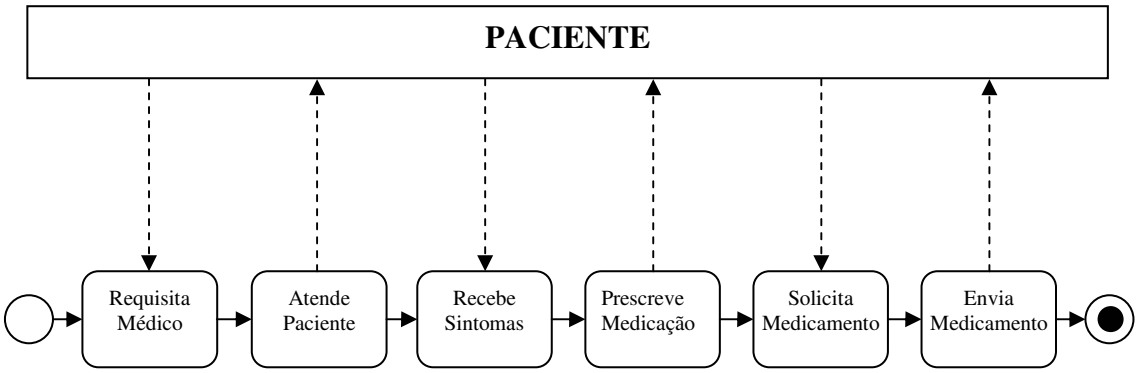


Figura 3.2 – Processo de Atendimento Abstrato.

3.1.3 Processos de Colaboração

Os processo de negócio do tipo colaboração modelam as interações entre dois ou mais processos de negócio. Nesta representação são apenas destacadas as atividades que se comunicam via troca de mensagens, entre processos distintos.

3.2 Diagrama de Processos de Negócio

O principal objetivo da BPMN é de ser uma notação simples, a fim de ser facilmente adotada por analistas de negócio. Além deste interesse, existe o requisito da BPMN ser suficientemente poderosa para representar processos de negócios de alta complexidade, o qual entra conflito com o primeiro. No intuito de compatibilizar estes desejos antagônicos, os elementos gráficos da BPMN são divididos em categorias, que serão vistas na próxima sub-seção.

3.2.1 Categorias de Objetos do Diagrama de Processos de Negócio

Para ser dotada de um mecanismo simples para prover modelos de processos de negócio e, ao mesmo momento, ser hábil na representação de processos cada vez mais complexos, a BPMN adotou a abordagem de categorizar os elementos existentes na notação. A notação define um pequeno grupo de categorias, facilitando o entendimento dos diagramas. É possível a adição de outros elementos a fim de tornar a notação mais poderosa e completa. As quatro categorias básicas de elementos são as seguintes:

Objetos de fluxo: São os objetos principais que definem o comportamento de processos de negócios. Esta categoria é composta por três tipos de objetos: Evento; Atividade e; Gatway.

Objetos de Conexão: São os objetos, que conectam os objetos de fluxo. Existem três tipos: Fluxo Seqüencial; Fluxo de Mensagem e; Associação.

Swimlanes: São entidades, pelas quais, são realizados os agrupamentos dos elementos Pools e Lanes, utilizados para particionar um diagrama BPMN de acordo com os responsáveis pela execução das atividades.

Artefatos: São objetos utilizados para prover informações adicionais sobre o processo. Apesar de modeladores e ferramentas de modelagem estarem livres para adicionarem tantos artefatos quanto forem necessários, BPMN tem por objetivo padronizar classes de artefatos dividindo-os nos seguintes grupos: Dados; Grupo e; Anotação.

3.2.2 Objetos do Diagrama de Processos de Negócio

As próximas tabelas exibem a lista completa dos objetos que podem ser utilizados em um diagrama BPMN. Devido à quantidade grande de elementos existentes na notação, os elementos serão mostrados em grupos, de acordo com seu significado na notação.

O Primeiro tipo de elemento apresentado é o evento, que é algo que ocorre dentro de um processo, afetando seu fluxo, normalmente tendo uma causa ou impacto. São representados por círculos e divididos em três tipos: inicial; intermediário e; final.

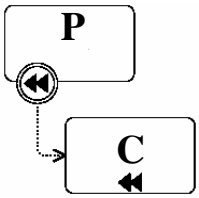
Tabela 3.1 – Elementos gráficos BPMN do tipo Evento.

Elemento	Descrição
 Inicial	Como o nome sugere, o evento inicial indica onde um determinado processo começa. Este tipo de evento inicia o fluxo do processo, desta forma não deve haver nenhum fluxo atingindo um evento inicial, apenas fluxo saindo do mesmo.
 Intermediário	Ocorre entre um evento inicial e final e afeta o fluxo de um processo, porém não possui os poderes para iniciar nem terminar um processo.
 Final	Como o nome sugere, o evento final indica onde um determinado processo termina. Este tipo de evento finaliza o fluxo do processo, desta forma, não deve haver nenhum fluxo saindo de um evento final, apenas fluxo atingindo do mesmo. Existem tipos especializados de eventos finais, que serão vistos nas próximas tabelas.

Os próximos elementos a serem mostrados, são do tipo Gatilho e Resultado. Eles são agrupados em nove categorias: Mensagem; Relógio (*Timer*); Erro; Cancelamento; Compensação; Regra; Link; Múltiplo e; Término. Todos os Gatilhos e Resultados serão expostos pela tabela abaixo.

Tabela 3.2 – Elementos gráficos BPMN do tipo Gatilho e Resultado.

Elemento		Descrição
Mensagem	 Inicial	Este tipo de evento é disparado com a chegada de uma mensagem, dando inicio a execução do processo.
	 Intermediária	Indica que a chegada de uma mensagem dispara um evento no meio de um processo. Fazendo com que o processo continue, caso ele esteja em pausa, ou mude o fluxo para o tratamento de uma exceção. Em um fluxo normal, ele é utilizado para enviar mensagem a outro participante.
	 Final	O evento final estabelece que uma mensagem deve ser enviada a um participante ao término do fluxo de um processo de negócio.
Relógio (*Timer*)	 Inicial	Este tipo de evento é utilizado para iniciar um processo em determinado momento. Este momento pode ser um determinado dia e hora, ou em um intervalo de tempo.
	 Intermediário	O Timer Intermediário é utilizado no decorrer de um processo de negócio, podendo representar uma data específica ou um intervalo de tempo. Quando utilizado no fluxo principal, serve como um mecanismo de atraso no processo.
Cancelamento	 Intermediário	É utilizado em sub-processos contidos em uma transação atômica. Ele deve aparecer anexado à fronteira do sub-processo, ficando associado a outra atividade, que dará continuidade a outro fluxo de execução no processo de negócio principal. O evento será ativado caso algum evento final de cancelamento seja atingido pelo fluxo do sub-processo em execução.
	 Final	Indica que a transação corrente deve ser cancelada, levando à ocorrência da ativação do evento intermediário de cancelamento, que por sua vez deve estar anexado à fronteira do sub-processo.

Compensação	 Intermediária  Compensação Anexada a um Processo	É utilizado para associar um sub-processo transacional a uma atividade de compensação, fazendo com que a atividade de compensação seja executada, caso o evento seja disparado. Esse evento, que deve aparecer anexado a fronteira do sub-processo, será ativado quando algum evento final de compensação é atingido pelo fluxo do sub-processo em execução. Uma atividade de compensação corresponde a uma atividade capaz de desfazer os efeitos causados pelo sub-processo. A representação desta configuração pode ser exemplificada de acordo com a segunda figura a esquerda, onde P corresponde ao sub-processo e C corresponde à atividade de compensação.
	 Final	Este tipo de evento final é utilizado dentro de um sub-processo transacional, para indicar a existência de um *rollback* no sub-processo. Caso este evento seja disparado pelo fluxo de execução, o evento intermediário de compensação anexado a fronteira do sub-processo será automaticamente disparado.
Regra	 Inicial	Este tipo de evento é disparado quando uma determinada condição a respeito do ambiente de execução do processo torna-se verdadeira. Fazendo com que o processo se inicie.
	 Intermediária	Este tipo de evento é apenas utilizado para tratamento de exceção. A regra corresponde a uma expressão que avalia algum dado em processamento, caso a avaliação produza o valor verdade, o evento será disparado.
Link	 Inicial	O Link Inicial é utilizado para ligar algum ponto em um processo ao início de outro. Processos ligados através de um *link* são vistos como sub-processos dentro de um processo maior.
	 Intermediário	Este tipo de evento é utilizado para ligar algum ponto no interior do processo a outro processo de negócio.
	 Final	Este tipo de evento é utilizado no final de um processo, ligando-o ao início ou meio de outro processo, via a utilização de eventos iniciais e intermediários, do tipo link, respectivamente.

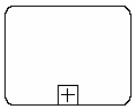
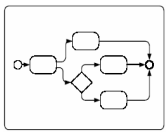
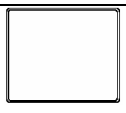
Múltiplo	 Inicial	Este tipo de evento indica que existem múltiplas formas para iniciar um processo, porém apenas uma delas será necessária. Bastando que um dos possíveis eventos ocorra.
	 Intermediário	Semelhante ao evento inicial do tipo múltiplo, porém aplicado ao interior de um processo.
	 Final	Este tipo de evento indica que múltiplas conseqüências irão ocorrer ao término de um processo. Como por exemplo, o envio simultâneo de várias mensagens.
Término	 Término	Este tipo de evento indica o ponto final da execução de um processo.
Conector	 Conector Página	O Conector de páginas, representado por um evento intermediário *Link*, é apenas utilizado em modelos de processos impressos, quando o processo ocupa nais de uma página. O Conector indica onde o fluxo seqüencial deixa uma página e onde da continuidade na página seguinte.
Erro	 **Intermediário**	Este tipo de evento é utilizado no tratamento de erro, seja para geração do erro quanto para captura. Um erro é levantado, caso o evento deste tipo faça parte do fluxo normal e seja atingido. O tratamento de um erro ocorre quando o evento está anexo à fronteira de uma atividade que gerou o erro. A forma de representar esta configuração é semelhante ao exemplo visto para o evento de compensação.
	 **Final**	Este tipo de evento representa a geração de um erro. O erro gerado por este evento pode ser tratado caso haja um evento intermediário de erro anexado a fronteira da atividade ou sub-processo causador do erro.

Tabela 3.3 – Elementos gráficos BPMN do tipo Atividade.

 Atividade	Uma Atividade é o termo genérico para a realização de alguma tarefa dentro de uma organização. Elas podem ser atômicas ou não. Existem três tipos de atividades: Processo; Sub-Processo e; Tarefa. Os dois últimos são representados por retângulos de bordas arredondadas, enquanto que processos não possuem fronteiras ou são representados pelo conteúdo dentro de um Pool.
 Tarefa	A tarefa corresponde a uma atividade atômica que faz parte de um processo. A Tarefa é utilizada na situação em que a atividade dentro do processo não pode ser decomposta.

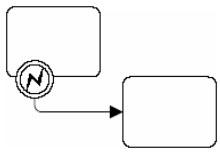
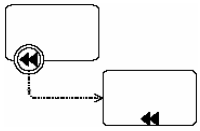
Um Processo encapsula uma coleção de atividades para a realização de uma tarefa. Já um Sub-Processo, que pode ser encapsulado ou expandido, pode ser utilizado dentro de um processo de maior magnitude, nesse contexto ele é visto como uma atividade do processo. O Processo pode ser modelado como tendo várias instâncias e ainda um sub-processo pode ter sua atomicidade garantida pela definição de uma transação sobre o mesmo. Todos estes aspectos são tratados na notação BPMN e vistos na Tabela 3.4.

Tabela 3.4 – Elementos gráficos BPMN do tipo Processo.

 Sub-Processo Encapsulado	Os detalhes de um sub-processo neste formato não são visíveis no diagrama. O sinal de mais, visto no inferior da figura, indica que a atividade é um sub-processo, que possui um nível mais profundo de detalhamento.
 Sub-Processo Expandido	Na forma expandida, o sub-processo é apresentado em detalhes. As atividades que fazem parte do sub processo tornam-se visíveis dentro da fronteira do processo.
 Múltiplas Instâncias	Esta representação, que é utilizada em objetos do tipo atividade ou sub-processos, indica a execução paralela de múltiplas instâncias do mesmo.
 Transação	Uma transação é representada por uma região que envolve um sub processo, dando o significado que tal sub-processo pertence a uma transação.

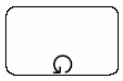
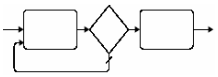
O Fluxo Seqüencial é utilizado para indicar a ordem em que as atividades são executadas dentro de um processo de negócio. Existindo seis tipos, detalhados em seqüência.

Tabela 3.5 – Elementos gráficos BPMN do tipo Fluxo.

Fluxo Normal	Fluxo normal é aquele que se inicia em um evento inicial e continua através das atividades de um processo, por muitas vezes seguindo caminhos alternativos e paralelos, se encerrando ao atingir um evento final. Este fluxo pode ocorrer de forma condicional ou não.
Fluxo não condicional	Um Fluxo Não Condicional corresponde ao fluxo que não é afetado por qualquer condição nem passa por um Gateway. A conexão entre duas atividades corresponde a um simples exemplo deste tipo de fluxo.
Fluxo Condicional	O fluxo condicional possui uma expressão condicional associada, devendo ser testada para que o fluxo prossiga. Este tipo de fluxo é identificado por um losângulo pequeno no início da seta.
Fluxo Default	Para os Gateways de decisão XOR e OR pode haver um fluxo de saída do tipo Default que será escolhido caso nenhum dos demais sejam.
 Fluxo de Exceção	Possibilita o desvio do fluxo normal de um processo. Ele é baseado em um evento intermediário, que representa uma exceção, que pode ocorrer durante a realização do processo, do qual ele faz parte.
Fluxo Mensagem	O fluxo de mensagens é utilizado para mostrar a comunicação entre dois participantes via troca de mensagens.
 Fluxo Compensação	Uma associação de compensação possibilita o desvio do fluxo normal do processo e é baseado em um evento intermediário de compensação.

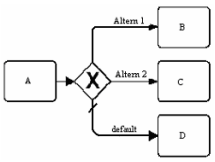
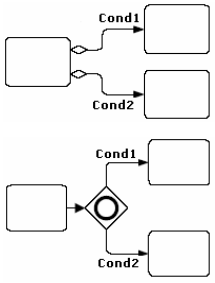
A notação de BPMN possui dois mecanismos para representar *loop* em um processo, sendo estes descritos na Tabela 3.6.

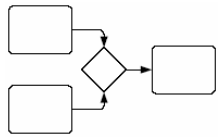
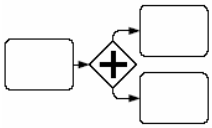
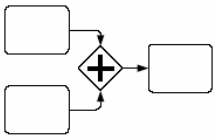
Tabela 3.6 – Elementos gráficos BPMN do tipo Loop.

 Loop Atividade	Este tipo de *loop* indica a ocorrência de multiplas execuções de uma atividade ou um sub-processo de forma sequencial.
 Loop Fluxo Seqüencial	Este tipo de loop é utilizado para conectar um dos possíveis fluxos de saída de uma atividade ao início da outra atividade.

Representados graficamente por losângulos, o Gateway é usado para realizar convergência ou divergência sobre um fluxo seqüencial. Dependendo do tipo, ele pode receber um ou mais fluxos de entrada e gerar um ou mais fluxos de saída. Ele pode representar os seguintes tipos de comportamento no fluxo de um processo: desvio (*branching*), bifurcação (*forking*), fusão (*merging*) e junção (*joining*). Marcadores colocados no interior do Gateway irão indicar seu tipo. Esses tipos são mostrados a baixo.

Tabela 3.7 – Elementos gráficos BPMN do tipo Geteways de Decisão.

 Geteway Exclusivo	Um Gateway Exclusivo (XOR) seleciona apenas um dos possíveis fluxos de saída em tempo de execução. A decisão pode acontecer baseado na avaliação dos dados processados pelas atividades ou pela ocorrência de algum evento.
 Geteway Inclusivo	Gateway Inclusivo define um ponto de decisão, onde um ou mais possíveis fluxos podem ser escolhidos ao mesmo tempo. Uma condição default pode ser usada para assegurar que haverá pelo menos um fluxo de saída. Existem duas versões para este tipo de Gateway, a primeira usa vários Fluxos Condicionais saindo de uma atividade e a segunda utiliza um Gateway OR, normalmente utilizado quando há combinação com outro Gateway em um fluxo de saída.

 Merge	BPMN utiliza o termo “Merge” para referenciar a combinação exclusiva de dois ou mais fluxos em um único fluxo.
 And  Fork  Join	O Gateway AND pode ser utilizado em operações de fork e join. No fork, ele realiza o papel de transformar um fluxo seqüencial de execução em diversos fluxos paralelos, possibilitando a execução paralela de atividades. Utilizado para representar o join, o gateway AND recebe mais de um fluxo de entrada e alinha todos em apenas um fluxo de saída. A atividade conectada pelo fluxo de saída só inicia sua execução quando todas as atividades que se conectam ao AND terminam suas execuções.

Os elementos gráficos apresentados em seqüência, são mais voltados a auxiliar a documentação e prover um nível maior de informações referentes ao processo de negócio modelado.

Tabela 3.8 – Elementos gráficos BPMN do tipo Artefato.

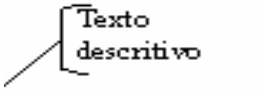
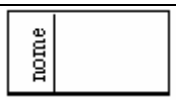
 Dados	Os dados são artefatos, pois não afetam diretamente o fluxo de execução ou o fluxo de mensagens, apenas oferece informações sobre o que as atividades necessitam para serem processadas e o que elas produzem após seu processamento.
 Grupo	Um grupo de atividades não deve afetar o fluxo das mesmas. Apenas sendo usado com propósito de documentação e análise.
 Anotação	As anotações são utilizadas apenas para prover informacoes adicionais sobre o funcionamento do processo.

Tabela 3.9 – Elementos gráficos BPMN do tipo Swimlanes.

 Pool	Um Pool representa um participante em um processo. Dentro do Pool existem várias atividades que pertencem ao contexto do participante em questão.
 Lane	Um Lane consistem em uma partição dentro de um Pool, estendendo todo comprimento do Pool, seja de forma vertical ou horizontal. É utilizado para organizar e categorizar as atividades dentro de um Pool.

Os elementos existentes na notação gráfica de BPMN, apresentados nas tabelas acima, dão uma boa visão do poder da notação. A sua completude, o projeto bem feito e a credibilidade da BPMI, sua mantenedora, nesta área foram alguns dos fatores que despertaram o interesse de inúmeras ferramentas de modelagem de processo de negócio a adotarem esta notação como uma notação padrão.

No Capitulo 5, onde uma linguagem para processos de negócio é apresentada, a BPMN é utilizada como fonte de inspiração na definição dos operadores da mesma. Ela também é utilizada em alguns exemplos de modelos de processos para que seja demonstrada a equivalência de modelos de processos definidos na BPMN com os processos escritos na linguagem.

O próximo capítulo é dedicado à apresentação do *framework* de Semântica de Ações utilizado como ferramenta para a definição da semântica da linguagem proposta.

Capítulo 4

Semântica de Ações

Desenvolvida originalmente por Peter Mosses [27] [26], juntamente com a colaboração de David Watt [39], Semântica de Ações consiste em um *framework* com grande potencial para representação de processamento de informações. Ela foi concebida com o objetivo de ser utilizada na especificação de linguagens de programação, servindo para realizar sua documentação e auxiliar na definição de padrões de implementação. Apesar de ser baseada em uma notação formal, é facilmente compreendida, fazendo com que as especificações baseadas em sua notação tornem-se bastante acessíveis a programadores [26]. Possibilitando, inclusive, tais especificações serem agregadas a descrições informais, na especificação de uma linguagem de programação.

Uma descrição em semântica de ações é extremamente modular, possuindo propriedades como: extensibilidade; modificabilidade e; reusabilidade. Esta última, bastante requerida por projetistas de linguagens de programação.

A notação de ações é composta por primitivas e combinadores de ações. Cada primitiva representa um tipo especial de processamento de informações enquanto que cada combinador representa uma particular mistura de controle de fluxo e vários tipos de fluxo de informações. A notação de ações foi projetada com primitivas e combinadores suficientes para expressar as formas mais padronizadas de controle de fluxo de dados. Também é inerente em Semântica de Ações a utilização de uma notação básica para dados. Dentre os tipos, encontram-se valores *booleanos*, números racionais, listas, tuplas e Maps.

Semântica de Ações possui uma estrutura composicional, onde a semântica de cada frase da linguagem é determinada pela semântica de suas sub-frases. Ações são entidades que representam potencialmente o processamento de informações, ou seja, o significado de programas e as frases que o compõem.

Semântica de Ações possui todas as propriedades desejáveis a um método para descrever linguagens de programação [41]:

- **Legibilidade**: Sua notação é baseada em palavras bem sugestivas, deixando de lado o uso de simbologias complexas e obscuras, aumentando a legibilidade de suas descrições semânticas. Outro ponto que contribui bastante para suas clareza consiste na utilização de conceitos bem conhecidos computacionalmente.

- **Modularidade**: Especificações baseadas em sua notação são construídas sobre um conjunto padronizado e pequeno de ações primitivas e combinadores de ações, que correspondem a um notável recurso utilizado pela notação de ações e ditam o fluxo de processamento de informações [27]. Tais propriedades possibilitam o uso de partes de uma especificação, baseada neste *framework*, na construção de novas especificações.

- **Abstração:** Sua notação é baseada em conceitos genéricos de processamento de informação e controle de fluxo, não se baseando em detalhes de implementação.

- **Comparabilidade:** A utilização de primitivas e combinadores de ações padronizados facilitam a comparação semântica entre linguagens descritas na notação de ações.

Uma descrição baseada em Semântica de Ações é composta por três módulos principais, sendo estes: Sintaxe Abstrata; Entidades Semânticas e; Funções Semânticas [27].

4.1 Sintaxe Abstrata

A sintaxe de linguagens de programação determina a legalidade de um programa e o relacionamento entre os símbolos e as frases encontrados nele. A sintaxe é definida sobre dois aspectos, sintaxe concreta e sintaxe abstrata.

A sintaxe concreta se concentra na análise de seqüências de caracteres para a validação de um programa. Ela é normalmente obtida pela análise léxica e sintática das frases, onde o primeiro tipo de análise se concentra em validar seqüências de caracteres como palavras legalmente válidas em um programa, conhecidas também como *lexemis;* já a

análise sintática valida a estrutural das frases, analisa a combinação dos *lexemes* para a formação de frases que sejam válidas em um texto que represente programa.

A sintaxe abstrata oferece uma interface apropriada entre a sintaxe concreta e a semântica de uma linguagem. Para atingir este objetivo a sintaxe abstrata ignora todos os detalhes que não sejam semanticamente significantes para representar essencialmente a estrutura composicional de um programa.

A notação adotada para representação da sintaxe abstrata de linguagens, no *framework* de semântica de ações, utiliza gramática livre de contexto, a qual faz uso de expressões regulares para sua representação. Para melhor demonstrar tal notação na representação de Sintaxes Abstratas, será apresentado um pequeno exemplo da Sintaxe Abstrata de uma linguagem de programação simplificada.

Gramática:

```
(1) Comando        ::= [Identificador ":=" Expressao] |
                       ["if" Expressao "then" Comando⟨"else" Comando⟩?]|
                       ["while" Expressao "do" Comando] |
                       ["begin" Comando "end"].
(2) Comando        ::= ⟨Comando ⟨ ";" Comando⟩* ⟩.
(3) Expressão      ::= Numeral | Identificador | ["(" Expressao ")"]|
                       [Expressao Operador Expressao].
(4) Operador       ::= "+" | "-" | "*" | "≠" | "and" | "or".
(5) Numeral        ::= [digito+].
(6) Identificador  ::= [letra (letra | digito)*].
(7) Digito         ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
(8) Letra          ::= "a" | ... | "z" | "A" | ... | "Z".
Closed
```

Figura 4.1 – Gramática de Linguagem Fictícia

Tal especificação é composta por um conjunto de equações. Onde cada equação é similar a um grupo de produções em uma gramática, usando-se '|' para separar as alternativas possíveis para cada elemento da gramática. Símbolos terminais são escritos entre aspas duplas, como "while" e "+". Enquanto que os não terminais são escritos livremente sem as aspas duplas e começando com letra maiúscula, como por exemplo: Expressao, Comando e Operador. As equações (2), (5) e (6) fazem uso das expressões regulares, por exemplo.

Na segunda alternativa da equação (1) (“if” Expressao “then” Comando ⟨”else” Comando⟩?) da gramática exemplificada, é utilizada a parte opcional ⟨”else” Comando⟩? , que equivaleria à utilização da seguinte alternativa:

```
[“if” Expressao “then” Comando ] |
[“if” Expressao “then” Comando  ”else” Comando]
```

Porém a utilização da alternativa que possui parte opcional, torna a gramática mais concisa.

Sempre é escrita, no final da gramática, a palavra **closed,** indicando que todas as equações da gramática já foram especificadas e a mesma não estaria aberta a inclusão de novas equações.

4.2 Entidades Semânticas

As entidades semânticas servem para representar o comportamento de programas, com independência de implementação. Em semântica de ações, são utilizados três tipos de entidades semânticas: Ações; Dados e; Yielders. O principal tipo entre os tipos de entidades consiste nas Ações, ficando em segundo plano os Dados e Yielders.

4.2.1 Ações

As ações são, essencialmente, entidades computacionais e dinâmicas. A execução de uma ação representa o processamento de informações e reflete o natural passo-a-passo computacional. A notação utilizada para representar ações é denominada de Notação de Ações.

O processamento de uma ação, que pode fazer parte de outra ação (ação pai), pode levar a uma das possíveis situações:

- ***completes***: Representa o término normal de uma ação;
- ***escapes***: Representa uma terminação com ocorrência de uma exceção, fazendo com que a ação pai seja desviada, caso o *escape* tenha sido capturado.
- ***fails***: Representa o abandono da execução de uma ação, caso a ação pai possua alternativas a ser executada, uma destas será escolhida, caso contrário à ação pai também será levada à situação de *fail*;

- ***diverges:*** Leva a uma execução infinita da ação, levando inclusive a ação pai entrar nessa situação.

A semântica de programas pode ser descrita utilizando ações, pois a sua execução corresponde ao comportamento de programas. As ações processam alguns tipos diferentes de informações, que podem ser classificados de acordo com a intensidade de propagação da informação, sendo estes tipos classificados de acordo com a seguinte relação:

- ***transient***: Corresponde a tuplas de dados que alimentam ações para a realização de processamento, também sendo utilizadas para representar informações oriundas da execução de ações. Este tipo de informação encontra-se disponível apenas para uso imediato em uma ação. Informações *transient* representam informações computadas durante a execução de um programa. Uma Coleção deste tipo de informação é representada em tuplas, facilitando, desta forma, o acesso a cada item de dado.

- ***scoped***: Mapeamento de identificadores (*tokens*) para dados. Os dados representados pelos *tokens* podem ser referenciados a qualquer momento durante a ação.

- ***stable***: Dados armazenados em células. As informações do tipo *stable*, podem ser criadas e alteradas por diversas ações, isto é realizado apenas fazendo referência à célula que guarda o dado. Este tipo de informação é persistido até o momento que é explicitamente destruído.

- ***permanent***: Dados comunicados entre ações distribuídas através do envio de mensagens.

4.2.2 Dados

Dados, em semântica de ações, consiste nas informações que são processadas pelas ações. Vários elementos bem conhecidos são utilizados na representação de dados, como: valores booleanos, números, caracteres, strings, listas, Sets e Maps. Para cada um dos tipos de dados relatados existe, na notação de ações, um conjunto de operadores

associados a esses tipos de dados. Novos tipos de dados podem ser especificados livremente para representação de informações especiais.

Na notação de ações a palavra *data* é utilizada para representar qualquer entidade abstrata que represente algum tipo de dado. Já a palavra *individual* representa um único elemento pertencente a um dos possíveis tipos de dados. Uma ação quando encapsulada em uma estrutura chamada *abstraction* também pode ser utilizada como um dado, permitindo ser manipulada por outras ações.

4.2.3 Yielders

Yielders são elementos que podem ser avaliados para produzir dados durante o processamento de uma ação. Os dados produzidos precisão acessar as informações correntes. Existem primitivas de Yielders previamente definidas na notação de ações. Como exemplo podemos apresentar o seguinte Yielder: the given number#2. Que, quando utilizado em uma ação, produz como resultado o valor numérico existente na posição 2 de uma tupla recebida pela ação corrente.

4.3 Funções Semânticas

As funções semânticas são utilizadas para mapear a sintaxe abstrata em entidades semânticas que representem o seu comportamento. As especificações de funções semânticas são realizadas utilizando-se equações semânticas. Onde, cada função semântica deve possuir apenas um argumento.

Para exemplificar, será ilustrada todas as funções semânticas necessárias para dar um significado semântica à sintaxe apresentada na Seção 4.1. As funções semânticas utilizadas são: **execute_, avalie_, the operation result_, the value of_.** A utilização de mais de uma equação semântica para representar uma função semântica é comum na definição de funções semânticas. A função semântica **execute_**, por exemplo**,** que opera sobre o elemento Comando, é composta de seis equações semânticas, com cada uma sendo aplicada a uma das possíveis formações do elemento não terminal Comando.

```
execute _  :: Comando → action[completing | diverging | storing].
(1) execute [I: Identificador ":=" E: Expressao] =
          | give the cel bound to I and avalie E
            then
          | store the given number#2 in the given cell#1.
(2) execute ["if" E: Expressao "then" C: Comando] =
          evaluate E  then
            | check the given truth-value and then execute C
            or
            | check not the given truth-value and then complete.

(3) execute ["if" E:Expressao "then" C1:Comando "else" C2:Comando]=
         evaluate E  then
            | check the given truth-value and then execute C1
            or
            | check not the given truth-value and then execute C2.
(4) execute ["while" E:Expressao "do"  C:Comando] =
     unfolding
       evaluate E  then
       |check the given truth-value and then execute C and then unfold
        or
       | check not the given truth-value and then complete.
(5) execute ["begin" C:Comando "end"] =   execute C.
(6) execute(C1:Comando";" C2:Comandos)=execute C1 and then execute C2.

avalie_  :: Expressao → ação [giving a value].
(7) avalie N:Numeral = give the value of N.
(8) avalie I:Identificador = give the value bound to I.
(9) avalie [ "(" E:Expressao ")" ] = avalie E.
(10) avalie [E1:Expressao O:Operador E2:Expressao]  =
      (avalie E1 and avalie E2) then give the operation-result of O.

the operation-result_:: Operador →
                        yielder[of a value][using the given value#2].
(11) the operation-result "+" = the number yelded by
        the sum of (the given number#1, the given number#2) .
(12) the operation-result "-" = the number yelded by
        the diference of (the given number#1, the given number#2) .
(13) the operation-result "*" = the number yelded by
        the product of (the given number#1, the given number#2) .
(14) the operation-result "≠" =
        not (the given valuer#1 is the given valuer#2) .
(15) the operation-result "and" =
        both of (the given truth-value#1, the given truth-value#2) .
(16) the operation-result "or" =
        either of (the given number#1, the given number#2) .

the value of_ :: Numeral → number
(17) the value of  N:Natural = number & decimal N .
```

Os nomes utilizados para representar as funções semânticas consistem de frases, que podem ser definidas livremente. Normalmente, são utilizadas frases bem sugestivas que venham a facilitar o entendimento da função.

O lado direito das equações semânticas utiliza a notação padrão, para ações e dados, provida por semântica de ações.

Cada função semântica deve ter sua funcionalidade especificada, como por exemplo, a especificação da função semântica: **avalie_::Expressao → ação[giving a value]** assegura que todas as equações semânticas utilizadas na sua especificação (equações 7, 8, 9 e 10) só podem receber como parâmetro um elemento do tipo Expressão, obedecendo a sintaxe abstrata definida previamente, e seu significado corresponde a uma ação que, quando executada, retorna um valor.

A utilização de [...] como parâmetro de uma equação semântica corresponde a aplicação da equação a um dos possíveis nós que a árvore sintática abstrata pode assumir de acordo com a especificação da gramática da linguagem. Já no exemplo apresentado pela equação 6, é utilizado (...), como parâmetro para a equação semântica, denotando desta forma a aplicação da equação semântica a uma tupla, onde cada elemento da tupla corresponde a um possível nó da arvore de sintaxe abstrata que o tipo do elemento da tupla pode assumir.

É uma prática bastante comum a representação de equações semânticas em termos de outras equações semânticas. Como pode ser observado na Equação 7 `(avalie N:Numeral = give the value of N)`, a qual faz referência a função semântica definida pela Equação 17 `(the value of  N:Natural = number & decimal N)`.

As funções semânticas devem ser definidas de forma completa e consistente sobre as árvores de sintática que define a linguagem. Deste modo, para cada elemento principal da sintaxe da linguagem é necessário à especificação de uma função semântica. Caso o elemento da sintaxe assuma mais de uma formação possível deve existir uma equação semântica para cada uma das possíveis formas deste elemento da sintaxe.

4.4 Facetas

A notação de ações é apresentada em uma divisão de grupos de ações e Yielders, onde cada um desses grupos é um tipo de Faceta. Excluindo a Faceta Básica, todas as demais (Funcional, Declarativa, Imperativa e Comunicativa) estão intimamente relacionadas às categorias de informações apresentadas anteriormente (*transient, scoped, stable, permanent*).

4.4.1 Básica

A faceta básica lida com ações básicas, que estão fortemente relacionadas a conceitos básicos de controle de fluxo. As ações apresentadas aqui servem como alicerce para a representação das demais ações existentes nas outras facetas.

O processamento de uma ação corresponde a um passo atômico, realizado por um ou mais agentes. Neste momento será considerada apenas a existência de um único agente realizador da execução de ações, onde passos de execução ocorrem em uma ordem bem definida, não sendo permitida a execução paralela. Neste cenário é assegurado que, em determinado passo de execução, a informação corrente disponível é bem definida. Desta maneira, a execução de uma ação é tida como a realização de passos seqüenciais de execução.

A execução de uma ação pode ser finita e terminar ou infinita e divergir. Nestas situações a execução das ações são consideradas como sendo dos tipos *complete* e *diverge*, respectivamente. Porém, em certa situações, é possível que a execução de uma ação escape da terminação normal, deixando de dar continuidade ao fluxo de execução natural. Neste caso, a terminação da execução da ação é tida como anormal e o tipo de terminação associada a esta situação é denominada de *failure*.

4.4.1.1 Ações e Combinadores

Adiante, será mostrada toda a simbologia que compõe a notação básica de ações. O símbolo action, elemento primordial da notação de ações, denota o conjunto de todas as ações que fazem parte da notação de ações, não restringindo apenas as ações básicas, mostradas nesta seção. As funcionalidades associadas a cada uma das ações (complete, escape, fail, commit, diverge e unfold) e aos combinadores de ações (unfolding_, indivisibly_, _or_, _and_, _ and then _ e _trap_), especificados abaixo, serão discutidos em seqüência.

- `complete : action`
- `escape : action`
- `fail : action`
- `commit : action`
- `diverge : action`
- `unfold : action`
- `unfolding_ :: action → action`
- `indivisibly_ :: action → action`
- `_ or _ :: action, action → action`
- `_ and _ :: action, action → action`

- `_ and then _  :: action, action → action`
- `_ trap _  :: action, action → action`

A execução da ação primitiva `complete` consome um único passo de execução em uma ação composta e termina normalmente, dando seqüência ao fluxo normal de execução. A ação `escape` indica uma terminação anormal, não permitindo a continuidade do fluxo normal de execução da ação composta, que atingiu o `escape`. Tal fluxo será apenas redirecionado caso a ação composta seja o parâmetro à esquerda de um `trap`, que direcionará a execução para a ação existente à direita do `trap`.

A execução da ação `fail` aborta a execução corrente, fazendo com que a alternativa em execução seja abandonada e, caso haja outras alternativas possíveis, uma destas será executada. A ação `A1 or A2` representa a escolha independente de implementação de uma das alternativas a ser executada. Caso uma das ações, existentes entre um combinador de ação `_or_`, atinja uma ação `fail`, a outra alternativa possível será escolhida para execução. Quando há a ocorrência de uma ação primitiva `commit` em uma alternativa em execução, todas as demais alternativas são descartadas por completo.

A execução de `A1 and A2` escalona passos de execução entre `A1` e `A2` até que ambas completem, ou até que uma delas atinja um `escape` ou `fail`. Caso se queira forçar a execução de uma ação em um passo atômico, deve-se fazer uso do `indivisibly`, onde execução da ação `indivisibly A` faz com que a ação `A` seja executada em um único passo, impedindo que passos de execução sejam intercalados entre esta ação e qualquer outra. Diferente de `A1 and A2`, a ação `A1 and then A2`, assegura que as ações `A1` e `A2`, serão executadas em uma ordem bem estabelecida, na qual `A2` só começara a ser executada após o término da execução de `A1`.

A execução da ação `diverge` nunca termina a execução. O combinador de ação `unfolding_` quando aplicada à ação `A` (ou seja, `unfolding A`) faz com que a ação `A` seja executada até a ação `unfold`, sub-ação da ação `A`, seja atingida, fazendo com que a ação `A` seja novamente executada.

4.4.1.2 Outcomes

Os outcomes são entidades auxiliares, utilizadas na especificação de ações, indicando os possíveis resultados da execução de uma ação. Na faceta básica os seguintes outcomes são introduzidos:

- `outcome = completing| escaping| committing| diverging| failing|`□`.`
- `failing = nothing.`

Os possíveis outcomes de uma ação são especificados da seguinte forma: `A[O]`, onde `A` corresponde à ação e "`O`" corresponde aos possíveis outcomes da ação, sendo estes separados por `|`.

Os outcomes `completing`, `escaping`, `diverging` e `failing` representam término normal, término anormal, não término e falha, respectivamente. Como `failing` é um outcome de ocorrência possível na execução da maioria das ações, caso em que a ação referencia uma informação indisponível, ele é considerado um outcome implícito, por isso, o mesmo é igualado a `nothing`, que representa ausência de outcome. O outcome `commit` permite a ocorrência de uma sub-ação `commit` durante a execução da ação.

4.4.2 Funcional

A faceta funcional da notação de ações é voltada para a especificação de fluxo de dados, onde informações do tipo `transient`, representadas em *tuplas* de dados, são produzidas e consumidas por ações.

As ações primitivas existentes na faceta funcional apenas são capazes de produzir informações do tipo transientes, nas situações em que o outcome resultante de sua execução forem `completing` ou `escaping`. Porém, quando o outcome resultante for `diverging` ou `failing`, a ação não produzirá, como resultado de sua execução, qualquer informação `transient`.

Os dados `transient` resultantes de uma ação correspondem aos dados oriundos de sua execução, que podem ser produzidos durante computações realizadas na execução desta, ou podem meramente corresponder a informações de entrada de uma ação que foram repassadas. Quando uma ação é representada por duas sub-ações, o resultado gerado pela sua execução pode corresponder à concatenação dos valores produzidos por ambas

sub-ações ou apenas pelos transientes resultantes da ação que realizou o último passo de execução, dependendo do tipo de combinador de ações utilizado.

A notação de ações funcional corresponde a uma extensão da notação básica de ações e de dados, com a inclusão de novas ações primitivas, um novo combinador, e alguns novos Yielders.

4.4.2.1 Ações e Combinadores

- `give _ :: yielder → action .`
- `escape whith _ :: yielder → action .`
- `regive :: action .`
- `choose _ :: yielder → action .`
- `check _ :: yielder → action .`
- `_ then _ :: action, action → action .`

A execução de `give Y`, onde `Y` é um yielder, retorna os dados transientes produzidos pela avaliação de `Y`. Caso `Y` gere algum resultado, `give Y` leva ao outcome `completing`, caso contrário, o outcome resultante será `failing`. Como uma *tupla* vazia gerada por `Y` consiste em um resultado produzido pelo yielder, `give()` faz com que o outcome resultante seja `completing`, sendo que, neste caso, retornando uma *tupla* vazia. A ação primitiva `regive` apenas repassa todos os transientes recebidos sem realizar qualquer tipo de processamento sobre os dados. A ação `escape whith Y` realiza a execução da ação `escape` e, em seguida, retorna os dados produzidos por `Y`.

A ação `choose Y` consiste em uma generalização de `give Y`, realizando uma escolha entre um dos transientes produzidos por `give Y` para ser o retorno resultante de sua execução. Da mesma forma de `give Y` a ação `choose Y`, falha quando Y não gera nenhum resultado.

A ação `check Y` necessita que `Y` seja avaliado para um valor *boolean*. Caso o valor seja *true,* a ação completa, outcome igual a `completing`. Caso contrário, ela falha, outcome igual a `failing`. É comum a utilização da ação `check Y` em testes para a execução de uma ação ou outra. Por exemplo, `(check Y and then A1) or (check not Y and then A2)`, onde, quando Y é avaliado para *true,* a ação `A1` é executada, caso contrário, a ação a ser executada será `A2`.

O combinador de ações `A1 then A2` representa a composição das ações `A1` e `A2`, onde `A2` só será executada após a execução de `A1`. Os dados recebidos pela ação `A1 then A2`

são apenas passados à ação `A1`, a qual pode realizar algum tipo de processamento sobre esses dados e gerar novos dados de saída, que serão apenas repassados a `A2`. Os transientes resultantes da execução de `A1 then A2` são os resultantes da execução de `A2`.

4.4.2.2 Yielders

Na faceta funcional novos yielders são introduzidos à notação de ações. Sendo estes:

- `given_ :: data → yielder`
- `given _#_ :: datim, positive-integer → yielder`
- `it : yielder`
- `then : yielder`
- `the _ yelded by _ :: data, yielder → yielder`

O Yielder `given Y` produz os dados resultantes da avaliação de `Y`, contanto que os dados recebidos para avaliação sejam do tipo `Y`, desta forma, `the given truth-value` (onde o `the` é opcional, tendo apenas a função de facilitar a legibilidade) deve resultar `true` ou `false`, por exemplo. Caso o valor recebido não seja um `truth-value`, a ação irá falhar. Já o yielder `given Y#n` acessa o n-ésimo elemento pertencente a tupla recebida pelo yielder `Y`. Sendo assim, a construção `the given truth-value#2`, é interpretada como um yielder que espera receber como parâmetro uma tupla, na qual o segundo elemento da tupla seja do tipo truth-value.

Os yielders `it` e `then` capturam transientes recebidos por uma ação, onde o `it` considera que apenas um item foi recebido pela ação.

O yielder `the _ yelded by _` enfatiza que o valor produzido pelo yielder no segundo parâmetro é do tipo de dado identificado pelo primeiro parâmetro. Como por exemplo, `the natual yelded by it`, enfatiza que o dado capturado por `it` é do tipo `natural`.

4.4.3 Declarativa

As ações pertencentes à faceta declarativa manipulam informações do tipo *scoped*, que têm uma propagação maior que as informações do tipo *transient*. Este tipo de informação permanece acessível durante o processamento da ação que a gerou, a menos que seja propagada explicitamente.

Informações do tipo *scoped* são representadas pelo mapeamento de identificadores com entidades, que representam um subconjunto de todos os tipos de dados existentes na

notação de ações, sendo este subconjunto denominado de *bindable*. O mapeamento de identificadores em *bindable* é denominado de *binding*.

4.4.3.1 Ações e Combinadores

A notação de ações declarativa estende a notação de ações básica com a introdução de novas ações a combinadores. Abaixo são encontradas algumas das mais utilizadas nesta faceta:

- `bind _ to _  :: yielder, yielder → action`
- `rebind  :: action`
- `_ hence_  :: action, action → action`

A ação `bind T to Y` cria um *binding* do token `T` para o *bindable* produzido pelo yielder `Y`. A ação `rebind` é utilizada apenas para repassar todos os *binds* recebidos na ação corrente para a próxima ação a ser processada. O combinador de ação `_hence_`, quando aplicado da seguinte forma: `A1 hence A2`, possibilita os *bindings* produzidos em `A1` serem utilizados em `A2`.

4.4.3.2 Yielders

Nesta faceta, novos Yielders são adicionados. Abaixo são mostrados dois desses:

- `current bindings  : yielder`
- `the_ bound to_  :: bindable, yielder → yielders`

O primeiro, `current bindings`, produz um *map*, contendo todos os *bindings* recebidos. O yielder `the d bound to T` produz um valor do tipo *d*, após a avaliação do *bindable* associado ao *token* T, no *map* de todos os bindings existentes. Por exemplo, o yielder: `the truth-value bound to b`, produz um valor do tipo *truth-value* (true ou false) que esteja associado ao *token* b.

4.4.4 Imperativa

Esta faceta é especializada no tratamento de informações do tipo *stable*, as quais possuem um tempo de vida maior que as informações do tipo *transiente* e *scoped*. Informações desse tipo representam valores atribuídos a variáveis em programas. As linguagens de programação representam informações desse tipo através de acesso randômico de memória.

Na notação de ações, informações do tipo *stable* são representadas através do mapeamento de *cells* para dados do tipo *storeable*. As *cells* são elementos que representam o endereço ou localização em uma área de memória de acesso randômico.

4.4.4.1 Ações e Combinadores

A notação de ações imperativa enriquece a notação de ações, com novas ações para manipulação das informações do tipo *stable*. Em seqüência, uma dessas ações é apresentada.

- `store _ in _  :: yielder, yielder → action`

A ação imperativa `store Y1 in Y2`, altera o dado armazenado em uma *cell* reservada, produzida por `Y2`, com o valor do dado de tipo *storable* produzido por `Y1`.

4.4.4.2 Yielders

Nesta faceta apenas dois novos yielders são introduzidos.

- `current storage  : yielder.`
- `the _ stored in _`

Onde, o `current storage`, dá uma visão do estado atual de todos os dados armazenados. Enquanto que `yielder the d stored in Y` produz o dado do tipo *d* armazenado na celular produzida por Y.

4.4.5 Reflexiva

A faceta reflexiva é volta para a manipulação de informações do tipo *abstraction*. Uma *abstraction* é utilizada para encapsular uma ação, permitindo sua manipulação como se fosse um dado. Tal tipo de dado é utilizado para representar as *procedures* das linguagens de programação. O tipo *abstraction* é definido da seguinte forma: abstraction ≤ datum, indicando que uma abstraction é um subconjunto dos datum. A forma de se criar uma *abstraction* sobre uma ação é realizada através do seguinte operador:

- `abstraction of _  :: action → abstraction`

Onde, a ação recebida como parâmetro corresponde a ação encapsulada na *abstraction*.

4.4.5.1 Ações e Combinadores

Nessa faceta apenas uma ação é especificada.

- `enact_ :: yielder → action`

Essa ação deve receber como parâmetro um Yielder, que deve produzir uma *abstraction* e ativar a execução da ação encapsulada na *abstraction*.

4.4.5.2 Yielders

- `aplication _ to _  :: yielders, yielders → yielders`
- `clousure _  :: yielder → yelder`

Assumindo que `Y1` produz `abstraction of A`, e `Y2` seja avaliado para um dado `d`, o uso de `aplication Y1 to Y2`, equivalente a `abstraction of (give d then A)`. De forma similar, quando `current bindings` são avaliados para `b`, `clousure Y1` equivale a `abastraction of (produce b hence A)`.

4.4.6 Comunicativa

A faceta comunicativa incrementa a notação de ações básica e reflexiva com o processamento de informações por um sistema distribuído de agentes. As informações manipuladas nesta faceta são do tipo *permanent*.

A comunicação entre os agentes ocorre de forma assíncrona, ou seja, após um agente enviar uma mensagem para outro agente, ele fica livre para seguir com seu processamento, não ficando na espera do momento em que a mensagem é tratada pelo agente receptor.

Mensagens transmitidas a um agente ficam armazenadas em um *buffer* ilimitado assim que ela chega no agente receptor. O fato de um agente receber uma mensagem durante o processamento de uma ação, não interrompe o processamento da mesma. O agente receptor só é informado da chegada da mensagem quando a ação em execução no agente checa o *buffer*.

Cada mensagem carrega a identificação do agente que a envia e do agente que deve recebê-la. Além dessas informações, uma mensagem pode possuir um conteúdo, que pode se um dado qualquer, inclusive uma *abstraction* (dado que encapsula uma ação).

Na notação de ações comunicativa tudo que pode ser comunicado entre agentes, é do tipo *communication.* Pertencente ao tipo *communication* existe dois tipos: *message* e *contract*.

4.4.6.1 Ações e Combinadores

Na notação de ações comunicativa, algumas novas ações são definidas:

- `send_ :: yielder → primitive-action`
- `remove _ :: yielder → primitive-action`
- `offer_ :: yielder → primitive-action`
- `patiently_ :: yielder → primitive-action`

A ação `send Y`, onde `Y` é do tipo `message`, inicia a transmissão de uma mensagem. A forma natural de se representar `Y` é da seguinte maneira: `a message [to Y1] [containing Y2]`, onde `Y1` identifica para que agente a mensagem deve seguir e `Y2` representa o conteúdo da mensagem. A ação `remove Y`, com `Y` produzindo uma `message`, realiza a retirada da mensagem do buffer de entrada do agente.

A ação `offer Y`, onde `Y` é do tipo `contract`, é especificada da seguinte forma: `a contract [to an agent] [containing abstraction of A]`, onde `A` é a ação a ser executada na chegada do `contract` no agente receptor. A ação `patiently A` representa um estado de espera até que `A` falhe, para que a ação termine sua execução.

Com base nas ações primitivas, definidas acima, outras ações, denominadas de híbridas, são definidas na faceta comunicativa, como: `receive Y`, `subordinate Y`, `subordinate-action`. Abaixo, segue o exemplo de como a ação híbrida `receive Y` é definida.

```
receive Y = patiently
            |choose a Y [in set or items of the current buffer] then
            |   remove the given message and give it
```

A ação `receive Y` representa a espera por tempo indeterminado à chegada da mensagem `Y`. O uso da ação `subordinate an agent`, cria um `agent` apto a receber uma `message` contendo uma `abstraction` para executar a ação encapsulada na `abstraction`. Este artifício é exemplificado abaixo, onde um `agent`, identificado por "`agent-exemplo`", é subordinado e, em seguida, é enviada ao mesmo uma `abstraction` encapsulando a ação. Isso faz com que a `action A` seja executada pelo `agent` identificado por "`agent-exemplo`" no momento da chegada da `message`.

```
        give "agent-exemplo" and subordinate an agente and then
        bind the given string#1 to then given agent#2 and then
    hence
        send a message [containing clousure of  abstraction of   A]
                       [to the agent bound to "agent-exemplo"]
```

4.4.6.2 Yielders

Em seguida, segue os yielders existentes na notação comunicativa de semântica de ações, que são utilizados nas ações mostradas anteriormente.

- `current buffer : yelder.`
- `performing-agent : yelder.`

O Yielder `current buffer` produz a lista de mensagens existente no buffer do agente, na ordem de chegada das mensagens.

Durante a execução de uma ação, o yielder `performing-agent` gera a identificação do agente em execução.

4.5 Ferramentas

Com o amadurecimento de estudos e pesquisas na área de Semântica de Ações, muitas ferramentas baseadas em sua notação vem sendo desenvolvidas e evoluídas, dando um maior suporte a este *framework*. Nesta seção serão listadas algumas aplicações baseadas em semântica de ações.

4.5.1 ASD Tools

ASD Tools vem sendo desenvolvida desde 1993 por Van Deursen e Petter Mosses [13]. Eles provêm um ambiente de suporte para uso de semântica de ações, incluindo facilidades para a utilização de *parsers*, editor de sintaxe textual, realização de checagem, e interpretador de descrições em semântica de ações. Tais funcionalidades aumentam a produtividade na escrita e manutenção de longas especificações em Semântica de Ações, além de serem bastante úteis no aprendizado da escrita na notação de ações. As especificações suportadas por ASD Tools devem são escritas ASCII, obedecendo à notação padrão para especificação de semântica de ações. ASD Tools foi implementada usando-se os sistemas ASF e SDF [22].

4.5.2 Cantor

O Cantor consiste em uma ferramenta para compilação de programas escritos em linguagens de programação, projetadas com o *framework* de Semântica de Ações [33]. O componente responsável pela compilação é escrito em Perl [38] e recebe como entradas a especificação da sintaxe e semântica da linguagem, ambas escritas em LATEX, utilizando a notação de ações. O compilador emite código para a linguagem de máquina abstrata RISC, o qual é compilado em código para o processador RISC. O compilador do Cantor possui módulo responsável por realizar checagem sintática, transformar programas em ações, compilar as ações geradas e um pseudo SPARC assembler.

4.5.3 Actress

O sistema Actress [29] é um gerador de compilador baseado em semântica de ações, recebe como entrada a descrição, em notação de ações, da linguagem fonte e, a partir desta descrição, gera um compilador. O compilador gerado traduz programas escritos na linguagem fonte para uma ação. Posteriormente é realizada a checagem da boa formação da ação e, finalmente, é realizada uma tradução desta ação em um programa C.

4.5.4 OASIS

O gerador de compiladores OASIS [32] é capaz de produzir compiladores eficientes e otimizados para linguagens *procedurais* e funcionais com funções de alta ordem e recursividade. O compilador, gerado automaticamente, produz códigos que é comparado em termos de eficiência a códigos gerados por compiladores escritos manualmente. O OASIS é formado basicamente por:

- um *pré-processador* escrito em PERL [38] que recebe as funções semânticas de uma linguagem e produz o gerador de ações para a linguagem especificada;

- um *analizador e otimizador de ações,* que recebe a ação programa e gera um código diretamente para linguagem assembly, realizando análises e otimizações sobre a ação programa recebida, a fim de garantir eficiência no código gerado;

Este sistema é a evidência mais convincente da aplicabilidade prática de geradores de compiladores baseados em semântica de ações.

4.5.5 Abaco (Algebraic Based Action COmpiler)

O sistema Abaco [28], consiste em uma ferramenta direcionada a dar suporte a escrita de especificações de linguagens de programação em semântica de ações. O Abaco combina orientação a objetos e semântica de ações para produzir implementações de linguagens de programação a partir da sua descrição semântica (especificada sobre a notação de ações).

Todas as ferramentas apresentadas demonstram o amadurecimento deste *framework* de Semântica de Ações, e todos os benefícios trazidos pelos trabalhos realizados nesta área de pesquisa se mostram bastante úteis na utilização deste *framework* para a definição de uma linguagem para modelagem de processos de negócios.

Capítulo 5

Action Notation Business Process Language

A Action Notation Business Process Language (ANBPL), foi concebida tomando-se como base o fundamento que um processo de negócio consiste em um conjunto de unidades de execução de tarefa, denominadas de atividades, conectadas entre si e que operam em harmonia para que o objetivo final do processo de negócio seja alcançado.

Da mesma forma que as técnicas para modelagem de processos de negócio se destinam a dar significado aos diferentes tipos de conexões e dependências entre as atividades pertencentes a um processo de negócio, a ANBPL tem como escopo representar, baseando-se em uma notação formal, usando o *framework* de Semântica de Ações, os possíveis tipos de conexões entre atividades. Neste contexto, as atividades são tidas como "caixas pretas", que realizam algum tipo de processamento. Sendo assim, não é parte do escopo da ANBPL a modelagem do significado de cada atividade, apenas ficando restrita à representação das conexões entre elas, que dão o significado global do fluxo de processamento do processo.

A sintaxe da ANBPL é baseada em um conjunto de operadores que expressam diferentes tipos de conexões entre atividades, definindo os possíveis controles de fluxo de processamento que ocorrem em um processo de negócio. Os operadores existentes na linguagem foram criados com inspiração em um subconjunto dos operadores gráficos existentes na BPMN [8], apresentada no Capítulo 3 deste documento. A BPMN foi adotada como base para a definição dos operadores de fluxo da ANBPL, por se tratar de uma notação que conquistou grande espaço no mercado, e, em meio a inúmeras notações para modelagem de processos, ela vem se consolidando como a notação universal para este fim, uma vez que a mesma é adotada por uma gama grande de ferramentas de modelagem de processos de negócio.

O significado semântico desta linguagem foi concebido utilizando o *framework* de Semântica de Ações [26], normalmente utilizado na especificação de linguagens de programação, apresentado no Capítulo 4. Devido à grande quantidade de trabalhos e desenvolvimentos na área de Semântica de Ações, como a construção de editores, interpretadores e geradores de compilador automáticos, baseados em sua notação, tal *framework* se mostrou bastante atrativo para a especificação desta linguagem. Abrindo espaço para a futura utilização da ANBPL em conjunto com interpretadores, que operam sobre a notação de Semântica de Ações, na construção de simuladores de processos de negócio.

A seguir listamos os passos que foram realizados na especificação da ANBPL:

1. **Definição da sintaxe da linguagem**: Neste primeiro momento a sintaxe da linguagem é definida tomando-se como ponto de partida um grupo de operadores de fluxo, tendo os elementos necessários para garantir uma boa representatividade para os processos de negócio que venham a serem escritos nesta linguagem.

2. **Definição de entidades semânticas intermediárias**: Nesta etapa algumas entidades semânticas são definidas com o objetivo de auxiliar o mapeamento da sintaxe da linguagem em elementos semânticos da notação ações.

3. **Definição das Funções Semânticas**: Nesta terceira e última etapa, a especificação da semântica da linguagem, baseada em sua sintaxe, é realizada com a utilização das Funções Semânticas do *framework* de semântica de ações.

Nas próximas três subseções deste Capítulo as etapas destacadas acima, que consistem nos passos que foram realizados na especificação da linguagem, serão apresentadas em maiores detalhes.

5.1 Sintaxe da Linguagem

A sintaxe de uma linguagem tem como objetivo definir as regras para formação das frases que compõem os programas. Cada linguagem possui sua própria regra de formação, que define como as frases são compostas por símbolos e outras frases. É na gramática que essas regras de formação são especificadas. Na gramática da linguagem todos os operadores e elementos que fazem parte de sua sintaxe são definidos.

A gramática da ANBPL, definida na Figura 5.1, é apresentada em sua completude. Vários símbolos não terminais são definidos, como: Connection; Connections; ConditionalActivity; ConditionalActivities; IdActivity; IdsActivities e; Process. Além destes, os símbolos terminais: “seq”; “seq-if”; “fork”; “join” e; “choice”. Eles foram criados para servir como operadores que definem os diferentes tipos de conexões entre atividades pertencentes a um processo negócio.

Gramática:

```
(1)   Connection             ::=  [IdActivity  “seq”  IdActivity]|
                                  [IdActivity “seq-if” ConditionalActivity]|
                                  [IdActivity “fork” IdsActivities]|
                                  [IdsActivities “join” IdActivity]|
                                  [IdActivity “choice” ConditionalActivities]
(2)   IdActivity             ::=  String
(3)   IdProcess              ::=  String
(4)   IdsActivities          ::=  IdActivity +
(5)   ConditionalActivity    ::=  (String , IdActivity)
(6)   ConditionalActivities  ::=  ConditionalActivity +
(7)   String                 ::=  String String | String Dig |
                                  ‘a’ | ... | ‘z’ | ‘A’ | ... | ‘Z’ |
(8)   Dig                    ::=  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(9)   Connections            ::=  Connection+
(10)  Process                ::=  “Process” IdProcess “is” Connections
Closed.
```

Figura 5.1 – Gramática da ANBPL

O símbolo não terminal `Connection` representa uma conexão entre atividades. Para cada tipo de conexão foi definida uma regra de formação na gramática. O símbolo `IdActivity` representa o identificador de uma atividade, podendo assumir qualquer valor, que atenda à regra de formação do elemento não terminal String.

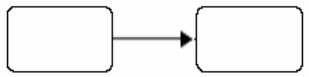
O elemento `ConditionalActivity` tem o papel de representar a execução condicional de uma atividade. A atividade é identificada pelo segundo elemento da *tupla*, que define este tipo de elemento. O primeiro elemento da *tupla* corresponde a um valor que deve ser utilizado em um teste para indicar se a atividade deve ou não ser executada.

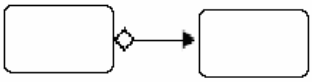
Utilizando a mesma notação para representar uma *tupla* de um ou mais elementos, os símbolos `IdsActivities`, `ConditionalActivities` e `Connections`, foram definidos para representar respectivamente *tuplas* de elementos do tipo `IdActivity`, `ConditionalActivity` e `Connection`.

Na definição de um processo de negócio, que segue a regra de formação do elemento `Process`, deve ser especificado o nome do processo e uma *tupla* de elementos do tipo `Connection`. O nome deve ser especificado de acordo com a regra definida pelo elemento `String`. A *tupla* de `Connection` é representada pelo elemento `Connections`. Pode fazer parte da `Connections`, qualquer que seja o tipo de `Connection` definido na gramática da linguagem.

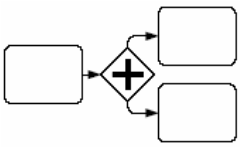
Como os operadores presentes na sintaxe da ANBPL foram definidos com inspiração na BPMN, na Tabela 5.1 será demonstrado o relacionamento dos elementos da notação BPMN com as diversas formações que o elemento do tipo `Connection` pode assumir. Para cada um dos tipos de conexão será dada uma descrição de como ocorreu o mapeamento dos símbolos da BPMN para as construções existentes na linguagem.

Tabela 5.1 – Mapeamento dos elementos pertencentes a notação gráfica de BPMN para elementos da gramática da ANBPL.

<table>
<tr><td colspan="2">Fluxo Não Condicional: É representado pela ocorrência de dois símbolos não terminais <code>IdActivity</code>, intercalados pelo símbolo terminal <code>“seq”</code>. Dando uma idéia de execução em seqüência. Onde a atividade identificada pelo <code>IdActivity</code>, à direita de <code>“seq”</code>, iniciará sua execução apenas quando a atividade identificada pelo <code>IdActivity</code>, à esquerda de <code>“seq”</code>, termine de processar e os dados gerados ao término de sua execução ficarão prontos para serem consumidos pela outra atividade.</td></tr>
<tr><td>  </td><td><code>Connection ::= [IdActivity “seq” IdActivity]</code></td></tr>
<tr><td colspan="2">Fluxo Condicional: É representado pelo símbolo terminal <code>“seq-if”</code> entre dois símbolos não terminais, onde o anterior ao <code>“seq-if”</code> é um <code>IdActivity</code>, representando o identificador da primeira atividade na seqüência de execução, e o posterior ao <code>“seq-if”</code> é um <code>ConditionalActivity</code>, o qual é composto por uma <code>String</code> e um <code>IdActivity</code>, onde este <code>IdActivity</code> define a segunda atividade a ser executada. O operador <code>“seq-if”</code> representa que a execução em seqüência, dependente de uma condição.

Este operador define que após a execução da primeira atividade, a condição para a execução da segunda dependerá do valor gerado pela primeira atividade coincidir com o valor existente na primeira parte da tupla, do <code>ConditionalActivity</code>. Apenas na situação em que os valores sejam os mesmos, a atividade identificada pelo <code>IdActivity</code> pertencente a <code>ConditionalActivity</code>, será executada.</td></tr>
<tr><td>  </td><td><code>Connection ::= [IdActivity “seq-if”
ConditionalActivity]</code></td></tr>
</table>

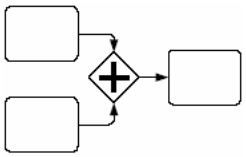
Fork: É definido pelo símbolo terminal "`fork`" ligando um `IdActivity` a uma *tupla* de `IdActivity`, representada pelo símbolo `IdsActivities`. Tal tipo de conexão representa a partição do fluxo de execução de um processo de negócio em vários fluxos de execução paralela. Ao término da execução da primeira atividade, identificada pela `IdActivity` à esquerda do "`fork`", todas as atividades que são identificadas pela *tupla* de `IdActivity`, à direita do "`fork`", iniciarão uma execução de forma simultânea, onde todos os dados produzidos pela atividade à esquerda do "`fork`" tornar-se-ão aptos a serem consumidos por estas atividades.



```
Connection = [IdActivity "fork" IdsActivities]
```

Join: Esse tipo de conexão é identificada pelo símbolo terminal "`join`", ligando uma *tupla* de `IdActivity`, identificado pelo elemento `IdsActivities` a um `IdActivity`. Representando desta forma a sincronização de vários fluxos de execução paralela em um único fluxo de execução.

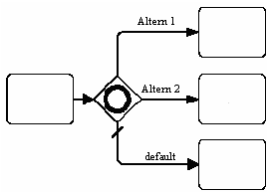
Nesta construção a atividade, identificada pelo `IdActivity` à direita de "`join`" apenas iniciará sua execução após todas as atividades identificadas pelo símbolo `IdsActivities`, à esquerda do "`join`", terminem sua execução.



```
Connection = [IdsActivities "join" IdActivity]
```

Geteway Inclusivo Baseado em Dados: Será representado pela utilização do símbolo não terminal `"choice"`, ligando um identificador de atividade, `IdActivity`, a um elemento do tipo `ConditionalActivities`.

Nesse tipo de conexão o operador `"choice"` representa a realização da escolha de quais atividades irão entrar em execução após o término da execução da atividade identificada pelo `IdActivity`, à esquerda de `"choice"`. Cada `ConditionalActivity` pertencente ao `ConditionalActivities`, terá o valor presente no primeiro elemento da *tupla*, que o representa, comparado com o valor produzido pela atividade, identificada por `IdActivity`. Todas as `ConditionalActivity`, cujos testes não falhem, terão a atividade indicada pelo `IdActivity`, segundo elemento da *tupla* que define o `ConditionalActivity`, executada.




```
Connection = [IdActivity "choice"
              ConditionalActivities]
```

Após apresentar a sintaxe da ANBPL, as próximas seções deste capítulo terão o papel de definir a semântica da linguagem.

5.2 Entidades Semânticas

Na definição da sintaxe da linguagem, vários elementos do tipo `Connection` foram definidos e tiveram seu comportamento descrito textualmente, porém, uma descrição informal não é suficiente para que a linguagem seja passível de execução. Por isso, é necessário definir a semântica da linguagem, utilizando um mecanismo formal para sua especificação. É com esta tarefa que o *framework* de Semântica de Ações será utilizado no projeto da linguagem.

Nesta linguagem as atividades, que fazem parte do processo de negócio, são vistas como unidades de processamento distribuídas que se comunicam através de diferentes tipos de conexões, que ditam o fluxo de execução do processo. Para expressar este

cenário será utilizada a faceta comunicativa de Semântica de Ações, a qual lida com entidades denominadas de `agent`, que se comunicam via a troca de mensagens.

Para expressar a semântica dos diversos tipos de conexão a ANBPL representa cada atividade como sendo uma unidade de execução, que se encontra hospedada em um `agent`. As conexões entre essas atividades são expressas através de troca de mensagens entre `agents`. A Figura 5.2 expressa esse significado. Onde **agent** representa um `agent`, que possui em seu interior uma **executionUnit,** da qual faz parte duas coleções de elementos do tipo `Actions`, uma denominada de **Receivers** e outra de **Senders**. A **executionUnit** significa a unidade de execução que define as condições necessárias para o processamento de uma atividade e o que deve ser realizado após seu processamento. **Receivers** e **Senders** definem respectivamente, a lista de mensagens que precisam ser recebidas, para que a atividade seja executada, e a lista de mensagens que serão enviadas após a sua execução.

Erro! Indicador não definido.

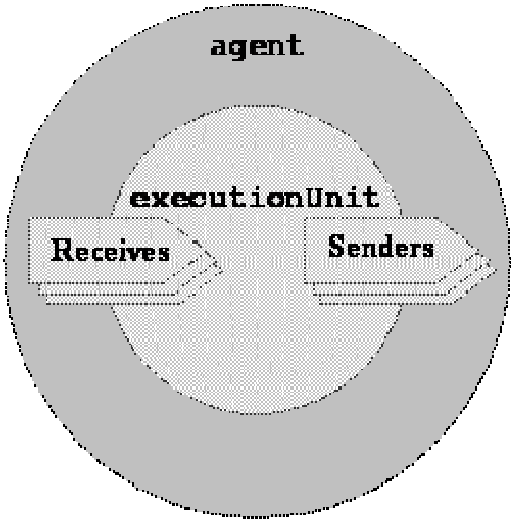


Figura 5.2 - Representação de uma atividade da ANBPL.

Os identificadores de atividades, representados por elementos do tipo `IdActivity`, existentes na sintaxe da ANBPL, encontram-se associados a elementos do tipo `action`, entidades semânticas da notação de ações, que representam unidades de processamento e no contexto da ANBPL definem o comportamento de uma atividade quando processada. A associação entre identificador de atividade (`IdActivity`) e uma `action`,

é representado pela ilustração presente na Figura 5.3. Tal associação é considerada como já provida pelo ambiente de execução do processo, uma vez que não é o objetivo desta linguagem a especificação do significado do processamento das atividades de um processo de negócio.

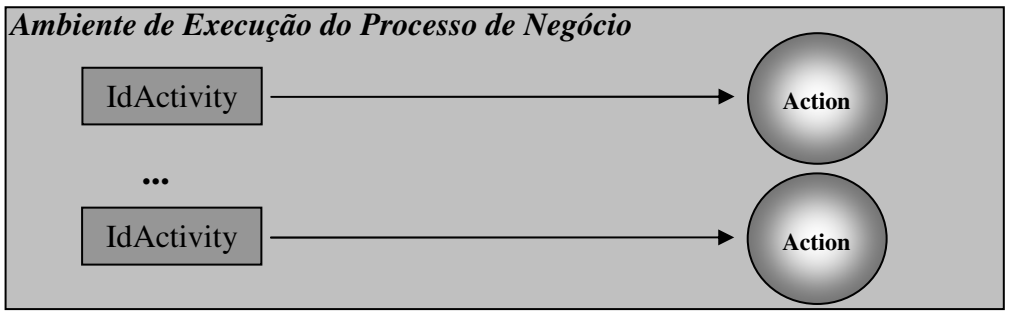


Figura 5.3 – Ambiente de Execução com `IdActivity` e `Action`.

Para expressar a semântica da linguagem, deve-se realizar o mapeamento da sintaxe da linguagem para entidades existentes na faceta comunicativa da notação de ações. Porém, a realização de tal mapeamento de forma direta mostrou-se complexo devido à forma com que os elementos da faceta comunicativa devem ser manipulados. Com o objetivo de facilitar esta tradução, foram criadas algumas entidades semânticas intermediárias, com o objetivo de servir como uma ponte entre a sintaxe da linguagem e a faceta comunicativa da notação de ações. A ilustração apresentada pela Figura 5.4 mostra o fluxo natural do mapeamento da sintaxe da linguagem para a faceta comunicativa da notação de ações.

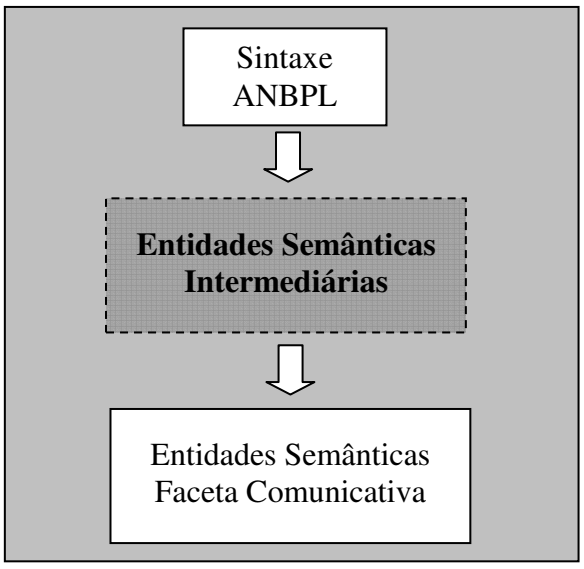


Figura 5.4 – Conversão da Sintaxe nas Entidades Semânticas.

As entidades semânticas intermediárias são definidas de acordo com a notação de dados utilizada por Semântica de Ações e na própria notação de ações. Entidades semânticas como listas, *tuplas* e `action` são as mais utilizadas na especificação das entidades intermediárias. Nas próximas subseções tais entidades serão apresentadas em grupos, juntamente com as operações que agem sobre elas.

5.2.1 Unidades de execução do processo

5.2.1.1 Tipos

- `executionUnit = (list of action, list of action)`
- `empty-executionUnit = (empty-list, empty-list)`
- `token = string.`
- `pollAgent = map [token to executionUnit]`

A entidade `executionUnit` define as pré-condições (mensagens que precisam ser recebidas) para execução de uma `action` e o que será realizado após sua execução (mensagens que serão enviadas). Tal `action` representa o processamento de uma atividade pertencente a um processo de negócio. Cada `action` desse tipo é identificada no ambiente de execução do processo por cada um dos `tokens`, que são chaves na entidade `pollAgent`. Enquanto que no contexto interno do `pollAgent` cada token representa a chave de uma `executionUnit`. Além de servir de identificador para duas entidades distintas, dependendo do contexto no qual são referenciados, os `tokens` quando concatenados no final da string `"agent-"`, serve como identificador de um `agent` no contexto de execução do processo de negócio. Durante a tradução da sintaxe da ANBPL para as entidades semânticas intermediárias, o elemento de sintaxe `IdActivity` deverá ser traduzido para elementos do tipo `token`, presentes na definição do `pollAgent`. Dessa forma, no modelo de memória de um processo de negócio na ANBPL, ilustrado pela Figura 5.5, cada `IdActivity` representa:

- uma associação a uma `action`, que representa o processamento de uma atividade no contexto do processo de negócio;

- uma associação a uma `executionUnit` no contexto do `pollAgent`;

- uma associação entre o identificador gerado pela concatenação da string "`agent-`" ao `IdActivity` com um `agent`.

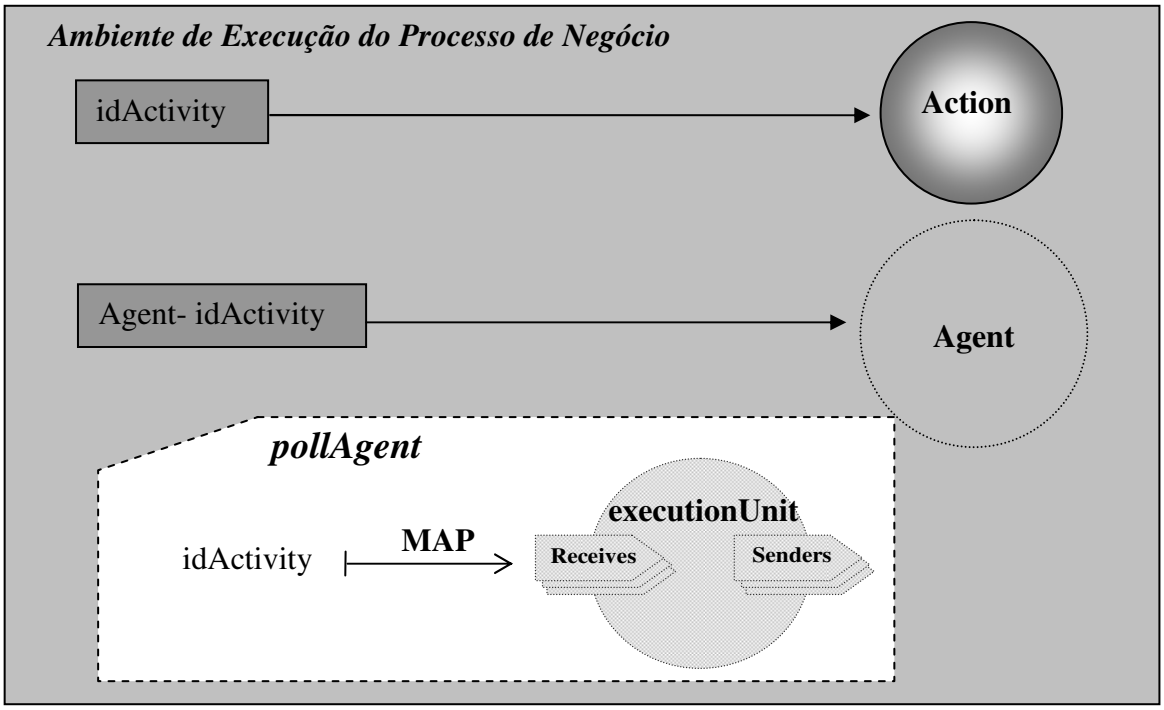


Figura 5.5 – Modelo de associações realizadas por idActivity no ambiente de execução da ANBPL

A `executionUnit`, como pode ser observada em sua definição, é composta por duas listas de entidades do tipo `action`. A primeira lista é formada por `actions`, onde cada uma representa grupos de mensagens que precisam ser recebidas para que o processamento da `action`, que representa uma atividade, ocorra. A segunda lista possui as mensagens que serão enviadas após cada execução dessa `action`. A entidade `empty-executionUnit` representa uma unidade de execução, cujas duas listas se encontram vazias. A entidade `token`, definida pela entidade `string`, corresponde a uma seqüência de caracteres juntamente com dígitos, utilizada na definição da entidade `pollAgent`, para representar chaves de unidades de execução. Cada `token` corresponde a um identificador de atividade, porém, no contexto de execução do processo, ligam-se a `actions`, que representam o comportamento de atividades, e no contexto do `pollAgent` ligam-se a `executionUnit`.

5.2.1.2 Operadores

A seguir, são apresentadas as entidades semânticas que representam operadores, que operam sobre as entidades semânticas definidas na seção anterior. Essas entidades são definidas por diversos operadores e a especificação do seu comportamento é realizada pelas equações apresentadas em seguida.

```
• getAnToken _                :: poolAgent -> token
• _ removeTheTokenMap_        :: pollAgent, token -> pollAgent
• addSenders( _ ,  _ )        :: action, executionUnit -> executionUnit
• addReceivers( _ ,  _ )      :: action, executionUnit -> executionUnit
• transToExecAction(_ , _)    :: token, executionUnit -> action
• tansPartSend(_ , _ )        :: token, executionUnit -> action
• transPartRec(_ , _)         :: executionUnit, action -> action
• separateByAnds _            :: list ot action -> action

(1) getAnToken pa: poolAgent =  first elements mapped-set pa.
(2) pa: pollAgent removeTheTokenMap t: token =
           pa omitting set of t.
(3) addSenders (send, (receivers: list of  action,
                       senders:   list of  action)) =
           (receivers, concatenation (senders, list of (send))).
(4) addReceivers(recs: action, (receivers: list of action,
                                senders:   list of action) ) =
           (concatenation (receivers, list of (recs)), senders).
(5) transToExecAction (t: token , e: executionUnit)=
           transPartRec (e, tansPartSend (t, e)).

(6) tansPartSend (t:token,  ( recs:  list of  action ,
                              sends: list of  action))  =
    the action bound to t and then separateByAnds sends.
(7) transPartRec( (empty-list, sends:list of action),
                   execThenSend: action) =
           commit.
(8) transPartRec((recs: list of  action, sends: list of action),
                 execThenSend: action) =
               head recs and then execThenSend
           and
               transPartRec ((tail recs, sends) , execThenSend).

(9) separateByAnds empty-list = complete.
(10)separateByAnds(acts: list ot action) =
         head acts and separateByAnds(tail acts).
```

A operação `getAnToken_` definida pela equação (1) tem o papel de retornar qualquer `token` da entidade `pollAgent` e, para atingir esse objetivo, primeiramente, o `pollAgent`, que é representado por um `map`, é transformado em um `set` através da aplicação da operação `mapped-set_` para, em seguida, ser executada a operação `elements_` que recebe como parâmetro um `set` e o transforma em uma `tupla`, para então o primeiro elemento da `tupla` ser retornado com a aplicação do operador de *tuplas* `first_`. Os tipos `set`, tupla, as operações `mapped-set_`, `elements_` e `first_`, são exemplos de entidades que fazem parte de notação de dados presente no *framework* de semântica de ações.

A operação binária `_removeTheTokenMap_` toma como argumento à esquerda um `pollAgent`, mapeamento de `token` para `executionUnit`, e à direita um `token`,

realizando a remoção do mapeamento existente no `pollAgent` que tem o `token`, parâmetro a direita, como chave. A equação (2) define como a remoção é realizada, onde, primeiramente, o `token` é transformado em um `set` via a aplicação da operação `set of_`. Após a criação do `set` contendo um elemento, o `token`, esse `set` é passado como parâmetro, à direita, para o operador binário `_omitting_`, juntamente com o `pollAgent`, à esquerda. O operador `_omitting_` realiza a exclusão de mapeamentos existente no map, à esquerda do operador, onde a chave dos mapeamentos que devem ser excluídos aparecem no `set` passado à direita do operador. Como o `set` passado contém apenas um elemento, o `token`, então o operador `_removeTheTokenMap_` garantirá a remoção do `pollAgent`, apenas o mapeamento que tem como chave o `token` passado como parâmetro à direita deste operador.

As operações `addSenders(_,_)` e `addReceivers(_,_)` que tomam como argumento uma `action` e uma `executionUnit`, transformam a `executionUnit` acrescentando uma `action` na lista de `actions` que representam o envio de mensagens e na lista de `actions` que representam o recebimento de mensagens, respectivamente. Como pode ser observado nas equações (3) e (4) a `action` que é passada como parâmetro para qualquer uma das duas operações é primeiramente transformada em uma lista contendo um único elemento para, depois, ser invocada a operação de concatenação de listas `_concat_`.

A operação `transToExecAction_`, definida pela equação (5), corresponde a um dos principais operadores apresentados aqui, uma vez que ele transforma uma `executionUnit` em uma `action`, que define o comportamento da `executionUnit` como uma unidade de execução que para ter sua execução ativada precisa receber determinadas mensagens e depois de completada sua execução mensagens são enviadas a outras `executionUnit`. Para representar esta situação a operação `transToExecAction_` utiliza outras operações auxiliares, sendo essas: `transPartSend(_,_)`, `transPartRec(_,_)` e `separateByAnds_`, definidas pelas equações (6), (7), (8), (9) e (10).

A operação `transPartSend(_,_)`,que opera sobre um `token` e uma `executionUnit`, gera uma `action` que corresponde a `action` referenciada pelo token e em seguida actions separadas pelo operador and, que representam envio de mensagens em paralelo,

por isso a utilização do operador `separateByAnds_` na definição do operador `transPartSend(_,_)`.

O operador `tansPartRec(_,_)` define que a execução de cada `action` da lista de `receivers` é uma precondição independente para que a `action`, segundo parâmetro do operador, seja executada. Cada `action` da lista de `receivers` do pollAgent, define o recebimento de uma ou mais mensagens. Esse operador amarra a execução da `action`, ao acontecimento de eventos de recebimento de mensagens. Para que o recebimento dessas mensagens ocorram outras `executionUnit` tiveram que ser executadas para em seguida enviaram mensagens, garantindo o sincronismo de execução do processo.

5.2.2 Tipos de Conexões

Esta seção descreve as entidades semânticas responsáveis por representar os três tipos de conexões entre unidades de execução, onde cada entidade consiste num tipo particular da `connection`. Também são apresentadas as operações que manipulam um `pollAgent`, atualizando as `executionUnit` contidas nele, a partir do momento em que entidades do tipo `connection` são adicionadas ao `pollAgent`.

5.2.2.1 Tipos

- ```
  connection =  connection(token, token ) |
                connection(token, token) when(string) |
                connection(list of token, token)
  ```

- ```
  connections = conection*
  ```

A entidade `connection` é representada por três diferentes tipos. Porém, antes de entrar nos detalhes sobre cada tipo, é preciso deixar claro que cada `token` existente na definição dos tipos de `connection` representam identificadores para entidades que, dependentemente do contexto em que são referenciados, possuem um significado diferente.

A conexão `connection(token,token)` identifica o tipo de conexão incondicional entre unidades de execução, onde os dois elementos do tipo `token`, presentes neste tipo de conexão, correspondem a identificadores de atividades. Neste contexto, a atividade identificada pelo segundo `token` só será executada após o término de execução da atividade identificada pelo primeiro `token`, incondicionalmente.

A conexão `connection(token,token)when(string)` indica uma conexão condicional, onde a `string`, existente na parte `when(string)` representa o valor a ser comparado com o valor produzido, na primeira posição da tupla de dados, pela atividade identificada pelo primeiro `token` da `connection` e, apenas quando os valores comparados forem iguais, a atividade, identificada pelo segundo `token` desta `connection` será executada. A entidade `connection(list of token, token)` indica que a atividade representada pelo segundo parâmetro será executada apenas quando todas as atividades identificadas pela lista de `tokens` terminarem sua execução. Finalmente, a entidade `connections` aparece representando uma *tupla* de elementos do tipo `connection`.

5.2.2.2 Operações

- `_append_ :: pollAgent , connections -> pollAgent`
- `_connect_ :: pollAgent , connection -> pollAgent`
- `createReceivers_ :: (list of token) -> action`
- `_updateSenders_ :: PollAgent, (list of token, token)`

```
(1) pa: pollAgent append cs: connections =
     if not (cs is ( )) then
        (pa connect first cs) append rest cs
     else  pa.

(2) pa: pollAgent connect connection (s: token, r: token ):connection=
     if (pa at s is nothing) then
         pa [s to
             addSenders(empty-executionUnit,
                        send a message
                            [to the agent bound to “agent-” ^ r])]
     else
         pa [s to
             addSenders(pa at s,
                        send a message
                            [to the agent bound to “agent-”^r])]

     if (pa at r is nothing) then
         pa [r to
             addReceivers(empty-executionUnit,
                          receive a message
                              [from the agent bound to “agent-”^s])]
      else
         pa [r to
             addReceivers(pa at r,
                          receive a message
                              [from the agent bound to “agent-”^s])].
```

```
(3) pa: pollAgent connect
    connection (s:token, r:token)when(c:string): connection =
     if (pa at s is nothing) then
         pa [s to
             addSenders(empty-executionUnit,
                | | check string#1 is c and then
                | | send a message [to the agent bound to “agent-”^r]
                | or
                | | check not string#1 is c and then complete
              )
            ]
     else
        pa [s to
            addSenders(pa at s,
                | | check string#1 is c and then
                | | send a message [to the agent bound to “agent-”^r]
                | or
                | | check not string#1 is c and then complete
             )
           ]
    if(pa at r is nothing) then
        pa [r to
            addReceivers(
               empty-executionUnit,
               receive a message [from the agent bound to “agent-”^s])
           ]
    else
        pa [r to
             addReceivers(
               pa at r,
               receive a message [from the agent bound to “agent-”^s])
            ].

(4) pa: pollAgent connect
        connection(senders: list of token, r: token): connection =
    pa updateSenders (senders,  r)
    if (pa at r is nothing) then
       pa[r to
          addReceivers(empty-executionUnit, createReceivers (senders)]
     else
       pa [r to
          addReceivers(pa at r, createReceivers (senders)].
(5) createReceivers (senders: list of token)  =
    if (senders is empty-lsit)
       then complete
    else
       receive a message [
          from the agent bound to “agent-”^(head senders)]
       and
          createReceiver(tail senders).
```

```
(6) pa: PollAgent updateSenders (senders: list of token,  r: token)
    if (senders is empt-list)
      then pa
    else
      if (pa at head senders is nothing) then
         pa[head senders to
            addSenders(
              empty-executionUnit,
              send a message [to the agent bound to “agent-”^r])]
      else
         pa[head senders to
            addSenders(
              pa at head senders,
              send a message [to the agent bound to “agent-”^r])]
    pa updateSenders (tail senders, r).
```

A primeira operação apresentada, `_append_`, recebe como parâmetros um `pollAgent` e um `connections`, *tupla* de entidades do tipo `connection`. Esta operação tem o papel de atualizar ou incluir, caso não exista, `executionUnit` no `pollAgent` de acordo com os identificadores de `executionUnit` presentes em cada item de `connections`, passado como segundo parâmetro. Como pode ser observada na equação (1), a definição da operação `_append_` é feita de forma recursiva, utilizando a operação `_connect_` para cada um dos itens da `connections`.

A operação `_connect_` tem o papel de atualizar o `pollAgent`, primeiro parâmetro da operação, de acordo com o `connection` passado como segundo parâmetro. Como existem três diferentes tipos de `connection`, sua definição é baseada em três equações: (2), (3) e (4), responsáveis respectivamente, por definir o que deve ser alterado no `pollAgent` quando os tipos de `connection` forem `connection(s:token,r:token)`, `connection(s:token, r:token)when(string)` e `connection(senders:list of token, r:token)`.

Quando o tipo de `connection` utilizado pelo operador `_connect_` for `connection(s:token, r:token)` o `pollAgent` será atualizado de forma que as `executionUnit` identificadas por `s` e `r`, no contexto do `pollAgent`, caso existam, sejam atualizadas e caso contrário, sejam criadas e associadas ao respectivo `token` e incluídas no `pollAgent`. De forma que a `executionUnit`, identificada pelo `token s`, tenha sua lista de `action`, que representa as mensagens a serem enviadas, atualizada com a inclusão de um `action` que define o envio de uma mensagem para o `agent` responsável por executar a `executionUnit` identificada pelo `r`. Enquanto que a `executionUnit`, identificado pelo `token` r, deve ter sua lista de `action`, que representa as mensagens que

precisam ser recebidas, atualizada com a inclusão de uma `action` que representa o recebimento de uma mensagem do `agent` que executa a `executionUnit` identificada pelo `token` s. Como o objetivo de cada `executionUnit`, presente no `pollAgent`, é de representar o comportamento de uma atividade em um processo de negócio, então uma `connection` desse tipo representa a situação em que uma atividade ficará em estado de espera para ser executada até que a outra termine sua execução e a partir do momento que uma conexão deste tipo é atualizada no `pollAgent` via a operação `_connect_`, isto faz com que o processo de negócio seja atualizado com um requisito a respeito da ordem de execução dessas atividades.

A operação `_connect_`, quando aplicada a uma `connection` do tipo `connection(s:token, r:token)when(string)`, definida na equação (3), é bem semelhante à definida pela equação (2), com a diferença peculiar que a `action` a ser incluída na lista de `action`, que representa os envios de mensagens, possua um teste para verificar se deve ou não enviar tal mensagem. Esse teste é realizado pela verificação de igualdade do parâmetro passado na clausula `when()` com o valor existente na primeira posição de uma *tupla* de dados gerada após a execução da `action` identificada por s, no ambiente de execução do processo.

Para o tipo `connection(senders:list of token, r:token)`, o operador `_connect_`, definido pela equação (4), primeiramente atualiza todas as executionUnit, identificadas por `senders`. Nessa atualização, para cada `executionUnit`, é adicionando um `action` na lista de `actions` de envio de mensagens. Essa tarefa é realizada com a utilização da operação auxiliar `_updateSenders(_,_)`, na equação (5). A action adicionada representa o envio de uma mensagem para o `agent` identificado por `"agent-r"`. Em seguida, o operador `_connect_` altera a `executionUnit` identificado pelo `token r`, adicionando em sua lista de `actions` de recebimento de mensagens uma action que representa a necessidade de recebimento de mensagens de cada um dos `agents` que encapsulam as `executionUnit`, definidas em `senders`. Para indicar a necessidade de recebimento de todas as mensagens, as `actions`, que definem recebimento de mensagens, foram combinadas pelo `combinador` de ações `and`. Tal combinação é realizada pela operação `createReceivers(_)`, equação (6).

5.2.3 Manipulação dos Agentes

Este tópico apresenta as operações responsáveis por criar e ativar os `agents`. Em um processo cada `agent` possui uma `executionUnit associada`, onde cada composição deste tipo representa uma atividade de um processo de negócio. A seguir serão apresentadas as operações responsáveis pela manipulação dos `agents`.

- `createAgents __:: pollAgent -> action`
- `initializateAgents __ :: pollAgent -> action`
- `activateAgents __:: pollAgent -> action`

```
(1) createAgents pa: pollAgent =
      |check pa is empty-map and then complete
     or
      |check not pa is empty-map and then
      | | | |give "agent-" ^ getAnToken pa
      | | | and
      | | |  |subordinate an agent
      | | and then
      | |  | bind the given token#1 to the given agent#2 and then
      | |  | createAgents pa removeTheTokenMap the given token#1

(2) initializateAgents pa: pollAgent =
      |check pa is empty-map and then complete
     or
      |check not pa is empty-map and then give getAnToken pa and then
      | | |send a message [containing clousure of abstraction of
      | | |                transToExecAction (it, pa at it)
      | | |                to the agent bound to "agent-"^it]
      | | |
      | | and
      | |  | initializateAgents pa removeTheTokenMap it
      | |  |

(3) activateAgents pa: pollAgent =
       | bind "endProcess" to complete
     and then
      | |createAgents pa
      |hence
      | | initializateAgents pa
```

A operação `createAgents_`, recebe um `pollAgent` como parâmetro e para cada uma das chaves dos mapeamento existentes no `pollAgent`, um `agent` é criado. Depois de criado, cada `agent` é ligado a um identificador, cuja regra de formação consiste na concatencação da string `"agent-"` com uma chave do mapeamento do pollAgent. A concatenação realizada na formação do identificador do `agent` é necessaria devido às chaves pertencentes ao `pollAgent` já serem utilizadas como identificador para `actions` no ambiente de execução do processo. Portanto não podem ser também utilizadas como identificador dos `agents`.

A operação `initializateAgents_` tem o papel de ativar a execução dos `agents`. Para isso, tal operação age sobre cada um dos `agents` existentes, encapsulando em uma `abstraction` a `action` gerada pela aplicação da operação `transToExecAction(_,_)`, e em seguida enviando essa `abstraction` para o `agent`. O fato de um `agent` receber uma mensagem contendo uma abstração faz com que esse `agent` seja ativado e a `action` encapsulada na `abstraction`, recebida no interior da mensagem, seja executada.

Com o papel de juntar as funções de criação e inicialização dos `agents`, é definida a operação `activateAgents_`, a qual, além de realizar as funções mencionadas sobre os `agents` ela cria uma associação do identificador de atividade `"endProcess"` a `action complete`, representando o término da execução de um processo de negócio. Tornando `"endProcess"` um identificador chave que representa o término de um processo.

5.3 Definição da Semântica

Após definida a sintaxe da linguagem, a qual é especificada pela gramática apresentada na Seção 5.1, é necessário dar significado às possíveis construções existentes nesta gramática. Como o ponto de partida para a escrita de um processo de negócio nesta linguagem consiste na utilização da regra de formação especificada pelo símbolo não terminal `Process`, da qual faz parte outro símbolo não terminal, denominado de `Connections`, representado por uma *tupla* de elementos do tipo `Connection`, é necessário dar o significado semântico desses elementos.

As funções semânticas, como são utilizadas na especificação de linguagem utilizando o *framework* de Semântica de Ações, possuem o papel de realizar o mapeamento da sintaxe da linguagem em entidades semânticas da notação de ações, que definam o seu comportamento. Tal mapeamento pode ser realizado diretamente para a notação de ações. Porém, na definição desta linguagem foram criadas outras entidades, apresentadas na Seção 5.2, que devem servir como uma notação intermediária para as entidades semânticas requeridas, facilitando a realização deste mapeamento.

Em seguida, são apresentadas as funções semânticas adotadas para expressar a semântica da linguagem, juntamente com as equações que as definem.

```
• run_:: Process -> action
(1) run [“Process” p “is” cs] =
    activateAgents(empty-map append createAllConnections cs).

• createAllConnections_::Connections -> action
(1) createAllConnections cs: Connections =
      check cs is ( ) and then complete
    or
      check not cs is ( )
      and then (createConnections first cs,
                createAllConnections rest cs).
• createConnections_ :: Connection -> connection*
(1) createConnections [id1: IdActivity “seq” id2: IdActivity] =
       connection (id1, id2).
(2) createConnections [id1: IdActivity “seq-if”
    (cond: string, id2: IdActivity):ConditionalActivity] =
       connection (id1, id2) when (cond).
(3) createConnections [id1:IdActivity “fork” ids:IdsActivities] =
      if (ids is ())
         then ()
      else
         (connection(id1, first ids),
          createConnections(id1 “fork” rest ids)).

(4) createConnections [ids: IdsActivities “join” id2: IdActivity] =
    connection(list of ids, id2)

(5)createConnections[id1:IdActivity “choice”
                     acs: ConditionalActivities] =
    if (acs is ( ))
       then ()
    else
      (createConnections (id1 “seq-if” firs acs),
       createConnections (id1 “choice” rest acs)).
```

A primeira função semântica apresentada, função semântica `run()`, recebe como parâmetro um `Process`, que representa um processo de negócio, e define como é realizado o mapeamento desse elemento da sintaxe da linguagem para a notação de ações, utilizando as entidades semânticas, definidas na Seção 5.2. Na realização deste mapeamento é utiliza a operação `activateAgents()`, a qual mapeia um `pollAgent` em uma `action`. O `pollAgent`, passado para a operação `activateAgents_`, é produzido via a utilização da operação `append_` aplicada a um `empty-map` e uma *tupla* de `Connection`. O `empty-map`, passado como parâmetro, à esquerda do `append_`, representa um `pollAgent` vazio, enquanto o parâmetro à direita do `append_` é gerado através da função semântica `createAllConnections( )` quando aplicada a uma `Connections`.

A função semântica `createAllConnections( )` opera de forma recursiva, chamando a função semântica `createConnections( )` sobre cada um dos elementos da *tupla* de

`Connection` recebida como parâmetro. A função semântica `createConnections( )` é especificada por cinco equações, onde cada equação trata um tipo particular de `Connection`, mapeando a `Connection` em uma *tupla* de uma ou mais entidades semânticas do tipo `connection`.

5.4 Processos de Negócios na ANBPL

Com o objetivo de esclarecer a utilização e funcionamento da ANBPL, nas duas próximas subseções serão mostrados dois processos de negócios. Onde cada processo é primeiramente modelado em BPMN, para, em seguida, ter sua codificação demonstrada utilizando a ANBPL. Posteriormente é apresentado como seria o processo na notação de ações, após o código escrito em ANBPL sofrer todas as transformações, de acordo com a especificação da linguagem.

Como a ANBPL foi idealizada tomando como base a notação de modelagem de processos BPMN, os processos exemplificados são apresentados nesta notação para, em seqüência, serem apresentados em ANBPL.

Na Figura 5.6, pode-se observar as etapas de transformações que ocorrem na tradução do modelo de um processo na BPMN para entidades semânticas. Tal tradução começa com sua representação em diagrama BPMN. Em seguida o processo é traduzido em código escrito em ANBPL, após a transformação T1, para, posteriormente, ser demonstrado o equivalente na notação de ações, após a transformação T2.

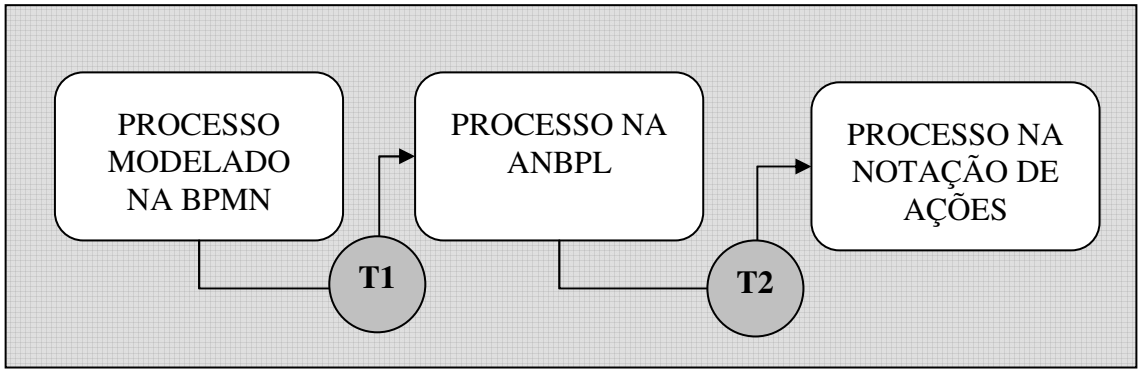


Figura 5.6 – Tradução de um processo da BPMN para a notação de ações.

A seguir são descritos os passos que devem ser realizados em cada uma das etapas de tradução.

- **T1**: Tradução de um processo de negócio modelado na BPMN para código fonte em ANBPL:

    - para cada atividade existente no processo, criar um `IdActivity` que a represente;
    - caso uma atividade apareça conectada a um evento de final de processo, este deve ser considerado como uma atividade e o `IdActivity` utilizado para representá-lo deve se chamar: "`endProcess`";
    - com base na Tabela 5.1, traduzir os componentes gráficos, existentes no processo modelado em BPMN, para elementos do tipo `Connection`, existentes na sintaxe ANBPL;
    - escrever o processo de acordo com a sintaxe do elemento `Process ("Process" String "is" Connections)`, onde no elemento `String` um nome deve ser dado ao processo e após o símbolo terminal "is" deve ser colocado uma *tupla* de `Connection`, contendo todas as `Connection` criadas pelo passo anterior.

- **T2**: Passos para traduzir um processo codificado em ANBPL para a Notação de Ações:

    - aplica-se a função semântica `run_`, apresentada na seção 5.3, sobre o processo descrito em ANBPL, que com o auxílio das demais funções e entidades semânticas definidas, realizará a tradução do processo em ANBPL para a notação de ações.

Nas próximas sub-seçoes serão apresentados dois processos de negócio modelados em BPMN, juntamente com a demonstração das etapas de transformação destes processos desde sua definição em BPMN para o equivalente na notação de ações, passando pela sua representacao em ANBPL. As etapas de transformação correspondem exatamente às etapas **T1** e **T2** descritas acima.

5.4.1 Pintura de Peças

O processo de pintura de peças, já apresentado na Seção 2.2.3, utilizando a técnica de modelagem de processos IDEF3, agora será descrito utilizando BPMN. Como pode ser observado abaixo, este pequeno processo é composto por três atividades: Pintar Peça; Secar Peça e; Testar Pintura.

O processo inicia com um evento inicial, ativando a atividade Pintar Peça, que é seguida pelas atividades Secar Peça e Testar Pintura. Após a atividade Testar Pintura é realizado um teste para averiguar a qualidade da pintura, caso esta seja aprovada, o processo finaliza com a chegada do fluxo de execução do processo em um evento final, caso contrário, o fluxo de execução do processo é redirecionado para a atividade Pintar Peça.

Processo na BPMN

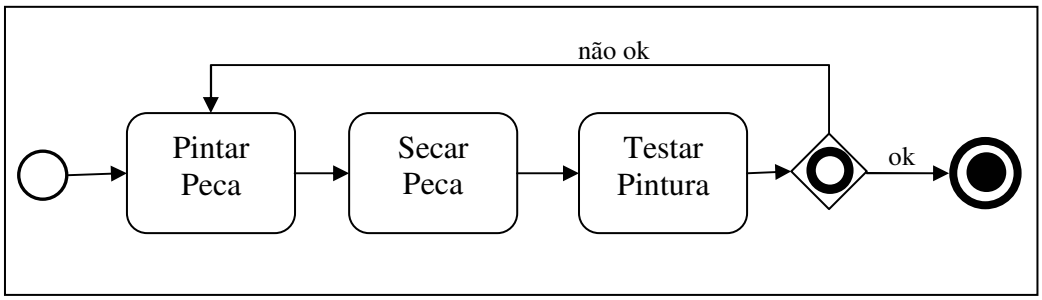
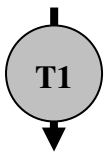
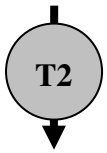


Figura 5.7 – Processo Pintura de Peças Descrito na BPMN.



Processo na ANBPL

```
Process pinturaPecas is ((pintarPeca seq secarPeca),
                         (secarPeca seq testarPintura),
                         (testarPintura choice
                                        (("ok", endProcess),
                                         ("naoOk", pintarPeca))))
```



Processo na Notação de Ações

```
bind “endProcess” to complete
and then
  give “agent-pintarPeca” and subordinate an agent and then
    bind the given token#1 to the given agent#2 and then
  give “agent-secarPeca” and subordinate an agent and then
    bind the given token#1 to the given agent#2 and then
  give “agent-testarPintura” and subordinate an agent and then
    bind the given token#1 to the given agent#2 and then
  give “agent-endProcess” and subordinate an agent and then
    bind the given token#1 to the given agent#2 and then
  complete
 hence
  send a message [containing clousure of abstraction of
      the action bound to “pintarPeca”
    and then
      send a message [to the agent bound to “agent-secarPeca”] ]
   [to the agent bound to “agent-pintarPeca”]

  and
  send a message [containing clousure of abstraction of
     receive a message [from the agent bound to “agent-pintarPeca”]
    and then
     the action bound to “secarPeca”
    and then
     send a message[to the agent bound to “agent-testarPintura”]]
   [to the agent bound to “agent-secarPeca”]
  and
  send a message [containing clousure of  abstraction of
      receive a message [from the agent bound to “agent-secarPeca”]
    and then
      the action bound to “testarPintura”
    and then
      check string#1 is “ok” and then
      send a message to the agent bound to “agent-endProcess”
     or
      check not string#1 is “ok” and then complete
    and
      check string#1 is “naoOk” and then
      send a message to the agent bound to “agent-pintarPeca”
     or
      check not string#1 is “naoOk” and then complete ]
   [to the agent bound to “agent-testarPintura”]
  and
  send a message[containing clousure of  abstraction of
       receive a message
          [from the agent bound to “agent-testarPintura”]
     and then
       the action bound to “endProcess” ]
  [to the agent bound to “agent-endProcess”].
```

De acordo com os passos que ditam como deve acontecer as etapas de tradução **T1** e **T2**, o processo de pintura de peças sofreu todas as transformações necessárias para chegar ao estágio de representação na notação de ações, mostrado acima. Na etapa **T1**

as três atividades e o evento de final de processo, presentes no diagrama que representa o processo em BPMN geraram os identificadores de atividades: `pintaPeca`, `secarPeca`, `testarPintura` e `endProcess`.

Após observar as conexões entre atividades no modelo deste processo em BPMN, Figura 5.7, foi identificada a existência de três conexões entre atividades, sendo representados por três elementos do tipo `Connection`. Onde duas utilizam o operador `seq`, indicando processamento seqüencial das atividades e a outra utiliza o operador `choice`, representando dois caminhos condicionais e excludentes de fluxo de processamento. Na definição da última `Connection` foi considerado que a atividade TestarPintura tem como resultado de sua execução um valor que pode ser "ok" ou "naoOk", dependendo respectivamente da avaliação positiva ou negativa do teste da qualidade da pintura realizada. Dependendo dessa avaliação, deverá ser enviada mensagem para o `agent`, identificado por "agent-pintarPeca" ou o para o `agent`, identificado por "agent-endProcess".

A transformação **T2** ocorre apenas com a aplicação da função semântica `run( )` sobre o código fonte do processo descrito em ANBPL, gerado após a transformação **T1**. De acordo com o resultado observado, para cada `idActivity` existente na definição do processo é definido um identificador de `agent` através da concatenação de `"agent-"` ao `idActivity`.

Em seguida, várias mensagens são criadas e enviadas a cada `agent`. As mensagens possuem em seu conteúdo uma `abstraction`, encapsulando a `action` que define o processamento de uma atividade. Cada `action` deste tipo é composta por três partes:

1. recebimento de mensagens de outros `agents` ;
2. a `action` associada ao identificador de atividade;
3. envio de mensagens para outros `agents;`

A primeira e a terceira parte podem não ocorrer. A primeira é omitida no caso que a atividade é a primeira a ser executada em um processo, portanto não dependendo do recebimento de mensagens de outro `agent,` para que sua execução ocorra. A terceira parte fica de fora no caso da `abstraction` que é enviada ao `agent` identificado por `"agent-endProcess"`, que define o término de um processo, portanto não necessitando enviar mensagem para que qualquer outro `agent` dentro do processo seja ativado.

O processo apresentado na seguinte seção é de uma magnitude maior que o de pintura. Porém todas as etapas de tradução apresentadas no processo de pintura são seguidas no processo de confirmação de viagem.

5.4.2 Confirmação de Viagem

Este processo, descrito em BPMN pela Figura 5.8, é responsável por descrever as atividades inerentes a um processo de confirmação de viagens, realizadas por uma agência de turismo. Seu papel é garantir que todas as solicitações de reservas necessárias a uma viagem sejam realizadas, além de confirmar que tais reservas foram processadas com sucesso, para que a viagem possa ser confirmada.

O processo é composto por doze atividades, começando com a atividade Acessa Informações Viagem, onde as informações pertinentes à viagem, a ser realizada, são levantada. Após o processamento desta atividade, as três atividades seguintes podem ser realizadas em paralelo: Solicita Reserva Carro; Solicita Reserva Hotel e; Solicita Reserva Vôo. Para cada atividade de solicitação existe uma atividade de verificação da reserva, sendo essas: Verifica Reserva Carro; Verifica Reserva Hotel e; Verifica Reserva Vôo. As atividades de verificação das solicitações das reservas podem seguir dois fluxos possíveis. Caso a verificação de reserva seja confirmada, uma atividade de confirmação daquela reserva é processada, caso contrário, a solicitação da reserva é realizada novamente.

Apenas após a confirmação de todas as reservas (carro, hotel e vôo), a viagem será confirmada, serviço que é realizado pela atividade Confirma Viagem, finalizando o processo em seqüência.

Processo na BPMN

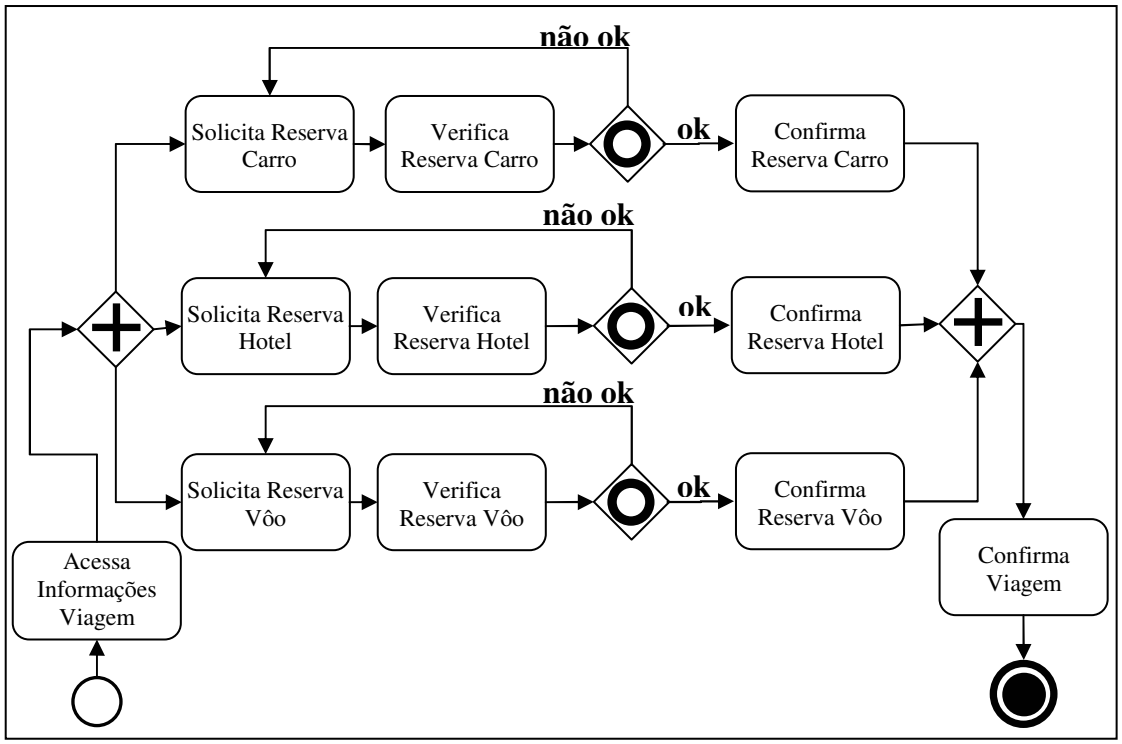
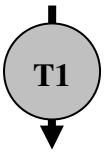
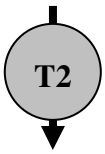


Figura 5.8 – Processo de Confirmação de Viagem na BPMN.



Processo na ANBPL

```
Process confirmacaoViagens is
    ((acessaInfViag  fork (solResCarro, solResHotel, solResVoo)),
     (solResCarro seq verifResCarro),
     (verifResCarro choice ((“ok”, confResCarro),
                            (“naoOk”, solResCarro)),
     (solResHotel seq verifResHotel),
     (verifResHotel choice ((“ok”, confResHotel),
                            (“naoOk”, solResHotel)),
     (solResVoo seq verifResVoo),
     (verifResVoo choice ((“ok”, confResVoo),
                          (“naoOk”, solResVoo)),
     ((confResCarro, confResHotel, confResVoo) join confViagem),
     (confViagem seq endProcess) )
```



Processo na Notação de Ações

```
bind “endProcess” to complete
and then
  give “agent- acessaInfViag” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-solResCarro” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-verifResCarro” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-confResCarro” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-solResHotel” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-verifResHotel” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-confResHotel” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-solResVoo” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-verifResVoo” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-confResVoo” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-confViagem” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
  give “agent-endProcess” and subordinate an agente and then
    bind the given token#1 to then given agent#2 and then
hence
  send a message [containing clousure of  abstraction of
    the action boundo to “acessaInfViag”
    and then
      send a message [to the agent bound to “agent-solResCarro”]
    and
      send a message [to the agent bound to “agent-solResHotel”]
    and
      send a message [to the agent bound to “agent-solResVoo”] ]
  [to the agent bound to “agent-acessaInfViag”]
and
  send a message[containing clousure of  abstraction of
    receive a message
      [from the agent bound to “agent-acessaInfViag”]
    and then
      the action boundo to “solResCarro”
    and then
      send a message
        [to the agent bound to “agent-verifResCarro”]
    and
    receive a message
      [from the agent bound to “agent-verifResCarro”]
    and then
      the action boundo to “solResCarro”
    and then
       send a message
         [to the agent bound to “agent-verifResCarro”] ]
  [to the agent bound to “agent-solResCarro”]
and
```

```
send a message[containing clousure of  abstraction of
  receive a message
    [from the agent bound to “agent-acessaInfViag”]
  and then
    the action boundo to “solResHotel”
  and then
    send a message [to the agent bound to “agent-verifResHotel”]
 and
  receive a message
    [from the agent bound to “agent-verifResHotel”]
  and then
    the action boundo to “solResHotel”
  and then
    send a message [to the agent bound to “agent-verifResHotel”]
[to the agent bound to “agent-solResHotel”]
and
send a message[containing clousure of  abstraction of
  receive a message
    [from the agent bound to “agent-acessaInfViag”]
  and then
    the action boundo to “solResVoo”
  and then
    send a message [to the agent bound to “agent-verifResVoo”]
 and
  receive a message
    [from the agent bound to “agent-verifResVoo”]
  and then
    the action boundo to “solResVoo”
  and then
    send a message [to the agent bound to “agent-verifResVoo”]  ]
[to the agent bound to “agent-solResVoo”]
and
send a message[containing clousure of  abstraction of
  receive a message
     [from the agent bound to “agent-solResCarro”]
  and then
     the action boundo to “verifResCarro”
  and then
    check string#1 is “ok” and then send a message
      [to the agent bound to “agent-confResCarro”]
   or
    check not string#1 is “ok” and then complete
  and
    check string#1 is “naoOk” and then send a message
      [to the agent bound to “agent-solResCarro”]
   or
    check not string#1 is “naoOk” and then complete  ]
[to the agent bound to “agent-verifResCarro”]
and
```

```
send a message[containing clousure of  abstraction of
   receive a message [from the agent bound to “agent-solResHotel”]
   and then
      the action boundo to “verifResHotel”
   and then
      check string#1 is “ok” and then send a message
        [to the agent bound to “agent-confResHotel”]
      or
      check not string#1 is “ok” and then complete
     and
      check string#1 is “naoOk” and then send a message
        [to the agent bound to “agent-solResHotel”]
      or
      check not string#1 is “naoOk” and then complete  ]
 [to the agent bound to “agent-verifResHotel”]
and
 send a message[containing clousure of  abstraction of
   receive a message [from the agent bound to “agent-solResVoo”]
   and then
      the action boundo to “verifResVoo”
   and then
      check string#1 is “ok” and then send a message
        [to the agent bound to “agent-confResVoo”]
      or
      check not string#1 is “ok” and then complete
     and
      check string#1 is “naoOk” and then send a message
        [to the agent bound to “agent-solResVoo”]
      or
      check not string#1 is “naoOk” and then complete  ]
 [to the agent bound to “agent-verifResVoo”]
and
 send a message[containing clousure of  abstraction of
   receive a message[from the agent bound to “agent-verifResCarro”]
   and then
      the action boundo to “confResCarro”
   and then
      send a message
          [to the agent bound to “agent-confResViagem”] ]
 [to the agent bound to “agent-confResCarro”]
and
 send a message[containing clousure of  abstraction of
   receive a message[from the agent bound to “agent-verifResHotel”]
   and then
      the action boundo to “confResHotel”
   and then
      send a message
          [to the agent bound to “agent-confResViagem”] ]
 [to the agent bound to “agent-confResHotel”]
and
 send a message[containing clousure of  abstraction of
   receive a message[from the agent bound to “agent-verifResVoo”]
   and then
      the action boundo to “confResVoo”
   and then
      send a message
          [to the agent bound to “agent-confResViagem”] ]
 [to the agent bound to “agent-confResVoo”]

 and
```

```
send a message[containing clousure of  abstraction of
    receive a message
      [from the agent bound to “agent-confResCarro”]
    and
      receive a message
        [from the agent bound to “agent-confResHotel”]
    and
      receive a message
        [from the agent bound to “agent-confResVoo”]
    and then
      the action boundo to “confResViagem”
    and then
      send a message
          [to the agent bound to “agent-endProcess”] ]
  [to the agent bound to “agent-confResViagem”]
and
  send a message[containing clousure of  abstraction of
    receive a message
      [from the agent bound to “agent-confViagem”]
    and then
      the action boundo to “endProcess”  ]
  [to the agent bound to “agent-endProcess”]
```

Como resultado da tradução **T1**, o processo de negócio descrito em BPMN é convertido de acordo com a sintaxe de ANBPL. Nessa nova representação o processo é escrito com um total de nove elementos do tipo Connection. Cada conexão entre atividades é representada por um elemento do tipo Connection. A *tupla* composta por todos esses elementos Connection é utilizada para representar o processo de negócio em ANBPL.

A Figura 5.9, corresponde à Figura 5.8 transformada com o destaque dos pontos de conexão entre atividades. Na fase de tradução **T1,** para cada ponto de conexão destacado, um elemento do tipo `Connection` é definido. O processo de negócio, de acordo com a notação da ANBPL, é definido pela tupla de todas as `Connection` criadas. Em seguida será mostrada a equivalência entre cada um dos pontos de conexão de atividades e o elemento `Connection` criado para o representar.

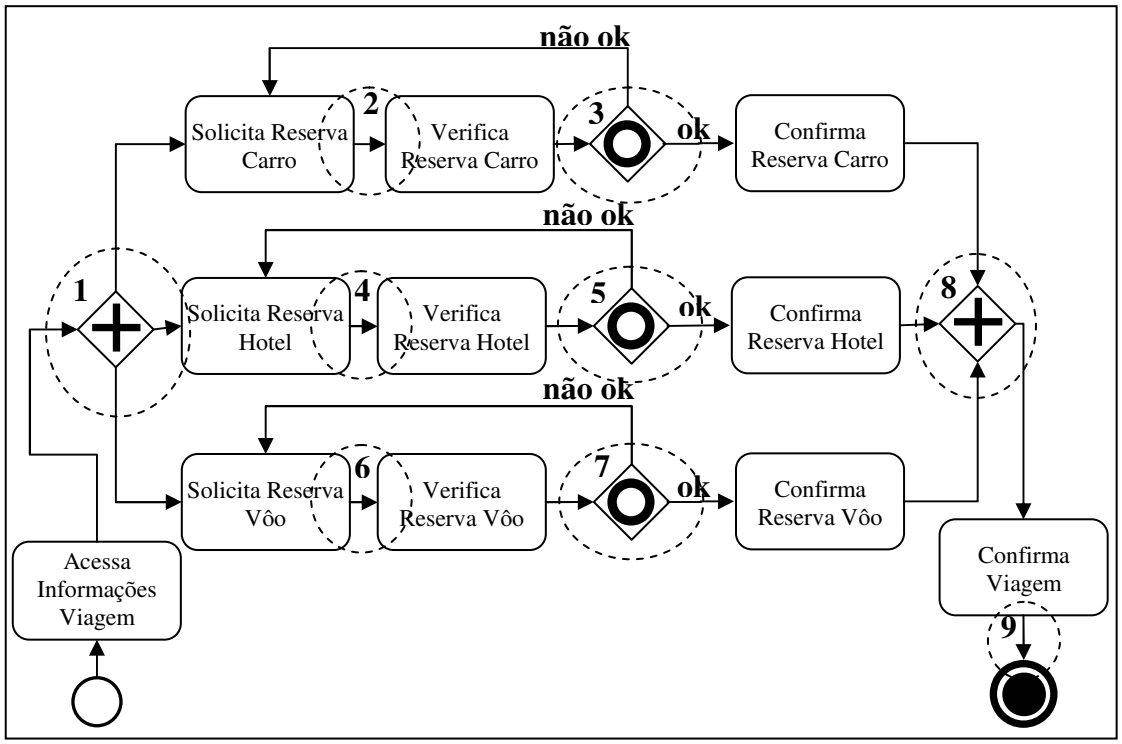


Figura 5.9 – Processo de Confirmação de Viagem na BPMN. Com Destaque Nos Pontos de Conexão Entre as Atividades.

A Conexão 1 interliga uma atividade a outras três, garantindo o processamento independente e paralelo das atividades onde as setas apontam. Para representar essa situação foi criada a `Connection: (acessaInfViag fork (solResCarro, solResHotel, solResVoo)`. As conexões 2, 4 e 6, são as mais simples, garantindo apenas o processamento seqüencial das atividades envolvidas. A conexão 2, por exemplo, é representada por `(solResCarro seq verifResCarro)`.

As conexões 3, 5 e 7, definem um fluxo condicional, podendo direcionar para a atividade de confirmação da solicitação correspondente ou para uma nova tentativa de solicitação, dependendo se o valor gerado pela atividade de verificação for "ok" ou "nãoOk", respectivamente. Para expressar esse comportamento é utilizado a `Connection`, que utiliza o operador `choice`, como pode ser observado, por exemplo, em: `(verifResCarro choice (("ok", confResCarro), ("naoOk", solResCarro))`.

Para que a atividade "confirma viajem" seja realizada é necessário que as atividades de confirmação de reserva de hotel, de carro e de vôo sejam finalizadas. Para expressar esse comportamento, a seguinte `Connection` foi adicionada a descrição do processo

((confResCarro, confResHotel, confResVoo) join confViagem). Para indicar que a atividade identificada por confViagem é a última do processo, foi definido um conexão de seqüência, ligando a atividade confViagem a atividade identificada por endPeocess, representando o término do processo de negócio .

Após a transformação **T1**, tem-se o processo descrito em ANBPL. Aplicando a transformação **T2**, sobre essa representação**,** o processo é traduzido para a notação de ações. Na notação de ações é definido um agent para cada atividade do processo, e as regras que ditam o fluxo do processo são representas por actions, que definem mensagens que precisam ser recebidas para o processamento de uma atividade e mensagens que são enviadas após seu processamento.

Na notação de ações, o processamento de cada atividade é realizado por um agent dedicado a esta atividade. O que deve ser executado por cada agent fica em uma action, encapsulada em uma abstraction, que apenas será executada após o envio dessa abstraction para o agent em questão. A action, encapsulada na abstraction, é construída conforme as Connection definidas na descrição do processo na ANBPL.

Capítulo 6

Conclusão e Trabalhos Futuros

Simplicidade e praticidade foram os principais conceitos em mente para o projeto da ANBPL. Considerar um processo de negócio como uma coleção de conexões entre atividades foi um fator primordial para que a representação de processos escrito nessa linguagem fosse o mais simples possível.

A escrita de um processo de negócio na ANBPL, a partir do modelo de um processo especificado em BPMN, torna-se fácil devido à linguagem manter um relacionamento claro entre tipos de conexões de atividades e os operadores da linguagem. Tal relacionamento é demonstrado na descrição da linguagem, que acompanhado de diretrizes que ditam como especificar processos em ANBPL a partir de um modelo de processo em BPMN, torna a tarefa da escrita de um processo na linguagem uma tarefa mecânica.

6.1 Trabalhos Relacionados

Essa linguagem foi pensada de forma completamente diferente das demais linguagens ou projetos de linguagens para execução de processos de negócio, visto que, as definidas até então, como: XPDL [42], BPML (*Business Process Languagem*) [2] e BPEL (*Business Process Execution Language*) [7][4], têm como foco a utilização de XML para representação dos processos de negócio, diferentemente da ANBPL, que tem uma representação completamente textual e baseado em uma sintaxe, de forma semelhante a uma linguagem de programação.

No projeto desta linguagem se buscou a todo o momento a definição de uma sintaxe, que possibilitasse a descrição de processos de forma textual, procurando reduzir o esforço ao se escrever um processo de negócio. O uso de Semântica de Ações na especificação de uma linguagem com este propósito contribuiu bastante para a consolidação do caráter inovador empregado neste trabalho. Visto que, na literatura são

encontrados trabalhos referentes à utilização de Semântica de Ações para a especificação de linguagens de programação.

6.2 Contribuições

Com uma maneira para representar processos de negócio, bem diferenciada das linguagens para execução de processos, propostas até então, a ANBPL surge como uma alternativa diferenciada para, no futuro, ser utilizada como ferramenta para realizar a execução de processos de negócio.

O uso do *framework* de Semântica de Ações [27] na especificação da semântica da linguagem foi de extrema importância, devido à rica representatividade do processamento distribuído de tarefas, oferecido pela Faceta Comunicativa deste *framework*. A equivalência entre as atividades de um processo de negócio e os agentes da Faceta Comunicativa, do ponto de vista de processamento de tarefas, que interagem, foi um ponto de inspiração de extrema importância na definição da ANBPL.

O projeto da ANBPL pode ser utilizado como fonte de estudo para interessados no aprendizado no *framework* de Semântica de Ações. Principalmente no que se refere a forte utilização da faceta comunicativa da notação de ações. Uma vez que a utilização prática de tal faceta não é tão detalhada na literatura que trata o *framework* de Semântica de Ações.

6.3 Trabalhos Futuros

Na definição da ANBPL, os principais elementos da BPMN, que ditam o fluxo do processo de negócio, foram levados em consideração para a definição da sintaxe da linguagem. Porém, devido à dimensão e a completude da BPMN, muitos elementos pertencentes à notação de BPMN não foram contemplados nesta primeira versão da linguagem, ficando a sugestão para trabalhos futuros à evolução da ANBPL com a extensão da sintaxe para contemplar outros elementos gráficos existentes na BPMN, tornando a linguagem cada vez mais rica e completa.

Além da extensão da sintaxe da ANBPL existe a possibilidade de definir a forma de representar o processamento particular das atividades de um processo, visto que, nesta primeira versão apenas as associações entre as atividades e o controle do fluxo de

execução do processo é tratado. Definindo como ocorrerá à especificação do processamento das atividades de um processo será possível à utilização da linguagem em conjunto com ferramentas de modelagem de processos para a execução de processos. Essa realização contribui para que ferramentas de modelagem possam ter na ANBPL um mecanismo útil para a execução mecanizada de processos de negócio.

Dado que os operadores presentes na ANBPL, que representam os tipos de conexões entre atividades, foram definidos a partir dos elementos do diagrama BPMN, o mapeamento automático de modelos de processos de negócio, definidos na BPMN, para descrição do processo, na ANBPL, torna-se uma tarefa completamente viável a ser realizada. Para que tal mapeamento seja realmente de utilidade, dando grandes contribuições a área de modelagem e simulação de processos, seria desejável a criação ou evolução de uma ferramenta visual para modelagem de processos, baseada na BPMN. A partir desta ferramenta seria possível criar o modelo de um processo utilizando BPMN e a transformação do modelo gráfico para a descrição do processo em ANBPL ocorreria de forma automática quando se desejasse executar o processo. Como à definição da semântica da ANBPL é realizada sobre o *framework* de Semântica de Ações, tal ferramenta poderia utilizar interpretadores e compiladores, já existentes e baseados na notação de ações, para a execução de processos de negócio, como por exemplo o Abaco [28].

A descrição semântica da linguagem foi realizada de acordo com a notação utilizada pelo *framework* de Semântica de Ações, denominada de notação de ações. Porém, não foi utilizada nenhuma ferramenta para validação de tal descrição. Ficando como sugestão para trabalhos futuros a utilização de uma ferramenta, como o Abaco, capaz de interpretar a notação de ações e validar a corretude da utilização dessa notação na descrição da ANBPL.

REFERÊNCIAS

[1] AITKEN S., KINGSTON J., Representing Process Models in Cyc: An Application to Crisis Management. Edinburgh. University of Edinburgh. 1998.

[2] ANDREWS T., et all. Business Process Execution Language for Web Services. version 1.1, 136 pp. 2003.

[3] APRIL J., et al. Enhancing Business Process Management With Simulation Optimization. In: BPTRENDS, 2005.

[4] ARKIN, A. Business Process Modeling Language. 2002. Disponível em http://xml.coverpages.org/ni2002-06-26-b.html. Acesso em: 1 Março 2004.

[5] ARQUIN A., INTALIO, BPMI.org, Business Process Modeling Language, Disponível em: http://www.bpmi.org/bpml-spec.htm. Acesso em: 15 nov. 2005.

[6] BORN, G. The Way to Design, Document and Re-engineer Business Systems. In: Process Management to Quality Improvement. 1994.

[7] BPMI.org, Business Process Manager Iniciative. Disponível em: http://www.bpmi.org. Acesso em: 12 nov. 2005.

[8] BPMI.org, Business Process Modeling Notation (BPMN). v. 1.0, 2004. Disponível em: http://www.bpmi.org. Acesso em: 12 nov. 2005.

[9] Business Modeling with the UML and Rational Suite Analyst Studio. Rational Software White Paper. 2002.

[10] CESARE, S.; LYCETT, M.; PATEL, D. Business Modeling with UML: distilling directions for future research, Enterprise information systems IV, Kluwer Academic Publishers, Hingham, MA, 2003.

[11] DATTA, A. Automating the Discovery of AS-IS Business Process Models: Probabilistic and Algorithmic Approaches, Information Systems Research, vol. 9, no. 3, pp 275-301. 1998.

[12] DAVENPORT, T. H. Process Innovation: Reengineering Work Through Information Technology, Harvard Business School Press, 1993.

[13] DEURSEN, A.V.; MOSSES, P. D. ASD: The action semantic description tools. In: Conf. on Algebraic Methodology and Software Technology, 1996, Munich, v. 1101, p. 579-582.

[14] FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. Integration Definition For Function Modeling (IDEF0), n. 183, 1993. Disponível em http://www.idef.com/Downloads/pdf/idef0.pdf. Acesso em: 15 nov. 05.

[15] HOMMES, B. J.; REIJSWOUD, V. V. Assessing the Quality of Business Process Modeling Techniques. In: 33rd Annual Hawaii International Conference, 2002. Hawaii, v.1, p. 1007.

[16] IBM RATIONAL SOFTWARE. Disponível em: http://www-306.ibm.com/ software/rational. Acesso em 14 fev. 2004.

[17] Integrated Definition Methods. IDEF3 Process Flow and Object State Description. Disponível em: http://www.idef.com/idef3.html. Acesso em: 14 fev. 2004.

[18] JACKOWSKI, Z. Business Modeling with UML: A Business Process Centred Architecture. Disponível em: http://www.agilealliance.com/articles/ jackowskizygmuntbusin/file. Acesso em: 15 nov. 2005.

[19] JANG, M.; KIM, Y. Process Modeling for Customer Service Process Redesign: A Conceptual Framework'. In: Integrating Technology & Human Decisions: Global Bridges into the 21st Century, v. 2, p. 1582-1584. 1999.

[20] KIM, H.; KIM, Y. Dynamic Process Modeling for BPR: A Computerised Simulation Approach. In: Information & Management, v. 32, no. 1, p. 1-13. 1997.

[21] KIM, Y. G. Process Modeling for BPR: Event - Process Chain Approach. Proceedings of the 16th International Conference on Information Systems, 1995, Amsterdam.

[22] KLINT, P. A. Meta-environment for Generating Programming Environments. In: ACM Transactions on Software Engineering Methodology, p. 176-201, 1993.

[23] KOBRYN, C.; WEIGERT T. OMG UML 2.0 RFPs. 2000, Disponível em: ftp://ftp.omg.org/pub/docs/ad/00-09-05.pdf. Acesso em: 14 fev. 2005.

[24] KOCK, N., MCQUEEN, R., Knowledge and Information Communication in Organisations: An Analysis of Core, Support and Improvement Processes, Knowledge and Process Management, vol. 5, no. 1, p 29-40. 1998.

[25] MAYER R. J., WITTE P. S. Delivering Results: Evolving BPR From Art To Engineerin. Disponível em http://www.idef.com/Downloads/pdf/bpr.pdf. Acesso em 14 fev. 2004.

[26] MOSSES, P. D. A Tutorial on Action Semantics. In: Tutorial notes for Formal Methods Europe, Barcelona, 1994.

[27] MOSSES, P. D. Action Semantics. Cambridge University Press. 1992. 329 pp.

[28] MOURA H. P, MENEZES L. C. S. The ABACO system - an Algebraic Based Action COmpiler. In: Algebraic Methodology and SoftwareTechnology, v. 1548, p. 527–529, 1999.

[29] MOURA, H. P. Action Notation Transformations. Glasgow, 1993. Tese de Doutorado, Dept. of Computing Science, Univ. of Glasgow.

[30] NICKOLS, F. The Difficult Process of Identifying Processes, Why It Isn't as Easy as Everyone Makes It Sound, 1999. Disponível em: http://home.att.net/~nickols/diffcult.htm. Acesso em 14 fev. 2004.

[31] OMG. OMG Unified Modeling Language Specification. 2000 UML. Disponível em: http://www.rational.com/media/uml/post.pdf. Acesso em 14 fev. 2005.

[32] ØRBAEK P. OASIS: An optimizing action-based compiler generator. In: CC'94, Proc. 5th Intl. Conf. on Compiler Construction, Edinburgh, v. 786, pp. 1-15. Springer-Verlag, 1994.

[33] PALSBERG J. An automatically generated and provably correct compiler for a subset of Ada. In: Fourth IEEE International Conference on Computer Language, San Francisco, California, 1992.

[34] PEPPARD, J., ROWLAND, P. The Essence of Business Process Reengineering, Prentice Hall, USA, p. 169-175. 1995.

[35] RAINER, F. Aligning Strategies Processes and IT: Case Study. IEEE Engineering Management Review, v. 28, n. 3, p. 81-91, 2000.

[36] RICHARD J. M., et. al. Information Integration For Concurrent Engineering (Iice) Idef3 Process Description Capture Method. Report Wright-Patterson Air Force Base, Ohio 45433-7604. 1995.

[37] UML Business Modelling Primer. Disponível em: http://gmckinney.info/resources/ UMLBusinessModelling100.pdf. Acesso em: 14 fev. 2004.

[38] WALL, L.; SHWARTZ R. L. Programming Perl. O'Reily & Associates. 2º ed., 1996.

[39] WATT D. A. Executable semantics descriptions. In: Software Practice and Experience, 16(1):13 43, 1986.

[40] WATT, D. A. Programming Language Syntax and Semantics. Prentice Hall International (UK) Ltd, 1991.

[41] WATT, D. A., MOSSES P. D. Action semantics in action. Não Publicado 1987.

[42] WfMC, The Workflow Manager Colition. Disponível em: http://www.wfmc.org/. Acesso em: 15 nov. 2005.

[43] YU, E.S.K., et. al. New Concepts and Tools for Re-Engineering, IEEE Expert, p. 16-23, 1996.