UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

José Rodrigo Ferreira Neri

APLICAÇÃO DE WORKFLOW NA ORGANIZAÇÃO DE SISTEMAS MULTIAGENTE

Florianópolis (SC)

2013

José Rodrigo Ferreira Neri

APLICAÇÃO DE WORKFLOW NA ORGANIZAÇÃO DE SISTEMAS MULTIAGENTE

Dissertação de Mestrado submetida ao Programa de Pós Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Dr. Jomi Fred Hübner
Coorientador: Prof. Dr. Carlos Henrique Farias dos Santos

Florianópolis (SC)

2013

José Rodrigo Ferreira Neri

APLICAÇÃO DE WORKFLOW NA ORGANIZAÇÃO DE SISTEMAS MULTIAGENTE

Esta Dissertação de Mestrado foi julgada aprovada para a obtenção do Título de "Mestre em Engenharia de Automação e Sistemas", e aprovada em sua forma final pelo Programa de Pós Graduação em Engenharia de Automação e Sistemas.

Florianópolis (SC), 20 de dezembro 2013.

Prof. Dr. Jomi Fred Hübner
Coordenador

Prof. Dr. Jomi Fred Hübner
Orientador

Prof. Dr. Carlos Henrique Farias dos Santos
Coorientador

Banca Examinadora:

Prof. Dr. Jomi Fred Hübner
Presidente

Prof. Dr. Gustavo A. Giménez Lugo

Profa. Dra. Jerusa Marchi

Prof. Dr. Marcelo Stemmer

À minha mãe, Lurdes (in memoriam), que sempre acreditou nos estudos como forma de aprendizagem para a alma.

AGRADECIMENTOS

Este trabalho não teria sido concluído sem a ajuda de muitas pessoas. Em primeiro lugar a meu Pai Neilton, ao meu irmão Pablio e a minha irmã Pamela pelo apoio e incentivo em todos os sentidos e que, mesmo com a distância, sempre estiveram ao meu lado oferecendo todas as condições possíveis para que eu pudesse alcançar meus objetivos.

Aos meus amigos de pesquisa e colegas da UFSC, tanto do DAS (Departamento de Automação e Sistemas) quanto de outros laboratórios, que compartilharam comigo estes anos de estudo e trabalho. Pelas experiências e amizades com os quais tive oportunidade de aprender muito. Em especial ao Caio e Dani que sempre me apoiaram e sempre tentaram me animar, e ao Maicon por vários anos de estudos e entretenimentos juntos.

Ao orientador Jomi Hübner pela sua orientação, por sua experiência e dedicação na área e pela amizade construida durante todo o curso.

A CAPES e PróEngenharia pela bolsa de pesquisa fornecida ao longo desses anos.

A todas as pessoas que me ajudaram direta e indiretamente para a realização deste trabalho.

"Conhecimento não é aquilo que você sabe, mas o que você faz com aquilo que você sabe".

Aldous Huxley

RESUMO

Esta dissertação apresenta uma proposta para integração de um sistema de workflow a um modelo organizacional de sistemas multiagentes. Essa integração soluciona algumas limitações que os modelos organizacionais costumam apresentar e também adiciona propriedades importantes dos sistemas de workflow. São proposta três alternativas de integração (modelo híbrido, modelo workflow com missões e modelo workflow) e realizado uma comparação entre elas, sendo que a melhor alternativa é utilizada para a integração do sistema de workflow denominado *Bonita* ao modelo organizacional *Moise*. Além do modelo, uma linguagem de programação normativa foi estendida com a finalidade de suportar a agregação do workflow. Uma arquitetura de implementação para viabilizar o modelo proposto é apresentada. Essa arquitetura é baseada na arquitetura original do *Moise* que fornece operações e propriedades observáveis com o intuito de permitir que os agentes interajam com uma organização. A fim de avaliar a proposta, é apresentado um exemplo completo da utilização do modelo e um comparativo entre os padrões de fluxo de workflow. O comparativo mostra que o modelo com agregação de workflow suporta mais padrões de fluxo que o modelo sem a agregação. Por fim são apresentadas algumas vantagens da utilização do modelo proposto.

Palavras-chave: Organização. Agentes. Workflow

ABSTRACT

This work proposes a form of integration of a workflow system to a multiagent systems organizational model. This integration solves some limitations which organizational models generally present, also adding important propreties from workflow systems. Three integration approaches are proposed (hybrid model, workflow model with missions and workflow model), and compared. The best of them is used to integrate the workflow system called *Bonita* to the organizational model *Moise*. In addition to the model, a normative programming language has been extended in order to manage the integration with the workflow. One architecture of implementation is proposed in order to make the proposed model feasible. This architecture is based in the original one from *Moise*, which offers operations and observable properties, allowing agents to interact with an organization. In order to evaluate the proposed solution, a complete example of its use is presented, together with a comparative of workflow patterns. This comparative shows the model which uses workflow supports more flow patterns than the model without workflows. At last, some advantages of the proposed model are presented.

Keywords: Organization. Agents. Workflow.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

A área de Sistemas MultiAgentes (SMA) estuda o comportamento de um grupo organizado de agentes autônomos (DEMAZEAU; BRIOT, 2002). Para que esses agentes possam interagir satisfatoriamente, eles devem ser capazes de se coordenar, cooperar e negociar, possibilitando que tanto a sociedade de agentes quantos os agentes individualmente possam atingir seus objetivos (WOOLDRIDGE, 2002).

De modo geral, uma organização é composta por grupos de agentes que se relacionam entre si a fim de alcançarem objetivos comuns. Estrutura organizacional é frequentemente vista como um meio de gerenciar dinâmicas complexas em sociedades. Isto implica que abordagens para modelagens organizacionais devem incorporar ambos os aspectos estruturais e dinâmicos de tais sociedades (DIGNUM et al., 2002; FERBER; MICHEL; BAEZ, 2005). As características que tornam o estudo das organizações um desafio bastante pesquisado é que elas são sistemas complexos, dinâmicos e adaptativos que evoluem (JENNINGS, 1999).

Modelos organizacionais auxiliam na visualização da estrutura e do funcionamento de uma organização, procuram prever e definir soluções que facilitem à a obtenção dos objetivos da organização. Durante as últimas décadas, muitos modelos organizacionais foram propostos e refinados. Dentre estes modelos estão AGR, TAEMS, Moise, STEAM, ISALANDER e Opera descritos na seção 2.3. Entretanto, os modelos existentes ainda não são completos quando são considerados aspectos relacionados a sua dimensão funcional. A dimensão funcional de um modelo organizacional é formada a partir da decomposição de um objetivo em uma estrutura de árvore, onde a raiz é a meta global e as folhas são as metas locais, ou seja, é caracterizada pela especificação de metas, decomposição de metas e a relação entre essas metas. Essa estrutura utilizada por esses modelos organizacionais possue algumas limitações, tais como:

- Não permite que haja um encadeamento condicional: utilizado para modelar uma escolha entre duas ou mais alternativas, por exemplo: *if* e *then* da lógica de programação;

- Não permite que haja um encadeamento iterativo: algumas vezes é necessário executar a mesma tarefa, ou o mesmo grupo de tarefas, múltiplas vezes até que uma dada condição seja alcançada;

- Não possui tratamento de exceção: onde podem ser previstas as

situações excepcionais que acontecem durante a execução de uma tarefa.

Desta forma, esses modelos organizacionais não conseguem trabalhar com fluxos de trabalho complexos que estão facilmente presentes em sistemas de workflow. Sistemas de workflow são uma tecnologia capaz de coordenar e sincronizar a maneira com que as atividades de uma organização são executadas para a realização de uma determinada tarefa. Workflow é definido pela WfMC como "a automação total ou parcial de um processo de negócio, durante a qual documentos, informações e tarefas são passadas entre os participantes do processo, de acordo com um conjunto de regras de procedimentos"(WFMC, 1999).

1.1 OBJETIVOS DA PESQUISA

Considerando-se essas limitações apresentadas por alguns dos modelos organizacionais existentes, o que se propõe neste trabalho é a utilização de um modelo conceitual para integração de um sistema de workflow com um modelo organizacional de sistemas multiagentes. O objetivo é utilizar um sistema workflow para controlar o fluxo das metas da dimensão funcional de uma organização, tratando as limitações apresentadas acima e adicionando outras propriedades importantes que sistemas de workflow podem oferecer.

Diante do acelerado crescimento e dos benefícios trazidos pela tecnologia de gerenciamento de workflow, além dos benefícios trazidos pelos agentes de software, introduzindo certo grau de autonomia e mobilidade nos workflows, acreditamos que pesquisar a união de tais tecnologias e apresentar um estudo elaborado nos parece um ganho e uma contribuição significativa à área de organização de sistemas multiagentes. As seguintes perguntas de pesquisa que motivaram este trabalho são:

1. Como agregar sistemas de workflow à especificação funcional de um modelo organizacional de sistemas multiagentes?

2. Quais as vantagens na utilização de um sistema workflow em modelos organizacionais de sistemas multiagentes?

1.2 METODOLOGIA

Afim de atingir os objetivos definidos neste trabalho e responder as perguntas de pesquisa desta dissertação, o trabalho é realizado de maneira exploratória e experimental, sendo dividido em uma série de etapas.

Inicialmente é feito a um levantamento do estado da arte onde foram estudados e definidos alguns conceitos principais sobre a teoria de SMA, os aspectos envolvidos sobre SMA, gerenciamento de processos e sistemas de workflow. Nesta etapa, também foram estudados diversos modelos organizacionais e ferramentas de workflow. Para a obtenção de referencial bibliográfico foram consultados alguns repositórios de publicações, tais como:

- http://ieeexplore.ieee.org/
- http://dl.acm.org/
- http://citeseerx.ist.psu.edu/
- http://scholar.google.com/
- http://academic.research.microsoft.com/

Uma vez realizado o levantamento bibliográfico e o estado da arte, foi definido um modelo conceitual para integrar um sistema de workflow a um modelo organizacional de SMA. Para a escolha do modelo conceitual foram analisadas e comparadas três abordagens diferentes: o modelo híbrido, o modelo workflow com missões e o modelo workflow puro. Após conceber o modelo conceitual, é estendida linguagem de programação normativa para organização. Essa linguagem possui o objetivo de dar suporte a agregação de uma ferramenta de workflow a um modelo organizacional.

Na etapa seguinte é definida uma arquitetura de implementação para viabilizar o funcionamento do modelo conceitual proposto. Os requisitos principais que a arquitetura deve seguir são: A arquitetura deve ser genérica o suficiente para que possa ser integrada com diversos sistemas de workflow; A arquitetura deve disponibilizar meios para que os agentes possam executar operações sobre o ambiente e a organização e devem ser disponibilizado as propriedades observáveis da especicação organizacional.

Finalmente, após a implementação da arquitetura proposta, é apresentado um exemplo completo de sua utilização com o intuito de

mostrar o seu funcionamento e validar o modelo proposto. Na validação foram analisados diversos aspectos relacionados ao uso do modelo, apresentando as suas vantagens com o objetivo de verificar se o modelo atingiu os objetivos da dissertação.

1.3 ESTRUTURA DO TRABALHO

O trabalho está estruturado em seis capítulos. No segundo e terceiro capítulos são apresentados uma breve revisão bibliográfica e o estado da arte relacionado à organização de SMA e sistemas de workflow. Nestes capítulos são analisados e comparados modelos organizacionais e ferramentas de workflow. No quarto capítulo propõe-se um modelo conceitual para integração de um sistema de workflow a um modelo organizacional. São avaliadas três proposta para o modelo conceitual e a melhor é apresentada em detalhes. No capítulo cinco é apresentada a arquitetura de implementação, são mostrados os detalhes em relação à arquitetura construída de acordo com o modelo proposto. Nele também é apresentada como foi feita a integração do modelo proposto em uma plataforma de programação e execução de SMA. Também são apresentadas as ferramentas utilizadas na implementação e as etapas utilizadas para construção do modelo proposto. Neste capítulo na seção de avaliação é ilustrado um exemplo completo de utilização do modelo, mostrado uma comparação entre padrões de workflow e apresentado as vantagens encontradas. Finalmente, no capítulo seis são apresentadas as conclusões da pesquisa e as extensões sugeridas como trabalhos futuros.

2 ORGANIZAÇÃO DE SISTEMAS MULTIAGENTES

Este capítulo introduz conceitos referente a sistemas multiagentes, organizações de sistemas multiagentes. Além disso, é apresentada uma análise de alguns modelos organizacionais e também um comparativo entre eles. Por fim, o modelo organizacional Moise é descrito em detalhes.

2.1 SISTEMAS MULTIAGENTES

Existem diversas definições para agentes, entretanto uma definição exata e definitiva de agente não é o objetivo deste trabalho. Um agente em sua forma mais simples é tudo aquilo que pode ser considerado capaz de interagir em um ambiente por intermédio de seus sensores e atuadores. Os sensores auxiliam o agente a realizar a percepção do ambiente, no caso de um agente robótico, através de câmeras e detectores da faixa de infravermelho e os atuadores servem para realizar ações no ambiente como se locomover pelo ambiente (RUSSELL; NORVIG, 2004). Essa ideia simples é ilustrada na Figura 1.

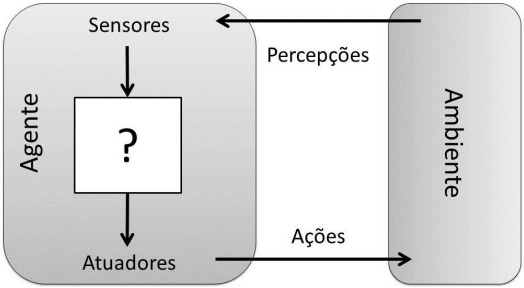


Figura 1: Agentes interagem com ambientes por meio de sensores e atuadores (RUSSELL; NORVIG, 2004).

Segundo (COPPIN, 2004) um agente é uma entidade capaz de realizar alguma tarefa, geralmente para ajudar um usuário humano. Os agentes podem ser biológicos (pessoas ou animais, por exemplo), robóticos ou computacionais. Apesar dessa definição não estar incorreta, ela omite os elementos ambiente, sensores e atuadores mesmo eles sendo fundamentais para a análise de um sistema de agente.

Algumas características fundamentais de agentes são: agirem de

forma autônoma, sem a intervenção de um ser humano na tomada de decisões, levando à satisfação dos seus objetivos; serem capazes de interagir com outros agentes utilizando alguma linguagem de comunicação inspirada nas dos humanos (REIS, 2003).

Os objetivos que devem ser alcançados pelos agentes podem ser comuns ou não entre eles, caso sejam comuns dizemos que os agentes são cooperativos, caso contrário dizemos que são competitivos (RUSSELL; NORVIG, 2004). Agentes competitivos estão unicamente interessados no bem pessoal próprio, agentes cooperativos possuem uma noção de bem social e estão preocupados em atingir objetivos compartilhados com outros agentes.

Um conjunto de agentes que interagem em um ambiente comum ou trabalham juntos para realizar um determinado conjunto de tarefas ou objetivos são chamados de sistemas Multiagentes (SMA). O objetivo da área de SMA é a definição de modelos genéricos de agentes, interações e organizações que possam ser instanciados dinamicamente dado um certo problema (HÜBNER, 2003). Algumas das características de SMA apresentadas em (JENNINGS, 1999) são:

- Organização: refere-se à organização das atividades que são necessárias para o processo de solução do problema;
- Cooperação: ocorre quando há o trabalho de um ou mais agentes para realizar uma tarefa;
- Negociação: acontece quando algum agente tenta obter auxílio para a realização dos seus objetivos através do oferecimento de alguma vantagem;
- Comunicação: refere-se à forma de comunicação, qual protocolo será utilizado para a troca de informação entre os agentes.

Na Figura 2 é possível observar uma visão de mundo orientada a agentes onde é possível perceber que um simples agente é insuficiente para resolver a maioria dos problemas. Como esses agentes possuem capacidades diferentes de percepção e ação no ambiente, eles serão capazes de influenciar diferentes partes do ambiente, por isso eles precisam interagir uns com os outros agentes para atingir seus objetivos individuais ou acessar os recursos do ambiente.

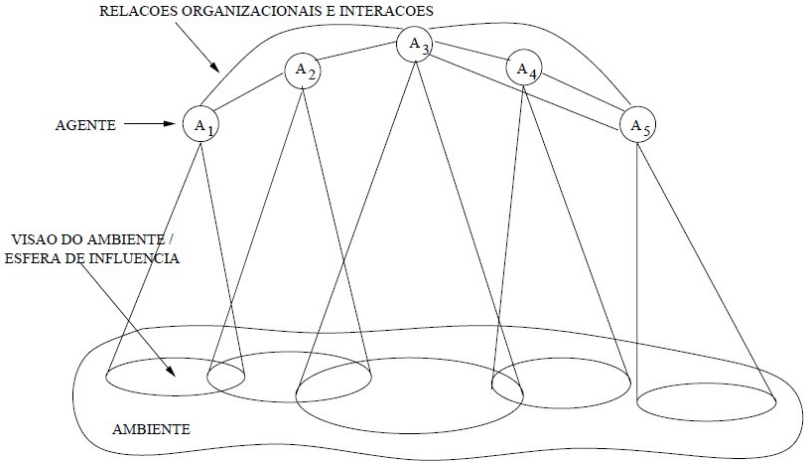


Figura 2: Visão de um Sistema Multiagente (JENNINGS, 1999).

2.2 MODELOS ORGANIZACIONAIS DE SMA

Em uma organização há uma união entre os agentes que formam um grupo. A organização pode ser usada para definir objetivos globais e os agentes relacionam-se a para atingirem esses objetivos. Sempre que um grupo puder ser representado de forma explícita, ele será chamado de uma organização. Numa Organização são definidos os meios através dos quais o projetista do sistema pode garantir que cada agente desejará e realizará a resolução dos problemas propostos.

A organização pode mudar os objetivos dos agentes, dar-lhes informações, ou dar-lhes ordens, pode ser usada para ajudar os agentes a executar suas tarefas ou para alocar os recursos necessários para a execução dessas tarefas (GERS, 2002). Segundo (WEISS, 1999), não existe um grande consenso sobre o conceito de organizações, abaixo estão listadas algumas características que normalmente estão associadas a este conceito:

- Constituídas de múltiplos agentes (humanos ou artificiais);
- Estão associados às tecnologias de resolução de problemas de grande escala;
- Estão comprometidas com uma ou mais tarefas;
- São sistemas altamente ativos;

- Capaz de afetar e ser afetada pelo ambiente;
- Possui conhecimento, cultura, memórias, história e capacidades distintas a um agente simples.

Em (COUTINHO; SICHMAN; BOISSIER, 2005), os autores citam quatro dimensões que são utilizadas na maioria dos modelos organizacionais de SMA. Essas dimensões podem ser utilizadas para classificar as abordagens organizacionais e suas características. São elas:

- Estrutural: A dimensão estrutural está ligada à especificação de papéis, grupos e relacionamentos entre estes, que podem ou não ser definidos a partir de objetivos organizacionais;
- Funcional: A dimensão funcional caracteriza-se pela especificação e decomposição de metas e a relação entre elas;
- Dialógica: A modelagem dialógica caracteriza-se pela especificação de estruturas de interação direta entre papéis por troca de mensagens tendo em vista a realização de objetivos organizacionais, diálogos, cenas e protocolos;
- Normativa: Na dimensão normativa são definidas as normas que interrelacionam e regulamentam elementos funcionais, estruturais e dialógicos.

Modelos organizacionais auxiliam na visualização da estrutura e do funcionamento de uma organização, procuram prever e definir soluções que facilitem à organização atingir os seus objetivos. Nesse capítulo são analisados e comparados seis modelos organizacionais: AGR, TAEMS, Moise, STEAM, ISALANDER e Opera. Cada modelo é brevemente descrito, sendo que o Moise é apresetando com mais detalhes na seção 2.3, por ser o modelo utilizado como referência nesta dissertação.

2.2.1 AGR

AGR (*Agent*, *Group*, *Role*) (FERBER; GUTKNECHT; MICHEL, 2003) é uma evolução do modelo Aalaadin (FERBER; GUTKNECHT, 1998). O modelo AGR é baseado em três conceitos primitivos, agente, grupo e papel.

Um agente é uma entidade ativa, que se comunica e que pode desempenhar papéis dentro de grupos. Nenhuma restrição é imposta sobre a arquitetura de um agente ou sobre as suas capacidades mentais.

Um grupo é um conjunto de agentes compartilhando características comuns. Um grupo pode ser utilizado tanto como um contexto para padrões de atividades, como também para particionamento de organizações. Um agente pode participar simultaneamente de mais de um grupo e dois agentes podem se comunicar somente se pertencerem ao mesmo grupo. Um papel é a representação abstrata de uma posição funcional de um agente em um grupo. Um agente ao entrar em um grupo precisa desempenhar ao menos um papel e um papel pode ser desempenhado por vários agentes.

Em AGR existem dois tipos de restrições estruturais: correspondência e dependência. Na restrição de correspondência um agente que desempenhar um determinado papel deve automaticamente desempenhar outro papel. A restrição de dependência afirma que para um agente desempenhar um papel ele necessita obrigatoriamente estar desempenhando outro papel, por exemplo, um agente está autorizado a desempenhar um papel de "diretor", somente se ele desempenha um papel de "investigador".

2.2.2 TÆMS

TÆMS (*Task Analysis, Environment Modeling, and Simulation*) (DECKER; LESSER, 1994; DECKER, 1996; LESSER, 2002) é um modelo organizacional que possui como primitiva básica o conceito de tarefa. TÆMS possui uma linguagem para descrever a estrutura de tarefas que viabiliza a análise e a simulação da organização.

A estrutura de tarefas é uma forma de representação da decomposição de tarefas em uma árvore. O nível mais alto de uma árvore (raiz) é chamado de grupo de tarefas e representa o objetivo do agente. As folhas desta árvore são chamadas de métodos e são as unidades de tarefas que os agentes podem executar. A estrutura de tarefas possui três níveis: objetivo, subjetivo e generativo.

O nível objetivo descreve a estrutura de tarefas numa visão completa ou 'real' que resolve um problema em um determinado período de tempo.

O nível subjetivo descreve como os agentes visualizam e interagem com a resolução do problema ao longo do tempo. Os agentes participantes da execução de uma tarefa não veem a tarefa como um todo, mas somente a parte que a organização do sistema lhes permite. Dessa forma, o nível subjetivo descreve a estrutura de tarefas do ponto de vista de cada agente, de maneira local.

Por fim, o nível generativo descreve as características estatísticas necessárias para gerar várias visões objetivas e subjetivas. A partir destas visões geradas, é possível avaliar quais estruturas de tarefa são mais adequadas em um dado domínio de resolução de problemas.

2.2.3 Moise

Moise (*Model of Organization for multI-agent SystEms*) (HANNOUN et al., 1999; HÜBNER, 2003) é um modelo organizacional para projeto de SMA baseado em noções como papéis, grupos e missões. Para o Moise, a organização é vista como um meio para reduzir a complexidade do problema a ser resolvido, através do esclarecimento e da divisão de tarefas entre os agentes e da definição de relações entre eles. Moise define uma especificação explícita da organização utilizada pelos agentes para raciocinar sobre a organização e também utilizada como uma plataforma que impõe aos agentes uma especificação a ser seguida.

Moise possui três dimensões: a estrutural, a funcional e a normativa. Na dimensão estrutural, o Moise define a estrutura interna de uma organização, a especificações de grupos, papéis e seus relacionamentos (ligações entre papéis, restrições e hierarquias). A dimensão funcional do Moise é formada a partir da especificação de esquemas sociais (compostos de planos e missões) que visam atingir uma meta global. Uma meta global representa um estado do mundo desejado pela organização, enquanto uma meta local é um objetivo de um único agente. Planos determinam a coordenação da realização das metas. Uma missão é um conjunto de metas locais que podem ser atribuídas a um agente através de seus papéis, sendo que esse é responsável pela satisfação de todas as metas da missão. E por fim, a dimensão normativa liga os aspectos funcionais e aspectos estruturais, indicando quais as responsabilidades dos agentes nos planos globais.

2.2.4 STEAM

STEAM (*a Shell for TEAMwork*) (TAMBE, 1997), é um modelo organizacional baseado na teoria das intenções conjuntas (LEVESQUE; COHEN; NUNES, 1990) e permite a representação explícita dos objetivos, planos e compromissos conjuntos de um time. Isso é utilizado na prática para permitir que vários membros de um time possam manter uma visão coerente dos planos e objetivos de sua equipe. Duas hie-

rarquias separadas são utilizadas para especificar a estrutura interna e o funcionamento de um time: uma hierarquia de subtimes e papéis e uma hierarquia de atividade conjunta.

A hierarquia de subtimes e papéis possui um time que pode ser recursivamente composto por subtimes e agentes individuais. Uma característica desta hierarquia organizacional é que ela possui dois tipos de papéis: papéis persistentes e papéis de tarefas específicas. Papéis persistentes são as atribuições de papéis de longo prazo para agentes individuais ou subtimes da organização. Essas atribuições geralmente não mudarão no curto prazo. Papéis de tarefas específicas são as atribuições de papéis de curto prazo, baseado na situação e tarefa atual.

A hierarquia de atividade conjunta se baseia em uma representação explícita das atividades do time, na forma de operadores de equipe ou plano reativo. Um operador representa uma atividade ou objetivo a ser realizado. Operadores de equipe expressam atividades conjuntas de um time, ao contrário dos operadores individuais que expressam as atividades próprias de um agente.

2.2.5 ISLANDER

ISLANDER é uma linguagem declarativa para a especificação de instituições eletrônicas. (ESTEVA; DAVID; SIERRA, 2002). Estas instituições estabelecem como as interações deverão ser estruturadas dentro de uma organização. Uma instituição eletrônica é composta por quatro elementos básicos: *framework* de diálogo, cenas, estrutura performativa e normas.

Um framework de diálogo define as ilocuções válidas que podem ser trocadas entre os participantes, os papéis participantes e os relacionamentos entre esses papéis. Ilocuções válidas são aquelas que respeitam uma ontologia comum, uma linguagem de comunicação e uma linguagem de representação de conhecimento.

Uma cena é composta por um conjunto de agentes que interagem seguindo um protocolo de comunicação bem definido. Protocolos de comunicação definem os diálogos que um papel pode ter em cada cena. Estrutura performativa é composta por cenas e especifica as relações que ocorrem entre essas cenas. Essas relações entre as cenas definem que agentes, desempenhando determinados papéis, podem passar de uma cena para outra, dadas algumas restrições.

As normas de uma instituição eletrônica definem os compromissos, obrigações e direitos dos agentes participantes.

2.2.6 OperA

OperA (*Organizations per Agents*) (DIGNUM et al., 2002; DIGNUM, 2004) é um modelo organizacional onde a organização de agentes é representada por quatro elementos: a estrutura social, a estrutura de interações, a estrutura de normas e a estrutura de comunicação.

A estrutura social descreve os papéis, dependências funcionais entre papéis, grupos e objetivos. Papéis são os principais elementos da estrutura social e a partir deles é definida e identificada as atividades e serviços necessários para atingir os objetivos da sociedade. Na definição de papéis, objetivos podem ser divididos em subobjetivos. Grupos proporcionam meios para a agregação de um conjunto de papéis e são usados para a definição de normas que envolvam todos os papéis do grupo.

A estrutura de interação é definida como um conjunto de cenas. Cenas representam os possíveis diálogos e interações envolvendo papéis, onde mensagens são trocadas pelos agentes a fim de alcançar os objetivos relacionados à cena. Exemplos disso são os diálogos necessários para um contrato chegar a um acordo, ou o fluxo de trabalho representado por um processo de vendas.

Na estrutura normativa, especificam-se normas relacionadas a papéis, normas de cenas e normas de transições. Existem três tipos de normas: de permissões, de obrigações e de proibições.

Já na estrutura de comunicação, definem-se a linguagem de comunicação, a linguagem de representação de conhecimento e a ontologia de domínio utilizadas nos mecanismo de comunicação da organização.

2.2.7 Comparação entre os modelos organizacionais

Foi realizada uma comparação (Tabela 1) entre os modelos descritos. Para a escolha dos modelos foi utilizado como critério principal a existência de uma infraestrutura organizacional que permite a implementação de organizações de agentes. Os modelos que satisfazem a esse critério são: AGR, STEAM, MOISE e ISLANDER. Já o Modelo TAEMS satisfaz parcialmente a este critério.

Os critérios utilizados na comparação foram as dimensões estrutural, funcional, dialógica e normativa. Um modelo que possua o sinal (-) em alguma dimensão, significa que essa dimensão não é suportada pelo modelo. O sinal (+) representa o nível de detalhe que a dimensão possui. Este comparativo é baseado nos trabalhos de (COUTINHO; SI-

CHMAN; BOISSIER, 2005; COUTINHO, 2009) e permite verificar o poder de expressão de cada modelo organizacional.

	Dimensão			
Modelo	Estrutural	Dialógica	Funcional	Normativa
AGR	++	+	-	−
STEAM	+	−	+++	−
TAEMS	-	−	+++	−
Moise	+++	−	++	+
ISLANDER	+	+ + +	-	++
Opera	++	+ + +	+	++

Tabela 1: Quadro comparativo entre os modelos organizacionais.

Na comparação buscou-se identificar os modelos de acordo com suas dimensões organizacionais. Pela Tabela 1 pode-se observar que o modelo AGR é voltado pra dimensão estrutural, STEAM voltado pra dimensão funcional e TÆMS é um modelo essencialmente funcional. Por outro lado Moise combina as dimensões estrutural funcional e normativa e ISLANDER combina as dimensões estrutural, dialógica e normativa. Apesar de Opera não possuir infraestrutura organizacional, ele é o o único modelo analisado no comparativo que apresenta conceitos cobrindo todas as dimensões organizacionais.

Foi escolhido o modelo organizacional Moise para ser utilizado nesse trabalho. A escolha do Moise se deu pelo seguinte motivo: dentre os modelos organizacionais que possuem a dimensão funcional e possuem infraestruras organizacionais, o Moise é o mais completo. Apesar de STEAM e TÆMS possuírem a dimensão funcional mais rica em detalhes que o Moise, este possui mais detalhes nas dimensões estrutural e normativa.

É importante frisar que dentre os modelos organizacionais comparados que possuem dimensão funcional, todos apresentam as limitações apresentadas no capítulo 1.

2.3 DETALHAMENTO DO MODELO ORGANIZACIONAL MOISE

Nesta seção iremos apresentar o modelo Moise (HANNOUN et al., 1999; HÜBNER, 2003), escolhido para ser utilizado nessa dissertação e brevemente descrito na seção 2.2.3. Inicialmente iremos falar do Moise *Organisation Modelling Language* (Moise OML) e da entidade orga-

nizacional. Em seguida iremos apresentar a *Normative Programming Language* e por fim discutiremos o linguagem de programação de organização normativa.

2.3.1 Moise OML

Moise propõe a OML, uma linguagem declarativa utilizada para representar de forma explicita uma Especificação Organizacional do ingles *Organizacional Specification* (OS). Uma OS é guardada em arquivo XML e pode ser decomposta em três dimensões: estrutural (SS), funcional (FS) e normativa (NS) (HÜBNER; SICHMAN; BOISSIER, 2007).

```
<OS = SS, FS, NS>
```

A dimensão estrutural do Moise define a estrutura interna de uma organização, a especificações de grupos, papéis e seus relacionamentos (ligações entre papéis, restrições e hierarquias). A Figura 3 apresenta um exemplo de uma especificação estrutural onde foram definidos quatro grupos. O grupo time (grupo raiz) é composto pelos subgrupos exploração, defesa e ataque. Uma instância do grupo time deve possuir uma única instância do grupo exploração, uma do grupo defesa e uma do grupo ataque. Além disso, o grupo time está ligado ao papel de capitão. Os agentes do grupo exploração podem assumir o papel de nave ou o papel de explorador e a nave tem autoridade sobre o explorador. O grupo defesa é composto de zero ou infinitas unidades de defesa e o grupo ataque é composto de zero ou infinitas unidades de ataque e o papel capitão tem autoridade sobre os papéis unidade de ataque e defesa.

A dimensão funcional do Moise é formada a partir da especificação de esquemas sociais que visam atingir uma meta global. Desta forma, um esquema social é formado a partir da decomposição de uma meta global em metas locais. Essa decomposição é realizada através de planos e as metas locais são alocadas aos agentes a partir de suas missões. Uma meta global representa um estado do mundo desejado pela organização, enquanto uma meta local é um objetivo de um único agente. Planos determinam a coordenação da realização das metas. No Moise uma meta global pode ser decomposta em três planos: sequencial, paralelo e escolha,. Por outro lado uma missão é um conjunto de metas locais que podem ser atribuídas a um agente através de seus papéis, sendo que esse agente é responsável pela satisfação de todas as metas da missão.

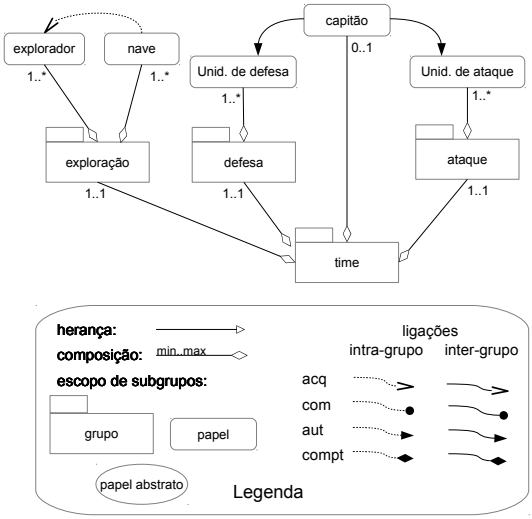


Figura 3: Exemplo da dimensão estrutural do Moise.

A Figura 4 apresenta um exemplo da especificação funcional de um esquema social. Na raiz do esquema tem-se a meta global de destruir o inimigo. Para satisfazer a meta global, três submetas em paralelo precisam ser satisfeitas: a meta ataque, a meta defesa e a meta recurso. Essas três metas são decompostas em três planos, e esses planos são compostos por metas locais. As metas dos planos ataque e recurso devem ser alcançadas em forma sequencial, ou seja, só é possível alcançar a meta g2 depois que a meta g1 for alcançada. Já as metas do plano defesa podem ser alcançadas de forma paralela, ou seja, pode-se alcançar a meta g8 e g6 ao mesmo tempo.

As missões são assumidas pelos agentes no momento em que estes assumem um papel em um grupo. Conforme especificado na Figura 4, o agente que se compromete com a missão m1 é responsável pela satisfação de todas as metas desta missão, no caso, as metas g1 e g9. Já o agente que assumir a missão m2 deve alcançar as metas g2 e g8.

A relação entre a especificação estrutural e especificação funcional é feita pela especificação normativa. Na especificação normativa são descritas as missões com as quais um papel tem permissão ou obrigação de se comprometer. Na Tabela 2 é possível observar que o papel de nave está obrigado a alcançar as metas da missão m1, o capitão tem obrigação de alcançar a missão m2, e assim por diante.

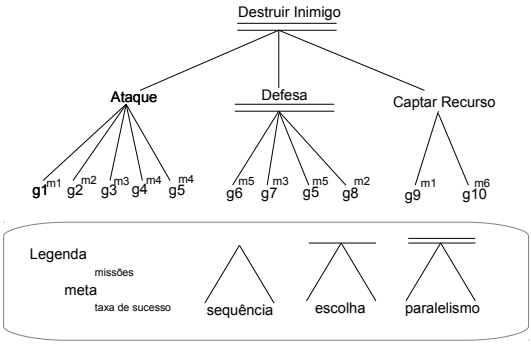


Figura 4: Exemplo da dimensão funcional do Moise.

Papel	Relação Deôntica	Missão
nave	obrigação	m1
capitao	obrigação	m2
conselho	obrigação	m3
unidade ataque	obrigação	m4
unidade defesa	obrigação	m5
explorador	obrigação	m6

Tabela 2: Exemplo da dimensão normativa.

2.3.2 Entidade Organizacional

Conforme visto anteriormente as três dimensões organizacionais do Moise formam a OS a qual não inclui agentes, pois tem um caráter mais abstrato. A OS é instânciada por um conjunto de agentes formando uma *Organizational Entity* (OE), que estabelece a posição dos agentes no contexto do sistema (Figura 5).

2.3.3 NPL

A Linguagem de Programação Normativa, do inglês `Normative Programming Language` (NPL) é baseada em três primitivas (obrigação, permissão e proibição) e duas estratégias de aplicação (sanção e regimentação) (TINNEMEIER; DASTANI; MEYER, 2009; LóPEZ; LUCK; D'INVERNO, 2002; GARCíA-CAMINO et al., 2009). Uma sanção é uma estratégia reativa aplicada após a ocorrência de uma violação e regimentação é uma

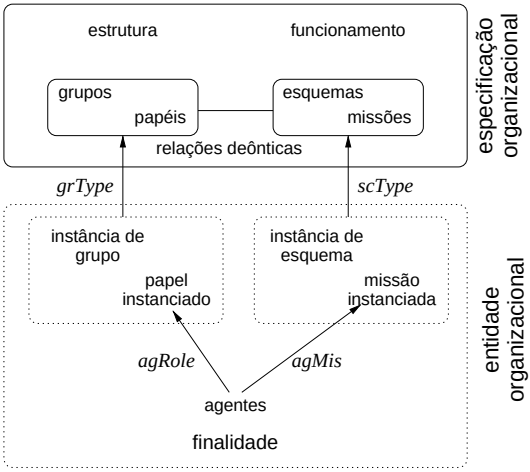


Figura 5: Entidade Organizacional (HÜBNER, 2003)

estratégia preventiva em que os agentes não são capazes de uma violação. NPL é baseada nas seguintes suposições (HüBNER; BOISSIER; BORDINI, 2011):

1. proibições são representadas tanto por arregimentação quanto por uma obrigação para que um agente decida como controlar a situação (impor alguma sanção);

2. permissões são definidas por omissão (tudo é permitido por padrão);

3. sanções são representadas como obrigações (algum agente é obrigado a aplicar a sanção).

4. normas são consideradas consistentes.

De acordo com a grámatica apresentada na Figura 6 a sintaxe NPL é composta por um conjunto inicial de fatos, um conjunto de regras de inferência (como em Prolog) e um conjunto de normas. Uma norma NPL possui seguinte forma:

$$\texttt{norm } id \texttt{ : } \varphi \texttt{ -> } \psi$$

onde id é um identificador único da norma, φ é a condição de ativação da norma e ψ é a consequÊncia da ativação da norma. Dois tipos de consequências são possíveis:

- `fail(`r`)`: representa uma norma regimentada, onde r representa a razão para a falha;
- `obligation(`a`; `r`; `g`; `d`)`: representa o caso onde uma obrigação para o agente a é criada. O parametro r é a razão da obrigação, onde um agente a é obrigado a alcançar o objetivo g, antes do tempo d.

```
np      ::= "np" atom "{" ( rule | norm )* "}"
rule    ::= atom [ ":-" formula ] "."
norm    ::= "norm" id ":" formula "->" ( fail | obl ) "."

fail    ::= "fail(" atom ")"
obl     ::= "obligation(" (var | id) "," atom "," formula "," time ")"

formula ::= atom | "not" formula | atom ( "&" | "|") formula
time    ::= "`" ( "now" |
              number ( "second" | "minute" | ...))
              "`" [ ( "+" | "-" ) time ]
```

Figura 6: Gramática para NPL (HüBNER; BOISSIER; BORDINI, 2011).

O ciclo de vida de uma obrigação (Figura 7) possui os seguintes estados:

- *Active:* é o estado inicial de uma obrigação recém criada.
- *Fullfilled:* o estado da norma muda para *fulfilled* quando um agente a cumpre o objetivo g antes do tempo d.
- *Unfullfilled:* o estado da norma muda para *unfulfilled* quando o ogente a não cumpre o objetivo g dentro do tempo d.
- *Inactive:* caso a condição de ativação de uma norma deixe de ser verdadeira o estado muda para inactive.

2.3.4 NOPL

A Linguagem de Programação de Organização Normativa, do inglês `Normative Organisation Programming` Language (NOPL) é uma classe particular de NPL especializada para Moise OML. Possui a mesma sintaxe e semântica de NPL mas usando fatos, regras e normas específicas para o modelo Moise. Regras de tradução (*Translation rules t-rules*) são utilizadas para traduzir uma OS definida em Moise OML em programas NOPL, um para cada grupo ou esquema definido pela

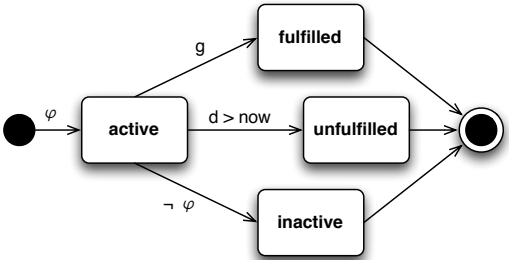


Figura 7: Ciclo de vida de uma obrigação em NPL (HüBNER; BOISSIER; BORDINI, 2011).

OS. Em seguida são apresentados fatos, regras e normas geradas para programas NOPL de esquema. Programas normativos de esquema possuem os seguintes fatos:

- `goal(`m`,`g`,`$pre-cond$`,'ttf'):` é um fato que define os argumentos para uma meta: m é o identificador da missão ligado a meta, g é o identificador da tarefa, $pre - cond$ são as precondições da meta e $'ttf'$ é o tempo que a meta tem para ser executada.

- `mission_role(`m`,`ρ`):` o papel ρ tem permissão ou obrigação de se comprometer com a missão m.

- `mission_cardinality(`m`,`min`,`max`):` é um fato que define a cardinalidade de um missão, ou seja, apresenta o número mínimo e máximo de agentes que podem se comprometer com a missão.

Os seguintes fatos dinâmicos estarão disponíveis em tempo de execução no artefato que gerencia a instância de um esquema.

- `play(`a`,`ρ`,`gr`):` o agente a possui o papel ρ na instância do grupo identificado como gr.

- `committed(`a`,`m`,`s`):` o agente a assume a missão m no esquema s.

- `responsible(`gr`,`s`):` o grupo instanciado gr é responsável pelas missões do esquema instanciado s.

- `achieved(`s`,`g`,`a`):` a meta g no esquema s foi alcançada pelo agente a.

As regras são utilizadas para a inferência de novos estados para o esquema. De acordo com as regras apresentadas a seguir, um esquema é considerado bem formado se a cardinalidade mínima de cada missão está satisfeita. A regra `mplayers` informa quantos agentes se comprometeram com a missão M. Uma meta é considerada habilitada apenas quando todas as suas pré-condições foram satisfeitas. Uma missão é considerada cumprida somente quando todas as metas da missão foram satisfeitas e o esquema é finalizado quando todas as metas do esquema foram satisfeitas.

- `well_formed(`T`):- mplayers(`m`,`S`,`Vm`) & Vm` $\geq$ `minmpS(`m`) & Vm` $\leq$ `maxmpS(`m`).`
- `mplayers(`M`, `P`, `mP`):- .count(committed (_,`M`,`S`),` mP`).`
- `enabled(`S`,`G`):- goal(_,`G`,`PCG`,_) & all_satisfied(`S`,`PCG`).`
- `mission_accomplished(`S`,`M`):- .findall(Goal, goal(`M`,`$Goal$`,_,_,_), MissionGoals) & all_satisfied(`S`,MissionGoals).`
- `all_satisfied(`S`,[`G`|`T`]):- achieved(`S`,`G`,_) & all_satisfied(`S`,`T`).`
- `is_finished(S):- satisfied(`S`,`gr`).`

Existem três classes de normas em NOPL para esquemas: normas para metas, normas para propriedades e domínios de normas. Para a classe de metas é definido a seguinte norma genérica que expressa a semântica Moise para comprometimento:

```
norm ngoal:
     committed(A,M,S) & goal(M,G,_,D) &
     well_formed(S) & enabled(S,G)
  -> obligation(A,ngoal,achieved(S,G,A),‘now’ + D).
```

Esta norma é incluída em cada programa NOPL gerado para um esquema. Ela pode ser lida como, quando um agente A está comprometido com a missão M, o esquema está bem formado e a meta G está habilitada. Então o agente A é obrigado a alcançar a meta G dentro do prazo D.

Para a classe de normas de propriedades, somente a propriedade de cardinalidade de uma tarefa será apresentada, pois as outras propriedades são controladas de maneira similar. A seguir é apresentada

uma norma para cardinalidade de uma missão. A norma define a consequência de uma situação ondem existem mais agentes comprometidos com a missão do que é permitido na especificação do processo. Na primeira norma acontece uma regimentação e na segunda a consequência é uma obrigação.

```
norm mc:
     mission_cardinality(M,_,MMax) &
     mplayers(M,S,MP) & MP > MMax
  -> fail(mission cardinality).

norm mc:
     mission cardinality(M,_,MMax) &
     mplayers(M,S,MP) & MP > MMax &
     responsible(Gr,S) & play(A,ρ,Gr)
  -> obligation(A,mc,committed(A,m,_),‘now’ + ‘ttf’).
```

A norma apresentada a seguir, representa a classe de normas de domínio. Essa norma é utilizada para um agente que possui um papel em um grupo e é responsável pelo grupo. Neste caso se a cardinalidade da missão ainda não foi alcançada e a missão ainda não foi concluída o agente é obrigado a se comprometer com a missão.

```
norm id:
     plays(A,ρ,Gr) & responsible(Gr,S) &
     mplayers(m,S,V) & V < maxmp(m) &
     not mission_accomplished(S,m)
 -> obligation(A,id,committed(A,m,S),‘now’ + ‘ttf’).
```

Essa ultima norma utiliza regimentação para proibir que um agente se comprometa com uma missão que ele não possui permissão.

```
norm mission_permission:
     committed(A,M,S) &
     not (plays(A,R,_)& mission_role(M,R))
  -> fail(mission_permission(A,M,S)).
```

2.4 CONSIDERAÇÕES

Este capítulo apresentou alguns conceitos importantes da área de organização de SMA e realizou um comparativo entre alguns modelos

organizacionais. Dentre os modelos comparados, o Moise pareceu ser a melhor alternativa para a concepção do modelo conceitual. Foram introduzidas as linguagens NPL e NOPL, essas duas linguagens normativas são importantes para o entendimento desta dissertação. No próximo capítulo serão apresentados conceitos referentes a área de sistemas de workflow e gerenciamento de processos.

3 SISTEMAS DE WORKFLOW

Este capítulo apresenta alguns conceitos necessários para um melhor entendimento sobre gerenciamento de processos de negócio, tecnologia de workflow e modelagem de processos de negócio. São avaliadas ferramentas de Workflow e a melhor alternativa será utilizada na integração ao modelo organizacional Moise.

3.1 GERENCIAMENTO DE PROCESSOS DE PRODUTO

Em qualquer organização, pequena ou grande, coexistem e interagem entidades (fornecedores, clientes, produtos e serviços) e operações (produção, marketing e vendas). Cada uma destas operações implica em múltiplos processos de negócio, que viabilizam determinado resultado. É importante comentar que as organizações crescem, tornando suas operações confusas e desordenadas. A visão de processos ameniza o impacto do crescimento não ordenado.

Um processo pode ser definido de diferentes formas. Para (DAVENPORT, 2004), um processo de negócio é um conjunto ordenado de atividades de trabalho no tempo e no espaço, com um começo e um fim. Além disso, há uma entrada, uma transformação e uma saída. A transformação ocorrida deve gerar valor para a entrada e criar uma saída mais útil e eficaz que adiciona valor ao cliente do processo.

Segundo a WfMC (*Workflow Management Coalition*), um processo é "um conjunto coordenado de atividades (sequenciais ou paralelas) que são interligadas com a finalidade de alcançar um objetivo, dentro de uma organização", sendo atividade conceituada como "uma descrição de um fragmento de trabalho que contribui para o cumprimento de um processo"(WFMC, 1999). Processos de negócio são, portanto, um sequenciamento lógico de atividades previamente estabelecidas, cujo objetivo é determinar como o trabalho será realizado em uma organização. Uma estrutura de processos de negócio bem concebida é um fator crítico para o sucesso da organização.

Um processo de negócio pode ser automatizado, facilitando o trabalho de tarefas repetitivas e diminuindo a tramitação de documentos impressos que podem ser perdidos ao longo do processo. A tecnologia de workflow automatiza um processo de forma total ou parcial (WFMC, 1999).

Quanto mais gerenciados forem os seus processos, maiores são

as chances que eles tragam os benefícios e resultados esperados pela organização. Entender como funcionam os processos e quais são os tipos existentes é importante para determinar como eles devem ser gerenciados para a obtenção do máximo resultado (GONCALVES, 2000).

O Gerenciamento de Processos de Negócio, do inglês *Business Process Management* (BPM) é definido como uma técnica moderna que suporta processos de negócios, utilizando um software para especificar, controlar, executar e analisar processos empresariais nos quais envolvam pessoas, empresas, aplicações, documentos e outras fontes de informações (AALST; HOFSTEDE; WESKE, 2003). BPM utiliza conceitos, métodos e técnicas para implantar, modelar e monitorar processos, visando a otimização dos resultados das organizações através da melhoria continua dos processos de negócio.

3.2 O CICLO DE VIDA DOS PROCESSOS

(BALDAN et al., 2007) propõe um modelo para o ciclo de vida do BPM (Figura 8) constituído por quatro fases: Planejamento, modelagem e otimização de processos, execução de processos e controle e análise de dados.

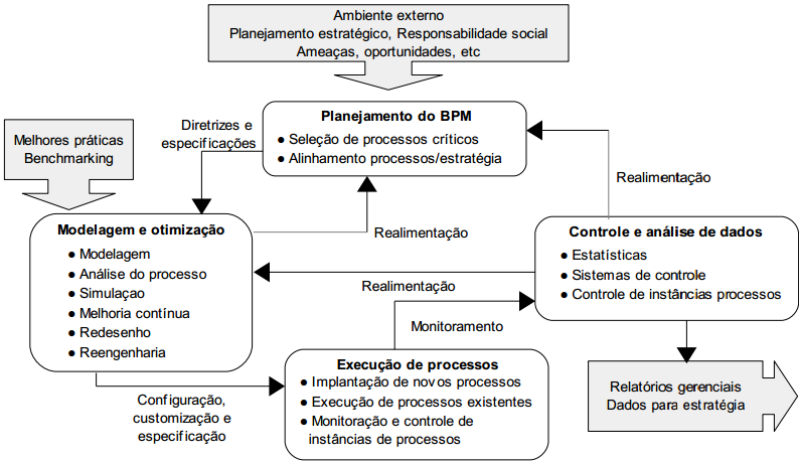


Figura 8: Ciclo de vida de um processo, Adaptado de (BALDAN et al., 2007).

Planejamento do BPM - Comporta as etapas de definição

dos processos chave para a estratégia da organização, levantamento dos principais pontos fracos dos processos em uso na organização, identificação de oportunidades (novos produtos) que necessitem de mudanças ou adaptação do processo atual. Perceber que processos sem problemas podem sofrer melhorias e proporcionar a visão global do modelo de processo, para (BALDAN et al., 2007) "ter uma Visão Global de processos ajuda na compreensão do funcionamento da empresa".

Modelagem e otimização de processos - atividades que permitem gerar informações sobre estado atual do processo (*As Is*) e a otimização e modelagem do estado desejado do processo (*To be*), quando aplicável; prover dados de integração entre processos; fazer simulações, inovações e redesenhos; adotar as melhores práticas e modelos de referência. Na fase de modelagem de processos, ferramentas de software têm demonstrado um grande valor e vem sendo usadas para facilitar a modelagem, documentação e integração entre modelos.

Execução de processos - Atividades que garantem a implementação e a execução dos processos existentes e melhorados, buscando a realização do plano, a ação daquilo que foi estudado e planejado; acompanhamento de processos implantados; monitoria e controle da execução das instâncias de processo e realização das mudanças de curto prazo.

Controle e análise de dados - São geradas as informações sobre o comportamento dos processos, em verificação dos cumprimentos às metas estabelecidas pela organização. Essas informações são utilizadas na elaboração de indicadores de comparação com os antigos processos, possibilitando a compreensão e avaliação dos cenários propostos. Os resultados desta fase geram informações que posteriormente devem realimentar o planejamento do próximo ciclo de BPM.

Para dar suporte a fase de controle e análise de dados são utilizados os sistemas BAM (*Business Activity Monitoring*), onde os processos são instrumentados com sensores para monitorar suas atividades e variáveis. A tecnologia BAM é baseada em estatísticas e pode identificar, por exemplo, que uma certa atividade demora mais do que o previsto, porque existe falta de recursos para executá-la de forma mais eficiente.

3.3 PRINCIPAIS CONCEITOS SOBRE A TECNOLOGIA DE WORKFLOW

Sistemas de workflows é uma tecnologia capaz de coordenar e sincronizar a maneira com que as atividades de uma organização são executadas para a realização de uma determinada tarefa. Workflow é a automação de um processo de negócio, por inteiro ou em parte, durante a qual, informações, tarefas e documentos são passados de um participante para outro, respeitando um conjunto de regras procedurais (WFMC, 1999). Sistemas de gerência de workflow, apesar de bastante antigos, receberam recentemente a atenção da comunidade científica, pela sua gama de aplicações.

Segundo (GEORGAKOPOULOS; HORNICK; SHETH, 1995) os sistemas de workflow podem ser classificados nos seguintes tipos:

Workflows Ad Hoc: descrevem processos simples onde é difícil encontrar um esquema para a coordenação e cooperação de tarefas, não há um padrão fixo para o fluxo de informações entre as pessoas envolvidas, são workflows pouco estruturados e aconselháveis para a modelagem de tarefas não estruturadas. Envolvem coordenação, colaboração e decisão humana durante o seu fluxo. Podem sofrer grandes transformações durante sua execução. A Figura 9 apresenta um workflow simplificado tipo Ad hoc envolvendo o processo de revisão de artigos para publicação.

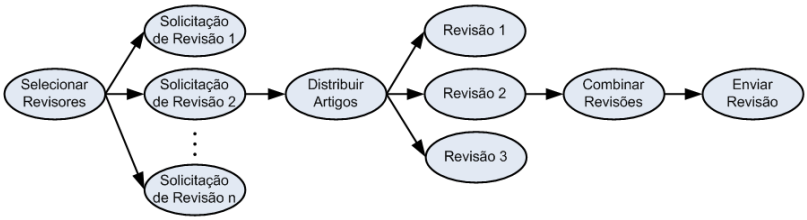


Figura 9: Workflow Ad hoc para revisão de artigos. Fonte: Adaptado de (GEORGAKOPOULOS; HORNICK; SHETH, 1995).

O processo mostrado inicia-se selecionando os revisores, em seguida distribuindo os artigos para os revisores selecionados, possibilitando que estes executem as revisões em colaboração e por fim enviando para os autores.

Workflows de produção: envolvem atividades altamente estruturadas que descrevem processos de informação complexos. Existe a necessidade de acesso a múltiplos sistemas de informações. A ordenação

e coordenação dos processos deste tipo de workflow podem ser automatizadas. Este tipo de workflow automatiza processos de negócios complexos, similares, por exemplo, a linha de montagem de um carro, suporta grandes volumes de informações, acompanhamento de tarefas e documentos compartilhados. A Figura 10 ilustra um workflow para um processo de requisição de seguro saúde.

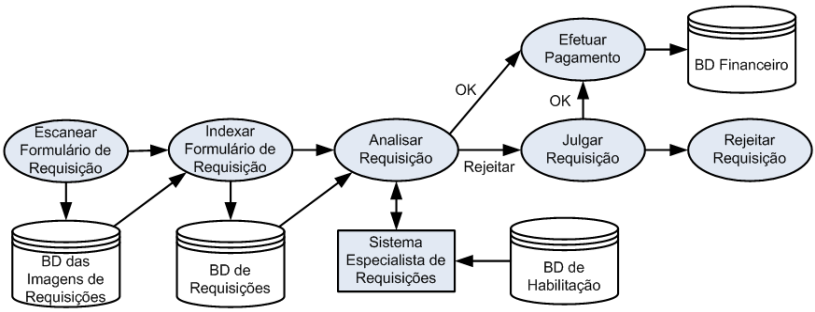


Figura 10: Requisição de atendimento através de um seguro saúde

Observa-se nesse workflow a presença de banco de dados e sistemas especialistas que são acessados ao longo do processo, auxiliando na automatização das tarefas.

Workflows administrativos: envolvem atividades fracamente estruturadas, repetitivas, previsíveis e com regras simples de coordenação de tarefa, possuem características semelhantes aos workflows de produção, entretanto são menos exigentes e não tem a necessidade de acessos múltiplos à sistemas de informação. A ordenação e coordenação dos processos deste workflow podem ser automatizadas.

A Figura 11 apresenta um workflow administrativo, utilizando novamente o processo de revisão de artigos, presumindo que os mesmos revisores são convidados para revisão de todos os artigos e que produzem revisões individuais que são consideradas pelo editor para tomar a decisão final.

A discussão sobre sistemas de workflow envolve vários conceitos básicos necessários para um melhor entendimento do tema proposto. A Figura 12 apresenta um esquema de relacionamento entre os termos básicos de workflow proposto pela WfMC. A seguir são definidos alguns desses conceitos:

Sistema de Gerência de Workflow em inglês (*Workflow Management System* - WfMS): Um sistema que permite a definição, gerenciamento e execução de workflows. WfMS provê a auto-

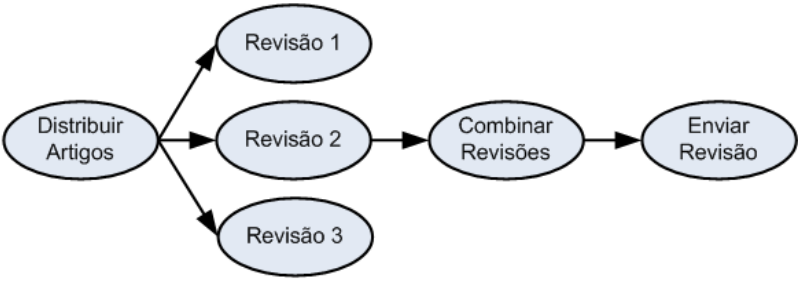


Figura 11: Workflow Administrativo para revisão de artigos. Fonte: Adaptado de (GEORGAKOPOULOS; HORNICK; SHETH, 1995).

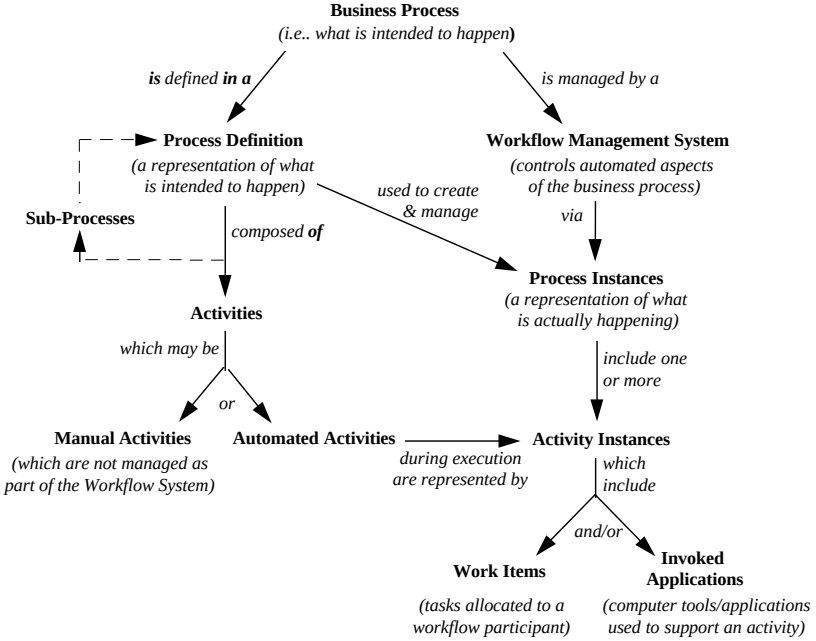


Figura 12: Glossário de workflow: relacionamento entre os conceitos básicos (WFMC, 1999).

matização de processos de negócio, garantindo que as atividades desse processo ocorram na sequência definida. Gerencia a alocação de recursos (humanos ou computacionais), que são responsáveis pela execução de determinada tarefa.

Processo: Conjunto de atividades que se relacionam a fim de atingirem um objetivo comum dentro de um contexto organizacional. Possui critérios que indicam o início e o final do processo e informações sobre as suas atividades, tais como participantes aplicações de TI associadas, dados, etc. Um processo pode conter tanto atividades automáticas quanto manuais.

Sub processo: É um processo, executado ou invocado por outro processo (ou subprocesso) do qual faz parte.

Atividade: Corresponde a uma etapa, tarefa ou unidade de trabalho executada dentro do processo. Uma atividade pode ser manual ou automatizada, ocorre sob a responsabilidade de um ator, entretanto pode ser executada por vários atores.

Instância de um Processo ou de uma Atividade: Quando um processo é iniciado, o sistema de gerência de workflow cria uma instância de processo, que por sua vez gera as instâncias das suas atividades. Uma instância é uma representação interna da definição do processo, ela possui uma identidade única, que a identifica externamente. Várias instâncias podem ser executadas ao mesmo tempo, sendo controladas pelo sistema de gerenciamento de workflows.

Participante do workflow: O participante de um workflow ou ator é um recurso (humano ou de sistema) que executa o trabalho representado por uma instância de atividade do workflow.

Papel: Um papel é o conjunto de competências necessárias para que um ator torne-se apto para a execução de uma dada atividade. Um ator pode possuir vários papéis, uma vez que um ator é associado a um papel, que por sua vez é associado a uma ou mais atividades.

Item de Trabalho: Representação de um trabalho a ser executado por um participante em uma instância do processo. Os itens de trabalho são apresentados ao participante através de uma lista de trabalho. Uma atividade consiste de um ou mais itens de trabalho.

Transição: É um ponto de um determinado momento de execução de uma instância do processo, que indica que uma atividade está completada e a atividade seguinte pode ser iniciada. Uma transição pode ser condicional ou incondicional, onde a sequência de atividades dependem ou não de condições de transição.

Controle do Fluxo de Atividades: Um fluxo de atividades representa uma ordem parcial de atividades relacionadas e as respectivas dependências entre estas.

Rota: Caminho lógico que determina a ordem em que as atividades serão executadas. Roteamentos podem representar uma sequência simples de atividades, bifurcações de diferentes tipos, *loop* e diferentes

tipos de sincronização.

Exceção: Uma exceção é um evento não previsto na especificação do processo, e que pode ocorrer durante a execução do workflow.

Motor do workflow: É o núcleo da arquitetura de um workflow responsável por controlar a execução de um determinado conjunto de instância de processos. Permite a navegação entre as várias atividades do processo, que podem envolver operações sequenciais ou paralelas, imposição de prazos, etc. Controla a entrada e saída de participantes do workflow.

3.4 BPMN

A *Business Process Modeling Notation* (BPMN), ou Notação de Modelagem de Processos de Negócio, é uma notação gráfica utilizada para representação de processos de negócios. A BPMN Foi desenvolvida em 2004 pela *Business Process Management Initiative* (BPMI) e atualmente é mantida pelo *Object Management Group*, desde que as duas organizações se juntaram em 2005. A versão atual da BPMN é a 2.2.

O objetivo principal da BPMN é fornecer uma notação padrão que seja facilmente compreendida por todos os usuários do negócio, desde os analistas de negócio, responsáveis pela primeira definição do processo, até os desenvolvedores que irão implementar o processo (WHITE, 2004). Outro objetivo da BPMN é prover uma especificação que possibilite a passagem de diagramas escritos em BPMN para linguagens projetadas para a execução de processos de negócio, tais como a BPEL4WS (*Business Process Execution Language for Web Services*).

A BPMN fornece uma notação para expressar os processos em um único diagrama de processo de negócio (*Business Process Diagram* - BPD). O BPD é composto por um conjunto de elementos gráficos. Estes elementos permitem o desenvolvimento de diagramas que são bastante familiares para a maioria dos analistas de negócio, pois são parecidos com fluxogramas (WHITE, 2004).

A seguir são apresentados os principais elementos que compõem um diagrama de processo de negócio. Uma atividade é representada por um retângulo de cantos arredondados e demonstra que tipo de trabalho (passo do processo) deve ser executado. Pode ser uma tarefa ou um subprocesso. O subprocesso, diferentemente de uma atividade normal, possui um sinal gráfico '+' na parte inferior do retângulo, como

é possível verificar na Figura 13.

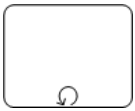
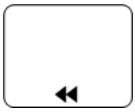
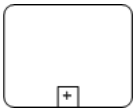
Elemento	Nome	Descrição
	Tarefa	Representa um trabalho ou atividade que um participante atua.
	Loop	Execução da mesma tarefa diversas vezes.
	Compensação	Logo após a finalização desta tarefa, o fluxo é desfeito e/ou retorna para o elemento que o solicitou.
	Sub-processo	Representação de um sub-processo no fluxo. Detalhes desse processo não estão visíveis no diagrama.

Figura 13: Tipos de atividades em BPMN.

Um evento é representado por um círculo e é algo que ocorre (ex.: a chegada de uma mensagem) no início, meio ou fim do processo. A Figura 14 apresenta os principais tipos de eventos em BPMN.

Um *gateway* (passagem) é representado por um losango e determina disjunções, sincronizações e junções de fluxos. Na Figura 15 é possível verificar os principais tipos de *gateway* em BPMN:*AND* e *OR*.

Os objetos conectores representam as transições entre as atividades do processo. Os três tipos diferentes de conectores existentes na BPMN são apresentados na Figura 16.

Um objeto separador (*swimlane*) é um mecanismo visual que organiza diferentes atividades em categorias de mesma funcionalidade. Na Figura 17 podem ser visto os dois tipos de objetos separadores existentes na BPMN:

A Figura 18 apresenta um processo de compra com cartão bancário descrito em BPMN. Os papéis envolvidos no processo são: cliente, banco e vendedor. O processo é iniciado com um *gateway*, onde o cliente precisa escolher se vai pagar a compra com cartão de crédito ou de débito, dependendo da escolha, o cliente vai assinar a fatura ou

Elemento	Nome	Descrição
	Simples	Indica o início, o meio e o fim de um processo (respectivamente).
	Mensagem	Uma mensagem chega para o participante e aciona/interrompe o fluxo.
	Temporal	Um temporizador para iniciar um modo de espera no processo.
	Exceção/Erro	Tratamento de uma exceção/erro específico.
	Múltiplo	Significa que múltiplos eventos podem ocorrer para acionar/interromper o fluxo, bastando apenas um dos eventos especificados para que seja aplicado o fluxo.
	Compensação	Identifica que um segmento do fluxo deve ser desfeito ou retornado para sua origem.
	Regra de Negócio	Condição ou regra para que o fluxo seja seguido.

Figura 14: Elementos que representam eventos em BPMN.

Elemento	Nome	Descrição
	Decisão Exclusiva	Significa que o fluxo deve seguir apenas um caminho a partir de uma condição.
	Bifurcação/Junção Paralela	Significa que o fluxo possui paralelismo e é subdividido ou juntado em outros segmentos.

Figura 15: Tipos de *gateway* em BPMN.

digitar a senha do cartão. Em seguida o banco verifica o saldo do cliente, caso o saldo seja insuficiente uma exceção de saldo é lançada e o processo termina, caso contrário, o banco efetua a tranferência do valor para o vendedor. Por fim o vendedor entrega o recibo para o cliente e a compra é efetuada, finalizando o processo.

Elemento	Nome	Descrição
	Transição sequencial	Mostra a ordem que as atividades são executadas.
	Transição de associação	Associa informações com objetos de fluxo, textos ou elementos gráficos.
	Transição de mensagem	Mostra o fluxo de mensagens entre dois participantes de processos diferentes.

Figura 16: Tipos de objetos conectores em BPMN.

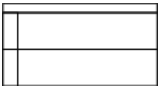
Elemento	Nome	Descrição
	Raia	Representa um participante no processo.
	Piscina	Representa um conjunto de participantes no processo.

Figura 17: Tipos de objetos separadores em BPMN.

3.5 WADE

WADE (*Workflow and Agent Development Environment*) é uma plataforma de software desenvolvida sobre a camada do JADE (*Java Agent DEvelopment Framework*). É um projeto que possui código fonte aberto, distribuído pela Telecom Italia sobre a licença LGPL (*Lesser General Public License Version 2*)(CAIRE; GOTTA; BANZI, 2008). Wade não é apenas um *add-on* para JADE, mas sim uma plataforma completa para o desenvolvimento de aplicações distribuídas baseada no paradigma de agentes, explorando o conceito de workflow para definir as lógicas do sistema e permite que um grupo de agentes possam, de forma cooperativa executar tarefas complexas definidas como workflow.

JADE (*Java Agent Development framework*) é um *framework* para desenvolvimento de aplicações de agentes em Java que seguem o padrão FIPA (*Foundation For Intelligent, Physical Agents*). FIPA é uma associação internacional de várias companhias que trabalham para especificar tecnologias de agentes genéricas. JADE implementa uma plataforma para o desenvolvimento de sistemas distribuídos. Agentes

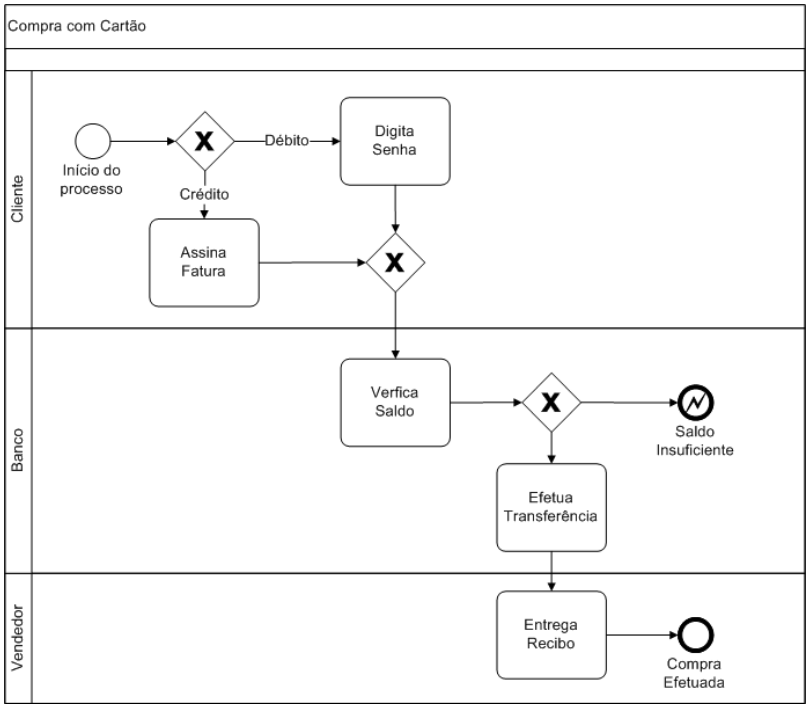


Figura 18: Exemplo BPMN - Compra com cartão.

JADE possuem como forma básica de comunicação mensagens assíncronas baseadas no paradigma *peer-to-peer*. A estrutura dessas mensagens é baseada na linguagem ACL (*Agent Communication Language*) do padrão FIPA. É permitida a comunicação entre agentes JADE com outros tipos de agentes fora da plataforma.

WADE vem com um ambiente de desenvolvimento chamado WOLF (*WOrkflow LiFe cycle management environment*) que facilita a criação de aplicações baseadas em WADE. WOLF é um *plug-in* para Eclipse que facilita o desenvolvimento de aplicações baseadas em WADE, proporcionando um editor gráfico de fluxo de trabalho e uma série de mecanismos para apoiar os programadores no desenvolvimento na fase de testes (CAIRE et al., 2008).

De acordo com a Figura 19, JADE fornece um ambiente de execução distribuído, comunicação peer to peer entre agentes, abstrações de agentes e comportamentos, gerenciamento básico de um ciclo vida de um agente e mecanismos de descoberta. WADE adiciona

a JADE suporte para a execução de tarefas definidas de acordo com a metáfora de workflow e uma série de mecanismos que auxiliam a gestão da complexidade da distribuição, tanto em termos de administração e tolerância a falhas.

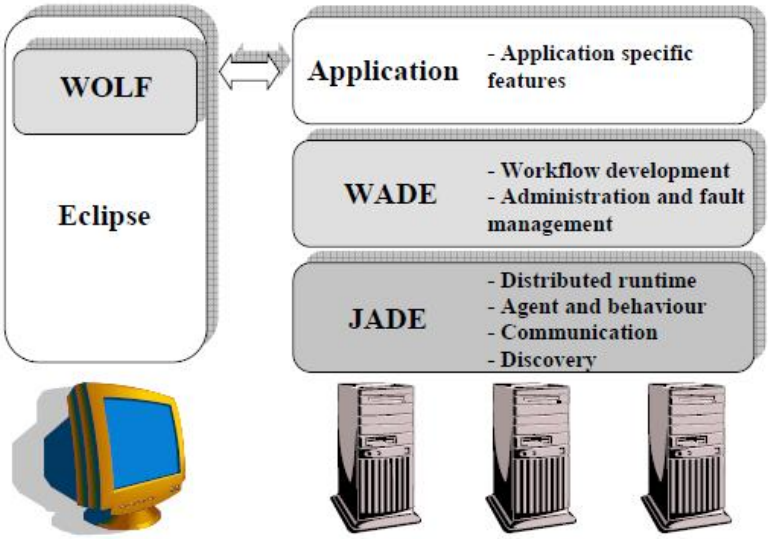


Figura 19: Plataforma WADE.

A Figura 20 apresenta a arquitetura WADE. Os componentes JADE estão representados em amarelo e os componentes específicos WADE em azul. Os componentes JADE são:

Container: um *container* é um abiente que permite a habitação de um ou vários agentes, juntamente com todos os recursos necessários para a execução desses agentes. O conjunto de todos os *containers* formam a plataforma JADE.

Main Container: Plataformas JADE obrigatoriamente devem conter o *Main Container*. Ele representa o ponto de partida para a inicialização da plataforma, ou seja, é o primeiro *container* a ser inicializado e todos os outros devem juntar-se a ele através de um processo de registro.

Os components específicos WADE são:

BootDaemon process: Há um processo *bootDaemon* para cada host na plataforma. O *bootDaemon* é responsável pelo *container* de ativação em sua máquina local.

Configuration Agent (CFA): O agente de configuração sempre é executado no *Main Container* e é responsável pela interação com os *bootDaemons* e por controlar o ciclo de vida da aplicação.

Controller Agent (CA): Existe um *Controller Agent* dentro de cada *container* da plataforma. Os *Controller Agents* são responsáveis pela supervisão das atividades no *container* local e por todos os mecanismos de tolerância a falhas implementado pelo WADE.

Engine Agents (WEA): Agentes incorporam micromotores de workflow que são capazes de executar fluxos de trabalhos.

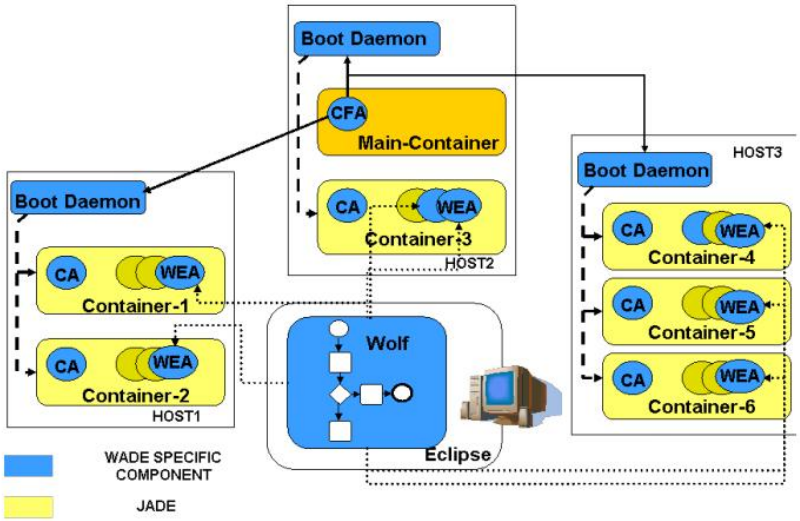


Figura 20: arquitetura WADE.

A fim de facilitar as operações de importação e exportação de workflows, WADE adota o meta-modelo de workflow derivado da linguagem XPDL (XPDL, 2013). As três principais atividades que compõem este meta-modelo são descritas a seguir.

Tool activities: consistem em invocar uma ou mais ferramentas de *software* externas, fora do processo de workflow.

Subflow activities: consistem em invocar um subfluxo de trabalho, delegando a outro agente ou outros agentes as tarefas desse subprocesso. Os agentes podem estar hospedados em *hosts* remotos. O mecanismo de delegação é implementado pela extensão do protocolo FIPA *contract-net protocol*.

Code activities: atividades especificadas através de um código

Java embutido na definição do processo de workflow. Ao contrários das *Tool* e *Subflow activities*, as *Code activities* não pertecem ao metamodelo XPDL. Essas *activities* são uma extensão WADE proprietária.

A seguir, são descritas algumas características da plataforma WADE.

- Cada agente pode incorporar um micromotor de workflow e um processo complexo pode ser realizado por um conjunto de agentes que colaboraram, cada uma executando uma parte do processo.

- Especialistas no domínio, bem como programadores, podem compreender e validar as lógicas do sistema e fluxo de trabalhos.

- Sua aplicação é adequada para execução de tarefas longas e complexas.

- É utilizado pela TI (Telecom Italia) em dois sistemas de missão crítica em Gestão da Rede, NNEM (*Network Neutral Element Manager*) e WIZARD.

- Traz a abordagem de workflows do nível de processos de negócio ao nível da lógica interna do sistema.

3.6 COMPARATIVO ENTRE FERRAMENTAS DE BPM

O primeiro passo para escolher uma ferramenta de BPM foi a definição dos pré-requisitos essenciais que uma ferramenta de BPM deve possuir para entrar na comparação. O segundo passo foi o levantamento de características desejáveis para as ferramenta BPM. O último passo consiste na comparação das ferramentas selecionadas. Os pré-requisitos essenciais para a seleção das ferramentas BPM são:

Open Source: a ferramenta precisa ser livre, ou seja, além de ser gratuita e não existir restrições para o seu uso, seu código fonte precisa ser disponibilizado e não ser vedada a distribuição de cópias.

Multi-plataforma: a ferramenta deve funcionar em vários sistemas operacionais: *Linux, Windows, MacOS*.

A partir desses pré-requisitos foram selecionadas algumas ferramentas de BPM: Jboss JBPM; Bonita Open Solution; Intalio; ProcessMaker; Activiti (Jboss). A seguir está um pequeno descritivo sobre cada uma dessas ferramentas.

3.6.1 JBOSS JBPM

JBOSS BPM é um gerenciador de processos de negócios (BPM) escrito em Java que é capaz de executar os processos descritos em BPEL ou seus próprios processos definidos na linguagem jPDL. Ele faz a ponte entre analistas de negócios e desenvolvedores. É distribuído sob a licença LGPL pela *JBoss Community* (JBPM, 2013).

Algumas tarefas do processo são executadas de forma automática, como mandar um e-mail ou invocar um EJB. Outras tarefas podem apresentar estados de espera, por exemplo, uma tarefa humana ou esperar que um cliente externo invoque um método de serviço.

3.6.2 Bonita Open Solution

O *Bonita Open Solution* (BOS) é uma solução BPM de código aberto. Criado em 2001, ele combina três soluções em uma: o *Bonita Studio*, utilizado para modelagem de processos baseados na notação BPMN 2.0; O*Bonita User Experience*, uma interface *web* (como um portal) para o usuário final interagir com os processos desenvolvidos; e o *Bonita Execution Engine* (BEE - Motor de execução do processos), que liga os processos aos sistemas externos e executa-os. BEE é bastante flexível, podendo ser adaptado a diversas infraestruturas de sistemas (BPMN, 2013).

3.6.3 Intalio BPMS

Intalio é um sistema de código aberto para Gerenciamento de Processos de Negócio que executa em diversos sistemas operacionais (*Linux, MacOS, Windows*). Sua versão gratuita possui 80% do código aberto e sua versão paga possui 100%. Sua interface gráfica utilizada para a definição de workflows é baseada na ferramenta de desenvolvimento Eclipse (ECLIPSE, 2013). Os processos de negócios definidos por meio dessa interface gráfica são baseados na linguagem BPMN. Estes processos são exportados para a linguagem WS-BPEL e são orquestrados pela máquina de execução de workflows Apache ODE (INTALIO, 2013).

3.6.4 ProcessMaker

O ProcessMaker é uma ferramenta web de BPM com suporte a design e automação de workflows. Possui versões gratuitas (Process Maker Open Source) e pagas, como SaaS (Cloud) ou locais (Enterprise). A ferramenta se distingue das demais por ser desenvolvida em PHP ao invés de Java, linguagem comum nas ferramentas de BPM (PROCESSMAKER, 2013).

3.6.5 Activiti BPM

O Activiti é um projeto de código aberto com licença Apache 2, criado pela Alfresco Software e liderado pelo criador do jBPM. Executa aplicações Java, em um servidor ou na nuvem. O Activiti é leve e facilmente integrável com aplicações existentes, além de suportar BPMN. O Activiti é acompanhado de um modelador visual (chamado Activiti Modeler) que pode ser usado para modelar diagramas sofisticados (ACTIVITI, 2013).

3.6.6 Comparação

Foram definidos alguns critérios para a avaliação das ferramentas candidatas. As seguintes propriedades ou características são desejáveis nas suítes BPM:

1. BPMN 2.0 - É preferível que a ferramenta suporte o padrão BPMN 2.0 e possua capacidade de importar e exportar para este formato.

2. *Business Rules Management* (BRM) - Possua componente BRM ou integre-se nativamente a um produto deste tipo. Um BRM permite separar as regras dos processos do código da aplicação, permitindo que usuários do workflow configurem estas regras de forma ágil e transparente.

3. *Business Activity Monitoring* (BAM) - Que possua um BAM para monitorar o fluxo dos processos.

4. Boa Documentação - É desejável que a ferramenta possua uma boa documentação. Será analisado se a documentação possui visão clara dos objetivos e propostas da ferramenta, abordando

instalação e configuração e o uso das funcionalidades da ferramenta. Também é considerado se a documentação contém tutoriais para aprendizagem da ferramenta.

5. Comunidade ativa - A ferramenta precisa ter uma comunidade ativa e, para avaliar esse aspecto, serão analisadas algumas questões como nível de atividade do fórum e se a ferramenta possui bom suporte por telefone e e-mail.
6. Java - Como o Moise é implementado em Java, é interessante que a ferramenta seja em Java também.
7. Usabilidade - A ferramenta deve ser intuitiva e fácil de usar.
8. Maturidade - O nível de maturidade da ferramenta é medido através do seu tempo de existência.

Depois de definidos os critérios considerados desejáveis para as a avaliação das ferramentas, foi atribuído uma pontuação. O simbolo (+) significa que não atende ou atende muito pouco ao critério e (+++) significa que atende totalmente ao critério.

Na Tabela 3 é possível observar a análise das ferramentas de acordo com os critérios definidos. Esse comparativo usa como base os trabalhos de (CARHUATOCTO, 2011; JUNIOR, 2011).

	Critério				
Ferramenta	Bonita	Jboss	Intalio	ProcessMarker	Activiti
BPMN 2.0	+++	+ + +	+++	+ + +	+++
BRE	++	+ + +	+	++	+
BAM	++	++	+	++	+
Documentação	+++	++	++	++	++
Suporte	+++	+ + +	+++	++	++
Usabilidade	+++	++	+++	++	++
Maturidade	+++	+ + +	++	++	++
Linguagem JAVA	+++	+ + +	+++	+	+++

Tabela 3: Quadro comparativo entre as Ferramentas de BPM.

Todas as ferramentas foram avaliadas com base nos critérios apresentados anteriormente. A ferramenta com a maior pontuação no comparativo foi a Bonita Open Solution. Algumas características interessantes dessa ferramenta é que ela possui uma comunidade bastante ativa no Brasil e possui uma boa usabilidade em comparação com as outras ferramentas apresentadas.

3.6.7 Considerações

Este capítulo apresentou conceitos importantes da área de gerenciamento de processos e sistemas de workflow. Foram analisadas algumas ferramentas de BPM e a melhor alternativa será utilizada na implementação do Modelo Conceitual. Foram escolhidas para o comparativo ferramentas de BPM que são utilizadas atualmente por grandes empresas no mercado de trabalho.

No próximo capítulo serão apresentadas três proposta para o modelo conceitual e a melhor alternativa será vista com maiores detalhes.

4 MODELO CONCEITUAL

Este capítulo apresenta três abordagens que permitem a integração de um sistema de workflow a um modelo organizacional, de forma a resolver as limitações apresentadas no capítulo 1. Essas abordagens apresentadas são definidas pelo autor e não se encontram na literatura. É realizado um breve comparativo entre as abordagens propostas, sendo que o modelo escolhido será explicado com maiores detalhes. Por fim é apresentado um exemplo completo de utilização do modelo.

4.1 PROPOSTA DE SOLUÇÃO

Para a escolha do modelo conceitual foram propostas três abordagens diferentes: o modelo híbrido, o modelo workflow com missões e o modelo workflow puro. Esta seção versa sobre diferentes abordagens para solução do problema, sem tentar descrevê-las detalhadamente.

4.1.1 Modelo Híbrido

O modelo híbrido é composto por esquemas sociais, planos, metas, missões e um sistema de workflow. Essa proposta foi concebida com o intuito de utilizar tudo que existe atualmente no Moise.

As modificações para ajustar o Moise à esse modelo consistem na adição de um novo tipo de meta chamado workflow. Esse tipo não utiliza os planos fornecidos pelo esquema social do Moise, mas sim o controle de fluxo fornecido por um sistema de worfklow.

Para cada meta global do tipo workflow existe um motor de workflow para controlar um fluxo de tarefas de um processo. A diferença é que as tarefas do processo são executadas por agentes. As tarefas do processo são alocadas de acordo com as missões dos agentes e a meta tipo workflow torna-se satisfeita quando todas as tarefas do processo são satisfeitas.

Por exemplo, na Figura 21 as metas globais *Destruir Inimigo* e *Defesa* possuem planos paralelos, a meta global *Ataque* possui um plano sequencial. As metas *Ataque* e *Defesa* são usuais do Moise, entretanto a meta *Captar Recurso* é do tipo workflow e utiliza um motor de workflow para controlar o fluxo das tarefas t9 e t10.

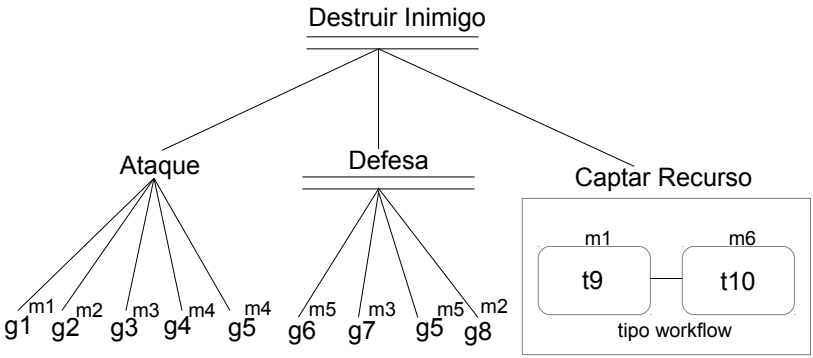


Figura 21: Modelo Híbrido.

4.1.2 Modelo workflow Com missões

O modelo workflow com missões é composto por um sistema workflow e missões do Moise. Os esquemas sociais são substituídos por processos e os papéis dos agentes continuam comprometidos com missões, o que significa que os agentes devem se comprometer com as tarefas do workflow que estão associadas às suas missões. As missões ainda permanecem com o intuito de não precisar alterar a dimensão normativa do Moise. Não existem mais planos, o fluxo das tarefas é controlado apenas pelo sistema de workflow.

Na Figura 22 é possível observar que cada tarefa possui uma missão associada. O agente que se compromete com a missão m1 deve cumprir as tarefas task1 e task3, e o agente que se compromete com a missão m2 deve cumprir as tarefas task2 e task4.

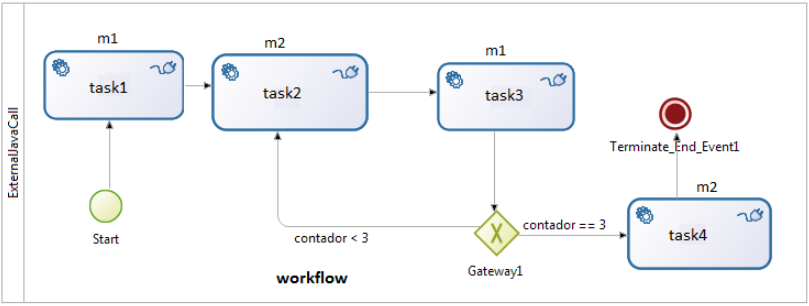


Figura 22: Modelo Workflow com Missões.

4.1.3 Modelo workflow puro

O modelo workflow puro(Figura 23) é semelhante ao modelo anterior porém não são mais utilizadas missões. Um agente se compromete com uma tarefa de acordo com o seu papel no grupo, não existindo mais a necessidade de assumir uma missão para só então assumir as tarefas dessa missão. Nesse caso, a dimensão normativa no Moise precisa ser alterada. Por exemplo, no lugar de determinar obrigações para com missões, irá determinar obrigações para com as tarefas.

Esse modelo foi concebido pensando em ser o mais parecido possível com a teoria de workflow, sendo mais fácil de ser utilizado por pessoas que já utilizam sistema de workflow. Nesse modelo os agentes se comprometem diretamente com as tarefas, através de um papel, como já acontece com os workflow, enquanto no modelo anterior um agente precisa se comprometer com missões para só então se comprometer com as tarefas dessa missão.

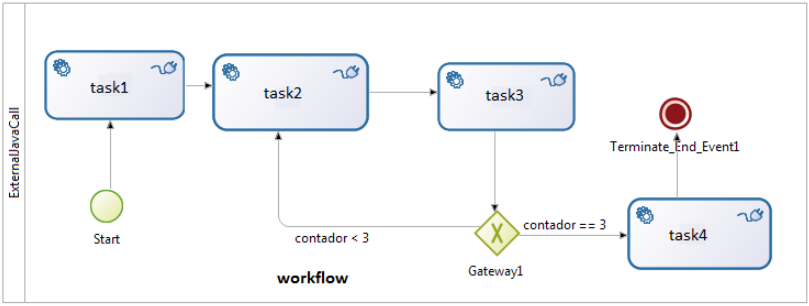


Figura 23: Modelo Workflow.

4.1.4 Comparativo entre os modelos

Foram analisadas as vantagens e desvantagens de cada um dos três modelos propostos (Tabela 4). Os critérios escolhidos para essa avalição foram: usuários Moise, usuários workflow, desempenho, encadeamento iterativo e dimensão normativa. Cada item possui uma pontuação de acordo com o seu modelo, sendo que (+) atende muito pouco ao item, (++) atende razoavelmente e (+++) atende totalmente. Todos os critérios foram definidos pelo autor e possuem os mesmo peso. A seguir são detalhados cada critério do comparativo.

	Modelo		
Critério	Híbrido	Workflow c/ missões	Workflow
Usuários Moise	+++	++	+
Usuários Workflow	+	++	+++
Novos usuários	+	++	+++
Desempenho	+	+ + +	+++
Encadeamento iterativo	++	++	+++
Dimensão normativa	+++	+ + +	+

Tabela 4: Quadro comparativo das vantagens e desvantagens apresentados pelos três modelos.

1. Usuários Moise: Diz respeito a usuários antigos que já utilizam o Moise. O modelo híbrido ganhou a maior nota nesse item, pois possui a vantagem de utilizar tudo que existe atualmente na dimensão funcional do Moise. Um usuário habituado ao Moise, não precisa aprender todos os conceitos de workflow, podendo se adaptar aos poucos, pois esses conceitos somente serão utilizados quando a meta é do tipo workflow.

2. Usuários workflow: Usuários de um sistema workflow teriam mais dificuldade para se adaptarem ao modelo híbrido, pois precisariam aprender conceitos de esquema social. O modelo workflow com missões ganhou uma nota inferior ao modelo workflow puro por utilizar missões, o que poderia confundir pessoas que utilizam sistema de worflow.

3. Novos usuários: São usuários que nunca utilizaram o Moise e não possuem conhecimento sobre sistemas de workflow. O modelo híbrido leva desvantagem nesse quesito em relação aos demais, pois usuários levariam mais tempo aprendendo conceitos (tanto de workflows, quanto do Moise) para utilizar todos os recursos do modelo hibrido.

4. Desempenho: Em relação ao desempenho o modelo híbrido ganhou a menor nota nesse quesito por possuir a necessidade de utilizar varias instâncias do motor de workflow, uma para cada meta do tipo workflow, causando problemas de desempenho e escalabilidade. Nos modelos workflow com missões e workflow é necessário apenas um motor de workflow para a dimensão funcional do Moise.

5. Encadeamento iterativo: Os modelos híbrido e workflow com missões possuem dificuldades de implementação de um encadeamento iterativo, onde uma determinada meta ou um grupo de metas podem ser alcançados várias vezes. Isso ocorre por causa do uso de missões, pois quando um agente cumpre todas as metas de sua missão, ele deixa a missão, não se comprometendo mais com as metas dessa missão. Assim, não é possível ao agente cumprir novamente uma meta dessa missão, já que ele abandonou a mesma. Uma forma de resolver isto é modificando a dimensão normativa, fazendo com que um agente apenas deixe a missão depois que o esquema social foi finalizado. O modelo workflow puro não possui esse problema de implementação, onde as tarefas são assumidas diretamente pelos agentes e eles nunca deixam uma tarefa.

6. Dimensão normativa: Como o modelo workflow puro não utiliza missões, a dimensão normativa precisa ser reimplementada no modelo. O modelo híbrido e workflow com missões utilizam missões, não necessitando grandes modificações na dimensão normativa.

Apesar dos modelos workflow com missões e workflow empatarem na comparação, foi escolhido o modelo workflow com missões para utilização nesse trabalho. Pela Tabela 4 é possível observar que o modelo workflow com missões é o único que atende razoavelmente ou totalmente aos seis itens do comparativo. Os quesitos considerados mais importantes e que definiram essa escolha foram: usuários Moise e dimensão normativa. Usuários Moise, para ficar mais de acordo com os conceitos originais do Moise. Dimensão normativa, por facilitar a próxima etapa desse projeto que é o desenvolvimento da proposta, nesse caso necessitando pequenas modificações na dimensão normativa.

4.2 AGREGAÇÃO DO WORKFLOW AO MOISE

Esta seção descreve o modelo conceitual escolhido para incorporação de um sistema de workflow ao modelo organizacional Moise. O modelo em, uma visão geral (Figura 24) é dividido em três componentes: a organização, o ambiente e os agentes.

O componente organização engloba o motor de workflow e o estado organizacional. O estado organizacional é um elemento importante na integração de um sistema de workflow ao Moise, já que nele estão

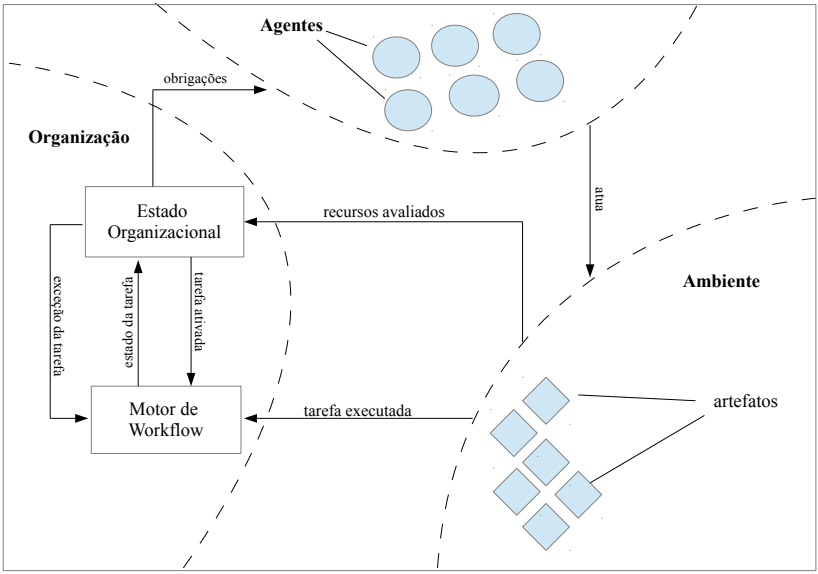


Figura 24: Visão geral da proposta dividida em três componentes: organização, ambiente e agentes.

contidas a especificação organizacional, a entidade organizacional, os fatos, regras e normas que os agentes têm obrigação ou permissão de se comprometerem. O estado organizacional avisa das possíveis exceções geradas por uma tarefa. Três tipos de exceções são possíveis: exceção de tempo, exceção de recursos e exceção de remoção. Uma exceção de tempo ocorre quando o tempo determinado para executar uma tarefa foi ultrapassado, uma exceção de recursos acontece quando não há recursos suficientes para a execução de uma tarefa e uma exceção de remoção ocorre quando uma tarefa é removida por um agente.

O ambiente segue a proposta de Agents & Artifacts (OMICINI; RICCI; VIROLI, 2008), onde o ambiente é formado por vários artefatos, que podem ser objetos do ambiente, como por exemplo, recursos necessários para a execução de uma tarefa. O ambiente informa ao sistema de workflow quando uma tarefa foi executada por um agente e informa ao estado organizacional os recursos disponíveis atualmente.

Os agentes atuam sobre o ambiente e sobre a organização. São responsáveis por cumprir as obrigações designadas pela organização, tais como: participar de um determinado grupo, assumir um papel na organização ou executar uma tarefa. Um agente pode receber e dar

ordem para outros agentes, pode monitorar o fluxo das tarefas através de métricas de processo fornecidas pelo workflow ou pelo estado organizacional, pode raciocinar sobre processos, verificar gargalos, atrasos, delegar tarefas, criar novas tarefas e cancelar ou remover tarefas. Caso uma tarefa não possua nenhum papel associado na organização, ela pode ser executada por um agente externo ou por um humano.

A Figura 25 apresenta o modelo conceitual de forma detalhada. O componente motor de workflow é constituído por processos e pelo estado das tarefas. Em um processo, existe um conjunto de tarefas interligadas de forma sequencial ou paralela, visando alcançar um objetivo comum da organização. O componente estado das tarefas armazena as exceções geradas pelo estado organizacional, recebe as tarefas executadas pelos agentes e informa aos processos e ao estado organizacional as mudanças nos estados das tarefas.

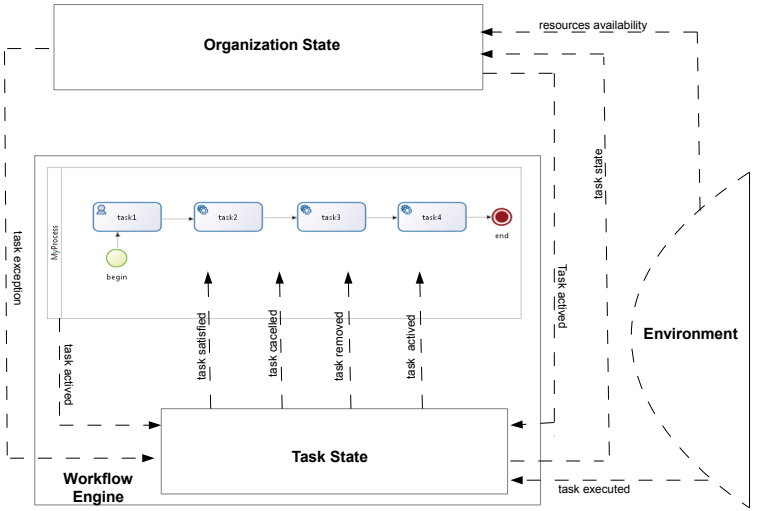


Figura 25: Modelo Conceitual.

Os estados possíveis de uma tarefa (Figura 26) são: inativa, ativa, satisfeita, cancelada ou removida. Todas as tarefas de um processo iniciam-se no estado inativa, caso os pré-requisitos para tornar uma tarefa habilitada sejam atingidos, a tarefa se torna ativa. Quando uma tarefa esta no estado ativa, significa que ela está pronta para ser executada por um agente, entretanto uma tarefa ativa pode ser can-

celada ou removida do processo. Para uma tarefa ativa tornar-se satisfeita, sua cardinalidade precisa ser alcançada, ou seja, uma tarefa precisa ser executada um número minímo de vezes para se tornar satisfeita.

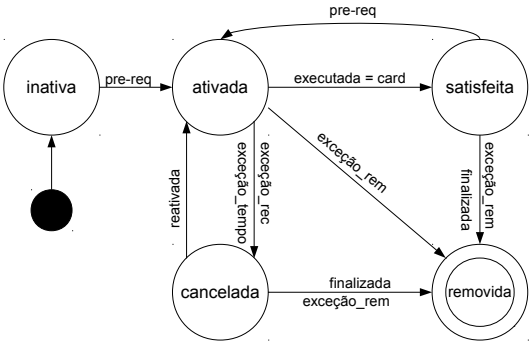


Figura 26: Estados possíveis de uma tarefa.

Uma tarefa ativa pode ser cancelada por um agente de monitoramento ou caso ocorra uma exceção de recurso ou uma exceção de tempo. Caso ocorra uma exceção de recurso, o fluxo do processo pode tomar um caminho alternativo, onde tarefas para captação de recursos serão ativadas. Uma vez que os recursos necessários para a execução da tarefa foram captados, a tarefa cancelada será reativada. Caso uma tarefa seja cancelada pelo agente de monitoramento ou por uma exceção de tempo, o fluxo do processo é interrompido até que um agente de monitoramento decida reativar a tarefa.

Uma tarefa satisfeita pode ser ativada novamente caso os seus pré-requisitos de habilitação sejam atingidos. Se uma exceção de remoção acontecer, a tarefa é excluída do processo e, ao contrário de uma tarefa cancelada, uma tarefa removida não pode mais se tornar ativa. Quando o processo termina, todas as tarefas são finalizadas.

Através das obrigações geradas pelo estado organizacional os agentes ficam cientes do momento no qual devem executar suas tarefas. Uma vez que a tarefa é executada, a obrigação do agente é cumprida. O modelo proposto não utiliza planos. O fluxo de tarefas é controlado por um motor de workflow, o que permite a criação de fluxos complexos, não possíveis na versão original do Moise.

4.3 EXEMPLO DE UM PROCESSO DE FABRICAÇÃO DE QUEIJO

Esta seção apresenta um exemplo completo da utilização do modelo escolhido. É mostrado a especificação estrutural, funcional e normativa do Moise. A especificação funcional, por ser foco deste trabalho, é apresentada em maiores detalhes, pois nela ocorrem mudanças na linguagem OML.

Na especificação estrutural (Figura 27) foi definido um grupo laticínio que é composto por três papéis: o fazendeiro, o entregador e o operário. Os papéis fazendeiro e operário possuem cardinalidade de 1 a 5, ou seja, no mínimo 1 e no máximo 5 agentes podem assumir esses papéis no grupo laticínio. Já o papel entregador possui a cardinalidade de 1 a 3. Os papéis fazendeiro e operário possuem autoridade sobre o papel entregador, essa autoridade é utilizada por exemplo, para solicitar que seja realizada alguma entrega do queijo.

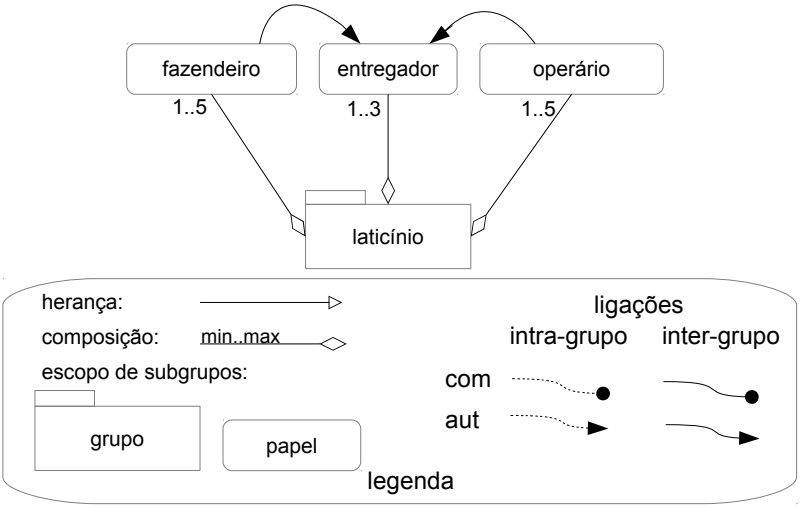


Figura 27: Especificação Estrutural.

A Figura 28 apresenta um processo de fabricação de queijo na especificação funcional. Neste processo, um laticínio precisa entregar uma encomenda de 1000 peças de queijo para um supermercado, para cumprir esse objetivo são utilizados três papéis como descrito na especificação estrutural: o papel fazendeiro, o papel entregador e o papel operário.

As tarefas são assumidas pelos agentes no momento em que estes

se comprometem com uma missão no processo. O agente que assumir o papel fazendeiro é obrigado a executar as tarefas da missão m1, que é tirar leite da vaca. O papel entregador é responsável pelas tarefas da missão m2 que são: entregar leite no laticínio e entregar a encomenda no supermercado. O fazendeiro precisa executar a tarefa da missão m3 de fazer o queijo.

O fluxo do processo inicia-se com a tarefa tirar leite da vaca, em seguida o leite é entregue ao laticínio para que produção do queijo possa ser iniciada. O queijo é produzido até o leite acabar ou até que a meta de 1000 peças de queijos seja alcançada. Caso o leite termine o fluxo do processo é reiniciado, voltando para a tarefa de tirar o leite da vaca.

Esse fluxo de processo não seria possível na dimensão funcional do Moise sem integração do workflow, pois os planos do Moise não suportam encadeamento condicional e nem o encadeamento iterativo. Sem encadeamento iterativo, uma meta satisfeita não ficaria mais ativa para ser satisfeita novamente, isso impediria a fabricação de mil peças de queijo. Sem encadeamento condicional, o fluxo das tarefas não conseguiria se decidir qual caminho tomar depois que a tarefa fazer queijo foi executada, ele ativaria ao mesmo tempo as tarefas entregar encomenda, tirar leite da vaca e fazer queijo.

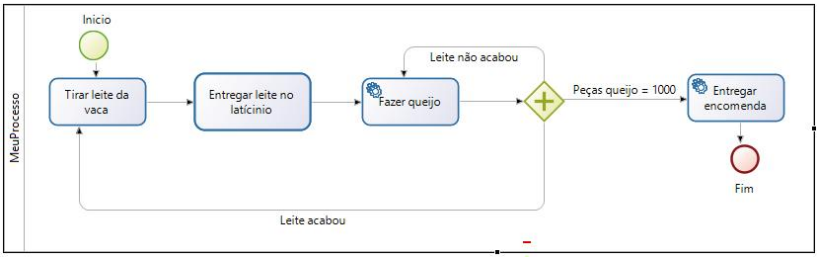


Figura 28: Especificação Funcional.

A relação entre a especificação estrutural e especificação funcional é feita pela especificação normativa. Na especificação normativa são descritas as missões com as quais um papel tem permissão ou obrigação de se comprometer. Na Tabela 5 é possível observar que o papel fazendeiro está obrigado a executar as tarefas da missão m1, o entregador tem obrigação de executar as tarefas da missão m2, e o operário tem obrigação de executar a tarefa da missão m3.

A Tabela 6 apresenta de forma sequencial, 18 passos do estado das tarefas do processo de fabricação de queijo. O estado das tarefas

Papel	Relação Deôntica	Missão
fazendeiro	obrigação	m1
entregador	obrigação	m2
operário	obrigação	m3

Tabela 5: Exemplo da dimensão normativa.

são: inativa, ativa, satisfeita, cancelada e removida. O objetivo desse exemplo é simular o estado das tarefas em tempo execução.

As quatros tarefas do processo de fabricação de queijo são task1 (tirar leite da vaca), task2 (entregar leite no laticínio), task3 (fazer queijo) e task4 (entregar encomenda).

No início da execução do processo, todas as tarefas iniciam no estado inativa, no passo 2 a tarefa1 é ativada e no passo 3 executada, enquanto as outras tarefas continuam no estado inativa. Logo após a tarefa1 se tornar satisfeita, a tarefa2 é ativada e em seguida executada e satisfeita. Um encadeamento em *loop* acontece no passo 8, quando a tarefa3 é satisfeita a tarefa1 volta a se tonar ativa(passo 9). Nota-se que o fluxo do processo apresentado funciona de forma sequencial, onde apenas uma tarefa pode se tornar ativa, caso o processo apresentado tivesse fluxo de tarefas em paralelo, duas ou mais tarefas tornariam-se ativa ao mesmo tempo. No passo 17 todas as tarefas estão no estado satisfeita. No passo 18, o processo terminou e todas as tarefas passam para o estado removida.

4.4 NOPL/W

Esta seção descreve a linguagem *NOPL/W* (em inglês, textit-Normative Organisation Programming Language for Workflow), uma extensão de NOPL (apresentada no capítulo 2). Para a elaboração da linguagem NOPL/W foi necessário enriquece-la com novos fatos, regras e normas necessárias para dar suporte à agregação do workflow ao Moise. A ideia principal é que uma OS em OML é traduzida de forma automática para diferentes programas em NOPL/W, tais programas definem o gerenciamento de normas para grupos e processos. Fatos dinâmicos são também introduzidos em NOPL/W para representar o estado corrente de uma organização. A seguir são apresentados os fatos, regras e normas que compõem um processo. Os que estão marcados com (*) já existiam em NOPL e são apresentados aqui para facilitar a

Passo	Tarefa1	Tarefa2	Tarefa3	Tarefa4
1	inativa	inativa	inativa	inativa
2	ativada	inativa	inativa	inativa
3	executada	inativa	inativa	inativa
4	satisfeita	ativada	inativa	inativa
5	satisfeita	executada	inativa	inativa
6	satisfeita	satisfeita	ativada	inativa
7	satisfeita	satisfeita	executada	inativa
8	satisfeita	satisfeita	satisfeita	inativa
9	ativada	satisfeita	satisfeita	inativa
10	executada	satisfeita	satisfeita	inativa
11	satisfeita	ativada	satisfeita	inativa
12	satisfeita	executada	satisfeita	inativa
13	satisfeita	satisfeita	ativada	inativa
14	satisfeita	satisfeita	executada	inativa
15	satisfeita	satisfeita	satisfeita	ativada
16	satisfeita	satisfeita	satisfeita	executada
17	satisfeita	satisfeita	satisfeita	satisfeita
18	removida	removida	removida	removida

Tabela 6: Estado das tarefas em tempo de execução.

compreensão da proposta.

4.4.1 Fatos

Para programas normativos de processos, os seguintes fatos são produzidos pela tradução de uma OS:

- `task(`m`,`t`,`$'ttf'$`,`idr`,`rt`):` é um fato que define os argumentos para uma tarefa: m é o identificador da missão ligado a tarefa, t é o identificador da tarefa, $'ttf'$ é o tempo que a tarefa tem para ser executada, idr é o identificador do recurso, rt são a quantidade de recursos necessários para a execução da tarefa.

- `mission_role(`m`,`p`)*:` o papel p tem permissão ou obrigação de se comprometer com a missão m.

- `mission_cardinality(`m`,`min`,`max`)*:` é um fato que define a cardinalidade de um missão, ou seja, são o número mínimo e

máximo de agentes que podem se comprometer com a missão.

4.4.2 Fatos Dinâmicos

Vários novos fatos foram adicionados a linguagem NOPL, por causa da adição de novos estados para as tarefas, esses estados podem ser alterado por um agente de monitoração. Outros fatos adicionados dizem respeito À utilização de recursos para execução de uma tarefa.

- `play(`a`,`ρ`,`gr`)*:` agente a possui o papel ρ na instância do grupo identificado como gr.
- `committed(`a`,`m`,`p`)*:` agente a assumiu a missão m no processo p.
- `resources(`idr`,`re`):` o recurso idr possui a quantidade re de recursos no ambiente.
- `responsible(`gr`,`p`)*:` o grupo instanciado gr é responsável pelas missões do processo instanciado p.
- `activated(`s`,`t`):` a tarefa t no processo p foi ativada.
- `executed(`t`,`p`,`a`):` a tarefa t no processo p foi executada pelo agente a.
- `satisfied(`s`,`t`):` a tarefa t no processo s foi satisfeita.
- `removed(`t`,`p`,`a`):` a tarefa t no processo p foi removida pelo agente a.
- `cancelled(`t`,`p`):` a tarefa t no processo p foi cancelada.
- `is_finished(`p`):` o processo p foi finalizado.

4.4.3 Regras

Além de fatos, foram definidas novas regras que são uteis para inferir estados a um processo. De acordo com a regra mission_acomplished(S, M), uma missão é considerada cumprida somente quando o processo é finalizado, diferente da linguagem NOPL, onde uma missão era considerada cumprida logo que todas as metas da missão fossem satisfeitas.

- `mission_accomplished(S,M):- is_finished(p).`

  Na regra `well_formed(`T`)`, um processo está bem formado quando a cardinalidade mínima de cada missão está satisfeita,

- `well_formed(T)*:- mplayers(m,S,Vm) & Vm >= minmpS(m) & Vm <= maxmpS(m)`

  A regra mplayers informa a quantidade de agentes que se comprometeram com a tarefa T.

- `mplayers(T,P,mP)*:-.count( committed(_,M,S), V).`

As regras assim como os fatos dinâmicos foram bastantes modificados, algumas regras não foram mais necessárias e por isso foram excluídas na linguagem NOPL/W.

4.4.4 Normas

Para a classe de normas de um processo, apenas a classe normas para meta precisou ser alterada. A classe de normas de propriedades e normas de domínio não serão mostradas pois não foram necessárias modificações neste tipo de normas no modelo conceitual proposto. A norma ngoal passou a se chamar ntask e é apresetanda a seguir.

```
norm ntask:
     committed(A,M,P) & task(M,T,D,IDR,RT) &
     well_formed(P)  &  actived(S,T)
  -> obligation(A, ntask, executed(A,P,T),‘now‘ + D).
```

Está norma é incluída em cada programa NOPL/W gerado para um processo. Ela pode ser lida como “quando um agente A está comprometido com uma tarefa T, o processo está bem formado e a tarefa está ativa, então o agente A é obrigado a executar a tarefa T dentro do prazo D”.

4.5 MUDANÇAS NA OML

Foram necessárias mudanças na OML da especificação funcional do Moise. Na especificação estrutural e normativa não ocorreram mudanças. Utilizando novamente o exemplo do laticínio é apresentada a seguir a especificação funcional em arquivo XML.

```
<functional-specification>
   <process id="laticinio">
      <task>
         <task id="t1" min="1" ds="tirar leite da vaca" />
         <task id="t2" min="1" ds="entregar leite latícinio" />
         <task id="t3" min="1" ds="fazer queijo" />
         <task id="t4" min="1" ds="entregar encomenda" />
       </ task >
      <mission id="m1" min="0" max="5">
         <task id="t1" />
      </mission>
      <mission id="m2" min="0" max="5">
         <task id="t2" />
         <task id="t3" />
      </mission>
      <mission id="m3" min="0" max="5">
         <task id="t4" />
      </mission>
   </process>
</functional-specification>
```

Na especificação funcional em XML acima é definido um processo laticínio, esse processo possui quatro tarefas (t1, t2, t3 e t4) e três missões (m1, m2 e m3). A tarefa t1 pertence a missão m1, as tarefas t3, t2 e t3 pertencem a missão m2 e a tarefa t4 pertence a missão m3. Cada missão possui cardinalidade 0 a 5. Foram excluídos os campos *scheme* e *goal* e adicionados no lugar os campos *process* e *task*. O campo *plan operator* foi excluído, dando lugar ao controle de fluxo por um motor de workflow. No modelo proposto a linguagem OML e notação BPMN formam a especificação funcional do Moise.

4.6 MODIFICAÇÕES NA SEMÂNTICA DO MOISE

Foram realizadas algumas modificaçõess na OS e OE do Moise. Essas modificações foram necessárias para suportar o modelo escolhido e são explicadas em detalhes a seguir.

4.6.1 Especificação Organizacional

Uma OS continua sendo composta por três especificações: estrutural, funcional e normativa:

$$<id, SS, FS, NS>$$

onde:

- *id* é um identificador único para uma *OS*;
- *SS* é o conjunto das especificações estruturais;
- *FS* é o conjunto das especificações funcionais;
- *NS* é o conjunto das normas.

Foram necessárias modificações na FS do Moise. A EE e NS não serão mostradas pois não necessitaram de alterações.

4.6.2 Especificação Funcional

Uma FS é definida pelos elementos da seguinte tupla:

$$<M,\ T,\ P>$$

onde:

- M é um conjunto de identificadores de missões da organização;
- T é um conjunto de identificadores das tarefas da organização;
- P é o conjunto de especificações do processo de uma organização, utilizando um sistema de workflow para o controle do fluxo das tarefas.

Uma especificação do processo (P) é definida pelos elementos seguintes da tupla:

$$<id,\ maxmp,\ minmp>$$

onde:

- *id* é um identificador único para o processo;
- *maxmp*: $M \rightarrow \mathbb{Z}$: é uma função que mapeia cada missão para o número máximo de compromissos existentes dessa missão no processo (cardinalidade máxima da missão);
- *minmp*: $M \rightarrow \mathbb{Z}$: é uma função que mapeia cada missão para o número mínimo de compromissos para o processo ser considerado bem formado (cardinalidade mínima da missão);

Uma tarefa é definida por uma tupla com os seguintes elementos:

$$\texttt{<}id\texttt{, }tm\texttt{, }card\texttt{, }ttf\texttt{, }idr\texttt{, }rt\texttt{>}$$

onde:

- *id* é um identificador único para a tarefa;
- *tm*: 2^M é o conjunto de missões que incluem a tarefa;
- *card*: $\mathbb{Z}$ é a cardinalidade da tarefa (quantidade de vezes que a tarefa precisa ser executada para ser considerada satisfeita);
- *ttf*: $\mathbb{Z}$ é o tempo que a tarefa tem para ser cumprida;
- *idr*: é o identificador de um recurso da tarefa; e
- *rt*: é a quantidade de recursos necessários para a execução da tarefa.

4.6.3 Entidade Organizacional

A OE passa a utilizar instâncias de processos no lugar de instâncias de esquemas, sendo definida a seguir por uma tupla com os seguintes elementos:

$$\texttt{<}OS\texttt{, }A\texttt{, }GI\texttt{, }PI\texttt{>}$$

onde:

- *OS* é um especificação organizacional da *OE*;
- *A* é um conjunto de agentes;
- *GI* é um conjuntos de Instâncias de Grupos criados pelo *OE*;
- *PI* é um conjunto de Instâncias de Processo PI criados pela *OE*.

Uma Instâncias de Processo (*PI*) é definida por uma tupla com os seguintes elementos:

$$\texttt{<}id\texttt{, }P\texttt{, }commitments\texttt{, }achievements\texttt{>}$$

onde:

- *id* é um identificador único da instância do processo;
- *P* é a especificação do Processo;

- *commitments*: $M \rightarrow 2^A$ é uma função que mapeia cada missão numa instância do processo ao conjunto de agentes que estão comprometidos com a missão;
- *achievements*: $A \rightarrow 2^T$ é uma função que mapeia cada agente para o conjunto de tarefas executas;
- *end*: identifica se o processo terminou.

4.7 CONSIDERAÇÕES

Este capítulo apresentou o modelo conceitual que possui o objetivo de resolver algumas limitações apresentadas na dimensão funcional de alguns modelos organizacionais. Foi apresentada também a linguagem NOPL/W, uma extensão da linguagem NOPL. Por fim, foi ilustrado um exemplo da utilização do modelo conceitual.

No próximo capítulo serão introduzidas a arquitetura de implementação e a validação da arquitetura e do mocelo conceitual.

5 ARQUITETURA

Este capítulo apresenta uma proposta de arquitetura que viabiliza o funcionamento do modelo sugerido para a integração de um sistema de workflow a um modelo organizacional de sistemas multiagentes. Esta é uma arquitetura possível, sendo que outras arquiteturas podem ser propostas com base no modelo conceitual apresentado no capítulo 4.

Inicialmente são apresentados os requisitos principais, a arquitetura de implementação e sua especificação. Em seguida são apresentadas as ferramentas utilizadas na implementação e as etapas para construção do modelo proposto. Por fim na seção de validação, é ilustrado um exemplo completo de utilização do modelo, mostrando um comparativo entre padrões de workflow e apresentandas as vantagens encontradas.

5.1 REQUISITOS PRINCIPAIS

A arquitetura desenvolvida deve atender aos seguintes requisitos:

- A arquitetura deve ser genérica o suficiente para que possa ser integrada com diversos sistemas de workflow;
- A arquitetura deve incluir uma ferramenta que permita a tradução da linguagem OML para NOPL/W;
- A arquitetura deve disponibilizar meios para que os agentes possam executar operações sobre o artefato de processo, tais como: comprometer-se com uma tarefa ou executar uma tarefa;
- Devem ser disponibilizadas as propriedades observáveis da especificação organizacional. Por exemplo, o estado das tarefas, o estado das obrigações e os agentes comprometidos com as missões;
- Deve ser utilizada a notação BPMN 2.0 na especificação funcional do Moise; O modelo deve executar sobre os sistemas operacionais *Windows*, *Linux* e *MacOS*;
- A arquitetura deve ser implementada na linguagem *Java* (versão 6 ou 7).

5.2 ARQUITETURA DE IMPLEMENTAÇÃO

A arquitetura atual do Moise é baseada no *ORA4MAS*, seguindo a ideia de *Agents & Artifacts* (OMICINI; RICCI; VIROLI, 2008; HüBNER; VERCOUTER; BOISSIER, 2010). No *ORA4MAS*, são fornecidas as propriedades observáveis e operações com o intuito de permitir que os agentes interajam com uma organização. Cada artefato é definido a partir de uma OS. A seguir é apresentada uma comparação entre a arquitetura atual do Moise e a arquitetura proposta nesta dissertação.

Na Figura 29 é apresentada a arquitetura atual do Moise. Ela possui 4 componentes: a OS, o *OML2NOPL Translator*, o *Scheme State* e o *NPL Interpreter*.

Numa OS (definida na seção 2.3.1.) é definida a especificação organizacional, que contém uma especificação estrutural, funcional e a normativa. O componente *OML2NOPL Translator* é responsável pela tradução de uma especificação em OML para a linguagem NOPL. Em *Scheme State* estão contidos os fatos dinâmicos que representam o estado atual da organização, assim como os objetivos *scheme* utilizados pelo *NPL Engine*. Por fim, o *NPL Interpreter* é composto por outros três componentes: *NOPL Program*, *NPL Engine* e *Obligations State*.

NOPL Program (apresentado na seção 3.6.7) define o gerenciamento de normas para grupos e processos. *NPL Engine* interpreta programas escritos em NOPL e, a partir de fatos dinâmicos, gerencia o estado atual das normas com as quais os agentes têm obrigação ou permissão de se comprometer, o estado atual das normas é guardadisão em *Obligation State*.

A arquitetura proposta pode ser observada na Figura 30. As principais modificações incluem alterações na OS, a adição do componente *Workflow Engine* e a mudança para utilização da linguagem NOPL/W em vez de NOPL.

As modificações na OS consistem na adição de uma notação BPMN, que assim como a OML, também é definida em XML. Desta forma uma OS passa a ser composta por uma notação BPMN e pela OML. É importante destacar que essas mudanças na OS ocorrem apenas na especificação funcional do Moise. Nas especificações estrutural e normativa a entrada de dados não possui a notação BPMN, sendo utilizada apenas a OML padrão.

O *Workflow Engine* é responsável por controlar um fluxo de tarefas a partir de uma notação BPMN e também modificar o estado das tarefas contidas no componente *Process State*.

O componente *Scheme State* passa a se chamar *Process State*.

Essa mudança de nome ocorreu para ficar de acordo com o que o componente se propõe, que é guardar os fatos dinâmicos de um *Process Artifact*, anteriormente chamado de *Scheme Artifact*. Outro ponto a ser considerado na modificação desse componente é a adição de novos estados para as tarefas. Esses estados são: *inactived*, *activated*, *executed*, *satisfied*, *cancelled* e *removed*.

Os estados de uma tarefa contidos no *Process State* foram discutidos na seção 4.2 e pode ser considerado o item mais importante utilizado na integração do sistema de workflow ao modelo organizacional Moise. Ele faz a ponte de comunicação entre um sistema de workflow e o Moise, uma vez que, com a adição da linguagem NOPL/W, quem passa a controlar o estado das tarefas é *workflow Engine*.

Outra modificação feita diz respeito ao componente de tradução, que passa a traduzir da linguagem OML para a linguagem NOPL/W e não para a linguagem NOPL como ocorre na arquitetura atual do Moise. O componente *NOPL Program* passa a se chamar *NOPL/W Program*, essa mudança de nome ocorreu porque na arquitetura atual é utilizado a linguagem NOPL/W (discutida na seção 4.4) no lugar da NOPL. Os componentes *NPL Engine* e *Obligations State* não sofreram modificações significativas. A única diferença se encontra no fato de que os dados enviados para o *NPL Engine* agora são escritos em NOPL/W e os fatos dinâmicos são fornecidos pelo *Process State* (não mais pelo *Scheme State*).

O funcionamento da arquitetura acontece da seguinte maneira: como ponto de partida é utilizado uma OS como entrada de dados. Uma OS contém uma notação BPMN e uma OML. A notação BPMN é utilizada pelo componente *Workflow Engine* para controlar o fluxo das tarefas de um processo. A linguagem OML é traduzida para programas NOPL/W através de uma ferramenta de tradução.

Workflow Engine é responsável pela alteração dos estados de uma tarefa, por exemplo, tornar uma tarefa ativa. Os estados das tarefas contidos no componente *Process State* e os *NOPL/W Programs* são a entrada do componente *NPL Engine*, que por sua vez gera as obrigações que os agentes devem satisfazer. Como exemplo de obrigações podemos mencionar a execução de uma tarefa ou o comprometimento com uma missão.

Para auxiliar os agentes no cumprimento de suas obrigações, o *Process Artifact* (Figura 31) disponibiliza propriedades observáveis e operações. Utilizando as propriedades observáveis, os agentes são capazes de raciocinar sobre a especificação organizacional e então executar as operações. As propriedades observáveis de um *Process Artifact* são:

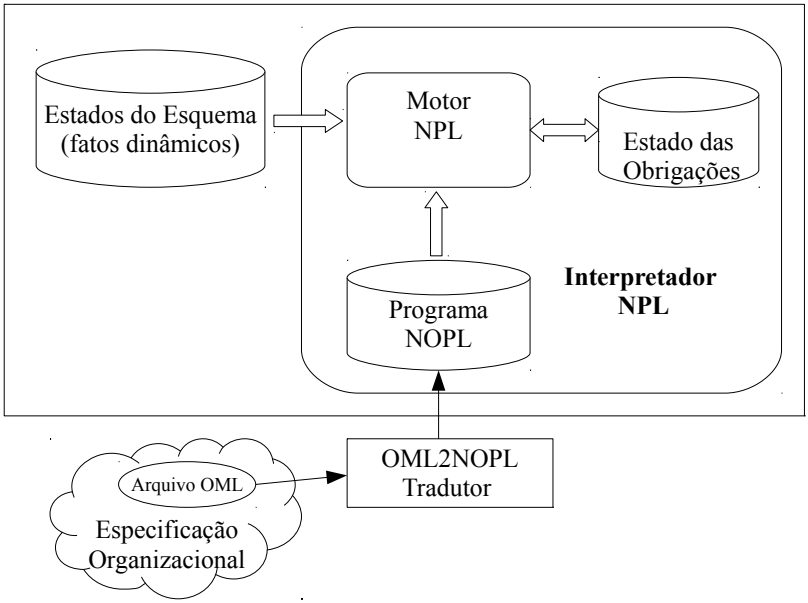


Figura 29: Arquitetura atual do Moise.

- ***PlayableMissions*:** Armazena todas as missões com as quais os agentes ainda precisam se comprometer.
- ***PlayersOfMissions*:** Apresenta todos os agentes comprometidos com o processo e com suas correspondentes missões.
- ***TasksState*:** Armeza o estado corrente das tarefas do processo. Os estados possíveis são: *inactived*, *activated*, *satisfied*, *cancelled* e *removed*.
- **ProcessSpecification:** Contém a especificação do processo na OS.
- **Obligations State:** Onde estão as obrigações ativas dos agentes.

As operações disponibilizadas para os agentes pelo *Process Artifact* são:

- **CommitMission:** Usado por um agente para se comprometer com uma missão;
- **TaskExecuted:** Esta operação é usada para definir o estado de uma tarefa como executada.

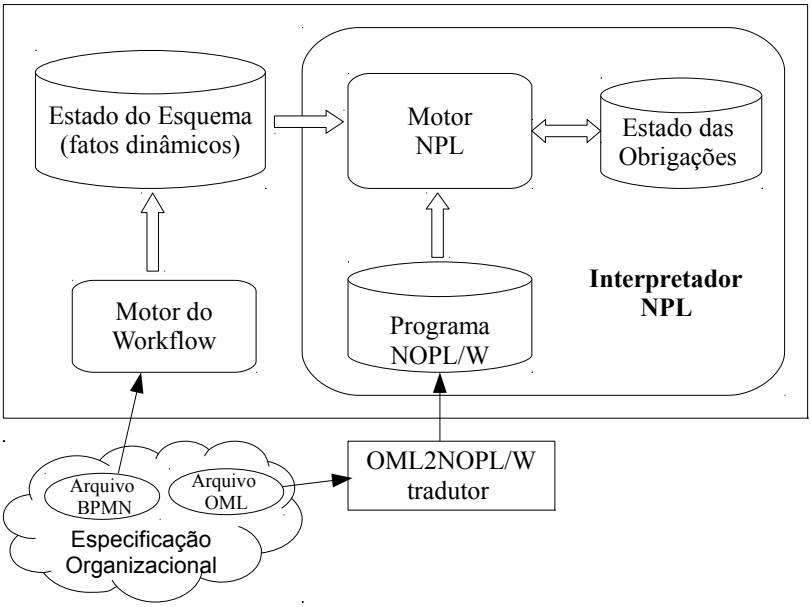


Figura 30: Arquitetura de implementação para o modelo proposto.

- **TaskCancelled:** Esta operação é usada para definir o estado de uma tarefa como cancelada.
- **TaskRemoved:** Modifica o estado de uma tarefa para removida.
- **TaskReactivated:** Utilizada para reativar uma tarefa que foi cancelada.

No modelo conceitual proposto, o ambiente é responsável por avisar à organização sobre as tarefas que foram executadas pelos agentes. Em (MAIQUEL, 2012) é discutido como isso pode ser realizado. Porém, por questão de tempo isso não será implementado na arquitetura proposta, passando essa responsabilidade do ambiente para os agentes.

5.3 CENÁRIO PARA GERAÇÃO DE UMA OBRIGAÇÃO

Para ilustrar o ciclo de vida de uma tarefa, desde a sua ativação, passando pela execução, até que a tarefa se torne satisfeita, foi utilizado um diagrama de sequências (Figura 32). Esse cenário possui o objetivo

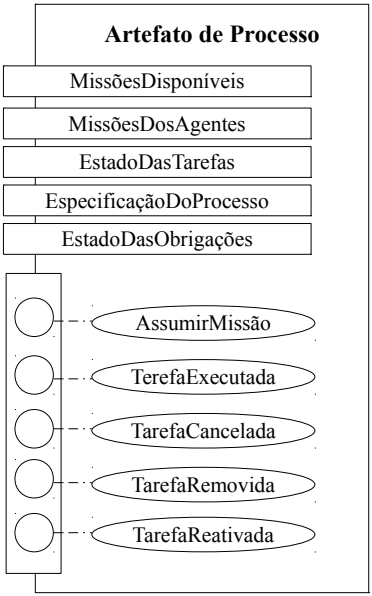


Figura 31: Artefato de Processo.

de facilitar a compreensão das entidades que compõem um artefato de processo e quais são as ações que essas entidades executam.

As entidades envolvidas no cenário são: *Workflow Engine*, responsável pela ativação de uma tarefa; *Process State*, onde estão armazenados os estados das tarefas; A entidade *NPL Engine* é responsável por gerenciar as obrigações e também os agentes, que são responsáveis por cumprir essas obrigações.

O diagrama de sequência inicia com a mensagem, *TaskActived(task1)*, passada da entidade *Workflow Engine* para a entidade Process State e depois encaminhada para a entidade *NPL Engine*. O objetivo dessa mensagem é informar que a tarefa *task1* está ativada e pronta para ser executada. A entidade *NPL Engine*, a partir de sua base de fatos, regras e normas constata que uma obrigação deve ser gerada para o *Agent1* e envia uma mensagem para esse agente avisando que ele possui a obrigação de se comprometer com a execução da tarefa *task1*. Uma vez comprometido com a obrigação, o *Agent1* possui um tempo máximo para cumpri-la. Assim que a tarefa é executada pelo *agent1*, ele encaminha uma mensagem para a entidade *NPL Engine* avisando que a tarefa foi executada.

A entidade *NPL Engine* verifica se a cardinalidade mínima de

execução da tarefa task1 foi alcançada e, caso tenha sido, a norma que gerou a obrigação muda seu estado para cumprida e uma mensagem avisando que a tarefa task1 foi satisfeita é enviada para as entidades Process State e Workflow Engine.

Por fim, quando Workflow Engine toma ciência que a tarefa *task1* foi satisfeita, é dada continuidade ao fluxo do processo e a tarefa *task2* é ativada, reiniciando o diagrama de sequência. A diferença é que agora é gerada uma obrigação para execução da tarefa *task2* e não para *task1*, uma vez que a próxima tarefa do fluxo do processo será ativada pela entidade *Workflow Engine*.

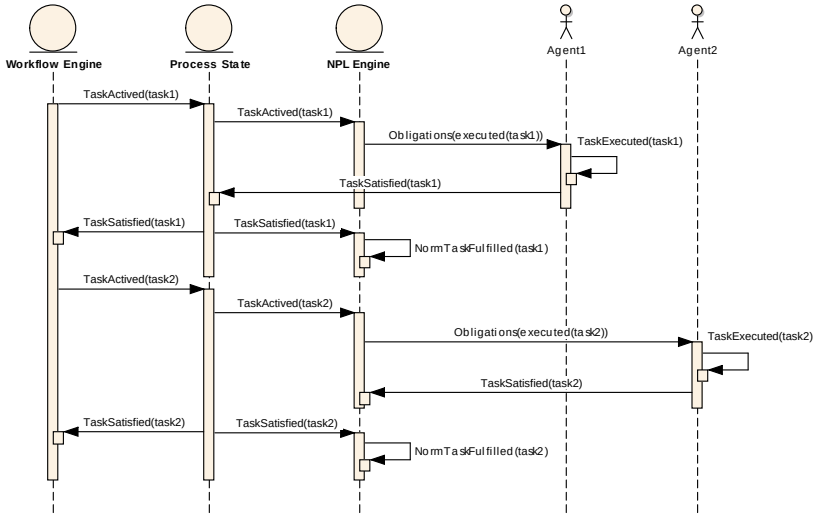


Figura 32: Diagrama de sequência do clico de vida de uma tarefa.

5.4 IMPLEMENTAÇÃO

Assim como o modelo proposto pode originar várias arquiteturas, sendo uma delas a apresentada neste cápitulo, a arquitetura apresentada pode ser implementada de diferentes maneiras. Como parte do trabalho gerado nesta dissertação, foi realizada uma implementação com a finalidade de validar o modelo e a arquitetura propostos.

Nesta seção são apresentadas as técnicas e ferramentas utilizadas no desenvolvimento da ferramenta proposta e as etapas seguidas na

implementação.

5.4.1 Ferramentas utilizadas

O modelo foi implementado utilizando o ambiente de programação *Eclipse Juno 4.2.2* e a linguagem de programação *Java* versão 7. Java permite a portabilidade da aplicação para diversas plataformas que suportam a linguagem.

Foi utilizado o Moise como modelo organizacional e para programação dos agentes foi utilizado a linguagem *Jason*. *Jason* é um interpretador para uma versão estendida da linguagem *AgentSpeak*. Foi utilizado o motor de workflow da ferramenta *Bonita Open Solution* e o banco de dados *MySQL Server 5.5*.

5.4.2 Etapas da Implementação

O processo de desenvolvimento do modelo foi dividido em três etapas:

- Modificações na especificação funcional do Moise.
- Implementação da linguagem NOPL/W.
- Implementação da estrutura de comunicação entre o Moise e o sistema de workflow.

5.4.3 Modificação na especificação funcional do Moise

A especificação funcional do modelo proposto é composta pela linguagem OML e a notação BPMN, ambas escritas em XML. A linguagem OML é validada por um *XML Schema*, que é uma linguagem para definição de regras de validação ("esquemas") em documentos no formato XML. Foi necessária a criação de um novo Schema para validar as alterações na linguagem OML.

Para exibição da especificação organizacional e os estados das tarefas de forma gráfica, foi necessária a criação de uma nova *XSL - Extensible Style Language*, que é usada para converter arquivos *XML* em *HTML*, de modo a permitir sua exibição.

Exemplo da *OML XML* pode ser vista em detalhes no apêndice B.

5.4.4 Implementação da linguagem NOPL/W

Para suportar o modelo proposto foi necessária a criação da linguagem NOPL/W, utilizada no lugar de NOPL. Depois de definida a nova linguagem NOPL/W foram necessárias alterações na classe *Os2NOPL*, chamada agora de *Os2NOPLW*. Essa classe é responsável pela tradução de uma OS para NOPL/W, nesse caso, transforma a linguagem OML em NOPL/W.

A modificação mais importante na classe *Os2NOPLW* é a não utilização de pré-condições, que eram utilizadas para informar quando uma tarefa estava habilitada. Agora, esse controle é feito pelo componente *Task State* (presente no componente *Process State*). Outra modificação estão nas regras: *well_formed*, *all_satisfied*, *is_finish* e *mission_acomplish*.

5.4.5 Implementação da estrutura de comunicação entre o Moise e o sistema de workflow Bonita

A comunicação entre o modelo organizacional Moise e a ferramenta de workflow Bonita ocorre através do componente *Workflow Engine* e *Task State*. Em *Task State* estão contidos os identificadores do processo, um identificador das tarefas e os estados das tarefas. Todos os três armazenados em um banco de dados *MYSQL Server*. *Task State* é representado pelo Modelo Entidade Relacionamento (MER) ilustrado na Figura 33. O MER é bastante simples, possuindo apenas 3 tabelas: *Process*, *Task* e *State*.

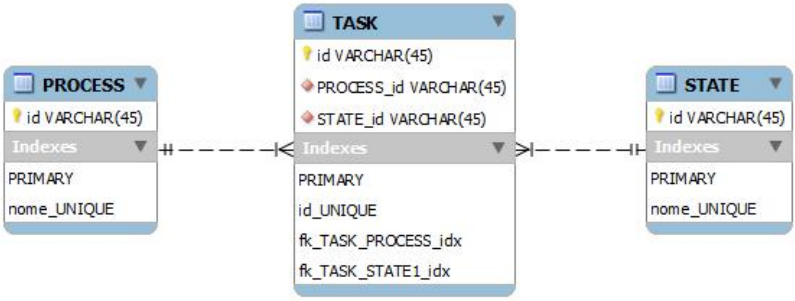


Figura 33: Modelo Entidade Relacionamento de *Task State*.

A tabela Process representa uma instância de um *Process Arti-*

fact. Cada instância de um *Process Artifact* possui um identificador único, guardado em cada linha da tabela Process. A tabela State representa um estado de um tarefa, cada estado de uma tarefa possui um identificado único (id). Os estados possíveis para uma tarefa são: *waiting*, *activated*, *satisfied*, *cancelled* e *removed*. A tabela *Task* representa uma tarefa, que por sua vez possui um id, está contida em um processo e possui apenas um estado.

O relacionamento das tabelas do MER apresentado na Figura 33 pode ser lido da seguinte forma: um processo pode possuir uma ou várias tarefas. Um estado pode estar relacionado com uma ou várias tarefas, ou seja, várias tarefas podem possuir o mesmo estado. Entretanto uma tarefa pode ter apenas um estado e estar contida em apenas um processo.

O componente *Task State* é responsável por tornar possível a comunicação entre os componentes *Workflow Engine*, *NPL Interpreter* e *Agents*.

5.4.5.1 Comunicação Workflow − TaskStateBD

Na comunicação do sistema de workflow com o componente *Task State* é utilizada a classe *TaskStateDAO* escrita em *Java*. A classe *TaskStateDAO* contém os métodos de acesso ao banco de dados, esses métodos podem ser utilizados para realização de consultas e alteração do conteúdo do banco de dados.

Os 3 métodos principais da classe *TaskStateDAO* são: *GetConnection()*, *SetStatetToActivated(process, task)* e *CanContinue(process, task)*. O método *GetConnection()* é responsável por estabelecer uma conexão com o servidor de banco de dados, *SetStatetToActivated(Process, task)* atualiza o estado de uma tarefa para ativada e o método *CanConitnue(process ,task)* trava o fluxo do processo até que o estado de uma tarefa até seja satisfeita. Esse método será explicado em detalhes na sequência.

Uma tarefa do sistema de workflow utiliza um conector do tipo *script grovvy* para acessar os métodos da classe *TaskStateDAO*. Conectores permitem integrar um solução de BPM aos *softwares* de sistema de informação. O Bonita BPM possui mais de 100 tipos conectores nativos, os mais comum utilizados são de bancos de dados, *e-mails*, *ERP*, *CRM*, sendo também possível adicionar novos conectores com o criador de conectores disponível na ferramenta.

Cada tarefa do workflow, que será executada por um agente,

chama dois métodos da classe *TaskStateDAO*. O primeiro método atualiza o estado de uma tarefa para *activated*. Está mudança de estado serve para informar a um agente que a tarefa está ativada e pronta para ser executada. O segundo método *CanContinue()* é utilizado para controle do fluxo do processo, nesse caso o fluxo do processo é interrompido e apenas retornará ao seu fluxo normal, quando a tarefa passar para o estado satisfeita ou removida. A seguir o algoritmo que define o método *CanContinue(process, task)* é apresentado.

```
1 Void CanContinue(String process, String task) $$

2    boolean continue = true;
3    String state = "";

4        while (continue)
5            state = GetTaskState(process, task);

6            switch (state)
7                case "waiting":
8                    continue = true;
9                    break;
10                case "activated":
11                    continue = true;
12                    break;
13                case "satisfied":
14                    continue = false;
15                    break;
16                case "cancelled":
17                    continue = true;
18                    break;
19                case "removed":
20                    continue = false;
21                    break;
22                default:
23                    break;
24
25            Thread.sleep(1000);
26
27
```

Na primeira linha é definida a assinatura do método, onde são passados os parâmetros: *process* (identificador de um processo) e *task* (identificador de um tarefa). Em seguida a variável *continue* recebe *true*.

As linhas de 4 até 26 são repetidas enquanto *continue* for *true*. Na linha 5 é obtido o estado atual de uma tarefa. Caso o estado atual da tarefa seja *waiting*, *activated*, ou *cancelled* a variável *continue* recebe *true*. Caso o estado da tarefa atual seja *satisfied* ou *removed*, *continue* recebe *false*. Na linha 25 o algoritmo dorme por 1 segundo a cada ciclo do *loop*.

Explicando em outras palavras, o algoritmo ficará *loop* até o estado da tarefa se tornar *satisfied* ou *removed*.

5.4.5.2 Comunicação Agent − NPL Interpreter

Agentes se comunicam com o componente *TaskState* através de *operation* do *Process Artifact*, discutido na seção 5.2. As *operations* disponíveis para os agentes são: *reactivated*, *cancelled*, *removed* e são utilizadas para alterar o estado de uma tarefa.

5.4.5.3 Comunicação NPL Interpreter − TaskState

O componente *NPL Interpreter* se comunica com *Taskstate* a partir de três métodos: *UpdateTask()*, *VerifyTaskActived()* e *IsTaskToSatisfied()*.

O método *UpdateTaskState(Process, task, state)* está disponível em *NPL Interpreter* para ser utilizado pelos agentes através de uma *operation*. O método *VerifyTaskActived()* é responsável por verificar se uma tarefa está no estado *activated*. O método *IsTaskToSatisfied()* verifica se a tarefa está no estado satisfeita, caso a tarefa esteja satisfeita, esse novo estado é informado ao motor de workflow para que o fluxo das tarefas possa continuar.

Resumindo, a alteração dos estados das tarefas ocorrem da seguinte maneira. Todas as tarefas iniciam no estado: *waiting*. Agentes podem alterar o estado de uma tarefa, através de uma operation, para: *activated*, *cancelled*, *removed*. Uma tarefa é alterado pro estado satisfeita apenas pelo *NPL Interpreter* quando a cardinalidade de execução da tarefa é cumprida: *satisfied*. O *Workflow Engine* pode apenas alterar o estado de uma tarefa pro estado: *activated*.

5.5 VALIDAÇÃO

Este seção possui o objetivo de ilustrar o modelo proposto e verificar se ele atingiu os objetivos da dissertação. Primeiramente, na seção 5.5.1 é mostrado um exemplo completo da utilização do modelo escolhido. Em seguida, na seção 5.5.2 é apresentado um comparativo de padrões de workflows e na seção 5.5.3 são apresentadas as vantagens da utilização do modelo proposto. As duas primeiras seções respondem, juntas, a primeira pergunta de pesquisa apresentada anteriormente. A

segunda pergunta será respondida na última seção.

5.5.1 Exemplo de uso do modelo proposto

Nesta seção será apresentado um exemplo de utilização do modelo proposto. Apenas aspectos específicos do problema serão explorados com o objetivo de demonstrar a aplicação de grande parte dos conceitos e dos relacionamentos que compõem o modelo.

O exemplo foi inspirado no processo de fabricação de queijo, mostrado no capítulo 4, onde é simulado o gerenciamento de um laticínio. As atividades envolvidas no processo vão da aquisição do leite até a fabricação do queijo e a entrega da encomenda. A seguir é explicado o passo a passo para a criação do processo na ferramenta Bonita BPM.

5.5.1.1 Definição do processo de fabricação de queijo na ferramenta Bonita

O primeiro passo para criação do exemplo foi a definição do processo de fabricação de queijo na ferramenta Bonita BPM. O processo pode ser observado na Figura 34 e foi definido com um editor gráfico utilizando a notação BPMN 2.0.

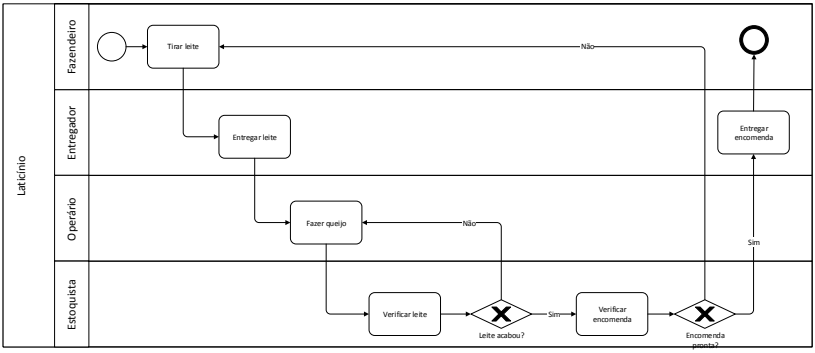


Figura 34: Processo de fabricação de queijo.

Foram utilizadas raias para separação dos papéis utilizados no processo. Os papeis são: *Fazendeiro*, *Entregador*, *Operário* e *Estoquista*. O papel *Fazendeiro* é responsável pela execução da tarefa *Tirar leite*, o papel *Entregador* é responsável pelas tarefas: *Entregar leite*

e *Entregar encomenda*. O *Operário* é responsável pela tarefa *Fazer queijo* e o papel de *estoquista* é responsável pelas tarefas: *Verificar leite* e *Verificar encomenda*. É importante frisar que nesse exemplo o papel *Estoquista* não é assumido por um agente e sim por um *script* escrito na linguagem *groovy*. Entretanto as tarefas *Verificar Leite* e *Verificar Encomenda* poderiam ser executadas tanto por um agente como por um humano.

O funcionamento do processo de fabricação de queijo ocorre da seguinte maneira: o processo é iniciado com a tarefa de tirar leite de vaca, essa tarefa é executada pelo agente que assume o papel de fazendeiro; após, o leite é entregue no laticínio pelo entregador para então o operário pode iniciar a fabricação do queijo. Após a fabricação do queijo, o estoquista verifica a quantidade de leite disponível em estoque. Enquanto o leite não acabar, o operário continua fazendo queijo. Quando o leite acabar, o estoquista verifica a quantidade de queijos produzida até o momento. Se a produção já for suficiente para atender a encomenda de 100 peças de queijo, o entregador faz a entrega no supermercado local. Caso contrário, o processo reinicia na tarefa de tirar o leite da vaca.

Depois da definição do processo pela notação BPMN 2.0 é preciso definir quais *scripts* cada tarefa do processo precisa executar. Cada tarefa que será executada por um agente precisa chamar dois métodos da classe *TaskStateDAO*. Esses métodos são utilizados para realizar o mapeamento entre as tarefas da notação BPMN com as tarefas contidas no componente *TaskState*. Por exemplo, com a chamada do método *UpdateTask(dayryIndustry, activated)* a tarefa *milk a cow* irá ativar a tarefa do banco de dados *milk a cow*.

Para o processo de fabricação de queijo foram utilizadas seis variáveis globais da ferramenta Bonita BPM. Essas variáveis são uteis para guardar os recursos que são produzidos ou consumidos por cada tarefa. Os recursos também são utilizados nos *gateways milk is over?* e *ready packages?* Por exemplo, dependendo do valor contido numa variável de recurso o fluxo do workflow pode tomar caminhos diferentes. As variáveis de recursos utilizadas foram:

- *cheese*: variável do tipo inteiro que informa a quantidade de queijos que foram produzidos até o momento.
- *milk*: variável do tipo inteiro que informa a quantidade de leite que o laticínio possui em estoque.
- *milkOver*: variável do tipo booleano que informa se o leite terminou ou não.

- *readyPackes*: variável do tipo booleano que informa se a meta da encomenda foi atingida ou não.

Toda tarefa que vai ser executada por um agente possui os métodos *SetTaskToActived (process)* e *CanContinue(process, task)*. Uma tarefa, quando executada, pode produzir ou consumir um recurso. Caso a tarefa produza um recurso, a variável do recurso será incrementada; caso ela consuma, a variável do recurso será decrementada. A seguir são apresentados os métodos e scripts que cada tarefa do processo executa:

1. A tarefa *MilkACow* executa os métodos: *SetTaskToActivated(dairyIndustry)*, *CanContinue(dairyIndustry, milkACow)* e *AddCheese()*.

2. A tarefa *DeliverTheMilk* executa os métodos: *SetTaskToActived(dairyIndustry)* e *CanContinue(dairyIndustry, deliverTheMilk)*.

3. A tarefa *MakeACheese* executa os métodos: *SetTaskToActived(dairyIndustry)*, *CanContinue(dairyIndustry, makeACheese)*, SubbMilk() e addCheese().

4. A tarefa *CheckMilk* executa o *script* seguinte:

```
1   if (milk > 0)
2      milkOver = false
3   else
4      milkOver = true
```

   Esse *script* informa se há leite no laticínio. Nesse caso, a variável *milk* terá um valor maior que zero.

5. A tarefa *check Packages* executa o *script* seguinte:

```
1   if (cheese >= 100)
2      readyPackages = true
3   else
4      readyPackages = false
```

   Esse *script* verifica se a encomenda de 100 peças de queijo foi alcançada.

6. A tarefa *DeliverThePackages* executa os métodos: *SetTaskToActived(dairyIndustry)*, *CanContinue(dairyIndustry,milkACow)*.

5.5.1.2 Especificação Funcional com OML XML

A definição da especificação funcional utilizando a linguagem OML é apresentado na Figura 35. Primeiro é definido o identificador do processo, em seguida são definidos os identificadores das tarefas e um texto descritivo dizendo o que elas fazem. Por último são definidas as missões e quais tarefas pertencem a cada missão.

```
<functional-specification>
    <process id="dairyIndustry" >
        <task id="end" >
            <task id="milkACow" min="1" ds="Milk a Cow" />
            <task id="deliveryTheMilk" min="1" ds="Deliver the Milk to Dairy Industry" />
            <task id="makeACheese" min="1" ds="Make A cheese" />
            <task id="deliveryThePackage" min="1" ds="Deliver The Packages" />
        </task>
        <mission id="m1" min="0" max="5">
            <task id="milkACow" />
        </mission>
        <mission id="m2" min="0" max="5">
            <task id="deliveryTheMilk" />
            <task id="deliveryThePackage" />
        </mission>
        <mission id="m3" min="0" max="5">
            <task id="makeACheese" />
        </mission>
    </process>
</functional-specification>
```

Figura 35: Especificação Funcional em OML XML.

5.5.1.3 Implementação dos Agentes

Depois da definição do processo e da especificação em *OML XML* é necessário codificar os agentes que irão assumir os papéis para execução das tarefas do processo de fabricação de queijo.

Os códigos dos agentes são escritos na linguagem Jason Agent Speak e podem ser conferidos no apêndice A. Foi criado um agente *Jason* para cada papel. Entretanto um papel poderia ser executado por mais de um agente: Foram utilizado os agentes: *Manager*, *Farmer* e *Deliveryman* e *Worker*.

O agente *Manager* não assume nenhuma tarefa do processo do laticínio. Ele é utilizado para gerenciar os grupos e processos instanciados em tempo de execução.

Após a criação dos grupos e processos, o agente *Farmer* entra no grupo laticínio, assume um papel no processo e espera pelas suas obrigações. O fonte dos agentes *Manager* e *Farmer* são apresentados nos apêndices A1 e A2 respectivamente.

Os códigos *Jason* dos agentes que assumem os papéis de *Deliveryman* e *Worker* são muito semelhantes ao código do agente *Farmer*. Esses agentes são diferenciados pelo tipo de papel que podem assumir e pelos planos para execução das metas.

5.5.1.4 Execução do Exemplo

Nesta seção será mostrado o exemplo de fabricação de queijo em tempo de execução. A Figura 36 apresenta a sequência de ações que ocorrem durante a execução do exemplo. A execução ocorre na seguinte ordem:

1. Inicia Workflow: utilizado iniciar o fluxo das tarefas do processo. Bonita BPM possui uma interface web para acompanhamento da execução das tarefas.
2. Criação do ambiente: nesta etapa o ambiente Cartago é criado.
3. Inicia organização: o modelo organizacional Moise é iniciado. Nesta etapa é utilizado a especificação organizacional para criação das instancias dos group artifact e process artifact.
4. Inicia agentes Jason: após a criação do ambiente e inicialização da organização os agentes Jason podem ser inseridos no ambiente e interagirem com a organização.
5. Cria obrigações: são criadas as obrigações para com os agentes.
6. Executa tarefas: nesta etapa os agentes já são capazes de executar suas tarefas.

   Caso ainda existam tarefas para serem executadas, o ciclo volta para a ação atualiza tarefas.

Na figura Figura 37 pode-se observar a interface gráfica do Moise, que permite a visualização da organização em tempo de execução. Essa figura está apresentando uma instância do processo. *Formation: ok*, significa que a regra de formação do processo ocorreu de forma satisfatória. Em *Player*, é mostrado quais as missões com as quais cada agente se comprometeu. Logo abaixo são mostradas as tarefas, os estados das tarefas, quem se comprometeu com cada tarefa e quem está executando uma tarefa naquele respectivo momento.

A partir da Figura 37, pode-ser também observar que as tarefas *milkACow* e *deliveryTheMilk* estão satisfeitas. A tarefa *makeAChesse*

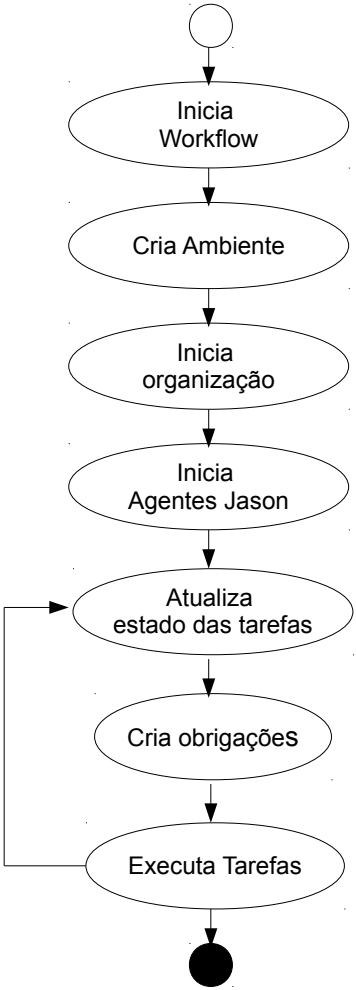


Figura 36: Sequência de execução do exemplo.

está ativada e as tarefas *deliveryThePackages* e *end* estão em estado de espera. A tarefa *end* é utilizada para indentificar o fim do processo.

Em *History* é apresentado o histórico de criação e cumprimento das obrigações. Observa-se que a obrigação para execução da tarefa *milkACow* foi criada e cumprida três vezes. Isso ocorreu por que os agentes operários fazem queijos até o leite acabar.

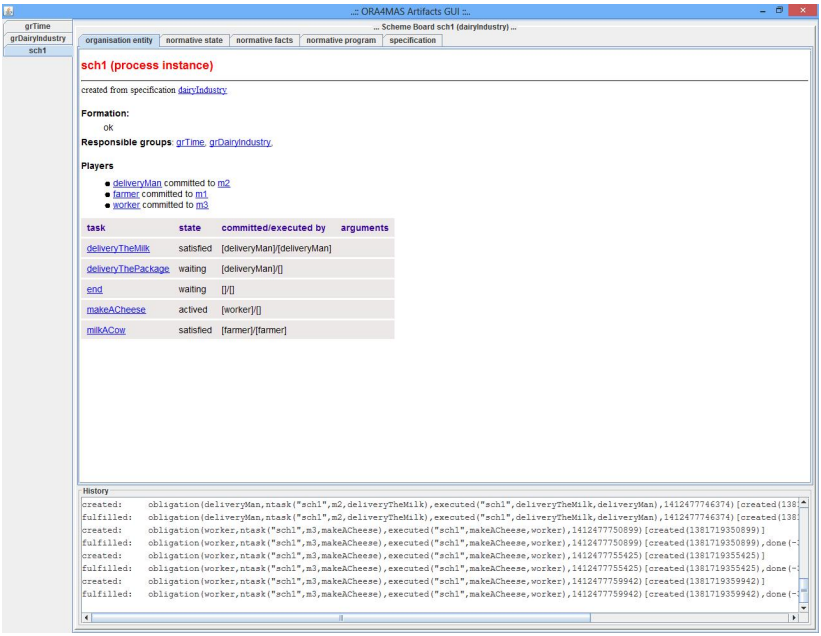


Figura 37: Interface gráfica do Moise.

5.5.2 Comparações de Padrões de workflow

Padrões são maneiras de descrever as melhores práticas e soluções para determinado problema de modo que seja possível aos outros reutilizar estas experiências. Assim como na área de Engenharia de Software, onde os padrões mais conhecidos são os Padrões de Projeto (Design Patterns) (GAMMA et al., 1995), a área de BPM e workflow apresenta abordagens para promover o reuso, tanto no nível conceitual quanto no nível de implementação do processo.

Padrões de workflow têm atraído a atenção de pesquisadores e da indústria de software devido as suas vantagens. Em (AALST et al., 2002) são descritos 21 padrões de workflow para controle de fluxo (e.g., sequencial, paralelo, condicional). Tais padrões são úteis tanto para a definição de workflows, como para validar o poder de expressão das linguagens e ferramentas de workflow (AALST; BARROS; HOFSTEDE, 2000). Os padrões descrevem diversas maneiras através das quais dados podem ser representados em definições de workflow.

A Tabela 7 sumariza os resultados da pesquisa em termos de

suporte aos padrões de workflow propostos. Para cada combinação, é indicado se o modelo suporta ou não o padrão. Como convenção, atribui-se *suportado* para padrões suportados direta e/ou indiretamente pela ferramenta, e *não suportado* para padrões não suportados pela ferramenta. Pela Tabela 7 é possível notar que o Moise com a agregação do workflow suporta 10 dos 13 padrões apresentados. Enquanto que o Moise sem a agregação suporta apenas 6 dos 13 padrões. Com suporte ao padrão *Arbitrary Cycles* uma ou mais tarefas podem repetir ciclicamente, permitindo *loop*. Isso demonstra vantagem de agregar workflow ao Moise, pois é possível resolver o problema de encadeamento iterativo e condicional.

	Ferramenta	
Padrão	Moise	Moise + workflow
1 Sequence	Suportado	*Suportado*
2 Parallel Split	Suportado	*Suportado*
3 Synchronization	Suportado	*Suportado*
4 Exclusive Choice	Suportado	*Suportado*
5 Simple Merge	Suportado	*Suportado*
6 Multi-choice	Suportado	*Suportado*
7 Synchronizing Merge	Não suportado	*Suportado*
8 Multi-merge	Não suportado	*Não Suportado*
9 Discriminator	Não suportado	*Não Suportado*
10 Arbitrary Cycles	Não suportado	*Suportado*
11 Implicit Termination	Não suportado	*Não Suportado*
12 MI Without Synchronization	Não suportado	*Suportado*
13 MI With a Priori Design Time Knowledge	Não suportado	*Suportado*

Tabela 7: Padrões de Fluxos

Outro ponto a ser considerado pela Tabela 7 é que o modelo proposto não suporta todos os 13 padrões de workflow por limitações da ferramenta *Bonita*, visto que existem outras ferramentas de workflow que suportam os 13 padrões apresentados. A consequência da ferramenta não suportar determinado padrão, está no fato dela não conseguir trabalhar com fluxos mais complexos.

5.5.3 Vantagens Encontradas

Abaixo são apresentadas algumas vantagens da utilização de workflow em organização de sistemas multiagentes:

Padrões de Fluxos Prontos: Na dimensão funcional do Moise sem agregação do workflow, a modelagem dos fluxos é difícil, pouco intuitiva e não há padrões de fluxos prontos a serem seguidos como em sistemas de workflow. Algumas vantagens de utilizar padrões de fluxos prontos são: Eles foram testados em diversos projetos, refletindo a experiência e conhecimento dos desenvolvedores que já utilizaram com sucesso esses padrões em seus trabalhos. Os padrões provêm uma solução pronta que pode ser aplicada a diferentes problemas; os padrões provêm um vocabulário comum de soluções que podem expressar, sucintamente, muitas soluções e por fim padrões de fluxos são reutilizáveis.

Interface gráfica intuitiva: Ao contrário do Moise, que não utiliza uma ferramenta para modelagem dos fluxos de metas da organização, a maioria das ferramentas de sistemas de workflow possuem um editor gráfico para modelagem dos fluxos de tarefas no padrão BPMN (Business Process Modeling Notation). Trata-se de uma notação padrão para o desenho de fluxogramas em processos de negócios.

Agentes podem raciocinar sobre workflow: Utilizando um componente BAM (Business Activity Monitoring) um agente monitora o workflow, verifica gargalos, atrasos, raciocina sobre o processo e cria novos processos em tempo de execução. Caso um agente se recuse a executar uma tarefa ele pode delegar a tarefa para outro agente ou mudar o seu papel, realizando assim o refinamento e ajuste de processos.

Utilização de um Motor de regras: Um componente BRM (Business Rules Management) é uma tecnologia que permite aos usuários finais definirem regras de negócio de forma declarativa. Regras de negócio ficam em um repositório separado, e podem ser consultadas pelos agentes em tempo de execução. Isso facilita a institucionalização das regras de negócio na organização, a transparência sobre as regras existentes e redução do esforço de manutenção de sistema.

Agentes podem utilizar os conectores do workflow: Agentes podem utilizar um banco de dados, acessar webservices, enviar e-mail, entre outras possibilidades que conectores do workflow oferecem.

Tarefas podem ser executadas por humanos ou por agentes: Alguns fluxos de tarefas podem ser totalmente automatizados, sendo todas as tarefas executadas por agentes. Em outros fluxos, algumas tarefas são executadas por humanos e outras por agentes, aumentando a interação entre eles.

6 CONCLUSÃO

Este trabalho apresentou uma proposta de utilização de um modelo conceitual para integração de um sistema de workflow a um modelo organizacional de SMA. Inicialmente foi realizada uma revisão bibliográfica e levantamento do estado da arte onde foram estudados e definidos alguns conceitos principais sobre a teoria de SMA, modelos organizacionais e de sistemas de workflow. Após o levantamento do estado da arte foi definido o modelo conceitual para agregação do sistema de workflow ao Moise. Após conceber o modelo conceitual, uma linguagem de programação normativa para organização foi estendida. Essa linguagem suporta a agregação de uma ferramenta de workflow a um modelo organizacional. Também foi definida uma arquitetura de implementação para viabilizar o funcionamento do modelo proposto, a arquitetura tem como requisito principal ser genérica o suficiente para poder ser integrada com diversos sistemas de workflow. Após a implementação da arquitetura proposta, foi apresentado um exemplo completo de sua utilização com o intuito de mostrar o seu funcionamento e avaliar o modelo proposto. Na avaliação foram analisados diversos aspectos relacionados ao uso do modelo, apresentando as suas vantagens com o objetivo de verificar se o modelo atingiu os objetivos da dissertação.

O objetivo principal desse trabalho foi atingido, na medida em que conseguiu resolver as limitações apresentadas por alguns modelos organizacionais, descritas na seção 1.3. Esse objetivo foi cumprido graças a definição do modelo proposto. Na seção 5.5.1 foi mostrado um exemplo completo da utilização do modelo e na seção 5.5.2 foi apresentado um comparativo de padrões de workflow.

O desenvolvimento dessa dissertação respondeu às perguntas de pesquisa, definidas na Seção 1.1 e que serão apresentadas a seguir.

1. Como agregar sistemas de workflow a especificação funcional de um modelo organizacional de SMA?

2. Quais as vantagens na utilização de um sistema workflow em modelos organizacionais de SMA?

A pergunta de pesquisa 1 foi respondia no capítulo 5. Inicialmente foram analisadas e comparadas três abordagens diferentes: o modelo híbrido, o modelo workflow com missões e o modelo workflow puro, essa comparação teve o objetivo de verificar qual abordagem melhor responde a pergunta de pesquisa 1. Após a comparação das três

propostas, o modelo conceitual escolhido foi definido e apresentado em detalhes.

A pergunta de pesquisa 2 foi respondida na seção 5.5.3, nela são apresentadas as vantagens da utilização do modelo proposto.

Uma contribuição deste trabalho está no fato do modelo organizacional poder agora utilizar a maioria dos padrões de workflow. Esses padrões são úteis tanto para a definição de fluxos de processos, como para validar o poder de expressão das linguagens e ferramentas de workflow.

A ferramenta desenvolvida foi uma contribuição importante para os desenvolvedores de SMA. Além disso, para a definição de uma especificação social é possível utilizar um editor BPMN, que facilita a modelagem do fluxo dos processos. Por fim, o estudo realizado permitiu realizar avanços na área de organização de SMA, uma vez que agora a dimensão funcional de modelos organizacionais pode utilizar de fluxos complexos de trabalho e é possível que tarefas sejam executadas tanto por agentes como por humanos.

6.1 TRABALHOS FUTUROS

Alguns possíveis trabalhos futuros foram identificados no decorrer deste trabalho. São eles:

- Suporte a outros padrões de workflow: atualmente existem mais de 40 padrões de workflow. Nesta dissertação foran comparados e testados apenas 13 destes padrões.

- Sugeri modificações na implementação da arquitetura. O modelo proposto utiliza-se de um banco de dados para guardar o estado das tarefas, entretanto poderia ser utilizado outras abordagens, como por exemplo, webservices.

- A inclusão de mecanismo para que o ambiente possa saber se a tarefa foi executada pelos agentes. Atualmente esse papel está sendo desempenhado pelos agentes.

- Desenvolvimento de uma ferramenta para que os agentes possam utilizar das métricas de processos disponibilizadas pelo sistema de workflow.

- Manual para utilização do modelo proposto.

- Estudo da possibilidade da organização e os agentes poderem utilizar dos conectores presentes nos sistemas de workflow.
- Utilização de aprendizagem por reforço na definição dos processos.

REFERÊNCIAS

AALST, W. V. D.; BARROS, A. P.; HOFSTEDE, A. H. M. Advanced workflow patterns. In: *Proceedings of the 7th International Conference on Cooperative Information Systems*. London, UK, UK: Springer-Verlag, 2000. (CooplS '02), p. 18–29. ISBN 3-540-41021-X.

AALST, W. V. D. et al. *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press, 2002.

AALST, W. van der; HOFSTEDE, A. H. M. T.; WESKE, M. Business process management: A survey. In: *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*. [S.l.]: Springer-Verlag, 2003. p. 1–12.

ACTIVITI. *Activiti BPM Platform*. 2013. Disponível em: <http://www.activiti.org/>.

BALDAN, R. et al. *Gerenciamento de Processos de Negócio: BPM - Business Process Management*. [S.l.]: Érica, 2007.

BPMN, B. *Bonita Open Solution*. 2013. Disponível em: <http://br.bonitasoft.com/>.

CAIRE, G.; GOTTA, D.; BANZI, M. Wade: a software platform to develop mission critical applications exploiting agents and workflows. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008. (AAMAS '08), p. 29–6.

CAIRE, G. et al. Wolf - an eclipse plug-in for wade. In: *Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Washington, DC, USA: IEEE Computer Society, 2008. (WETICE '08), p. 26–32. ISBN 978-0-7695-3315-5.

CARHUATOCTO, R. Jbpm, bonita, intalio, processmaker, activiti. qué bpm suite uso? *HOLISTIC SECURITY AND TECHNOLOGY*, p. 119–153, 2011. ISSN 0926-8782. Disponível em: <http://holisticsecurity.wordpress.com/2011/07/21/jbpm-bonita-intalio-processmaker-activiti-que-bpm- suite-uso/>.

COPPIN, B. *Artificial Intelligence Illuminated*. USA: Jones and Bartlett Publishers, Inc., 2004. ISBN 0763732303.

COUTINHO, L. R. *Interoperabilidade Organizacional em Sistemas Multiagentes Abertos baseada em Engenharia Dirigida por Modeloss*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2009.

COUTINHO, L. R.; SICHMAN, J. S.; BOISSIER, O. Modeling organization in mas: A comparison of models. *1º Workshop on Software Engineering for Agent-oriented Systems*, Uberlânida, MG, 2005.

DAVENPORT, T. H. *Reeengenharia de processos: como inovar na empresa através da tecnologia da informação*. [S.l.]: Campus, 2004.

DECKER, K. TÆms: A framework for environment centered analysis & design of coordination mechanisms. In: *In Foundations of Distributed Artificial Intelligence, Chapter 16*. [S.l.]: John Wiley & Sons, Inc, 1996. p. 429–448.

DECKER, K.; LESSER, V. Task environment centered design of organizations. In: *In Proceedings of the AAAI Spring Symposium on Computational Organization Design*. [S.l.: s.n.], 1994.

DIGNUM, V. et al. An organizational-oriented model for agent societies. *RASTA02 at AAMAS*, Bologna, Italy, 2002.

DIGNUM, V. A. *A model for organizational interaction: based on agents, founded organizational models*. Tese (Doutorado) — Utrecht University, 2004.

ECLIPSE. *Eclipse IDE*. 2013. Disponível em: <http://www.eclipse.org/>.

ESTEVA, M.; DAVID, d. l. C.; SIERRA, C. Islander: an electronic institutions editor. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*. New York, NY, USA: ACM, 2002. (AAMAS '02), p. 1045–1052. ISBN 1-58113-480-0.

FERBER, J.; GUTKNECHT, O. A meta-model for the analysis and design of organizations in multi-agent systems. In: *Proceedings of the 3rd International Conference on Multi Agent Systems*. Washington, DC, USA: IEEE Computer Society, 1998. (ICMAS '98). ISBN 0-8186-8500-X.

FERBER, J.; GUTKNECHT, O.; MICHEL, F. From agents to organizations: An organizational view of multi-agent systems. In: *In: LNCS n. 2935: Procs. of AOSE?03*. [S.l.]: Springer Verlag, 2003. p. 214–230.

FERBER, J.; MICHEL, F.; BAEZ, J. Agre: integrating environments with organizations. In: *Proceedings of the First international conference on Environments for Multi-Agent Systems*. Berlin, Heidelberg: Springer-Verlag, 2005. (E4MAS'04), p. 48–56. ISBN 3-540-24575-8, 978-3-540-24575-9.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GARCíA-CAMINO, A. et al. *Constraint Rule-based Programming of Norms for Electronic Institutions*. [S.l.]: Journal of Autonomous Agents and Multi-Agent Systems, 2009. 186?217 p.

GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib. Parallel Databases*, Kluwer Academic Publishers, Hingham, MA, USA, v. 3, n. 2, p. 119–153, abr. 1995. ISSN 0926-8782.

GERS, F. *Multi-agent system for distributed data fusion in peer-to-peer environment*. Tese (Doutorado) — University of Jyväskylä, Jyväskylä, 2002.

GONCALVES, J. E. L. s empresas são grandes coleções de processos. *AE - Revista de Administração de Empresas*, 2000.

HANNOUN, M. et al. Moise: Un modèle organisationnel pour la conception de systèmes multi-agents. *Em Acts des tèmes Journèes Francophones Intelligence Artificielle Distribuée & Systèmes Multi-Agents*, Hermès Science Publications, 1999.

HÜBNER, J. F. *Um Modelo de Reorganização de Sistemas Multiagentes*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2003.

HüBNER, J. F.; BOISSIER, O.; BORDINI, R. H. A normative programming language for multi-agent organisations. *Annals of Mathematics and Artificial Intelligence*, Kluwer Academic Publishers,

Hingham, MA, USA, v. 62, n. 1-2, p. 27–53, jun. 2011. ISSN 1012-2443.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. *INTERNATIONAL JOURNAL OF AGENT-ORIENTED SOFTWARE ENGINEERING*, v. 1, p. 370?395, 2007.

HüBNER, J. F.; VERCOUTER, L.; BOISSIER, O. *Instrumenting Multi-Agent Organisations with Artifacts to Support Reputation Processes*. [S.l.]: Springer US, maio 2010. 369-400 p.

INTALIO. *BPM - Intalio*. 2013. Disponível em: <http://www.intalio.com/segments/developers/bpm/>.

JBPM. *JBPM Jboos*. 2013. Disponível em: <http://www.jboss.org/jbpm>.

JENNINGS, N. R. Agent-oriented software engineering. In: *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: MultiAgent System Engineering*. London, UK, UK: Springer-Verlag, 1999. (MAAMAW '99), p. 1–7. ISBN 3-540-66281-2. Disponível em: <http://dl.acm.org/citation.cfm?id=646910.710805>.

JUNIOR, M. A. F. *Análise Comparativa entre Ferramentas BPM Gratuitas*. Dissertação (Mestrado) — UNIPLAC, 2011.

LESSER, V. Evolution of the gpgp/tÆms domain-independent coordination framework. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. New York, NY, USA: ACM, 2002. (AAMAS '02), p. 1–2. ISBN 1-58113-480-0.

LEVESQUE, H. J.; COHEN, P. R.; NUNES, J. H. T. On acting together. In: *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*. [S.l.]: AAAI Press, 1990. (AAAI'90), p. 94–99. ISBN 0-262-51057-X.

LóPEZ, F. L.; LUCK, M.; D'INVERNO, M. Constraining autonomy through norms. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*. New York, NY, USA: ACM, 2002. (AAMAS '02), p. 674–681. ISBN 1-58113-480-0.

MAIQUEL, B. *Uma Linguagem para Especificação da Dinâmica dos Fatos Institucionais em Sistemas Multiagentes*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2012.

OMICINI, A.; RICCI, A.; VIROLI, M. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, Hingham, MA, USA, v. 17, n. 3, p. 432–456, dez. 2008. ISSN 1387-2532.

PROCESSMAKER. *ProcessMaker Workflow Simplified*. 2013. Disponível em: <http://www.processmaker.com/>.

REIS, L. P. *Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer*. Tese (Doutorado) — FEUP, 2003.

RUSSELL, S.; NORVIG, P. *Inteligência artificial*. CAMPUS - RJ, 2004. ISBN 9788535211771. Disponível em: <http://books.google.com.br/books?id=wBMvAAAACAAJ>.

TAMBE, M. Towards flexible teamwork. *J. Artif. Int. Res.*, AI Access Foundation, USA, v. 7, n. 1, p. 83–124, set. 1997. ISSN 1076-9757.

TINNEMEIER, N.; DASTANI, M.; MEYER, J.-J. Roles and norms for programming agent organizations. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009. (AAMAS '09), p. 121–128. ISBN 978-0-9817381-6-1.

WEISS, G. (Ed.). *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press, 1999. ISBN 0-262-23203-0.

WFMC. *Workflow Management Coalition Terminology & Glossary*. Document Number WFMC-TC-1011 : Issue 3, 1999. Disponível em: <http://www.huihoo.org/jfox/jfoxflow/specification/03.Terminology_Glossary.p

WHITE, S. A. *Introduction to BPMN*. [S.l.]: IBM Corporation, 2004.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. [S.l.]: JOHN WILEY & SONS, 2002. ISBN 0047149691X.

XPDL. *XML Process Definition Language*. 2013. Disponível em: <http://www.wfmc.org/standards/xpdl.htm>.

APÊNDICE A – Código dos agentes Jason

Neste apêndice são apresentados os código fontes dos agentes Jason. Inicialmente é apresentado o fonte do agente manager, esse agente não executa nenhuma tarefa no laticínio, ele é responsável por criar e gerenciar os artefatos de grupo e processos. Em seguida é apresentado o fonte dos agentes Farmer e DeliveryMan.

A.1 FONTE DO AGENTE MANAGER

```
 include("check_obl.asl")  // some common plans for obedient agents
/* Initial Tasks */
!start.
/* Plans */
+!start
  <- .my_name(Me);
     createWorkspace("ora4mas");
     joinWorkspace("ora4mas",O4MWsp);

 makeArtifact("grTime","ora4mas.nopl.GroupBoard",["../dairyIndustry.xml", time,
 false, true ],IDtime);
 setOwner(Me);
 focus(IDtime);
     .print("group1 created");

 adoptRole(manager)[artifact_id(IDtime)];

 makeArtifact("grDairyIndustry","ora4mas.nopl.GroupBoard",["../dairyIndustry.xml",
 dairyIndustry, false, true ],IDexploracao);
 setOwner(Me);
 focus(IDexploracao);
 setParentGroup("grTime")[artifact_id(IDexploracao)];
     .print("group2 created");

   // wait for the other agents
   ?play(farmer,farmer,"grDairyIndustry");
   ?play(deliveryMan,deliveryMan,"grDairyIndustry");
   ?play(worker,worker,"grDairyIndustry");

    !run_process(proc1).

// general error handler for goal start
-!start[error(I),error_msg(M)] <- .print("failure in starting! ",I,": ",M).

+!run_process(P)
   <- makeArtifact(S,"ora4mas.nopl.ProcessBoard",["../dairyIndustry.xml", dairyIndustry,
      false, true ],ProcArtId);
      focus(ProcArtId);
      .print("process ",P," created");
  addProcess(P)[artifact_name("grTime")];
 .print("process is linked to responsible group").

-!run_process(P)[error(I),error_msg(M)] <- .print("failure creating process ",P," -- ",I,": ",

+?play(A,R,T) <- .wait(+play(_,_,_),100,_); ?play(A,R,T).

// signals
```

```
+normFailure(N)  <- .print("norm failure event: ", N).
+destroyed(Art)  <- .print("Artifact ",Art," destroyed").
```

A.2 FONTE DO AGENTE FARMER

```
 include("check_obl.asl")
!start.
!join.

/* Plans */
+!start
   <- lookupArtifact("grDairyIndustry",GrId); //get id from artifact
      adoptRole(farmer)[artifact_id(GrId)];
      focus(GrId).
-!start
   <- .wait(100);
      !start.

+!join
   <- .my_name(Me);
       joinWorkspace("ora4mas",_).
-!join
   <- .wait(200);
      !!join.

// keep focused on process my groups are responsible for
+process(L)
   <- for ( .member(S,L) )
         lookupArtifact(P,ArtId);
         focus(ArtId)
      .

+!quit_mission(M,S)
   <- .print("leaving my mission ",M," on ",P,"....");
      leaveMission(M)[artifact_name(P)].

/* Initial tasks */

/* Plans */
// application domain tasks
+!milkACow<- .print("Milk A Cow");
             .wait(500).

// conditions to leave missions
@lgss1[atomic]
+taskState(Process,end,_,_,executed)
    : .my_name(Me) & commitment(Me,m1,Process)
   <- !quit_mission(m1,Process).
```

A.3 FONTE DO AGENTE DELIVERYMAN

```
 include("check_obl.asl")

!start.
!join.

/* Plans */

+!start
   <- lookupArtifact("grDairyIndustry",GrId); //get id from artifact
      adoptRole(deliveryMan)[artifact_id(GrId)];
      focus(GrId).
-!start
   <- .wait(100);
      !start.

+!join
   <- .my_name(Me);
       joinWorkspace("ora4mas",_).
-!join
   <- .wait(200);
      !!join.

// keep focused on process my groups are responsible for
+process(L)
   <- for ( .member(P,L) )
         lookupArtifact(P,ArtId);
         focus(ArtId)
      .

+!quit_mission(M,S)
   <- .print("leaving my mission ",M," on ",P,"....");
      leaveMission(M)[artifact_name(P)].


/* Initial tasks */

/* Plans */

// application domain tasks
+!deliveryTheMilk<- .print("Deliver The MIlk To Dairy Industry");
        .wait(500).

+!deliveryThePackage<- .print("Deliver the Packages to Super Market");
        .wait(500).

// conditions to leave missions
@lgss1[atomic]
+taskState(Task,end,_,_,executed)
    : .my_name(Me) & commitment(Me,m2,Process)
   <- !quit_mission(m2,Process).
```

A.4 FONTE CHECK_OBL.ASL

```
// plans to handle obligations
+obligation(Ag,Norm,committed(Ag,Mission,Process),Deadline)
    : .my_name(Ag)
   <- .print("I am obliged to commit to ",Mission," on ",Process);
      commitMission(Mission)[artifact_name(Process)].

+obligation(Ag,Norm,executed(Process,Task,Ag),Deadline)
    : .my_name(Ag)
   <- .print("I am obliged to executed task ",Task);
      !Task[process(Process)];
      taskExecuted(Task)[artifact_name(Process)].

+obligation(Ag,Norm,What,Deadline)
   : .my_name(Ag)
   <- .print("I am obliged to ",What,", but I don't know what to do!").
```

APÊNDICE B – Especificação organizacional do exemplo de fabricação do queijo

Neste apêndice é apresentada a especificação organizacional em formato XML. Inicialmente é mostrada a especificação estrutural, em seguida a especificação funcional e finalmente a especificação normativa de um processo para fabricação de queijo.

B.1 FONTE DA ESPECIFICAÇÃO ORGANIZACIONAL

```
<?xml version="1.0" encoding="UTF-8"?>
<organisational-specification
    <structural-specification>
        <role-definitions>
         <role id="manager" />
            <role id="farmer" />
            <role id="deliveryMan"/>
            <role id="worker"/>
        </role-definitions>
        <group-specification id="time">
            <roles>
                <role id="manager" min="0" max="1"/>
            </roles>
            <links>
               <link from="farmer"  to="deliveryMan" type="authority"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
                <link from="farmer" to="farmer" type="communication"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
                <link from="worker" to="worker" type="communication"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
            </links>
            <subgroups>
              <group-specification id="dairyIndustry" min="1" max="1">
                  <roles>
                   <role id="farmer"   min="0" max="5" />
                   <role id="deliveryMan"   min="0" max="2" />
                   <role id="worker"   min="0" max="5" />
                  </roles>
                </group-specification>
            </subgroups>
         </group-specification>
    </structural-specification>

    <functional-specification>
        <process id="dairyIndustry" >
               <task id="end" >
                   <task id="milkACow" min="1" ds="Milk a Cow" />
                   <task id="deliveryTheMilk" min="1" ds="Deliver the Milk to Dairy Industry"
                   <task id="makeACheese" min="1" ds="Make A cheese" />
                   <task id="deliveryThePackage" min="1" ds="Deliver The Packages" />
               </task>
            <mission id="m1" min="0" max="5">
                   <task id="milkACow" />
            </mission>
            <mission id="m2" min="0" max="5">
                <task id="deliveryTheMilk" />
                <task id="deliveryThePackage" />
            </mission>
```

```
            <mission id="m3" min="0" max="5">
                   <task id="makeACheese" />
            </mission>
         </process>
    </functional-specification>

     <normative-specification>
        <norm id="n1" type="obligation" role="farmer"     mission="m1" />
        <norm id="n2" type="obligation" role="deliveryMan" mission="m2" />
        <norm id="n3" type="obligation" role="worker" mission="m3" />
    </normative-specification>
</organisational-specification>
```