COPPE/UFRJ

GESTÃO AUTONÔMICA DE PROCESSOS DE NEGÓCIOS

Pedro Calisto Luppi Monteiro Junior

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

Orientador: Jano Moreira de Souza

Rio de Janeiro

Março de 2009

GESTÃO AUTONÔMICA DE PROCESSOS DE NEGÓCIOS

Pedro Calisto Luppi Monteiro Junior

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

Aprovada por:

Prof. Jano Moreira de Souza, Ph D.

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Jonice de Oliveira Sampaio, D.Sc.

Prof. Carlos Alberto Kamienski, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2009

Monteiro Junior, Pedro Calisto Luppi

Gestão Autonômica de Processos de Negócios/ Pedro Calisto Luppi Monteiro Junior – Rio de Janeiro: UFRJ/COPPE, 2009.

XII, 113 p.: il.; 29,7 cm.

Orientador: Jano Moreira de Souza

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2009.

Referencias Bibliográficas: p. 105-111.

1. Computação Autonômica. 2. Processos de Negócios. I. Souza, Jano Moreira de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Titulo.

Agradecimentos

Agradeço em primeiro lugar ao meu orientador Jano Moreira de Souza e ao meu co-orientandor José Rodrigues Neto a toda ajuda dada no desenvolvimento desta dissertação.

Aos professores Xexéo, Jonice e Kamienski por aceitarem participar da banca de avaliação.

A todos da linha de computação autonômica por interagirem os seus trabalhos com o meu: Marcelino, Leonardo, Wallace e Luciano.

Por último, eu agradeço a minha família e aos meus amigos pelo apoio dado no decorrer deste trabalho.

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

GESTÃO AUTONÔMICA DE PROCESSOS DE NEGÓCIOS

Pedro Calisto Luppi Monteiro Junior

Março/2009

Orientador: Jano Moreira de Souza

Programa: Engenharia de Sistemas e Computação

A necessidade das organizações por respostas cada vez mais instantâneas às mudanças do mercado, associado ao seu dinamismo, e à procura por uma melhor posição estratégica, requer novas abordagens de gestão de processos. Os princípios da Computação Autonômica de auto-configuração, auto-cura, auto-otimização e auto-proteção podem ser adaptados a isso. Essa dissertação propõe uma gestão autonômica de processos de negócios através do ABPM (*Autonomic Business Process Manager*), que inclui um modelador e uma máquina de workflow com múltiplos agentes de software, baseada em regras de negócios. Ainda, foi gerada como contribuição do trabalho uma ontologia sobre processos de negócio, com problemas e ações.

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

AUTONOMIC BUSINESS PROCESS MANAGEMENT

Pedro Calisto Luppi Monteiro Junior

March/2009

Advisor: Jano Moreira de Souza

Department: System and Computing Engineering

Organization necessity for prompt responses, to cope with constant market changes, associated with its dynamics for always developing an improved strategic position, calls for new approaches to process management. Autonomic Computing principles (Self-Configuration, Self-Healing, Self-Optimization and Self-Protection) can be adapted to help organizations survive in such scenarios. This work proposes an autonomic business process management through ABPM, a tool that includes a modeler and a multiagent rule-based extension to workflow management systems. We show the viability of this proposal, and analyze its impacts. Other contributions of this work are ontology of business process problems and actions.

Índice do Texto

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

Este capítulo se inicia com a apresentação da motivação, do problema, dos objetivos, da justificativa e das contribuições da dissertação. Posteriormente, é exibida a metodologia utilizada e a definição do escopo da proposta. Por fim, é exposta a estrutura de divisão do trabalho.

1.1 – Motivação

Nesses últimos 50 anos, é inegável que a sociedade tem requerido, cada vez mais, resultados instantâneos (Murch, 2004) e isso se reflete nos negócios. Uma organização não pode desperdiçar tempo a espera de uma resposta. Como um exemplo disso, citado por Murch (2004), existem alguns restaurantes no Japão que, devido ao fato dos patrões quererem que os empregados comam rapidamente, cobram por tempo em que o freguês permanece no restaurante, ao invés de cobrarem pelo que consomem. Assim, os trabalhadores ficam mais tempo na empresa.

Esse mundo, agora, é confrontado com uma nova economia baseada em conhecimento, onde o dinamismo do processo é alto e seu impacto no gerenciamento é inevitável. A Computação Autonômica (Horn, 2001) surgiu para lidar com o aumento da complexidade dos sistemas computacionais. Esta dissertação propõe a utilização da computação autonômica para suportar a gerência de uma organização, pois é um modelo adequado nesse cenário dinâmico.

O termo autonômico se origina na medicina onde o sistema autonômico humano é definido como aquela parte do corpo responsável pela respiração, digestão e defesa de germes e vírus, sem que o ser humano tenha consciência do que se está fazendo, ou seja, sem executar ordens diretamente.

As organizações devem ter processos que se auto-gerenciem e se auto-adaptem às mudanças e aos desafios do mercado, criando assim uma gestão autonômica de processos de negócios. Portanto, a combinação da gerência de processos com os princípios da computação autonômica abre novas oportunidades a serem exploradas.

1.2 – Problema

Considerando essa necessidade de gerência dinâmica, pode-se observar um problema nos sistemas de workflows atuais. Neles são feitos encadeamentos de tarefas, que devem ser seguido à risca. Mas isso gera um problema: o que fazer quando uma tarefa não puder ser completada devido a algum problema do decorrer de sua execução? Simplesmente acabar com o workflow? Essa pode até ser uma ação plausível em alguns casos, porém nem sempre será. Esse problema normalmente não é tratado, e quando o é, é feito de maneira incompleta.

O problema em estudo é a ocorrência de uma exceção no decorrer de uma tarefa gerando uma anomalia e, assim, inviabilizando a finalização da atividade, de tal modo que não é possível redirecionar o workflow para um fluxo alternativo. Um exemplo dessa exceção que não permite a conclusão da tarefa seria a falta de algum recurso (sem papel não se faz um caderno). Ou então, como exemplo de exceção que gera uma anormalidade seria um custo acima do esperado, entretanto, caso não seja tratado no decorrer da execução da tarefa, corrigindo a causa, o custo poderia ficar insustentável para a organização.

1.3 – Objetivos

O objetivo principal é contribuir para automatizar a gestão de processos de negócios melhorando do tratamento das exceções lançadas em um workflow através de sua reconfiguração, tratando os impactos negativos gerados pelas anormalidades que

venham a ocorrer, assegurando que os recursos do workflow sejam disponibilizados de forma a minimizar a quantidade de problemas e protegendo o fluxo garantindo que será finalizado, ou seja, usando conceitos da Computação Autonômica.

Considerando esses objetivos, espera-se que o gerente responsável por um workflow possa modelar mais facilmente o workflow, sem ter que se preocupar, a todo o momento, em criar em cada atividade fluxos extras e repetitivos causados por exceções. A preocupação agora é definir regras autonômicas de negócios. Essas regras de negócio, na verdade, são as políticas de uma organização. Por exemplo, em um determinado tipo de workflow, o tempo pode ser crucial e cada atividade, quando atrasada, deverá ter mais um executor alocado. Nos sistemas atuais, deve-se modelar uma decisão em cada tarefa verificando se há atraso e em casos positivos uma nova atividade de alocar novo executor deveria existir. No sistema desenvolvido nesta dissertação, não é necessário criar várias decisões no fluxo, mas, sim, criar regras através das interfaces determinadas.

Com as regras autonômicas, também se espera uma diminuição da intervenção gerencial na execução do workflow, uma vez que o sistema saberá exatamente como agir quando uma determinada exceção ocorrer.

Com o intuito de atingir esses objetivos, é proposto o Gestor Autonômico de Processos de Negócios chamado de ABPM (*Autonomic Business Process Manager*). ABPM é um sistema de workflow que trata autonomicamente exceções que ocorram na execução de uma atividade ou do processo, seja corrigindo a execução da mesma ou propondo uma melhoria (otimização) no workflow.

Esse sistema possui uma arquitetura multi-agentes, baseada em regras para prover gerência autonômica de processo. Adicionalmente, a arquitetura possui múltiplos níveis, e com isso provê escalabilidade à solução, pois ela pode ser ampliada

futuramente a graus mais altos de abstração, mais perto do nível estratégico da organização, como no caso do *Balanced Scorecard* (Kaplan e Norton, 1996).

1.4 – Justificativa

Ao invés de ser feito de maneira automática por um sistema, as exceções podem ser monitoradas e corrigidas diretamente por um gerente. Porém, além de ser um trabalho extremamente difícil (como saber, por exemplo, se cada tarefa está atrasada ou com algum recurso em falta), ele também é custoso, pois a mão-de-obra gerencial é uma das mais caras e sobrecarregadas em uma organização. Por isso, esse workflow auto-gerenciável visa a diminuir a necessidade de intervenção (e esforço) gerencial. Pode acontecer de um determinado problema não poder ser tratado de maneira automática pelo sistema e, assim, um aviso é gerado a algum especialista, pois, em determinadas situações, a avaliação humana torna-se indispensável para a finalização correta da tarefa.

1.5 – Contribuições

Algumas contribuições foram geradas no decorrer do trabalho. Entre elas se destacam uma arquitetura de um sistema para gestão autonômica de processos, que foi implementada através do **ABPM** e da máquina autonômica de workflow nomeada de **AWE** (*Autonomic Workflow Engine*). Como conseqüência da arquitetura, o **diagrama de classes da aplicação** foi gerado, assim como os **diagramas de classes e atividade dos agentes de *software*** codificados. Finalmente, foi desenvolvida uma **ontologia sobre processos de negócios**, envolvendo os problemas que podem ocorrer do decorrer da execução e as ações a serem executadas no workflow.

1.6 – Metodologia

A partir da idéia principal, onde cada um dos processos teria propriedades autonômicas, foi realizado o primeiro passo, no qual uma pesquisa bibliográfica foi feita a fim de levantar trabalhos existentes relacionados a esta proposta.

O segundo passo foi à definição de um problema exemplo onde os conceitos poderiam ser aplicados. A partir desse exemplo, foi feita, na terceira etapa, a arquitetura geral do ABPM e, posteriormente, na quarta fase, foi codificado um sistema computacional que a suportasse. Para conseguir implementá-lo, alguns produtos foram gerados no decorrer do desenvolvimento. O primeiro deles foi o diagrama de classe da aplicação que, com algumas alterações, pode ser aplicado a qualquer sistema de workflow. Também foram geradas como contribuição duas ontologias, uma sobre os problemas que podem ocorrer do decorrer da execução do processo e outra sobre ações devem ser tomadas com intuito de eliminar esses problemas ou diminuir seus impactos negativos.

Após a implementação da arquitetura, o quinto passo foi a definição e a execução de um estudo de caso com o sistema desenvolvido. Esse estudo de caso envolveu alunos do Curso de Ciência da Computação da Universidade Federal do Rio de Janeiro e teve como objetivo comparar a modelagem com e sem as características autonômicas.

1.7 – Definição de escopo

Esta dissertação se propõe a trabalhar com a auto-configuração de workflows através de ações que podem alterar o fluxo de atividades. A auto-cura do fluxo de trabalho é encontrada com o tratamento automático de problemas. A auto-otimização é feita com a disposição dos recursos de forma mais otimizada possível. Finalmente, a

auto-proteção com relação ao negócio, ou seja, garantindo que o processo irá chegar ao seu fim.

Uma restrição do escopo é a codificação da redefinição de workflows através da procura de um novo fluxo. Apesar dessa ação estar disponível tanto na arquitetura quanto nas interfaces da aplicação, o algoritmo para a mesma não foi desenvolvido. Para a definição da otimização de workflows, pode-se usar como ponto inicial o trabalho de Pankratius e Stucky (2005), que formaliza mecanismos para verificação de similaridade e alteração de workflows, assim como a proposta de Muller (2002) que propõe uma adaptação dinâmica de workflow.

Este trabalho é parte de um maior, chamado de TEAM (*Tool for Enterprise Autonomic Management*) (Monteiro Jr *et AL*, 2008). A idéia desse projeto é prover uma Empresa Autonômica, ou seja, a partir dos processos de negócios, a organização pode obter graus elevados de gerência autonômica, como no caso do *Autonomic Balanced Scorecard* (Rodrigues Nt., 2007) ou, então, com a mineração de um arquivo de *log* com as regra e os problemas encontrados, para assim o sistema extrair algumas conclusões e assim reconfigurar o processo, seja em seu fluxo ou em seus recursos utilizados (Santos, 2008).. Também está inserida no TEAM a rede social autonômica e o *Autonomic Data Killing* (Pinheiro et al, 2009), que elimina da organização os dados desnecessários, facilitando assim a análise de informações.

1.8 – Estrutura da dissertação

Essa dissertação está dividida da seguinte forma: primeiramente uma introdução, seguida do Capítulo 2 com os conceitos, tecnologias e trabalhos relacionados. No Capítulo 3 é explicada a proposta e a arquitetura da solução e no seguinte são apresentados detalhes do sistema desenvolvido. Posteriormente, no Capítulo 5 é

apresentado um estudo de caso para validação e teste da ferramenta de modelagem, e, finalmente, é encerrada com a conclusão e com propostas para trabalhos futuros.

Capítulo 2 – Conceitos e tecnologias relacionados

Este capítulo trata de alguns conceitos fundamentais para o entendimento desta dissertação. Primeiro os dois principais assuntos da pesquisa são descritos: Computação Autonômica e Gestão de Processos de Negócios. Posteriormente, são explicadas as tecnologias utilizadas na definição e implementação da arquitetura: o sistema de workflow JBPM (*Java Business Process Management*), agentes de *software* e quadros negros. Por fim, é feito um resumo geral dos principais trabalhos relacionados.

2.1 – Computação Autonômica

O termo autonômico significa de acordo com dicionário Michaelis (Melhoramentos, 2004), que quem o possui tem autonomia, ou seja, um sistema autonômico é aquele que possui independência funcional. Esse termo se origina na medicina onde sistema autonômico humano é definido como aquela parte do nosso corpo que é responsável por tudo aquilo que o ser humano executa sem ordens explícitas como, por exemplo, a respiração, a digestão e a defesa de germes e vírus. O termo Computação Autonômica surgiu justamente derivado dessa definição (Murch, 2004).

A IBM foi quem pela primeira vez introduziu esse conceito na área da computação, e a empresa divulgou seu trabalho de pesquisa em Computação Autonômica em março de 2001 no discurso do seu então Vice-Presidente Sênior e Diretor de Pesquisa, Paul Horn, na Universidade de Harvard (Murch, 2004). Nessa palestra foi liberado o Manifesto da IBM, no é explicado todos os conceitos dessa nova tecnologia (Horn, 2001).

O manifesto da IBM (Horn, 2001) enumera algumas características que um sistema deve possuir para ser considerado autonômico:

- conhecer a si mesmo e seus relacionamentos: quando um recurso relacionado não estiver disponível, o sistema autonômico deve ter a capacidade de modificar para o outro. Logo, para saber quando um recurso vier a falhar, é necessário monitorá-lo e isso só é possível se o sistema conhecer seus relacionamentos.

- saber se configurar e re-configurar de acordo com condições variáveis e não previsíveis: quando modificar um recurso, o sistema saber configurá-lo para seu perfeito funcionamento.

- não se estabilizar com o estado atual: sempre procura por melhorias a serem feitas a partir de *feedbacks* da execução e de acordo com métricas definidas.

- ter propriedades de cura: o sistema deve saber encontrar problemas e achar caminhos alternativos para assegurar que execute sem percalços.

- se proteger: depois da criação da Internet, ataques maliciosas são cada vez mais normais, e um sistema autonômico deve detectar, identificar e proteger contra essas tentativas de invasão.

- conhecer o ambiente externo e o contexto que cerca suas atividades: um sistema autonômico deve informar a outros o seu estado e, também, descobrir novos fornecedores no ambiente.

- não deve existir isolado no mundo: por definição, um sistema autonômico não deve usar soluções proprietárias, e, sim, usar padrões abertos a todas as pessoas.

- sempre aperfeiçoar seus recursos: a complexidade deve ficar oculta ao usuário final, já que a proposta de autonomia é reduzir a complicação dos sistemas.

A computação autonômica tem algumas divisões e a mais aceita no meio comercial e acadêmico é em quatro dimensões: auto-configuração, auto-cura, auto-

otimização e auto-proteção (Murch, 2004). Essas dimensões são conhecidas pela sigla CHOP, onde cada letra representa o nome de uma dimensão em inglês (Configuration, Healing, Optimization e Protection). Alguns trabalhos, como Babaoglu *et al* (2005) propõem outras, até mesmo a chamada auto-* (auto-estrela), que inclui qualquer capacidade que um sistema pode fazer automaticamente.

A **auto-configuração** é a capacidade do sistema se instalar, se ativar e se atualizar automaticamente usando técnicas de configuração dinâmica de software. Isso significa identificar e documentar as características do item configurado. Essa dimensão também deve possibilitar a adaptação automática do sistema a mudanças no ambiente. Alguns exemplos citados por Murch (2004) são: instalação de novas versões; instalação, teste e entrega de pacotes de serviços; e instalação de pacotes, correções e modificações, juntamente com os testes necessários.

A **auto-cura** é a habilidade de descobrir, diagnosticar e corrigir potenciais problemas, para, assim, assegurar que esses sistemas executem sem interrupções. Com a complexidade e a grande quantidade de informações, podem ser levadas horas para se identificar e corrigir um problema, pois é consumido um grande tempo analisando *logs* e arquivos em memórias manualmente. Sistemas com auto-cura analisam e executam uma ação no exato momento em que ela ocorre diminuindo drasticamente o tempo perdido. Como exemplo, é citada a falha de índice de uma base de dados. Nesse caso, o sistema irá gerar um novo índice, testará e recarregará a base de dados trazendo o sistema de volta a produção. Outro exemplo seria a expansão de espaço de armazenamento de uma base de dados ou de um arquivo, de acordo com o crescimento da quantidade (volume) de dados.

A **auto-otimização** se caracteriza pela não comodidade com estado atual. Nessa dimensão, os objetivos pré-definidos serão sempre monitorados e o desempenho do

sistema para assegurar que o mesmo execute de forma ótima. Essa dimensão também trata da alocação e monitoramento de recursos da maneira mais adequada. Também será avaliado se com uma mudança no ambiente, algum progresso pode ser feito no sistema. Como exemplo, tem-se o balanceamento de carga de trabalho com a requisição de processamento extra para um determinado recurso. As estratégias de alocação de recursos também seriam um exemplo.

A **auto-proteção** diz respeito aos aspectos de segurança em todos os níveis do sistema. Essa dimensão envolve técnicas e algoritmos de criptografia, protocolos de segurança e mecanismos para monitorar e manter a integridade do sistema. O sistema deve antecipar, detectar, identificar e se proteger contra eventuais ataques. A auto-proteção trata, também, da identificação, detecção e proteção de propriedades de valores corporativos. Exemplos seriam cópia e recuperação de recursos, implementação de camadas de segurança, desligamento da rede quando houver tráfego suspeito e resolução de qualquer problema relativo à segurança.

Murch (2004) dividiu a computação autonômica em cincos níveis (básico, gerenciado, preditivo, adaptativo e autonômico). Esses níveis são evolutivos, onde o nível mais alto é uma evolução do penúltimo, o do penúltimo do antepenúltimo e assim por diante.

No nível básico, os profissionais são os responsáveis por monitorar os recursos e propor e executar soluções. Já no nível gerenciado, tem-se a coleta dos dados por sistemas e a exibição em alguns poucos consoles. No nível preditivo, além do monitoramento, é feita uma correlação entre os dados e o sistema propõe uma solução. A execução da ação que vai solucionar o problema encontrado é feita no nível adaptativo. Finalmente, no nível autonômico, há uma total integração do sistema com as regras de negócios e políticas da organização.

Kephart e Chess (2003) dizem que um sistema autonômico será, na verdade, uma coleção de elementos autonômicos que são definidos como sistemas individuais que contém recursos e entregam serviços a humanos ou outros elementos autonômicos. Seu comportamento interno, assim como seus relacionamentos, será gerenciado de acordo com políticas estabelecidas pelos humanos ou outros elementos autonômicos.

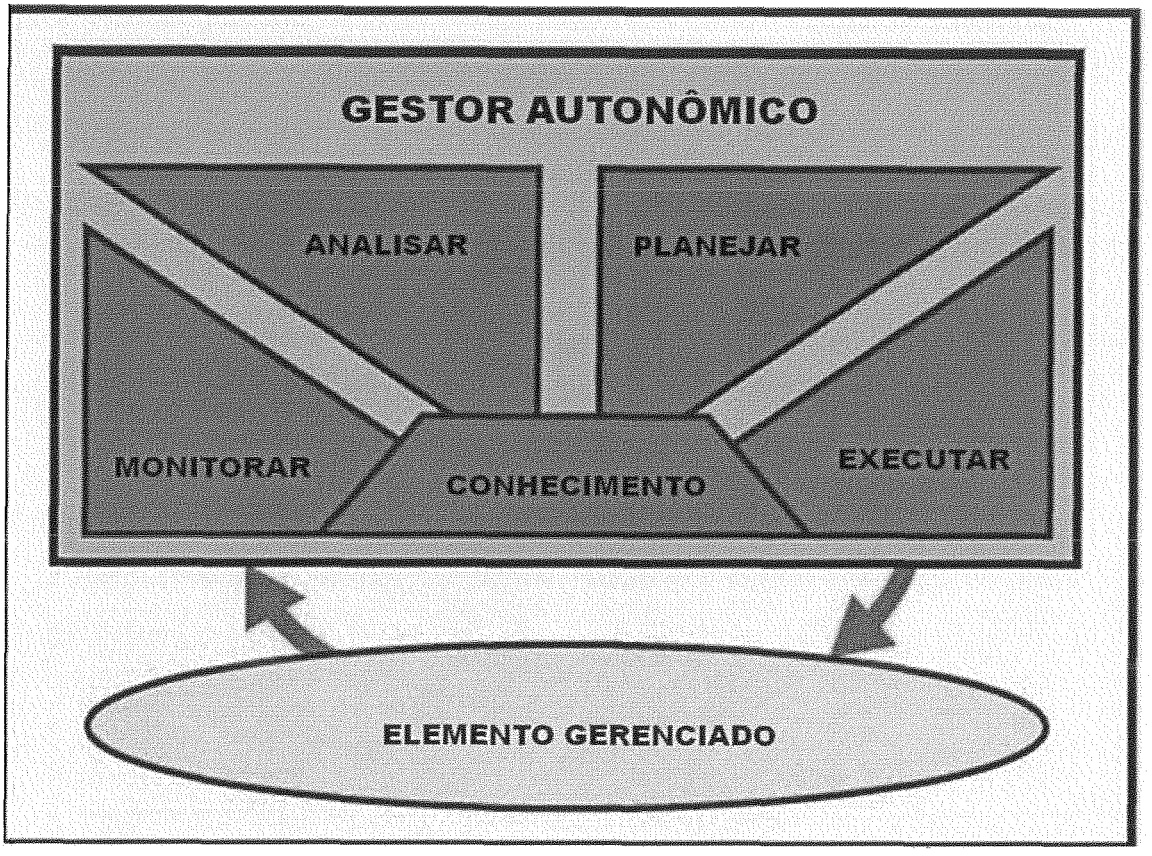


Figura 1 - Elemento Autonômico (Kephart e Chess, 2003)

A representação do elemento autonômico, mostrada em Kephart e Chess (2003), é vista na Figura 1, onde, na parte inferior se encontra o elemento gerenciado que pode ser de mais baixo nível como um recurso de tecnologia de informação (*hardware* ou *software*) ou de mais alto nível como um processo de negócio. A idéia é que esse elemento gerenciado e o ambiente no qual ele se encontra sejam monitorados e a partir da análise dos dados e informações coletados, juntamente com os conhecimentos que o elemento autonômico possui, sejam propostos planos de ação para uma posterior

execução, facilitando assim aquele que seria responsável pela supervisão do elemento gerenciado.

Para facilitar o entendimento do elemento autonômico, Kephart e Chess (2003) recomendam que, devido à autonomia e a interação voltada a um objetivo, ele seja encarado como um agente de *software*, e um sistema autonômico como um sistema multi-agente. Essa sugestão foi adotada, e como esses agentes de elementos autonômicos trabalharam em conjunto será explicado na definição da arquitetura no Capítulo 3.

O livro de Parashar e Hariri (2006) tem grande destaque na pesquisa de computação autonômica, reunindo trabalhos de diversos autores. Esse livro é dividido em duas partes. Na primeira, são reunidos trabalhos que visam explicar o paradigma da computação autonômica, enquanto que na segunda parte do livro são descritas arquiteturas e aplicações provenientes desse paradigma que mostram como a computação autonômica está sendo estudada e implementada.

Dois *surveys* têm o objetivo de definir o estado da arte da computação autonômica. Nami e Bertels (2007) focam nos sistemas de computação autonômica descrevendo suas características, arquiteturas, desafios e seus efeitos em medidas de qualidade como usabilidade, funcionalidade, segurança, sustentabilidade e portabilidade. Já McCann e Huebscher (2008) mostram que a computação autonômica não é apenas uma sigla da moda, mas sim um paradigma que está sendo usado amplamente na área de ciência da computação. Para isso, eles descrevem trabalhos já desenvolvidos que contribuem para a melhoria da computação, como o gerenciamento autonômico de energia elétrica para sistemas computacionais (Khargharia, Hariri, e Yousif, 2007).

2.2 – Gestão de Processos de Negócios

Para entender o que significa Gestão de Processos de Negócios (BPM - *Business Process Management*) é necessário estabelecer três conceitos básicos: processo, engenharia de processos e modelagem de processos.

Uma dificuldade em Gestão de Processos de Negócios é encontrar uma definição rigorosa do que significa processo. A definição aqui utilizada é a de Davenport (1994), onde diz que "um processo é [...] uma ordenação específica das atividades de trabalho no tempo e no espaço, com um começo, um fim e *inputs* e *outputs* claramente identificados: uma estrutura para a ação". Um processo de negócio contém um ou mais processos, e já processo é "qualquer atividade feita dentro da organização" (OMG, 2009). A OMG (2009) também define atividade como um termo genérico para qualquer trabalho dentro da organização, sendo classificado em processos, subprocessos e tarefas. A Figura 2 mostra a representação de processos e tarefas utilizados.

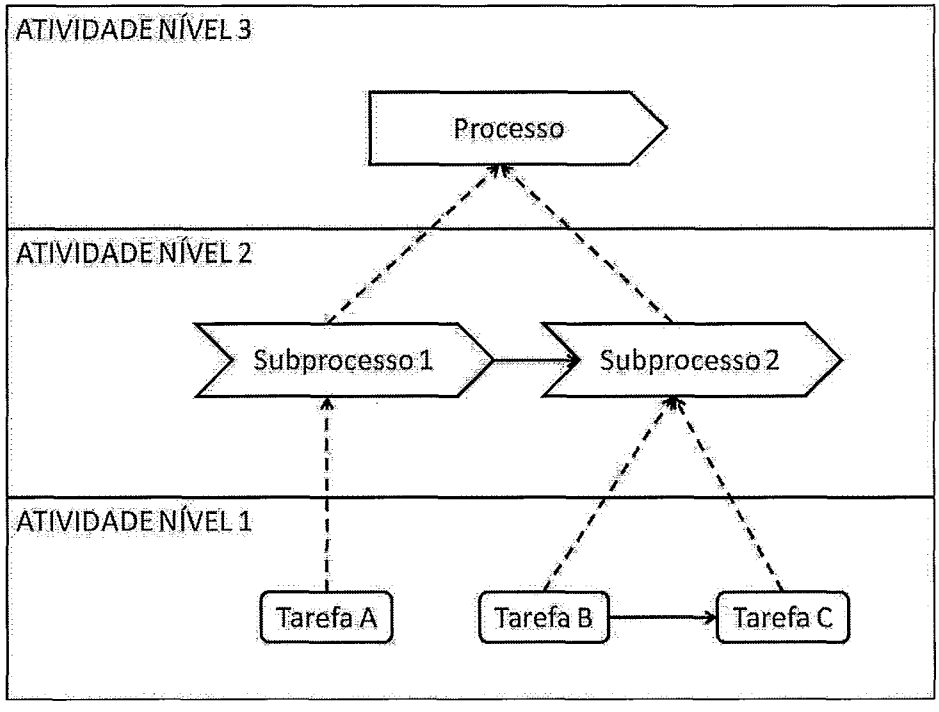


Figura 2 - Definição de atividades, processos e tarefas

Engenharia de processos pode ser entendida, segundo Paim (2002), "como uma arquitetura (*framework*) para entendimento, análise e melhoria dos processos dentro e dentre organizações". A Engenharia de Processos, de acordo o autor, habilita as empresas a se adaptarem mais rapidamente às mudanças do ambiente em que atuam através da gerência de seus processos. Ela deve prover conceitos e instrumentos para possibilitar a efetiva, eficaz e mínima utilização dos recursos, e para que os produtos ou serviços sejam gerados de forma eficiente, alcançando, assim, os resultados esperados e assegurando a sobrevivência das organizações em um novo contexto.

A modelagem dos processos é um instrumento essencial para a Engenharia de Processos. Paim (2002) lista alguns dos objetivos de uma modelagem de processos como propiciar um melhor entendimento e representação uniforme da empresa, suportar o projeto de novas partes da organização e prover um modelo para controlar e monitorar as operações da empresa.

Aalst, Hofstede e Weske (2003) apresentam o ciclo de vida do BPM e definem as fases de apoio a operacionalização dos processos de negócios: planejamento, configuração, execução e análise. A primeira fase é a de planejamento, onde os processos são levantados e modelados. Na configuração, o processo é colocado em um sistema de informação através dos seus modelos. Na execução, os processos de negócios são realizados no sistema de informação configurado. E a última fase é a de análise, onde os processos são analisados para se identificar problemas, gargalos e melhorias.

Um conceito altamente relacionado com processos de negócios é o de workflow. Na Figura 3, Aalst, Hofstede e Weske (2003) comparam a gerência de ambos levando-se em conta o ciclo de vida do BPM. Pode-se observar que a diferença básica entre eles

é que a gestão de processos de negócios suporta todas as fases, enquanto que a de workflow suporta as fases de planejamento, configuração e execução.

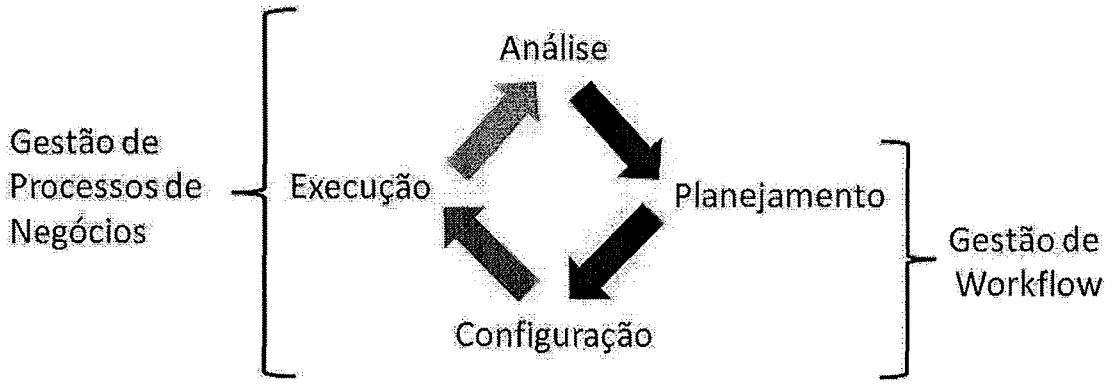


Figura 3 - Ciclo de vida do BPM para comparação entre gerência de workflow e processos (Aalst, Hofstede e Weske, 2003)

Nesse ponto, faz-se necessário uma breve explicação do que são workflows. Para falar em workflow, é necessário citar o *Workflow Management Coalition* (WfMC) que é uma organização sem fins lucrativos que foca em estabelecer parâmetros para padrões de definições de workflow. Em seu modelo de referência, Hollingsworth (1995) os define como a facilitação ou automação, através do computador, dos processos de negócios, seja em parte ou como um todo. E ainda expõe que eles devem se preocupar com a automação de procedimentos onde documentos, informações e tarefas são passadas entre os participantes de maneira a contribuir para, ou até mesmo alcançar, um objetivo organizacional.

Sistema de Gerenciamento de Workflow é, de acordo com a WfMC (Hollingsworth , 1995), "um sistema que completamente define, gerencia e executa workflows através da execução de programas de computador cuja ordem de execução é dirigida por uma representação computacional da lógica do workflow". Segundo Hollingsworth (1995), um Sistema de Gerenciamento de Workflow é o responsável pela gerência da seqüência das atividades e a correta invocação do recurso apropriado para executá-lo, seja ele um recurso humano ou recurso de tecnologia de informação. Esses sistemas são chamados também de máquinas de workflow.

Hollingsworth (1995) descreve que os Sistemas de Gerenciamento de Workflow devem ter características para prover suporte em três diferentes áreas:

- Construção: onde deve ter funções para definir, possivelmente através de modelagem, o processo e suas atividades.

- Controle em tempo de execução: onde deve ter funções para gerenciar o processo em um ambiente de operação e, também, funções de seqüenciamento das várias atividades.

- Interação em tempo de execução: onde deve ter funções para interagir com pessoas e tecnologias de informação para processamento dos vários passos da atividade.

2.3 – JBPM

JBPM (JBPM, 2006) é uma máquina de workflow (distribuída sobre a licença *Lesser General Public License* - LGPL) desenvolvido pela Red Hat Corporation. JBPM tem três principais módulos relacionados ao escopo do trabalho: o modelador "jPDL GPD" (*Java Process Designer Library-Graphical Process Designer*), a interface de execução denominada "Console Web Application" e uma biblioteca central para ser inserida em programas Java para manipular as informações que dizem respeito ao workflow na aplicação, chamada "jPDL Core Library".

O modelador jPDL GPD é um *plug-in* do Eclipse (Budinsky et al, 2003), que é uma plataforma extensível para desenvolvimento de *software* que permite tanto a codificação quanto a criação de modelos. Nele é possível definir o workflow baseado em diagrama de atividades da UML e o usuário (seja ele um analista de negócio ou um desenvolvedor de sistemas) também informa quem deve executar as tarefas e quais atributos e variáveis de controle pertencem a uma dada tarefa. Cada processo, além da parte visual, tem um documento XML associado, sendo que esse também pode ser alterado diretamente e, assim, reflete as mudanças no desenho do processo. Há sempre

uma sincronia entre modelo e XML. Após a definição completa do workflow, é necessário transmitir o arquivo XML para o Console Web Application.

O Console Web Application, como o próprio nome diz, é uma aplicação *web* e é executada no *middleware* JBoss (Fleury e Reverbel, 2003). Essa aplicação é desenvolvida em Java e o seu principal objetivo é ser uma interface de interação do usuário com tarefas em tempo de execução. O Console Web Application tem uma base de dados para armazenar as informações da aplicação. Originalmente, essa base de dados é descrita em um banco de dados, interno ao JBoss, chamado Hypersonic. Contudo, com algumas configurações, apresentadas no guia do usuário do JBPM (JBPM, 2006), é possível efetuar a migração para outros bancos. Para desenvolvimento do Console Web Application foi utilizada a biblioteca chamada jPDL Core Library.

A biblioteca central para manipulação de informações da aplicação, chamada jPDL Core Library, pode ser utilizada por qualquer aplicação que deseje operar processos e seus fluxos e foi desenvolvida em Java. Essa biblioteca manipula também um arquivo XML com a definição do workflow, que pode ter sido gerado pelo modelador ou por qualquer outra aplicação (desde que seja gerado no formato esperado), salvando-a na base ou carregando em objetos Java.

O ABPM foi desenvolvido como uma extensão do sistema de workflow JBPM, alterando seu modelador, para que o mesmo suportasse a definição de regras autonômicas. Esse modelador é baseado em diagramas de atividades da UML. Outro módulo do JBPM alterado foi à própria máquina de workflow para que suportasse as regras autonômicas construindo assim o AWE (*Autonomic Workflow Engine*). Para implementar as regras autonômicas, foi utilizado um sistema de produção *forward chaining* desenvolvido pela NASA chamado CLIPS (*C Language Integrated Production System*) (Giarratan, 2007). AWE utiliza agentes para monitorar o estado de

cada atividade escrevendo o valor de cada atributo em um espaço comum, representado por quadros negro. Esse quadro negro é monitorado por outros agentes, cada um monitorando uma dimensão da computação autonômica, verificando, via CLIPS, se alguma regra é disparada em função dos valores dos atributos monitorados e, caso afirmativo, ativando a execução de uma ação.

2.4 – Agentes e Quadro Negro

Primeiramente, é feita uma breve explicação dos conceitos de agentes e quadro negro, para posteriormente esclarecer o porquê desse tipo de abordagem ter sido escolhida.

Um agente é algo que tem habilidade para sentir o ambiente onde está inserido e agir nele (Russel e Norvig, 1995). O ser humano pode ser considerado um agente e os seus sentidos (visão, audição, paladar, olfato e tato) podem ser considerados sensores. Já em um robô os sensores podem ser, entre outros, seriam câmeras ou infravermelhos. E os agentes que são utilizados no desenvolvimento da aplicação são os agentes de *software* que têm códigos de *bits* como sensores (Russel e Norvig, 1995). Os agentes de *software* devem ser reativos, sentindo o ambiente e agindo de acordo com as mudanças no mesmo, devem ser autônomos controlando suas próprias atividades, devem ser proativos sempre se orientando para atingir um objetivo e, finalmente, devem ser executados sem interrupções.

Sistemas multi-agentes consistem em vários agentes de *software* que interagem entre si e para obter sucesso requerem a habilidade de cooperação, coordenação e negociação (Wooldridge, 2002).

Já quadro negro, de acordo com Shaw e Garlan (1996), é um estilo de arquitetura de repositório onde entidades fracamente acopladas compartilham um espaço comum. Corkill (1991) exemplifica a idéia de quadro negro pela metáfora de um

grupo de especialistas que estão tentando resolver um problema com o auxílio de um quadro negro de uma sala de aula. Inicialmente, nesse quadro negro está escrito o problema inicial e ele é atualizado por um especialista quando é encontrada uma informação relevante para a solução do problema. Esse processo se repete indefinidamente até que a solução seja encontrada.

Um sistema de quadro negro é dividido em três componentes principais: fonte de conhecimento, quadro negro propriamente dito e um componente de controle (Corkill, 2003).

As fontes de conhecimento são responsáveis por escrever as informações no quadro negro. No exemplo acima seriam os especialistas. Eles são independentes entre si e, em termos computacionais, podem ser de diferentes tecnologias. Na arquitetura proposta, detalhada adiante, no Capítulo 3, foram usados agentes de *software* como fonte de conhecimento. Eles monitoram as tarefas e escrevem seus atributos no quadro negro.

O quadro negro, propriamente dito, nada mais é do que um repositório comum de dados, onde todas as fontes de conhecimento possam ter acesso a ele. Com isso, o quadro negro conterá todos os dados, soluções parciais e outras informações relevantes para a solução do problema.

O componente de controle é separado das fontes de conhecimento e toma decisões em tempo de execução sobre o problema com as informações do quadro negro (Corkill, 2003). Para o componente de controle também foi utilizado agente de *software*, porém esse utiliza o CLIPS, sistema de produção baseado em regras desenvolvido pela NASA, para trabalhar com as regras de negócio.

Corkill (1990) cita as áreas que a abordagem de quadro negro é aplicada, e entre elas a área de controle de processo. O autor também afirma que não se deve utilizar a

arquitetura de quadro negro quando o conhecimento for facilmente representado por uma estrutura trivial, a aplicação não precisar tomar decisões dinâmicas e, também, não se comunicar com outras aplicações. A proposta aqui apresentada vai justamente contra essas três afirmações. Assim, foi decidido pelo uso de quadro negro, pois decisões devem ser tomadas a todo o tempo para corrigir o processo em andamento, o sistema se comunica com outras aplicações via serviços *web* e não há nenhuma estrutura de representação de conhecimento trivial que atenda à necessidade desta dissertação.

Para criação e definição dos agentes a serem utilizados, foi utilizado o modelo da funcionalidade emergente (Steels, 1990), o qual diz que uma funcionalidade não é ativada diretamente por um componente, mas sim indiretamente através de interações entre vários componentes primitivos direta ou diretamente. Cada componente individual tem sua própria interação com o ambiente, e é através de algum indicador do ambiente que esse componente é iniciado. Apesar de um componente ter uma funcionalidade particular, essa não é vista como uma funcionalidade do sistema, mas sim como uma contribuição para uma funcionalidade do sistema.

Como características do modelo emergente, Steels (1990) cita a tolerância a falhas em casos em que a funcionalidade combina o comportamento de vários elementos e a dinamicidade do sistema. A propriedade emergente não é controlada pelo sistema e seu valor pode mudar constantemente. As funcionalidades para controlar processos são decompostas em diversos componentes. Por exemplo, para se monitorar o estado de uma tarefa, cada uma de suas propriedades é monitorada separadamente.

Segundo Serugendo, Gleizes e Karageorgos (2005), um sistema multi-agente focado em auto-organização e emergência pode ter uma abordagem com mecanismos baseados em interações entre os agentes e *stigmergy*. De acordo com os autores, *stigmergy* é a característica em que o "trabalho estimula os trabalhadores". Com isso,

para a realização do controle e dos ajustes, utilizam-se informações que estão no ambiente, depositadas por outros agentes, sem um controle central. Como exemplo desse tipo de comportamento, é citado o caso das formigas, que deixam um feromônio no caminho de onde acharam comida, para que as demais possam achar também.

Assim, dadas as características de um sistema emergente, ou seja, onde uma funcionalidade não é ativada diretamente por um componente, mas sim indiretamente através de interações entre vários componentes, pode-se afirmar que o ABPM é emergente, pois apesar das exceções lançadas serem controladas pelos agentes, não há um controle dentro do comportamento do sistema como um todo e, ainda, há vários agentes de procedimento simples, que juntos compõe uma funcionalidade complexa. Ainda, pode-se observar que o ABPM possui características de *stigmergy*, já que um agente deixa um "rastro" para outro agente quando escreve no quadro-negro do processo pai informações a respeito do processo filho, e esse "rastro" é usado para ativação de regras. Considerando a conduta emergente referente às exceções lançadas, é possível observar que o comportamento do ABPM é diferente do caso de um workflow de condições, visto que não existem um comportamento pré-definido para o sistema, como um algoritmo procedural tradicional.

2.5 – Trabalhos Relacionados

Esta seção primeiramente introduz trabalhos relacionados a propostas de tratamento de exceções em processos de negócios e, posteriormente, apresenta alguns trabalhos de workflows com propriedades autonômicas.

2.5.1 – Exceções em processos de negócios

Um dos trabalhos de maiores destaque é o de Russell, Aalst, e Hofstede (2006) no qual é feita uma classificação das exceções e seus possíveis tratamentos. É uma

proposta geral independente de ferramenta de modelagem, tecnologia e linguagem de execução.

Inicialmente, os autores propõem um *framework* para classificação para tratamento de exceções baseada em padrões. Nessa classificação, primeiramente eles definem os eventos que lançam as exceções:

- falha de um item de trabalho: quando a razão da falha que não está na modelagem do processo, é necessário tratá-la para assegurar que o item de trabalho e o processo possam continuar funcionando.

- expiração de tempo: é útil especificar no projeto o que deve ser feito quando o prazo final não for respeitado.

- recurso indisponível: é normal um item de trabalho utilizar um ou mais recursos de dados em sua execução. Quando um recurso falhar, o item de trabalho não pode continuar.

- evento externo: sinais vindos de fora do processo podem impactar em um item do trabalho e requerem tratamento.

- violação de restrições: restrições são necessárias para manter a integridade e a consistência de um workflow. Quando violadas, alguma ação deve ser executada.

Posteriormente, os autores mostram uma série de ações que podem ser executadas em um item de trabalho, na ocorrência da exceção, nas atividades como exemplo tem-se: continuar a tarefa, realocar itens e re-iniciar, entre outros. Para o nível de processo, eles dividem as ações em:

- continuar executando a instância do workflow;

- parar somente a instância em que ocorreu a exceção; e

- parar todas as instâncias em execução do processo em que a exceção ocorreu.

Após o tratamento da exceção, pode ser necessário que uma ação de restabelecimento seja executada no workflow para corrigir possíveis efeitos colaterais gerados pela ação de tratamento da exceção. Os autores enumeram essas ações como: executar nenhuma ação, voltar ao estado anterior ou compensar os efeitos da exceção.

Com esse *framework* definido, os autores avaliam oito ferramentas de workflow líderes de mercado e chegam à conclusão que elas são limitadas no que diz respeito a tratamento de exceções.

Finalmente, é apresentada uma ferramenta gráfica, independente de tecnologia, para definir estratégias de tratamento de exceções, onde é oferecido suporte para definir e gerenciar derivações do fluxo normal de um workflow.

A proposta de Han, Thiery e Song (2006) é o tratamento de exceções em sistemas de gerenciamento de workflows médicos como diagnóstico, tratamento, terapia, administração de hospital, entre outros. Os autores definem o gerenciamento de exceções como a definição e o tratamento delas, eles também dissertam sobre a dificuldade de achar a granularidade da exceção, para não confundí-la com eventos não tão anormais que poderiam estar no fluxo normal do processo. Também é ressaltado o fato de que as exceções são em níveis de negócio, e não erros do sistema, como falta de memória ou divisão por zero.

Os autores afirmam, ainda, que o tratamento de exceções se divide em três grandes áreas:

- Classificação da exceção, definição de quando ela ocorre, quem a gera, quem a recebe e qual ação deve ser disparada;

- Tratamento da exceção. Como propagar para que outro interessado a possa tratar corretamente, tratar primitivamente com uma ação pré-definida ou tratar com uma ação a ser determinada em tempo de execução; e

- Análise da mesma. Ou seja, identificar todas e verificar, antes de executar o workflow, se todas estão tratadas. Também deve-se checar a eficácia do tratamento dado.

Essas áreas são utilizadas nesta dissertação, já que as exceções são divididas entres os tipos de recursos e são tratadas com uma ação.

Mourão e Antunes (2007) procuram tratar exceções sem tratamento planejado em um workflow. Exceções inesperadas são aquelas que necessitam de intervenção humana, pois elas não podem ser tratadas de forma automática, e, ainda, a organização não tem o conhecimento para resolver o problema. Citam como exemplo desse tipo de exceção a catástrofe nos controle de tráfego aéreo de 11 de setembro de 2001 nos Estados Unidos e a bancarrota de um cliente. Um diferencial dessa proposta é que, além das tradicionais ações de detectar a exceção, diagnosticar e executar uma ação, os autores também afirmam que a ação executada deve ser monitorada. Baseado nesse trabalho, foi desenvolvida uma forma de tratamento de exceções em que uma das ações possíveis seja avisar ao especialista para resolver o problema.

Já o trabalho de Casati *et al* (1999) visa tratar as exceções previsíveis, que se caracterizam por estar em um fluxo anormal do processo, e por isso não podem ser convenientemente modelada no fluxo normal através de tarefas e conexões. Elas também são assíncronas e normalmente são influenciadas por fatores externos. Também é usado o esquema de regras com condição e ação, e ao mesmo tempo com prioridades entre elas. Esse trabalho ainda alertou para a necessidade de formalização das regras.

A tese de doutorado de Muller (2002) concentra seus esforços na predição de falhas e reação a elas. O autor diz que os tradicionais sistemas de workflow não suportam flexibilidade suficiente para lidar com situações de falha que ocorram no decorrer da execução de um processo. Para lidar com isso, é proposto o AgentWork, um

sistema de gerenciamento de workflow baseado em agentes, que lida com a definição, execução e adaptação de workflow. O autor enumera as principais contribuições do trabalho como sendo:

- um modelo formal para definição, execução e estimação (comportamento futuro) de um workflow;

- um mecanismo para determinação e processamento de falhas e, ainda, ações de controle;

- um mecanismo para adaptação do workflow onde se originou a falha; e

- um mecanismo para tratar os workflows que se comunicam e colaboram com o workflow que originou a falha.

Explicado os trabalhos sobre exceções em processos de negócios, podemos citar alguns trabalhos de workflows autonômicos, o que é feito a seguir.

2.5.2 – Soluções Autonômicas para Workflow

No trabalho de Strohmaier e Yu (2006), os autores fazem uma classificação, baseado nos cinco níveis básicos, já explicado na Seção 2.1, descritos por Murch (2004), para sistemas de workflows autonômicos. No nível básico, tem-se a definição, a análise e a adaptação do workflow feita de forma manual pelo projetista. No nível gerenciado, a análise já é automatizada utilizando técnicas de estatísticas e monitoramento. No nível preditivo, o projetista conta com a ajuda de simulação, através de mineração e predição, para tomar suas decisões. O nível adaptativo é alinhado com os objetivos de negócios e sua característica principal é a automatização da adaptação (cura e otimização) do workflow. Se esta dissertação for vista como um trabalho isolado, a princípio ela se enquadraria nesse nível. No entanto, ela está associada a outros, Rodrigues Nt (2007) e Oliveira (2007), e esses trabalhos conjuntamente visam tornar uma organização autonômica, e não somente seus workflows. Assim, pode ser

enquadrada no nível autonômico, pois é alinhada com a visão estratégica da empresa e tem sua governança baseada em objetivos estratégicos.

Um dos trabalhos precursores foi o DynaFlow, publicado por Pinheiro *et al* (2007) e Vivacqua *et al* (2007). Esses trabalhos propõem um sistema distribuído de workflow dinâmico *peer-to-peer*, onde existe um publicador que disponibiliza as tarefas e os conhecimentos necessários a ela, e os executores que recebem todas as tarefas sobre as quais possuam conhecimento para realizar e escolhem as que desejam efetivamente executar.

Alguns trabalhos visam relacionar workflow e computação autonômica, porém com um enfoque diferente. Como exemplo, o trabalho de Lee (2007) que encaixa a adaptação de workflows científicos nos passos do elemento autonômico de monitorar, analisar, planejar e executar, sem tratar diretamente das dimensões autonômicas de autogerenciamento.

O FEEDBACKFLOW é uma arquitetura, proposta por Andrzejak, Herman e Sahai (2005), que também utiliza os passos do elemento autonômico de monitorar, analisar, planejar e executar. Ela visa gerar um workflow adaptativo, a partir de tarefas do sistema operacional como criar uma variável de ambiente, adicionar uma entrada no registro do Windows, iniciar um *script* ou reiniciar o sistema. Durante a execução do workflow, será feito um *feedback* com informações geradas dizendo se as tarefas foram executadas corretamente ou falharam, para, assim, atualizar o estado do sistema ou re-executar o workflow através de um plano alternativo.

Em Mangan e Sadiq (2002), são tratados workflows flexíveis como existem em tratamento de pacientes médicos, ensino, gestão de conteúdo *web* e em relacionamento entre pessoas. Os autores focam em como construir modelos para lidar com essa flexibilidade, porém não lidam com a execução do workflow em tempo real e nem com

possíveis reações para a cura de um problema. No entanto, a análise da definição do processo e suas abordagens de tratamento ajudam a entender a necessidade de se ter um componente que possa interferir na execução da tarefa, a qualquer momento fundamental para a definição da arquitetura do ABPM.

A idéia de Savarimuthu, Purvis e Fleurke (2004) é embutir agentes de *software* em um sistema para monitorar e controlar indicadores de desempenho através de dados obtidos por simulação do workflow. É feito um quadro de aviso com os parâmetros monitorados no sistema, para análise humana.

Buhler, Vidal e Verhagen (2003) propõem que um workflow formado por serviços *web* para ser adaptável deve ser formado por agentes de *software* porque assim não teria conhecimento apenas de si mesmo, mas também do ambiente como um todo. Também seria capaz de entender e utilizar ontologias, assim como seriam capazes de executar uma ação autônoma, de se comunicar e ter um comportamento colaborativo. Com essas características, possíveis problemas e adaptações podem ser resolvidos através das interações entre os agentes. A idéia de correção e adaptação de um workflow com múltiplos agentes pode ser adaptada. Outra idéia deles adequada a esta dissertação é o uso de container para manter o estado do workflow, com a diferença de ser utilizada a arquitetura de quadro negro.

Outro trabalho sobre workflow baseado em serviços *web* é o de Verma e Sheth (2005), no qual eles elevam os conceitos de computação autonômica para o nível de processos, sendo o comportamento do workflow baseado em políticas definidas pelo usuário. Os autores não propõem nenhuma arquitetura e nem apresentam nenhum sistema, apenas citam as características que um processo *web* autonômico pode ter dentro de cada dimensão autonômica, entre elas destacam-se a reconfiguração ótima de acordo com os fornecedores, o descobrimento dos melhores fornecedores de acordo

com as políticas da organização, cura de falha do sistema ou do negócio e a auto-otimização com as possíveis mudanças no ambiente.

Já em Heinis, Pautasso e Alonso (2005) é proposto um projeto juntamente com uma avaliação de uma máquina autonômica de workflow distribuído. Essa máquina automaticamente se configura baseada na carga de trabalho, procurando sempre se manter de forma equilibrada entre o nós. Os autores mostram como o paradigma da computação autonômica pode simplificar a entrega e manutenção de sistemas de workflow distribuído apresentando como seria a distribuição inicial da carga de trabalho e o que seria feito em caso de sobrecarga ou falha de algum nó. Se for considerado que a distribuição em um workflow é em função dos executores ao invés de ser em computadores, pode-se utilizar as técnicas de otimização de recursos para otimizar pessoas, mantendo sempre um equilíbrio entre os executores das tarefas, sem que um seja sobrecarregado e, também, se recuperando de possíveis falhas, ou ausências, de um deles.

Finalmente, um trabalho, que não trabalha com workflows, porém é interessante e vale a pena ser citado, é o de Aiber (2004). Nele, o autor propõe uma arquitetura para alinhar, de forma sempre auto-otimizada, a infra-estrutura de uma empresa com os objetivos de negócios da mesma. Como exemplo, ele cita que em um sítio de vendas um cliente de alto valor estratégico que pode ter mais banda alocada que outro, caso o último não tenha tanto valor. Esse trabalho de Aiber (2004) pode ser aproveitado com o objetivo de se chegar a uma empresa autonômica, para alcançar o nível autonômico proposto por Strohmaier e Yu (2006).

Capítulo 3 – Gestão Autonômica de Processos de Negócios

O início deste capítulo define alguns conceitos essenciais para o entendimento da arquitetura proposta e em seguida apresenta as ontologias de problemas e de ações que o sistema desenvolvido suporta. Posteriormente, é explicada a arquitetura da Gestão Autonômica de Processos de Negócios e de seu sistema gestor chamado de ABPM (*Autonomic Business Process Manager*) e, também, a arquitetura da máquina de workflow autonômico cuja sigla é AWE (*Autonomic Workflow Engine*, em inglês).

3.1 – Definições

Antes do detalhamento da proposta, são introduzidos alguns conceitos fundamentais para o seu entendimento: atributo, fato, recurso, condição, ação e regra. Exemplos dessas definições podem ser vistos na Tabela 1, com exceção de regra, pois ela, como será explicado mais adiante, é que a combinação dos conceitos anteriores.

Tabela 1 - Exemplo das definições de atributo, fato, recurso, condição e ação

Recurso	Atributo	Fato	Condição	Ação
Humano	Pessoas por Capacidade	Java = 1 pessoa	Java < 1 pessoa	Alocar pessoa
Temporal	Duração	Duração = 2 horas	Duração > 2 horas	Cancelar Tarefa

Um atributo é uma característica do processo que seja de interesse aos gerentes ou aos executores e pode afetar a execução do workflow, ou seja, os atributos são as

propriedades da tarefa ou do processo. Eles que são monitorados para se obter o conhecimento sobre o estado da tarefa.

Um fato é o valor de um atributo em um exato momento de tempo, ou seja, é uma verdade sobre o processo. Associado ao fato, existe um valor que pode ser um número inteiro, um texto, um valor booleano ou uma data, de acordo com seu tipo.

Recurso é uma classificação dos atributos. Um atributo pode ser sobre as finanças do processo, ou então, sobre os humanos envolvidos, relativos ao tempo de execução, à infra-estrutura envolvida ou a algum material. E cada uma dessas possibilidades trata de um tipo de recurso.

Uma condição é uma comparação de um fato com um determinado valor, através de algum dos tradicionais operadores de == (igual a), <> (diferente de), > (maior que), >= (maior ou igual que), < (menor que), <= (menor ou igual que). Várias condições podem ser combinadas, e assim é formado um conjunto de condições, e elas são interligadas por meio da conjunção E ou disjunção OU. O padrão utilizado foi a Forma Normal Conjuntiva, conhecida como uma conjunção de disjunções, ou seja, são cláusulas unidas por E, sendo que cada uma delas internamente é composta de vários OU. Por exemplo, (a OU b) E (c OU d) E (e OU f OU g). A escolha por essa representação foi feita para facilitar a construção e o entendimento da condição, separando claramente os literais, e ainda se deveu ao fato de qualquer expressão lógica poder ser definida nesse padrão.

Uma ação é uma intervenção na execução do workflow. Pode visar aperfeiçoá-lo ou então corrigir algum problema detectado. Essas ações podem ser diretas, alterando explicitamente a execução do processo, ou indiretas, avisando ao processo de nível superior que uma determinada condição é válida. Uma ação também pode ser a chamada de um especialista para resolução do problema.

Uma regra é construída usando condições e ações e tem a seguinte forma: "**se** o conjunto de condições é válido **então** execute uma ação". Para cada regra definida também é necessário associá-la a uma prioridade, prevendo assim o caso de quando existir duas condições válidas. Com isso, é possível determinar qual ação será executada. Ressaltando que para um mesmo processo não poderão existir duas regras com a mesma prioridade.

Considerando os conceitos acima, podem-se definir todos os sintomas de problemas que podem ocorrer na execução do processo, assim como as possíveis ações corretivas. E isso é feito através do uso de ontologias.

3.2 – Ontologias de problemas e ações

No decorrer da execução de um processo, podem ocorrer alguns problemas e ações devem ser tomadas com intuito de eliminá-los ou diminuir seus impactos negativos. O mapeamento dos problemas e das ações foi feito através de ontologia, gerando assim mais dois dos produtos deste trabalho.

Antes de detalhar as ontologias de problemas e a de ação, é apresentado na Figura 4 uma visão geral a respeito delas. Primeiramente, ela mostra uma estrutura na qual existe uma atividade que pode ser tanto um processo ou uma tarefa, onde um processo pode possuir vários outros processos, até que se alcance o nível de tarefa. Essa estrutura foi feita de acordo com o padrão de projeto Composite (Gamma *et al*, 1995).

É na atividade onde os problemas são apresentados, logo é ela que possui as regras, que são ativadas por um problema e executam uma ação. No nível de tarefa é onde são utilizados os recursos, que são divididos em: recurso humano, recurso material e recursos de infra-estrutura. São nesses recursos que se têm as falhas. Um recurso humano foi definido como sendo o conhecimento dos executores. O recurso material

pode ser um material propriamente dito, um documento ou uma informação. Já um recurso de infra-estrutura são os sistemas e os servidores.

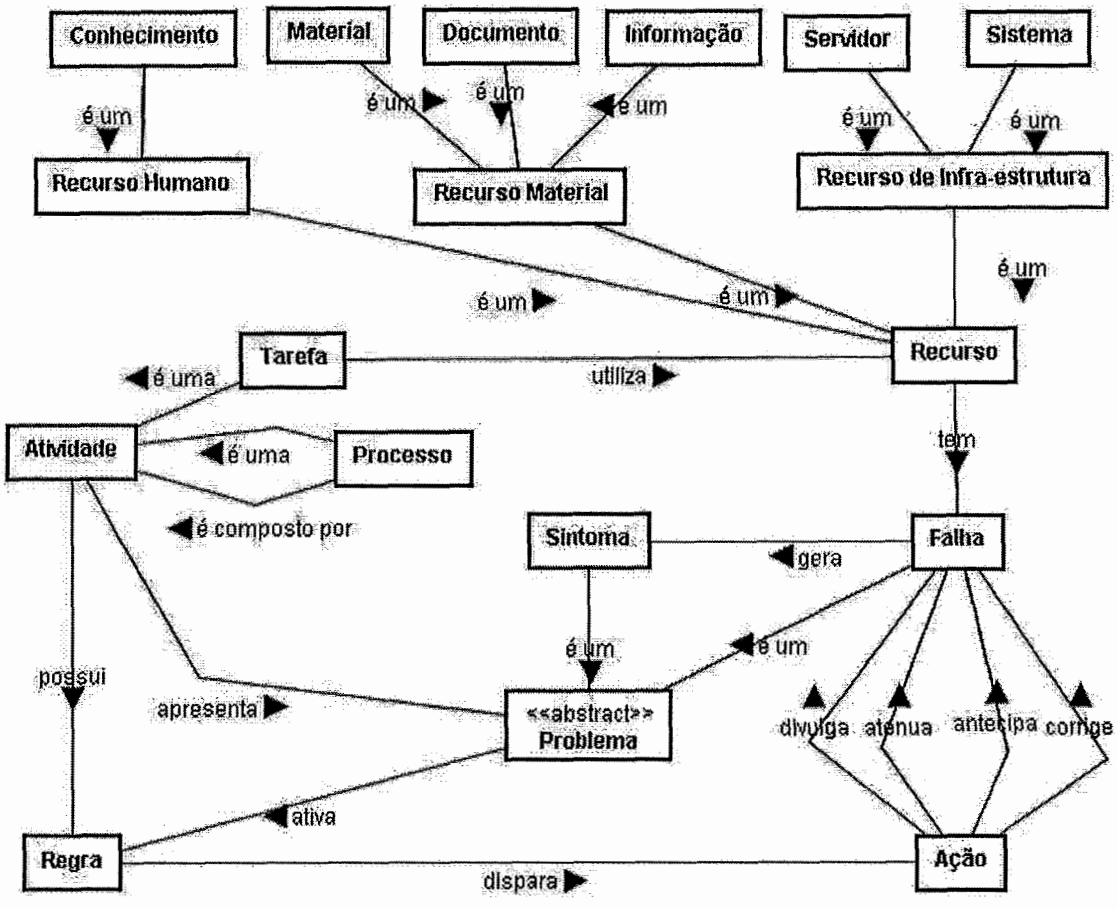


Figura 4 - Visão geral das ontologias

Para representar todos os problemas, foi feita uma ontologia que é mostrada na Figura 5. Problemas são apresentados em uma atividade. Um problema pode ser tanto um sintoma quanto uma falha e esse problema, apresentado por uma atividade, que é responsável pela ativação da regra. As falhas que geram sintomas, em analogia ao corpo humano, uma virose que gera o sintoma febre.

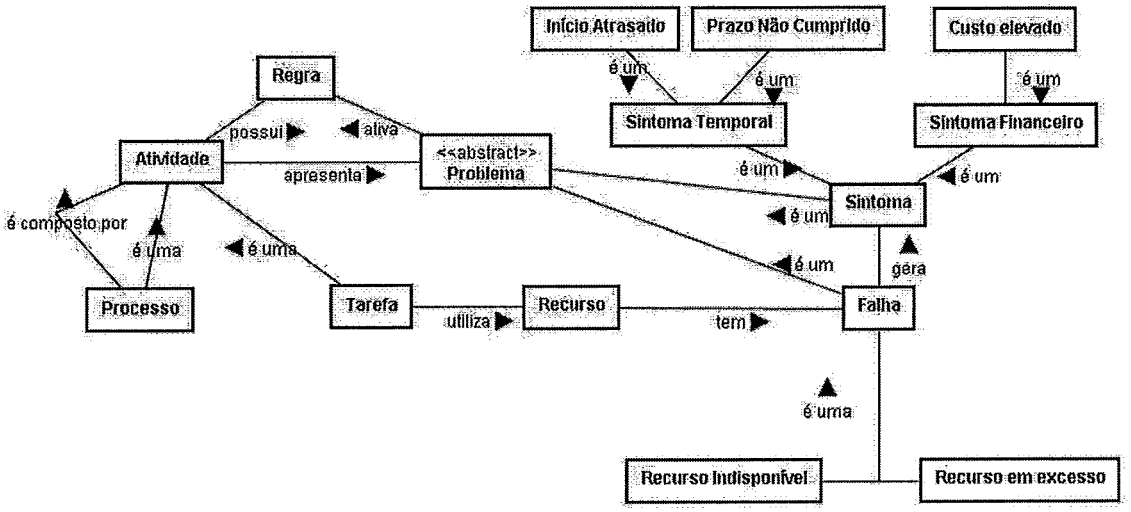


Figura 5 - Ontologias de problemas em um workflow

Eventualmente, falhas podem ser antecipadas pela observação dos sintomas. Os sintomas são divididos entre temporais e financeiros. Com relação ao tempo, um dos sintomas mais tradicionais em uma atividade é o atraso no início da mesma, assim como o não comprimento do prazo. Uma falha pode causar um custo desnecessário, e isso, obviamente, é ruim para a organização.

Os recursos humanos são mapeados de acordo com o conhecimento que a pessoa possui, pois não adianta em nada ter alguém executando uma tarefa se o mesmo não tem competência para tal. Tanto a falta de conhecimento quanto o excesso são falhas em um processo, pois geram um custo ou um desperdício desnecessário.

A infra-estrutura é composta por servidores e sistemas. Os servidores disponibilizam serviços que podem vir a se tornar indisponíveis e os sistemas são usados para busca e cadastro de informações da atividade, e também podem não estar acessível no momento necessário.

Quando a informação é vital a uma atividade, sua falta pode inviabilizar sua execução. O mesmo se aplica a um documento.

Diversas ações, previamente definidas, podem ser tomadas para que o processo seja prejudicado de forma menos custosa possível. Foi feita uma ontologia representando essas ações, e a mesma pode ser visualizada na Figura 6.

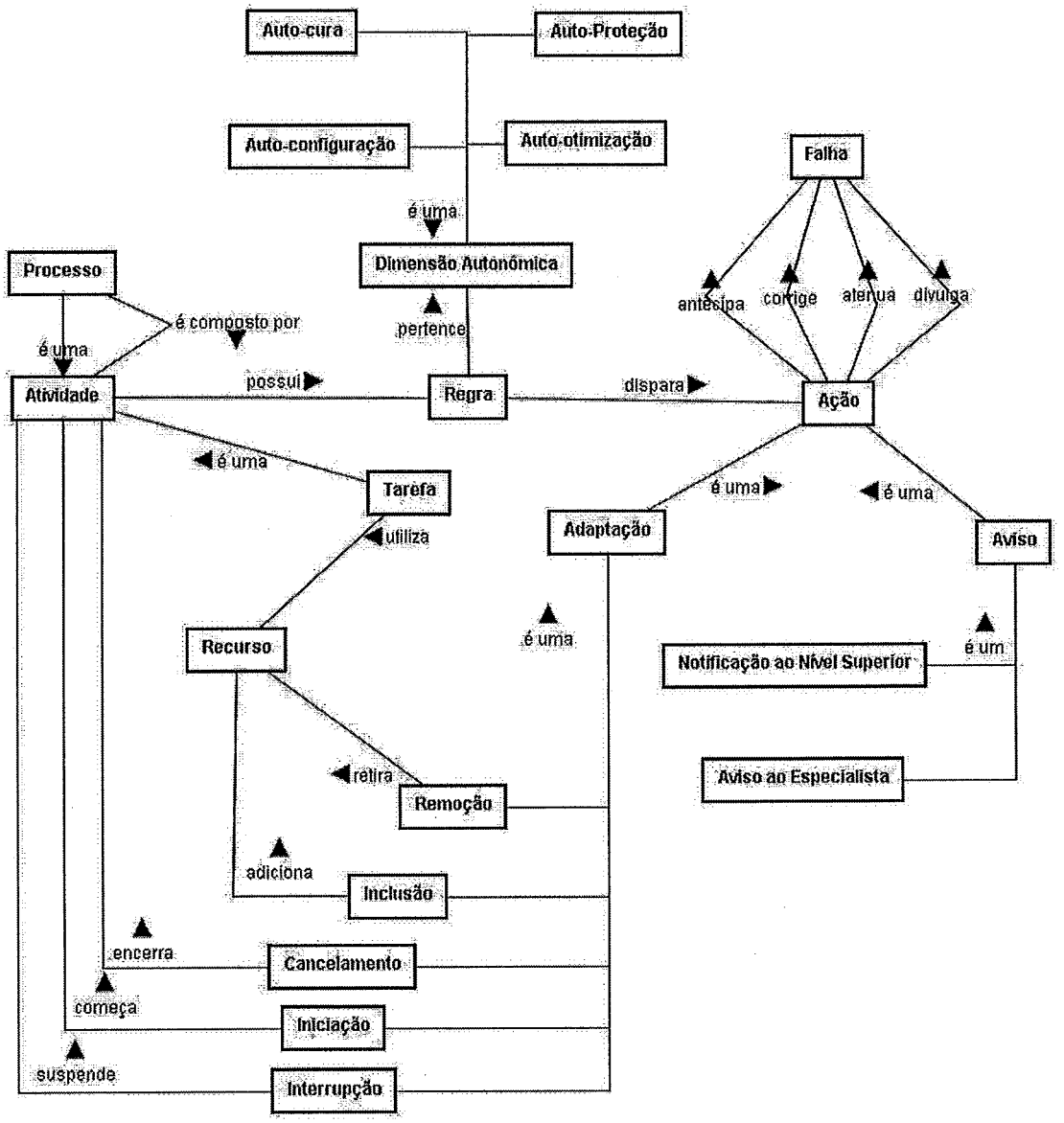


Figura 6 - Ações que podem ser executadas em um workflow

As ações são disparadas por uma regra, que pertence a uma das dimensões autonômicas, com o intuito de antecipar, corrigir, atenuar ou divulgar uma falha. Essas ações se caracterizam por serem de adaptação do workflow ou de aviso da ocorrência da falha.

Um workflow pode ser adaptado quando o fluxo de atividades é alterado, seja (re) iniciando, encerrando uma atividade ou então a suspendendo.

Com relação aos recursos, as ações de remoção podem retirar um recurso e as de inclusão podem adicionar um. Com isso, também se pode ter a ação de substituição de um recurso.

Considerando as ações de aviso, um workflow pode informar (por exemplo, através de correio eletrônico) a um especialista no problema (ou responsável pela atividade) para que o mesmo possa tomar a decisão correta para solução da atividade, quando não puder ter uma ação automática. Ou então a atividade pode avisar o nível superior, em termos de processo, para verificar se existe alguma solução que não seja pontual.

3.3 – Arquitetura da Gestão Autonômica de Processos de Negócios

A idéia da Gestão Autonômica de Processos de Negócios, representada na Figura 7, é que cada processo, qualquer que seja seu nível, possua características autonômicas acoplados a ele e, assim, tendo o chamado comportamento CHOP, o processo consiga se auto-gerenciar dentro das dimensões da computação autonômica, ou seja, tenha capacidade de auto-configuração, auto-cura, auto-otimização e auto-proteção.

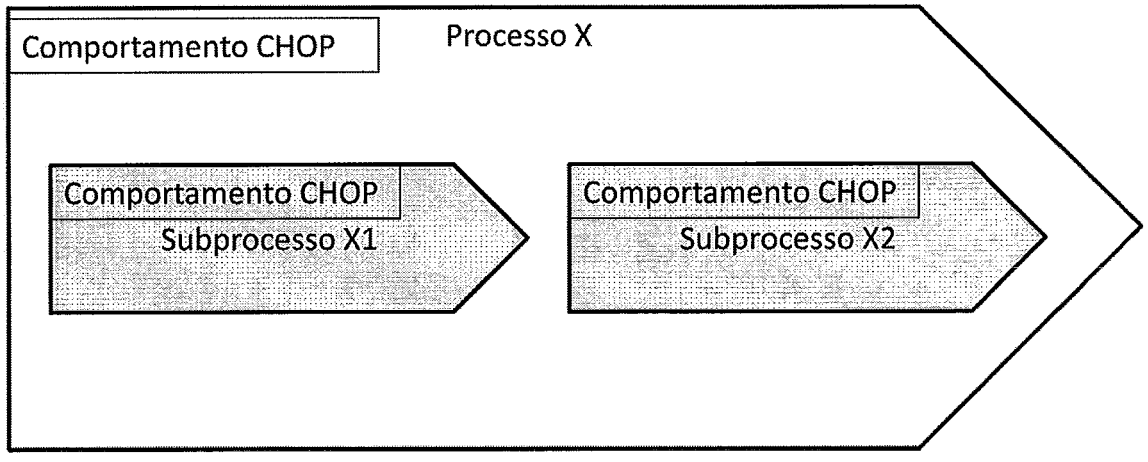


Figura 7 - Idéia Central

Para conseguir concretizar a idéia central, foi feito o Gestor Autonômico de Processos de Negócios (ABPM) que é a união de dois sistemas, o modelador e a máquina autonômica de workflow (AWE). O modelador é onde se definem as atividades e, nelas, as regras autonômicas de negócio, que serão executadas diante de

determinadas condições. O AWE é uma máquina de workflow, que além das funcionalidades convencionais de um sistema de workflow, possui agentes de *software* monitoram o estado de uma atividade e verificam se alguma ação, definida em uma regra de negócio, deve ser executada. O ABPM também se comunica com outros sistemas, via serviços *web*, esses sistemas são gerenciadores de recursos como o GCC (Oliveira, Souza e Perazolo, 2006) para recursos humanos ou o Tivoli (IBM Tivoli, 2008) para recursos de infra-estrutura. A visão geral da Gestão Autonômica de Processos de Negócios pode ser vista na Figura 8.

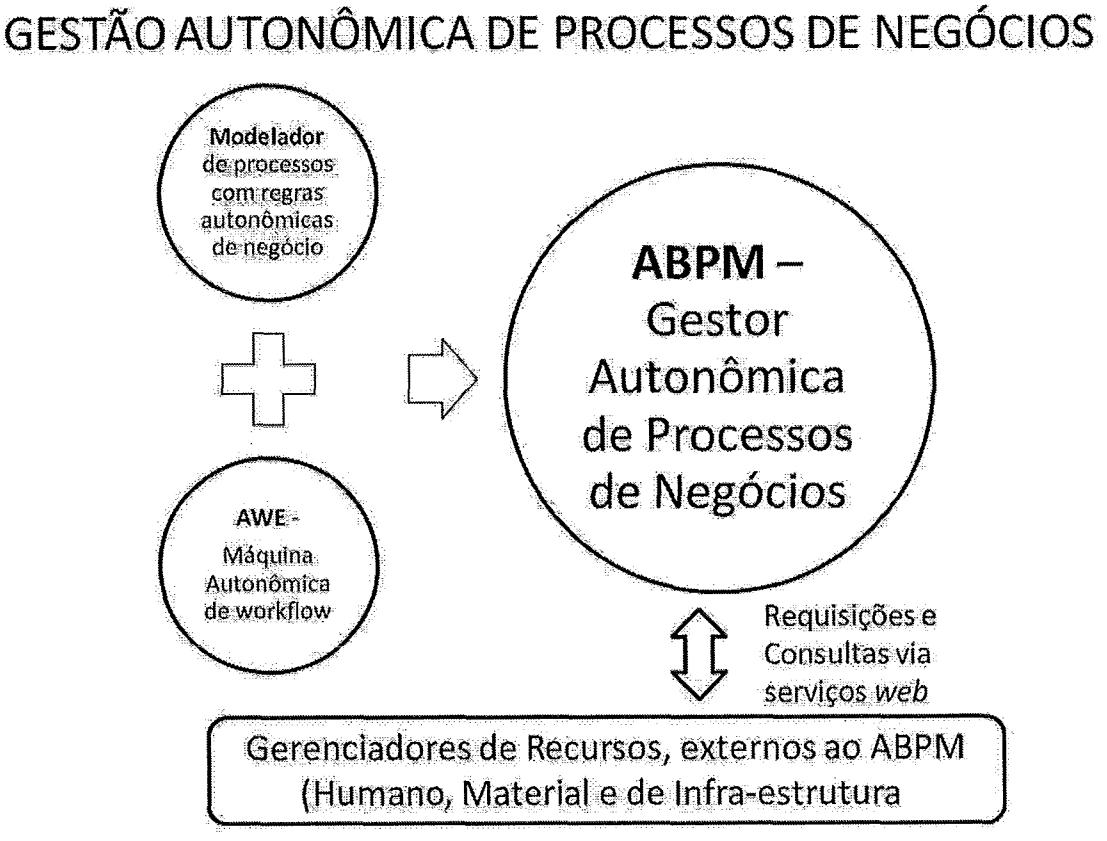


Figura 8 - Visão Geral da Gestão Autonômica de Processos de Negócios

3.4 – Arquitetura ABPM

ABPM foi desenvolvido a partir do JBPM. A Figura 9, proposta anteriormente por Monteiro Jr *et AL* (2008), apresenta a arquitetura do ABPM e nela podem ser

visualizados quais módulos originais do JBPM foram alterados e como os novos módulos do ABPM se integram com os originais depois das devidas evoluções.

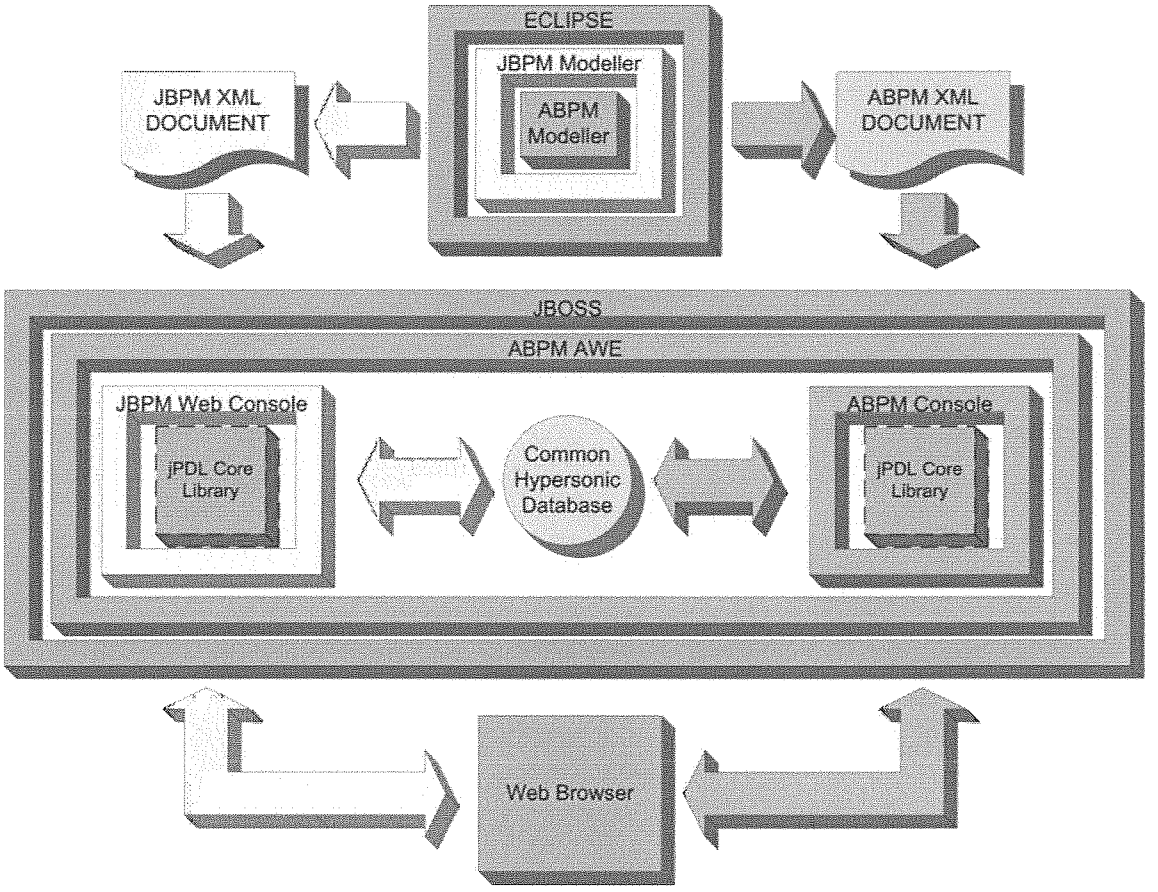


Figura 9 - Arquitetura ABPM

O primeiro módulo alterado foi o modelador jPDL GPD, chamado na Figura 9 de JBPM Modeller, desenvolvido como *plug-in* do Eclipse. Como foi feita a alteração desse *plug-in*, e não um novo para se comunicar com já existente, na arquitetura o ABPM Modeller está contido no modelador original. Em cada tarefa, assim como no processo, existem propriedades como nome, variáveis e executor. A alteração feita nesse módulo foi a possibilidade de adição das propriedades autonômicas (atributos e regras) a cada atividade e ao processo como um todo. O detalhamento das interfaces geradas para tal adição pode ser visualizado na Seção 4.1

Após as alterações, o modelador continua gerando o XML do JBPM com as informações a respeitos dos processos sem alterações, para que assim o workflow continue funcionando normalmente, caso não esteja sendo usado o ABPM.

O modelador agora gera um XML, específico do ABPM. O conteúdo do XML poderá ser observado na Seção 4.2.

Com a transmissão dos arquivos XML, o JBPM Web Console continua recebendo o documento que já era gerado anteriormente, porém o novo documento é recebido pela ABPM Web Console. A máquina de workflow autonômico, chamada de AWE, é onde estão os agentes que monitoram o processo a procura de ações a serem executadas. A arquitetura do AWE engloba ambos (JBPM e ABPM) os programas *web*. Essa arquitetura é mais bem detalhada na Seção 3.4.

Em termos de sistema, o usuário interage com os *web console* através do navegador da Internet, porém não será feito o detalhamento das interfaces, pois as originais do JBPM para iniciar um processo e executar uma tarefa não foram alteradas e as novas interfaces do ABPM são extremamente simples, sendo apenas de registro de usuários e visualização das regras de cada processo.

Tanto o ABPM Web Console quanto o JBPM Web Console utilizam a mesma base de dados. Para isso se tornar possível foi necessário alterar a base de dados para inclusão das novas informações do ABPM. As alterações feitas no diagrama entidade-relacionamento (adição de novas entidades e de novos relacionamentos dessas novas entidades com as que já existiam) podem ser vistas na Seção 4.3.1 – como diagrama de classes.

Algumas vezes, o ABPM Web Console necessita consultar outras aplicações para conseguir informações e isso é feito via serviços *web*. As informações necessárias são relativas a recursos, como quais as pessoas que podem substituir outras ou, então, se

um servidor, ou um sistema, está indisponível no momento. Ao receber as informações do serviço *web*, essas são avaliadas no ABPM, para verificar se elas podem participar do processo. Por exemplo, as pessoas sugeridas já podem estar alocadas em outra atividade naquele momento.

Pode-se observar que a máquina de workflow AWE está inserida na arquitetura ABPM, e nessa máquina é que se encontra a inteligência de monitoramento e de tomada de decisão sobre as ações. Por ser um tanto quanto complexa, ela necessita de uma arquitetura própria, vista a seguir na Seção 3.4.

3.5 – Arquitetura AWE (*Autonomic Workflow Engine*)

A arquitetura da máquina de workflow autonômica denominada AWE pode ser vista na Figura 10.

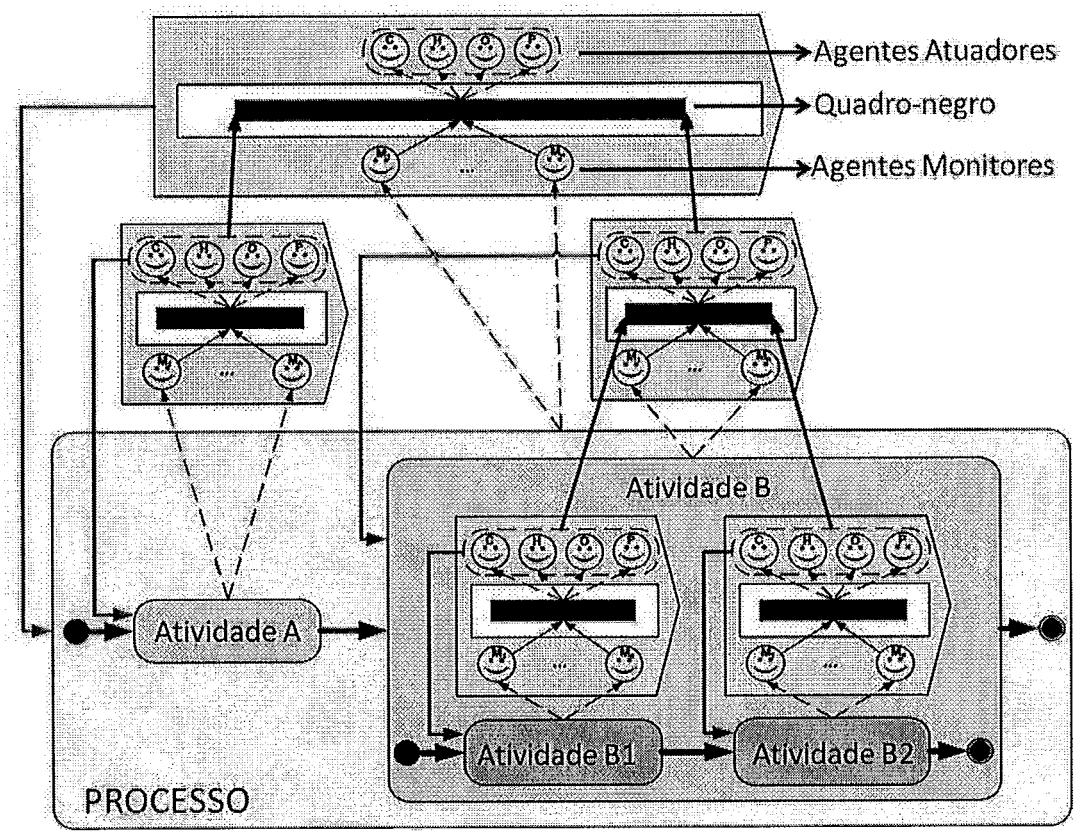


Figura 10 - Arquitetura AWE

O AWE possui um agente de *software*, que não está representado na Figura 10, para monitorar cada definição de processo chamado de agente Gerente da Definição do Processo. Esse agente, em termos de sistema, é considerado de mais alto nível, e é responsável por monitorar todas as instâncias do processo que estão em execução. Esse agente fica esperando pelo início de uma instância de processo, e sempre que isso ocorrer, ele inicia os devidos agentes monitores e executores. Após iniciado um processo, esse agente também monitora o início de uma tarefa, para também iniciar seus devidos monitores e executores. Assim, a partir desse agente é possível se obter os agentes de cada instância, seja de processo ou tarefa.

Na Figura 10, pode ser observado que cada instância de processo, assim como cada uma de suas atividades, independente do nível, é associada a uma estrutura, chamada de **elemento autonômico** de processo, formado pelos agentes monitores, pelo quadro negro local e pelos agentes atuadores. Ainda, o nível inferior sempre tem a possibilidade de se reportar ao nível superior.

Os **agentes monitores** estão representados no elemento autonômico na parte inferior, logo abaixo do quadro negro, e possuem a letra M*i* em seu interior, onde *i* é um numero seqüencial que identifica o agente. Eles têm a função de observar fatos sobre as atividades, e escrevê-los no quadro negro local, alterando o seu estado. O que é monitorado por esses agentes são os diversos tipos de recursos financeiro, humano, infra-estrutura, custo, material e tempo.

Os **quadros negros** locais são extremamente simples, eles servem apenas para receber e armazenar fatos sobre todos os recursos de uma atividade, recebidos através dos agentes monitores, persistindo assim o estado da atividade.

Como pode ser visto no elemento autonômico, cada quadro negro trabalha com quatro **agentes atuadores**, sendo cada um responsável por uma dimensão da

computação autonômica. Na Figura 10, eles estão posicionados logo acima dos quadros negro, e contêm, dentro deles, uma das letras C, H, O ou P. Onde a letra C representa a dimensão de auto-configuração, H para designar a auto-cura, O que diz respeito à auto-otimização ou P para auto-proteção.

Os agentes atuadores são os responsáveis pela tomada de decisão, ou seja, por selecionar qual ação é disparada após a interpretação dos fatos do quadro negro. Essa decisão é feita através de regras de negócios definidas pelo usuário. Em termos de implementação, ela é desenvolvida em cima de sistemas de produção de regras, no caso o CLIPS. O comportamento dos agentes se assemelha ao de um especialista no processo.

Um das ações é escrever fatos no quadro negro do nível imediatamente superior. Assim, além dos seus respectivos fatos, recebidos através dos seus monitores, o nível superior recebe também fatos do nível diretamente inferior. Esse comportamento é propagado até o nível mais alto do processo.

O funcionamento da arquitetura, como é visto na Figura 11, se inicia com a definição do processo pelo gerente. Para cada atividade e para o processo como um todo, são especificados os atributos, fatos, condições e ações que serão aplicados. Após definido e fechado o modelo, o workflow é disponibilizado para execução e o agente gerente da definição do processo é iniciado. Quando uma instância começa, os agentes monitores imediatamente percebem o estado do workflow e escrevem os devidos fatos em seus respectivos quadros negros. A partir daí, os atuadores também já vasculham o quadro negro em busca de fatos similares as condições das regras, para disparar uma ação quando necessário.

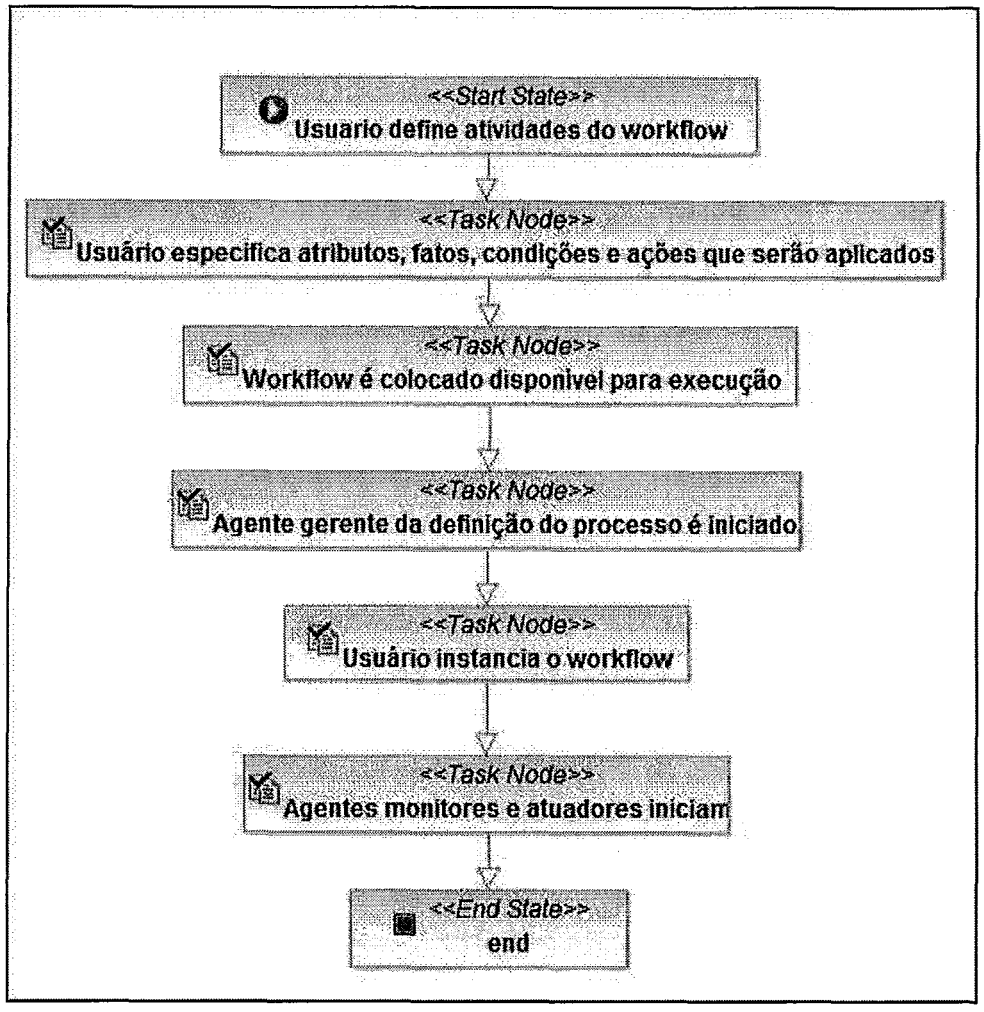


Figura 11 - Fluxograma de execução do AWE

Capítulo 4 – Gestor Autonômico de Processos de Negócio

O capítulo corrente detalha o sistema desenvolvido. Primeiramente, é apresentado o modelador juntamente com suas interfaces de interação com o usuário. Com essas interfaces é possível criar atributos e regras autonômicas. Posteriormente, é descrito o arquivo XML que é trocado entre o modelador e a máquina de workflow autonômico. Em seguida, é mostrado como foi desenvolvida a máquina de workflow autonômico através de diagramas de classes e atividades dos agentes. Depois, é explicado como os agentes monitores calculam os valores a serem colocados no quadro negro e como as regras são executadas. Finalmente, é exposto como é feito o tratamento das regras no CLIPS.

4.1 – Modelador

O modelador do ABPM, chamado de ABPM Modeller, é desenvolvido como *plug-in* do Eclipse a partir do módulo de modelagem do JBPM (JBPM Modeller). Foram adicionadas novas funcionalidades a esse último. Foi acrescida em cada tarefa, assim como no processo como um todo, a possibilidade de inserir as propriedades (atributos e regras) autonômicas.

4.1.1 – Criação dos atributos

Após a criação de uma tarefa, podem ser adicionados os atributos e regras autonômicas. Para efetuar a criação dos atributos, é necessário acessar as propriedades da tarefa ou do processo, escolhendo a aba autonômica e marcando a opção Atributos.

Para todos os atributos existe uma tela inicial padrão, que pode ser vista na Figura 12. Nela, primeiramente, deve ser escolhido o tipo de recurso e, posteriormente, qual atributo desse recurso será monitorado. Na Tabela 2, podem ser observados todos os tipos de recursos disponíveis com seus respectivos atributos possíveis, esses recursos foram definidos a partir da ontologia de problemas. Após optar por adicionar um atributo, são pedidos alguns parâmetros padrão. Em seguida ao preenchimento dos parâmetros, o atributo adicionado é listado, em forma de tabela, que, na interface da Figura 12, se localiza logo abaixo do botão "Add" e "Remove".

A tabela referente aos atributos listados contém o nome da tarefa, o tipo do recurso, o tipo e o nome do atributo e alguns parâmetros iniciais. O tipo de recurso é uma das opções da coluna esquerda da Tabela 2. Já o tipo de atributo se encontra do lado direito da mesma tabela. O nome do atributo é um identificador para quando um tipo de atributo puder ser adicionado mais de uma vez. Por exemplo, para o atributo Sistema pode ser adicionado "Planilha de Pessoas" e "Sistema de Gestão de Recursos Humanos". Quando um atributo não puder ser repetido para uma mesma tarefa, o nome do atributo terá o mesmo valor do tipo, por exemplo, Duração. Os parâmetros iniciais, quando puderem ter valor máximo, mínimo e esperado, são apresentados separados por ponto-e-vírgula (;), com o tipo (máximo, mínimo e esperado) seguido do sinal de igual (=) e do valor, por exemplo: "Máximo=3;Mínimo=1;Esperado=2;". Quando só puder ter valor esperado, é apresentado apenas o valor seguido de ponto-e-vírgula, por exemplo, "Disponível;".

A interface padrão inicial das tarefas (Figura 12) também permite remover um atributo através do botão "Remove", assim como, alterá-lo, bastando clicar duas vezes na linha correspondente ao atributo.

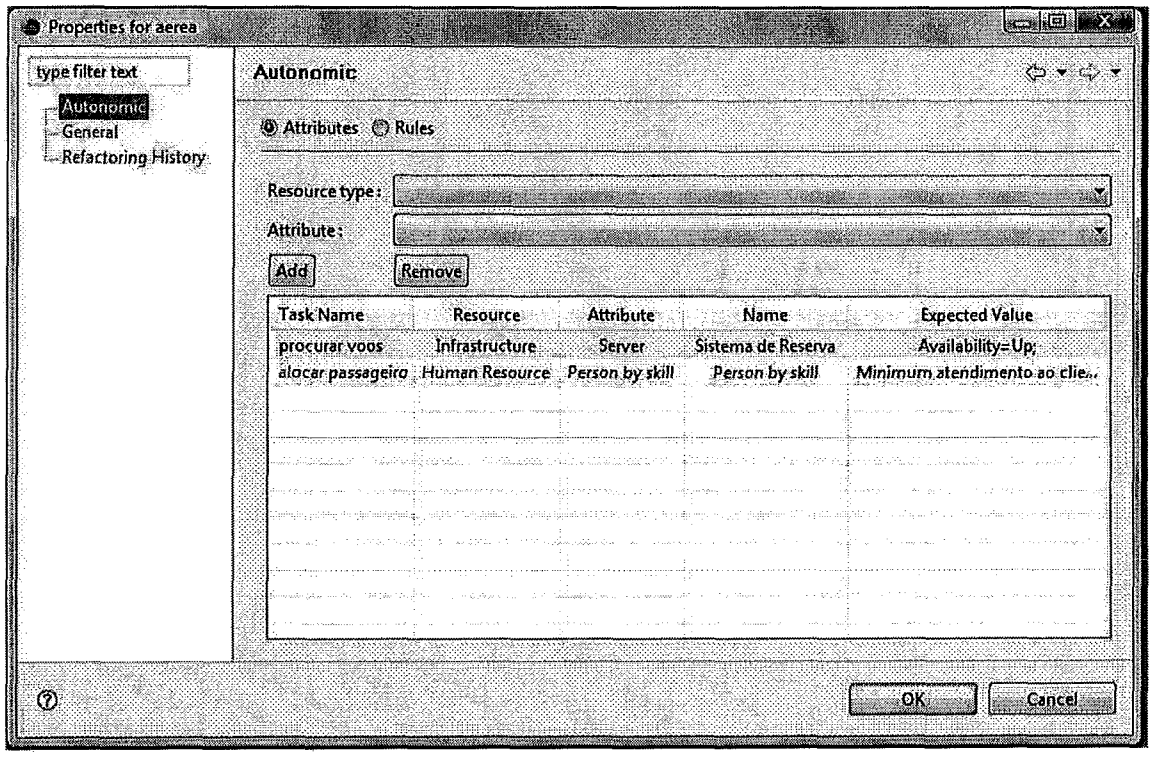


Figura 12 - Tela inicial para criação de atributos

Tabela 2 - Recursos X Atributos

Recursos	Atributos
Recurso Financeiro	Custo
Recurso Humano	Pessoas por Capacidade
Recurso de Infra-estrutura	Servidor
	Sistema
Recurso de Material	Documento
	Material
Recurso de Tempo	Duração
	Data Final

Como dito anteriormente, para cada tipo de atributo, alguns parâmetros são requisitados. É especificado nas subseções abaixo que parâmetros cada um deles requer.

4.1.1.1 – Criação do atributo Custo

A criação do Custo requer a especificação de um nome, porém esse nome deve ser um dentre alguns específicos correspondentes a moedas internacionais (Dólar, Real ou Euro). Eles estão em uma lista, e exigem um valor numérico para ser o valor mínimo, valor esperado e valor máximo, como pode ser visualizado na Figura 13. Sendo que o valor esperado não pode ser menor que o mínimo e nem maior que o máximo.

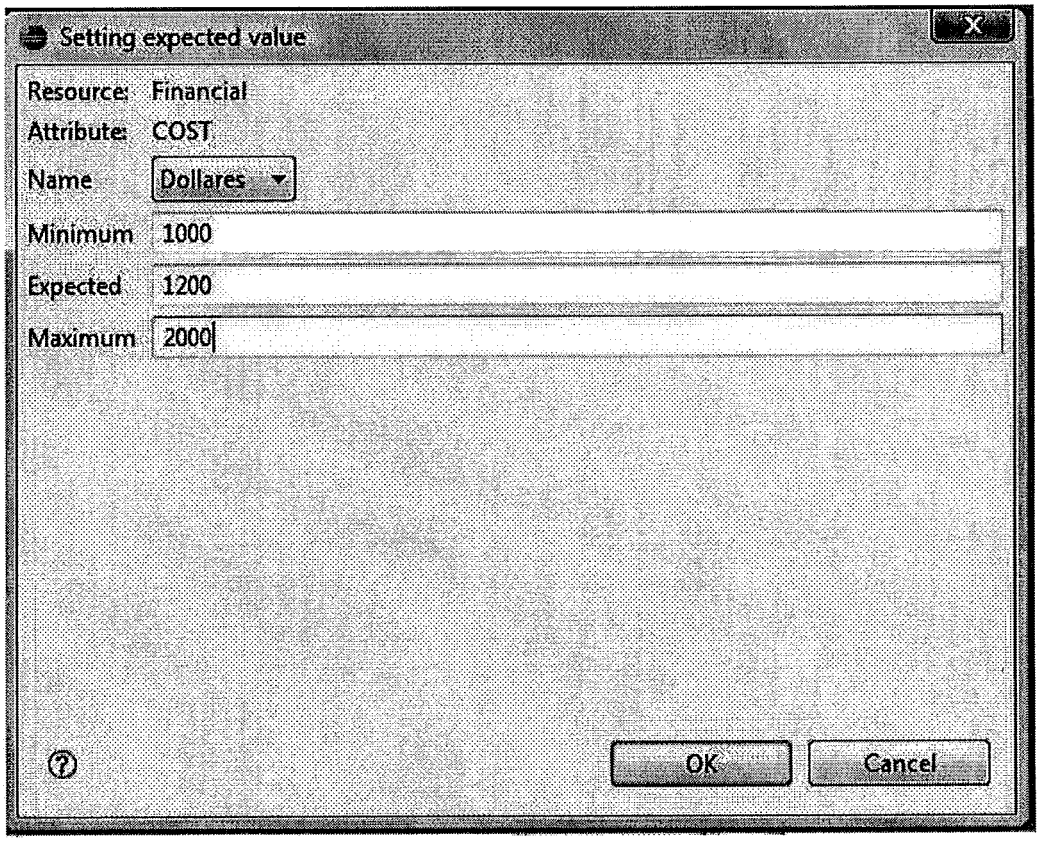


Figura 13 - Tela de inserção do atributo Custo

4.1.1.2 – Criação do atributo Pessoas por Capacidade

A criação de Pessoas por Capacidade é bem parecida com a de Custo e a Figura 14 apresenta a interface correspondente. Primeiramente, ela requer um nome específico que está em uma lista: Atendimento ao cliente, Cadastro de reservas de vôo, Cobrança, Fiscalização, Java, Banco de Dados e Vendas. E para definir esse atributo, deve-se informar qual o valor mínimo, esperado e máximo de pessoas com esse conhecimento

na tarefa. Assumiu-se que ou uma pessoa possui um conhecimento ou não o possui por completo, sem entrar em detalhes de quanto cada possui. Além disso, os conhecimentos não foram divididos em sub-conhecimentos.

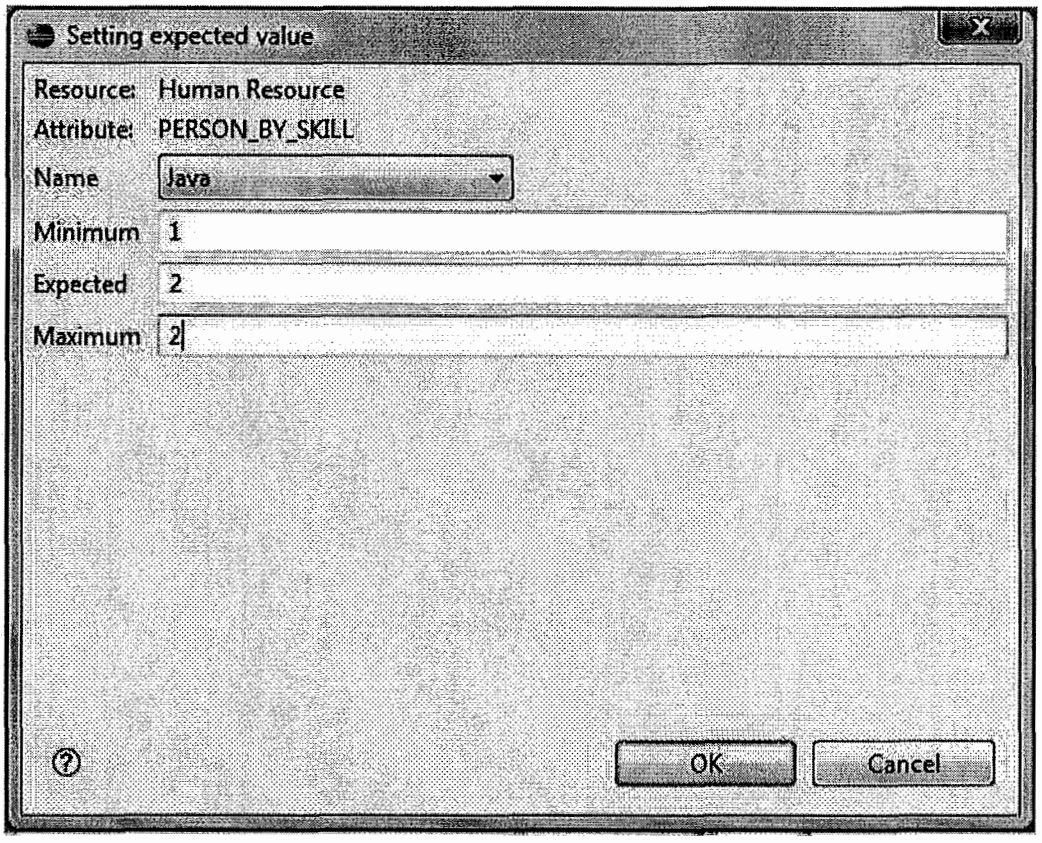


Figura 14 - Tela de inserção do atributo Pessoas por Capacidade

4.1.1.3 – Criação do atributo Servidor

Para Servidor, o nome é pedido, já que é permitido ter mais de um por tarefa. Pode ser qualquer valor textual como Oracle, CVS, entre outros. O outro parâmetro é a disponibilidade que poder ser Ativo ou Inativo. A Figura 15 demonstra a interface.

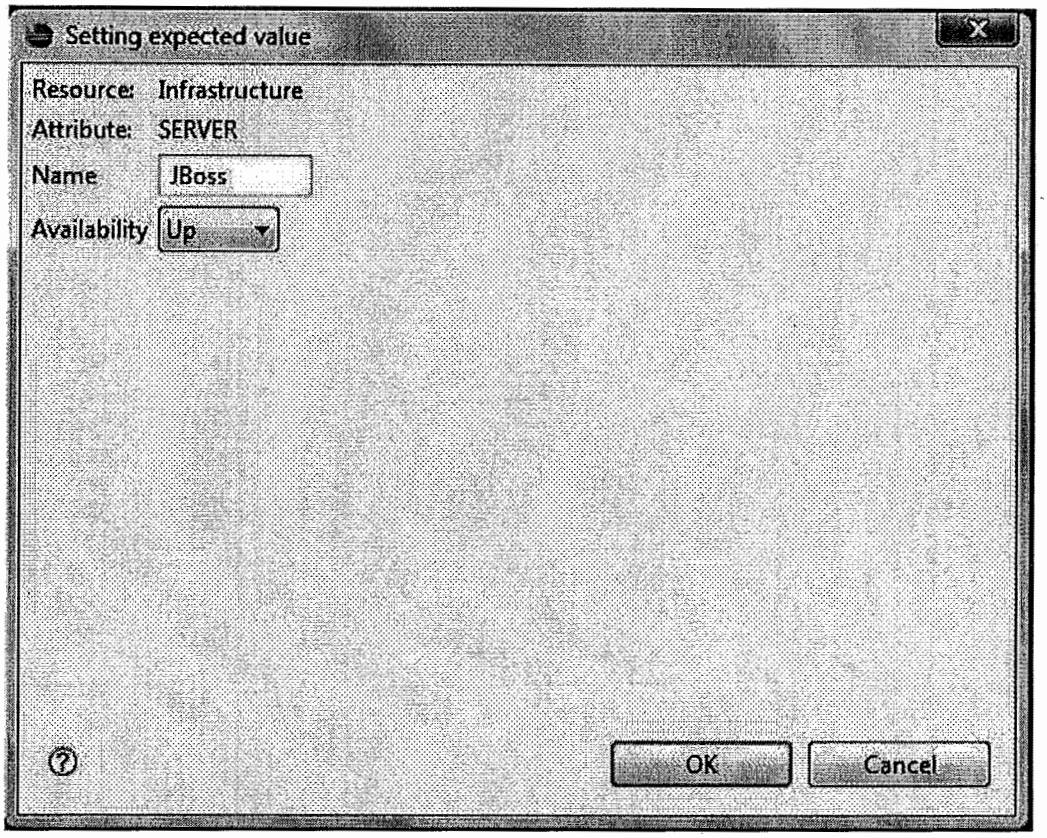


Figura 15 - Tela de inserção do atributo Servidor

4.1.1.4 – Criação do atributo Sistema

Em Sistema, mostrado na Figura 16, o nome é requerido. Pode ser qualquer valor textual como Sistema de Recursos Humanos, Gerenciador de Documentos, entre outros. O outro parâmetro é a disponibilidade que poder ser Disponível ou Indisponível.

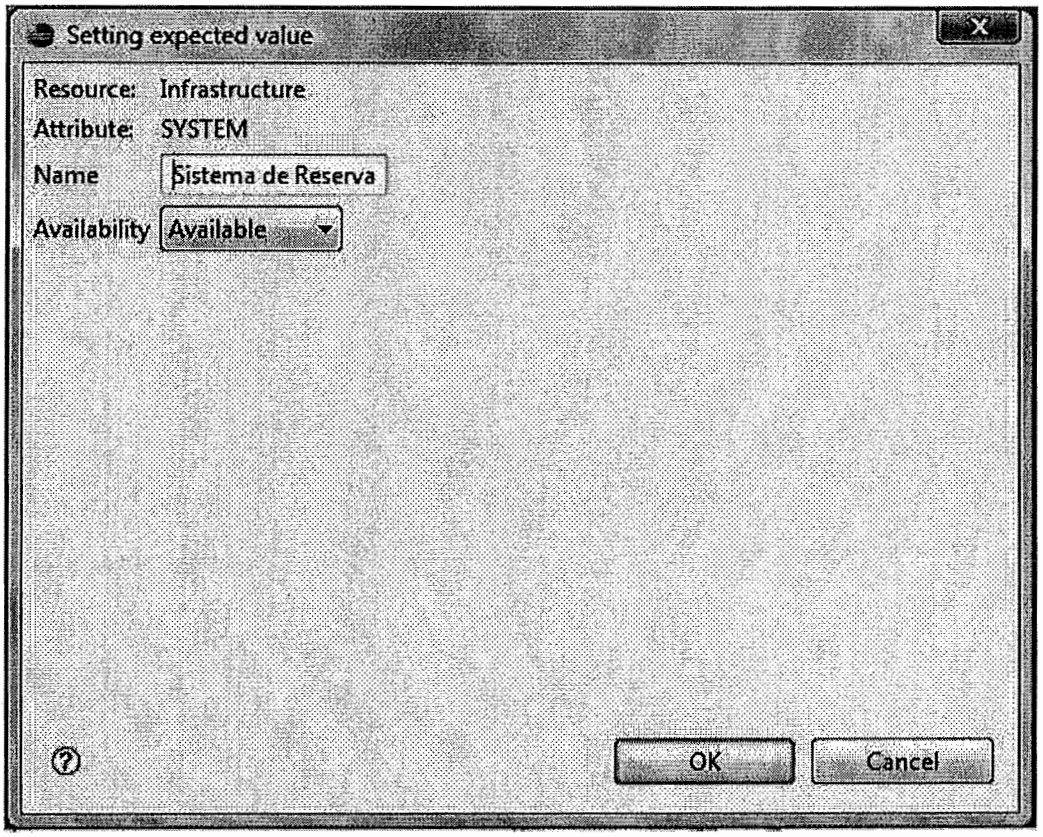


Figura 16 - Tela de inserção do atributo Sistema

4.1.1.5 – Criação do atributo Documento

A Figura 17 apresenta o atributo Documento, e pode-se verificar que o nome é pedido, podendo ser qualquer valor textual como RG, CPF ou Passaporte. O outro parâmetro é a disponibilidade que poder ser Disponível ou Indisponível.

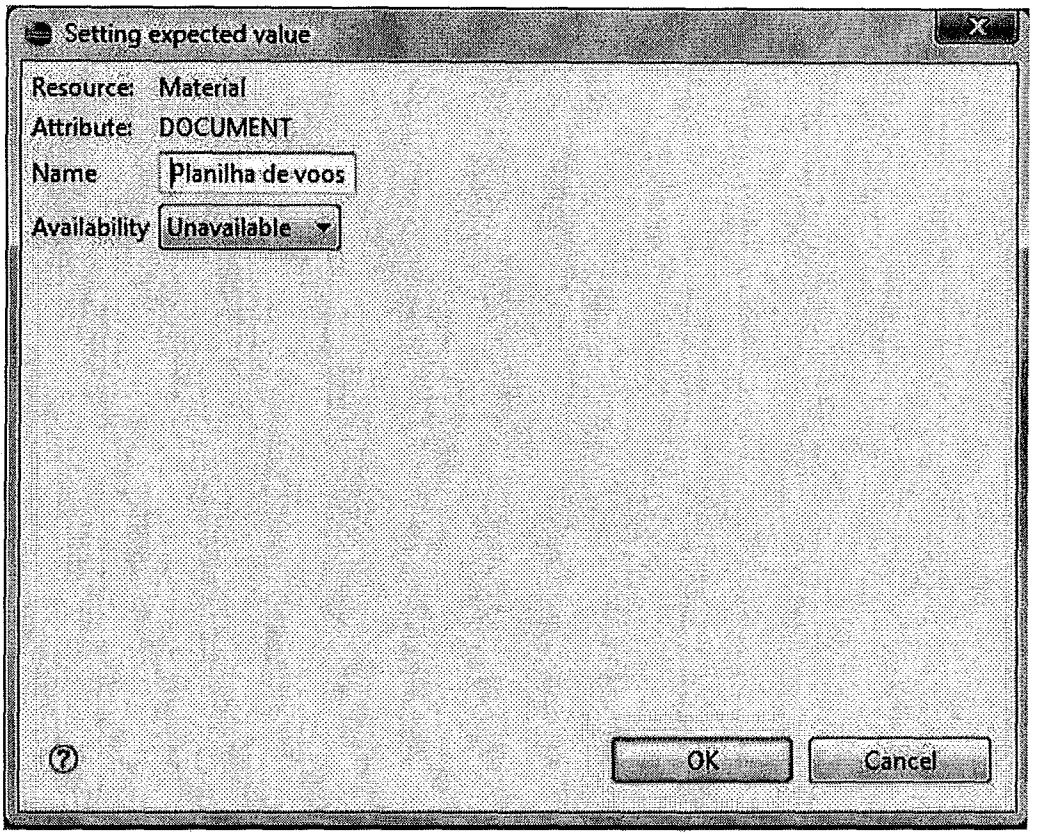


Figura 17 - Tela de inserção do atributo Documento

4.1.1.6 – Criação do atributo Material

Para Material, o nome é pedido. Pode ser qualquer valor textual como Papel, Ouro, etc. O outro parâmetro é a disponibilidade que poder ser Disponível ou Indisponível. A Figura 18 demonstra a interface.

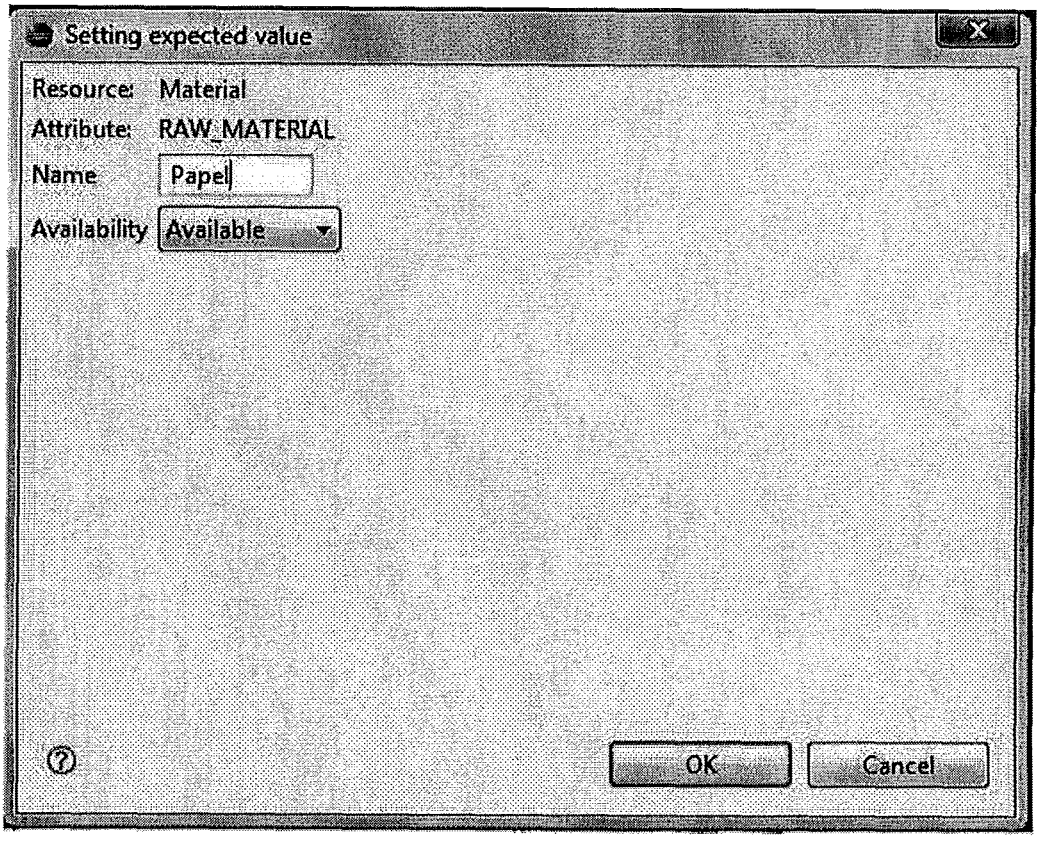


Figura 18 - Tela de inserção do atributo Material

4.1.1.7 – Criação do atributo Duração

A criação da Duração não requer a especificação de um nome e a unidade de tempo é sempre hora. Ainda exige um valor numérico para ser o valor mínimo, esperado e máximo como pode ser visualizado na Figura 19.

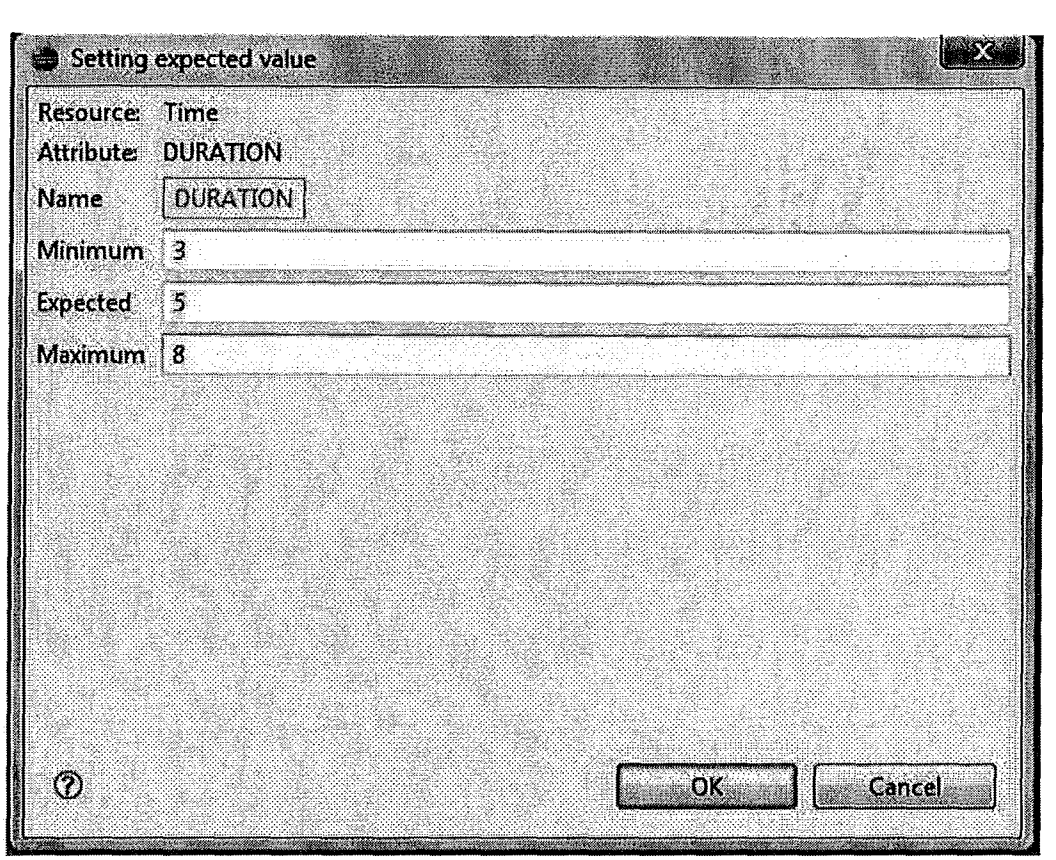


Figura 19 - Tela de inserção do atributo Duração

4.1.1.8 – Criação do atributo Data Final

A Data Final não requer nome e o único parâmetro que é necessário, visto na Figura 20, é a data final, cujo valor deve ser uma variável de controle, definida nas propriedades de controle da tarefa. A data final não pode ser uma data específica, pois um workflow é repetido várias vezes com datas finais diferentes. Por exemplo, na companhia aérea, a data final de embarque é o horário do vôo, e para cada vez que o processo for executado, essa data é diferente e seu valor estará nas variáveis de controle.

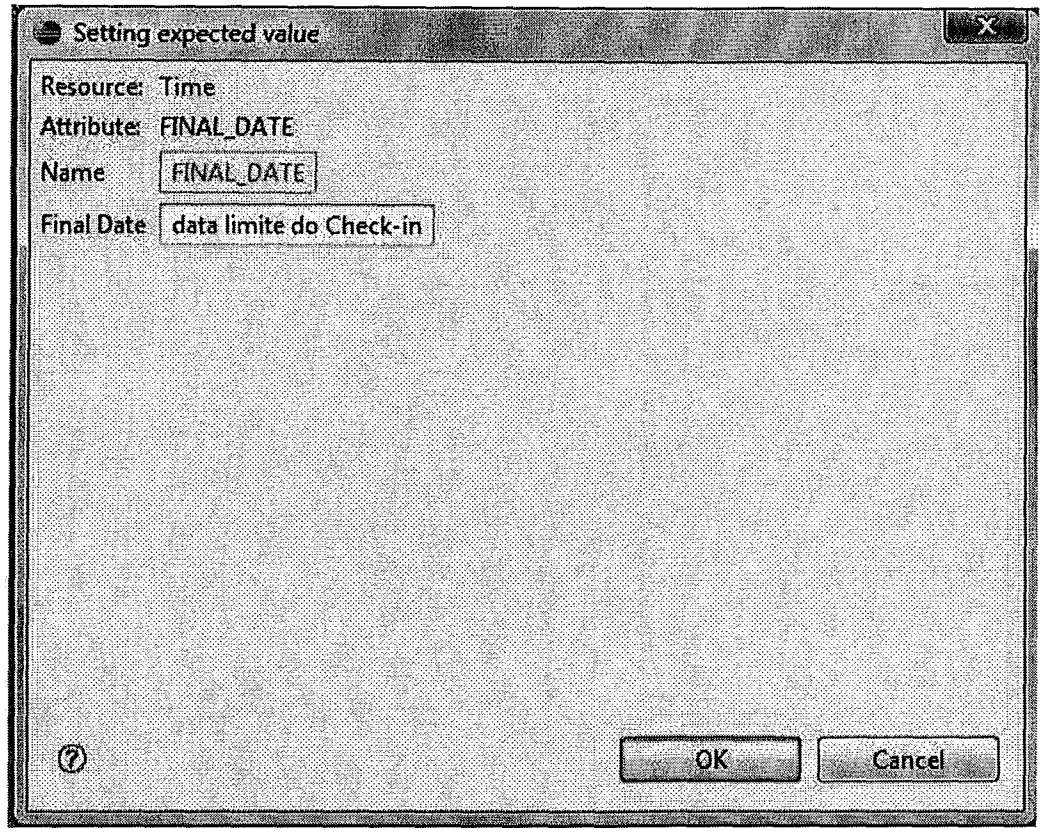


Figura 20 - Tela de inserção do atributo Data Final

4.1.2 – Criação das regras

Após a definição dos atributos, deve-se iniciar a montagem das regras. Assim como os atributos, todas as regras têm uma tela inicial comum, e essa pode ser vista na Figura 21.

Na tela inicial das regras, primeiramente, deve-se escolher também a ação que será executada. A dimensão autonômica serve apenas para classificação das regras, feita pelo gerente. As ações disponíveis são: Retornar a uma atividade anterior, Cancelar tarefa, Cancelar processo, Notificar especialista via correio eletrônico, Adicionar novo executor, Remover executor, Substituir executor, Usar outro sistema, Informar nível superior e Procurar Novo Fluxo. Com exceção da ação de "Procurar Novo Fluxo", todas as ações estão codificadas na aplicação desenvolvida.

Na tela exibida na Figura 21, há uma tabela com todas as regras definidas, onde é possível remover uma regra através do botão "Remove", assim como, alterar, bastando clicar duas vezes na linha correspondente. As regras são listas em ordem decrescente de prioridade. Pode-se também alterar a prioridade de uma regra subindo-a ou descendo-a da tabela. Assim, pode-se assegurar que não há duas regras com a mesma prioridade para uma tarefa.

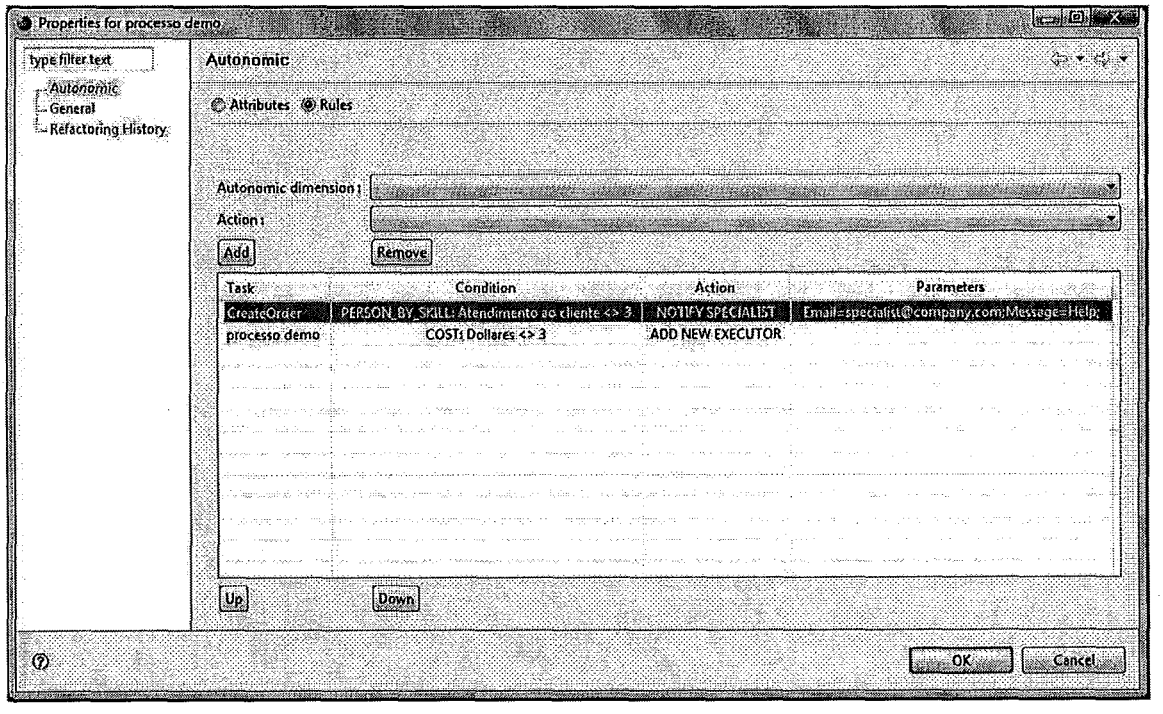


Figura 21 - Tela inicial de criação das regras

Ao adicionar uma regra, outra tela, essa de inserção dos parâmetros da condição e da ação é exibida. Na Figura 22, pode-se observar essa tela comum.

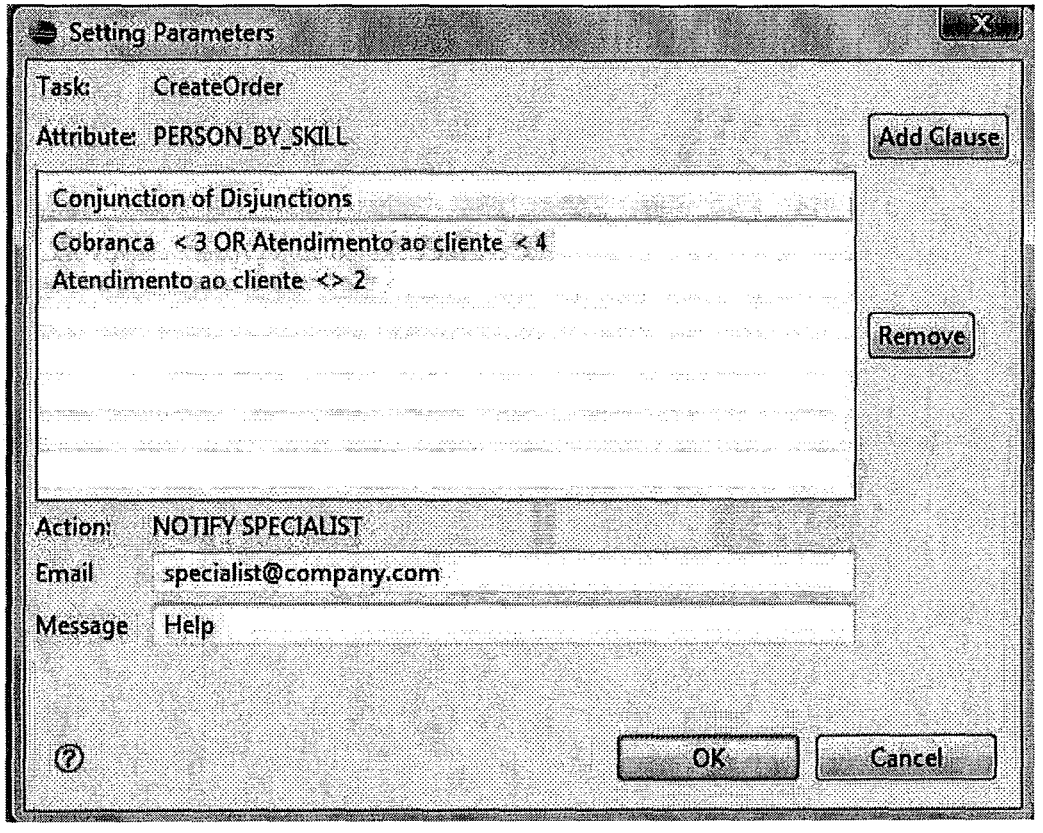


Figura 22 - Tela de inserção de parâmetros em uma regra

Na parte inferior da Figura 22, existe o campo "Action", que nos diz o nome da ação a ser executada, e logo abaixo dele, tem-se os parâmetros. Uma ação pode ter quantos for necessário, inclusive nenhum. A seguir, são apresentados quais são os parâmetros para cada tipo de regra:

- Retornar a uma atividade anterior: o campo "Atividade Anterior" espera o nome de uma atividade já definida no processo.

- Cancelar tarefa: não possui parâmetros.

- Cancelar processo: não possui parâmetros.

- Notificar especialista via correio eletrônico: possui o campo para preenchimento do correio eletrônico do especialista e, também, a mensagem que será enviada.

- Adicionar novo executor: não possui parâmetros.

- Remover executor: não possui parâmetros.

- Substituir executor: não possui parâmetros.

- Usar outro sistema: requer o nome do outro sistema que substituirá o que estava previsto.

- Informar nível superior: possui um campo para preenchimento do valor a ser informado para o nível superior. Esse valor deve ser um estado da tarefa. Quando uma tarefa é iniciada, isso é escrito no quadro negro superior o valor Iniciado. Também é informado automaticamente quando a tarefa é cancelada (finalizada sem sucesso) ou finalizada com sucesso. Esse valor informado ao nível superior é um estado, definido pelo gerente, que a tarefa pode estar entre o início e o fim dela.

- Procurar novo fluxo: não possui parâmetros.

Já na parte superior da Figura 22, tem-se a tabela de "Conjunção de Disjunções". Cada linha na tabela é uma disjunção e todas elas são unidas por um E, formando uma conjunção. Exemplificando, se existe na primeira linha da tabela "a OU b" e na segunda temos "C", logo se tem, no final, "(a OU b) E c". No ABPM, é possível adicionar várias linhas na tabela, para formar a condição da ação. Para adicionar uma linha, basta optar por adicionar uma cláusula ("Add Clause").

Ao optar por adicionar uma cláusula, uma nova tela, correspondente a Figura 23, surge. Na tabela, da parte superior da figura, cada linha representa um conjunto de "nome - sinal - valor" adicionada na parte inferior da figura, e cada linha é unida a outra através do operador OU. Na parte da adição, há primeiramente duas listas, uma com os atributos juntamente com seus nomes (exemplo: Material: Papel) ou das tarefas filhas possíveis, e outra com o sinal esperado (<, <=, =,>, >=, <>). Após a escolha do nome, deve ser preenchido o valor que é um texto livre ou é uma lista.

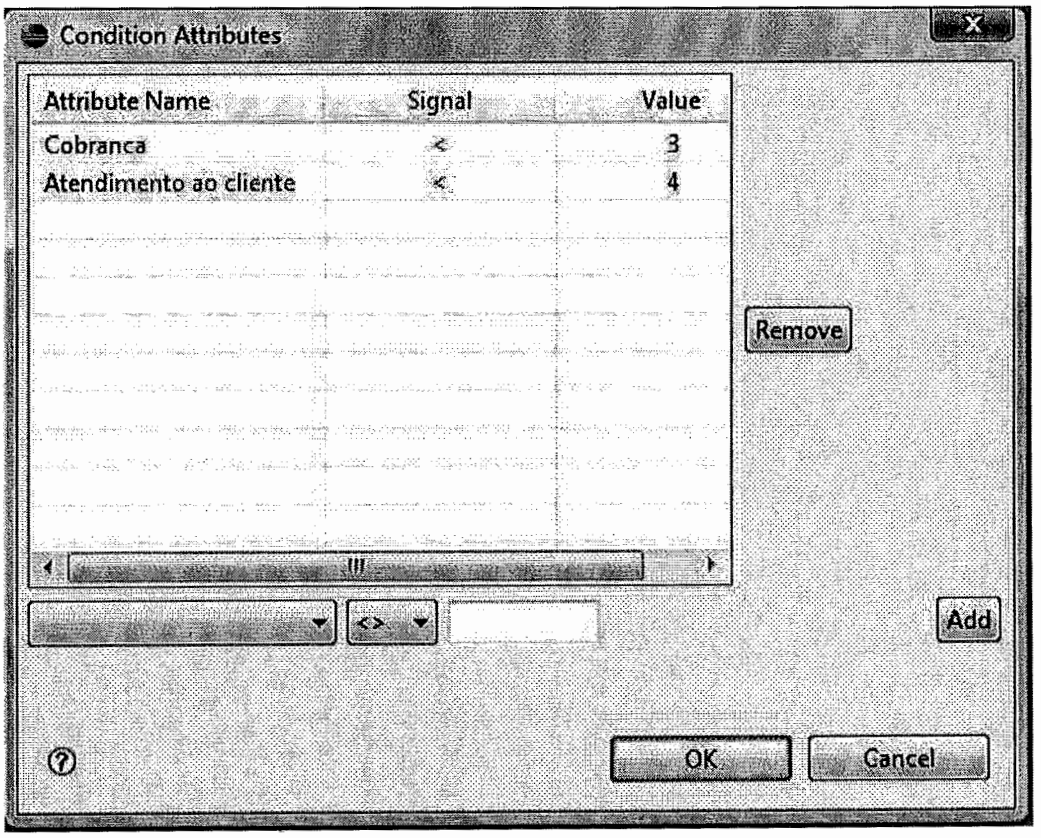


Figura 23 - Tela de formação da condição

Cada atributo pode ter alguns possíveis nomes e espera um tipo específico de resposta. Isso é detalhado na Tabela 3. Para as tarefas filhas, os valores estão numa lista de Iniciado, Cancelado (Finalizado sem sucesso), Finalizado com sucesso ou pode ser preenchido com um valor textual livre que coincida com o valor escrito por uma ação de Informar nível superior.

Tabela 3 - Nomes possíveis para atributos

Atributo	Nomes possíveis	Valor
Custo	Dólares	Valor numérico
	Real	
	Euro	
Pessoas por capacidade	Atendimento ao cliente	Valor numérico

	Cadastro de reservas de vôo	
	Cobrança	
	Fiscalização	
	Java	
	Banco de dados	
	Vendas	
Servidor	Livre	Uma listagem com os valores: Ativo e Inativo
Sistema	Livre	Uma listagem com os valores: Disponível ou Indisponível
Documento	Livre	Uma lista com os valores: Disponível ou Indisponível
Material	Livre	Uma lista com os valores: Disponível ou Indisponível
Duração	Duração	Valor numérico correspondente à hora
Data final	Data final	Valor textual correspondente a variável de controle, definida na propriedade de controle e não na autonômica, referente à data final.

4.2 – Interface XML entre modelador e ABPM Console

Na arquitetura ABPM, o modelador gera um arquivo XML, específico para o ABPM, com as informações para cada processo, qualquer que seja o nível, das propriedades (atributos e regras) autonômicas. O conteúdo desse XML pode ser observado na Figura 24.

```
<autonomicAttribute>
    <attributeMonitored
        resource="Infrastructure"
        attribute="Server"
        name="Sistema de Reserva"
        expectedValue="Availability=Up;"
    />
</autonomicAttribute>
<autonomicRule>
    <rule
        dimension="Self-healing"
        action="Use Another System"
        parameters="Other System=Planilha de Reservas;"
        value="Availability = Down"
    />
</autonomicRule>
```

Figura 24 - XML entre modelador e ABPM Console

O XML é dividido em duas *tags* principais: "autonomicRule" que representa o conjunto de regras autonômicas e "autonomicAttribute" que, por sua vez, representa o conjunto de atributos autonômicos. Essas duas *tags* são únicas para um processo (ou tarefa), ou seja, não se repetem na definição de uma tarefa.

O conjunto de atributos autonômicos a serem monitorados se caracteriza por uma ou mais *tags* nomeadas de "attributeMonitored". Cada vez que essa *tag* se repete, é um atributo a ser monitorado pelo agente. Nela, existem algumas propriedades que caracterizam o atributo: *resource*, *attribute*, *name* e *expectedValue*. Esses atributos são preenchidos na interface que foi explicada na seção anterior. *Resource* pode ser humano, temporal, entre outros. *Attribute* é o tipo de atributo como, por exemplo, documento, servidor, sistema. *Name* é o nome definido para o atributo como sistema gestão de documentos ou servidor de impressão. E, finalmente, *expectedValue* é o valor

que o atributo possui quando não está com problemas, podendo ter, para alguns atributos, valor mínimo e máximo também.

O conjunto de regras autonômicas agrupa várias *tags* do tipo *rule*. Cada uma delas representa uma regra que está à disposição do processo (ou tarefa). As propriedades dessa *tag* são: *dimension*, *action*, *parameters* e *clause*. *Dimension* é em qual área da computação autonômica (configuração, cura, proteção ou otimização) a ação se enquadra, e *action* é o tipo de ação que será executada na regra. Para execução da ação, alguns parâmetros são definidos e preenchidos na interface correspondente, e eles são carregados na propriedade *parameters*. E *clause* diz com qual cláusula a regra é disparada.

O arquivo XML será transmitido pelo modelador e recebido pela ABPM Web Console. Para tal, basta colocar os parâmetros necessários na interface de transmissão, que é apresentada na Figura 25. Os parâmetros são simples e intuitivos. O nome do servidor é o endereço *ip* da máquina onde o servidor se encontra, e a sua porta, aponta para ele está configurado. Os parâmetros "JBPM Deployer" e "ABPM Deployer" são, respectivamente, os nomes das aplicações do JBPM Web Console e ABPM Web Console.

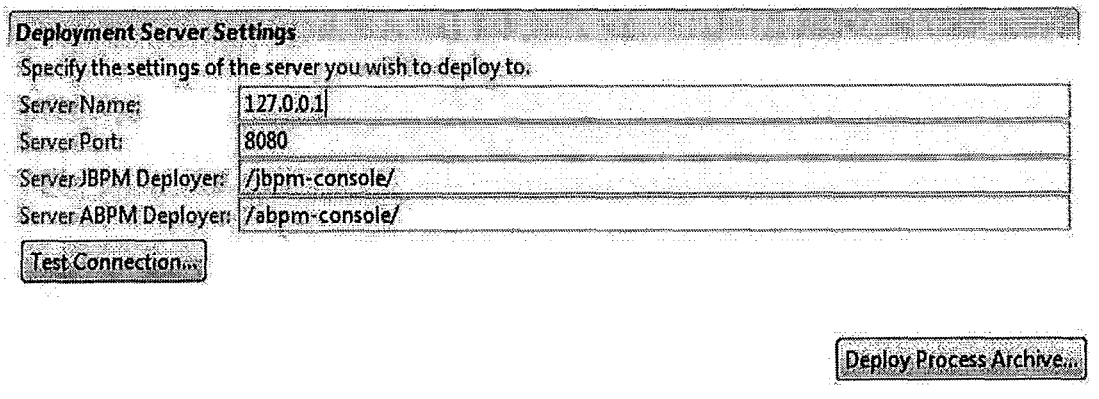


Figura 25 - Parâmetros para transmissão do XML entre modelador e AWE

4.3 – AWE: Máquina de Workflow Autonômico

A arquitetura do AWE já foi alvo de estudo na Seção 3.4, porém, são apresentados aqui os diagramas elaborados no desenvolvimento da máquina de workflow: diagrama de classes da aplicação, diagramas de atividade e diagrama de classe dos agentes.

4.3.1 – Diagrama de classes da aplicação

Para que tanto o ABPM Web Console quanto o JBPM Web Console utilizem a mesma base de dados, foram necessárias alterações na base inicial do JBPM. A Figura 26 mostra o diagrama de classes relativo a tal implementação. Como ambas as aplicações compartilham a mesma base de dados, são apresentadas apenas os conceitos adicionados, com exceção de tarefa e processo, que já existiam anteriormente. Mesmo assim, nesse caso, é apresentado somente os atributos necessários. Essa mudança não altera o funcionamento do JBPM Web Console, apenas permite o uso do ABPM Web Console. Esse diagrama de classe não se aplica exclusivamente ao JBPM, podendo ser aplicado a qualquer máquina de workflow, bastando para tal adaptar as associações existentes com processo e tarefa, caracterizando assim mais uma contribuição deste trabalho.

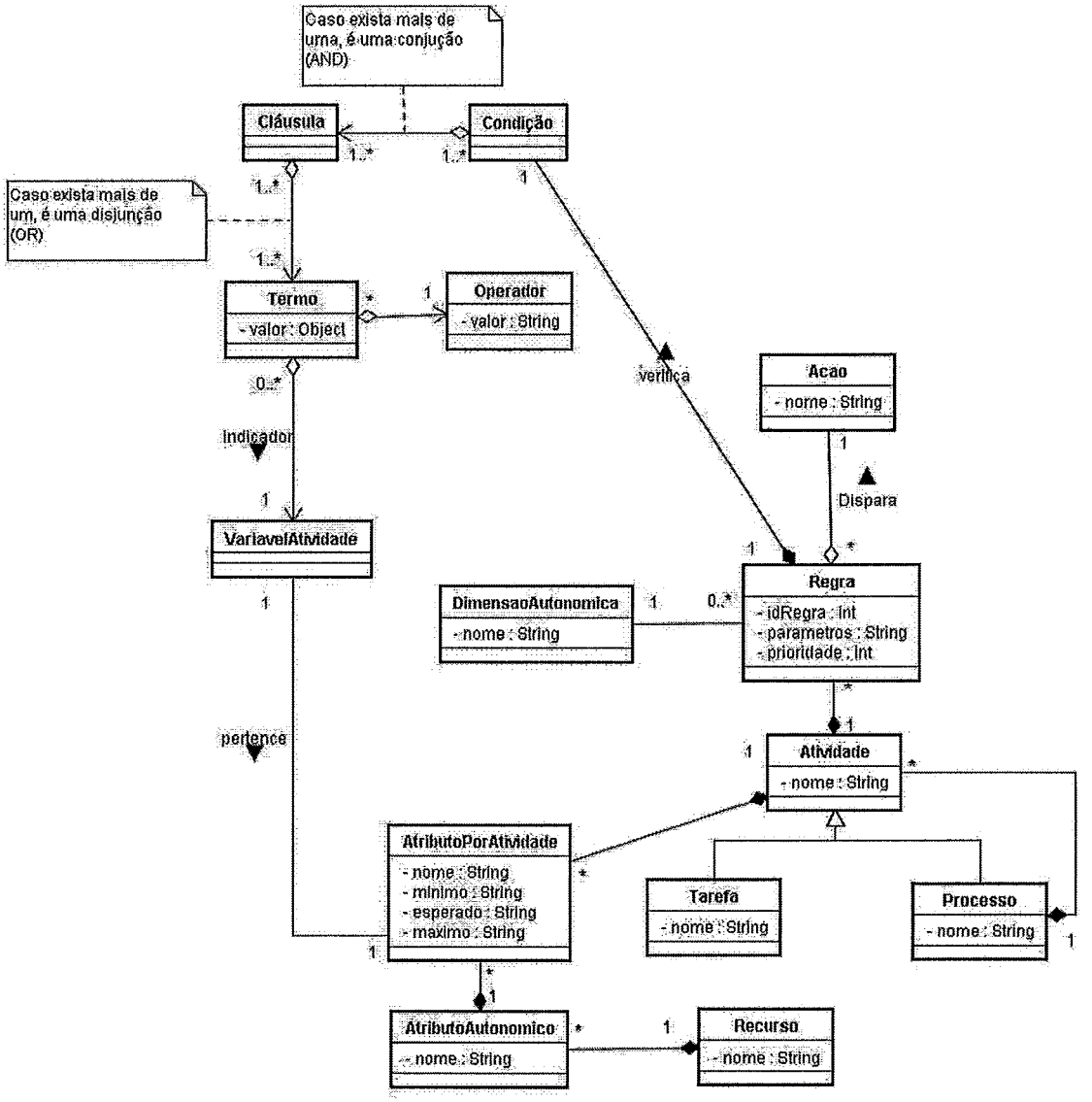


Figura 26 - Diagrama de Classes das Entidades do ABPM

Como "Tarefa" e "Processo" já pertenciam ao JBPM, em ambas é mostrado, por questões de simplificação, apenas o nome. Agora, as duas herdam de "Atividade", sendo que um processo pode ter vários subprocessos, até o nível mais baixo, que é a tarefa. Assim como nas ontologias da Seção 3.2, foi utilizado o padrão de projeto Composite (Gamma *et al*, 1995) para representar a estrutura de um processo ter subprocessos, finalizando em uma tarefa.

Outra classe é a "Atributo Autonômico", cujos valores estão em uma lista pré-determinada: custo, pessoas por capacidade, servidor, sistema, documento, material,

duração, data final e estado tarefa filha. Além do nome, um atributo autonômico armazena se seu o tipo é um valor numérico, textual ou booleano.

Um atributo autonômico pode pertencer a várias atividades, e o inverso também se aplica, ou seja, uma atividade pode possuir vários atributos autonômicos. Com isso, é formada uma classe, chamada de "Atributo por atividade". Nessa classe é armazenado o nome da instância do atributo (nome do documento, do servidor, da moeda, entre outros), os valores mínimo, esperado e máximo do atributo autonômico para uma determinada atividade.

A classe "Recurso" tem uma associação com a classe "Atributo Autonômico", onde um atributo autonômico pode pertencer a um único tipo de recurso e um recurso pode possuir vários atributos autonômicos. Sendo que "Recurso" possui apenas o nome como atributo e que seus valores são valores fixos: financeiro, humano, infra-estrutura, material e temporal.

Uma atividade também pode possuir uma série de regras, sendo cada uma delas é representada pela classe "Regra". Essa última classe verifica uma "Condição". Para ser possível que uma mesma ação possa ser definida para uma mesma tarefa (ou processo) com diferentes condições, em cada regra há um identificador único (atributo id).

Como foi definido que é usado como padrão a Forma Normal Conjuntiva, uma condição conterá várias conjunções, e essas, por sua vez, contêm várias disjunções. Assim, cada condição é composta por uma ou mais cláusulas. Cada "Cláusula" é conectada a outra pela conjunção E. Cada uma dessas cláusulas é composta por termos ("Termo"), sendo que caso exista mais de um, eles são interligados pela disjunção OU.

Um "Termo" tem a estrutura: "primeiro operando **operador** segundo operando", por exemplo: Duração > 30. "Operador" é um dos sinais: >, >=, <, <=, = ou <>. O

primeiro operando é um indicador, seja uma variável (atributo) da atividade (por exemplo: duração na atividade x). Já o segundo operando é um valor (atributo da classe "Termo"), podendo ser um valor estático um número, um valor booleano ou um valor textual.

Uma regra dispara uma "Ação" e para isso armazena os parâmetros necessários para execução. A classe "Ação" possui o nome como atributo e pode pertencer a várias dimensões autonômicas, representada pela classe "Dimensão Autonômica", de acordo com a definição do gerente.

Uma "Dimensão Autonômica" possui várias ações e como atributo tem o seu nome, cujos valores são fixos, ou seja, as dimensões da computação autonômica: auto-configuração, auto-cura, auto-otimização e auto-proteção.

4.3.2 – Diagramas de Atividade

Na seção corrente são apresentados os diagramas de atividade dos principais fluxos do sistema. O primeiro deles corresponde à atividade que ocorrer para se iniciarem os agentes dos processos, quando uma nova definição de processo chega através de arquivo XML. Posteriormente, são mostrados os diagramas do funcionamento dos agentes monitores e atuadores. E, por último, vê-se o diagrama de como a ação é disparada após a chegada da notificação de disparo.

Como pode ser visto na Figura 27, quando uma nova definição de processo chega ao AWE, essa é recebida pelo *servlet* nomeado de "UploadAction". Ele inicia o agente, da classe "ProcessDefinitionAgent", da definição do processo. Esse agente fica esperando pelo início de uma instância de processo, e sempre que isso ocorrer, ele inicia o agente da classe "ProcessInstanceAgent". Esse último inicia os devidos agentes monitores e executores e fica à espera do início das instâncias da tarefa. Sempre que uma instância de uma tarefa começa, é também iniciado um agente da classe

"TaskInstanceAgent" para ele, por sua vez, possa iniciar os agentes monitores e executores da tarefa.

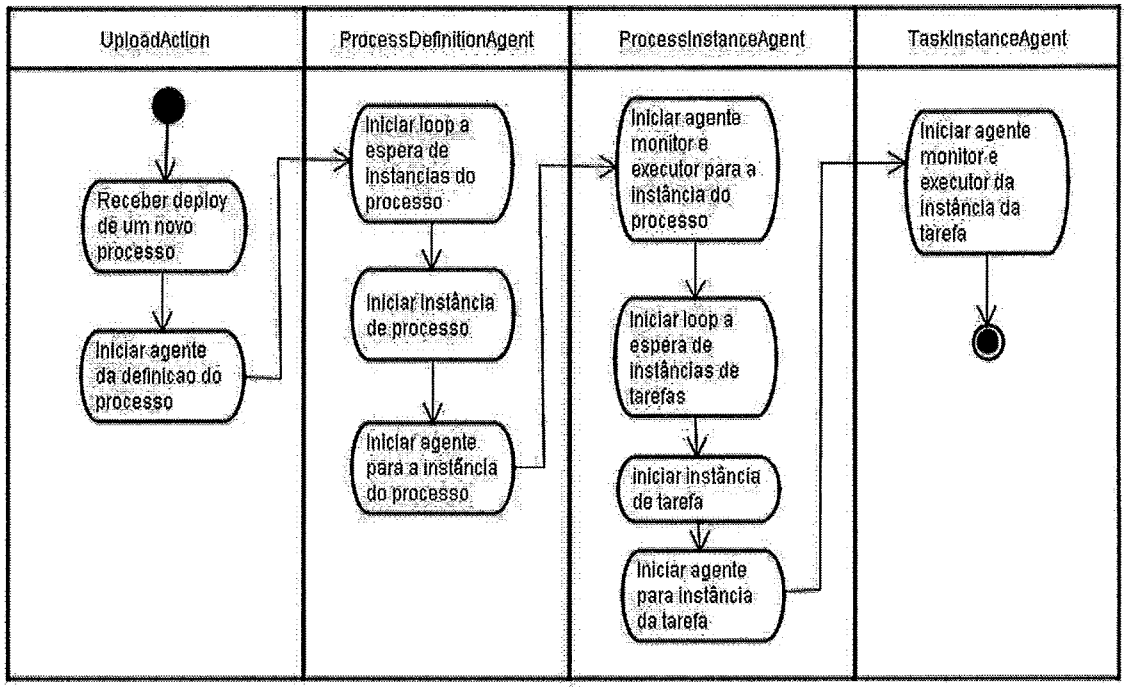


Figura 27 - Diagrama de atividades de iniciação dos agentes

As atividades dos agentes monitores não são difíceis de entender, como é demonstrado na Figura 28. Primeiramente, o agente monitor calcula o valor atual do atributo na tarefa e o compara com o valor armazenado no quadro negro. Caso o valor atual seja diferente do armazenado, ele atualiza o quadro negro. Caso contrário, o agente espera um tempo pré-determinado para verificar novamente. Após a atualização do quadro negro, ele também espera um tempo pré-determinado para verificar novamente o valor do atributo.

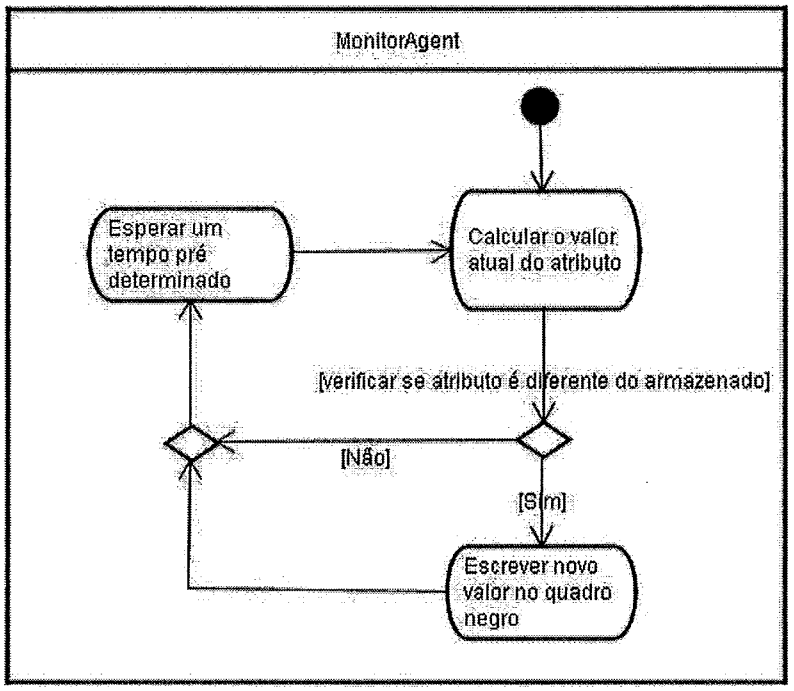


Figura 28 - Diagrama de atividades dos agentes monitores

O agente Atuador começa seu ciclo, como pode ser visto em Figura 29, recuperando as regras referentes à dimensão autonômicas que ele monitora. Assim que recuperadas, ele as coloca no CLIPS, que é um sistema de produção de regras que está inserido no AWE e tem a função de receber as definições das regras autonômicas e os valores dos atributos, e assim verificar se alguma condição está válida, e quando estiver avisar novamente os agentes atuadores. Após isso, ele inicia a busca pelos valores dos fatos associados às condições da regra e, caso eles tenham sido modificados, os insere no CLIPS, como é feita essa inserção é explicado na Seção 4.5. E esse, por sua vez, retira o valor antigo, insere o novo, verifica se alguma regra está válida, e caso esteja, notifica o agente Atuador.

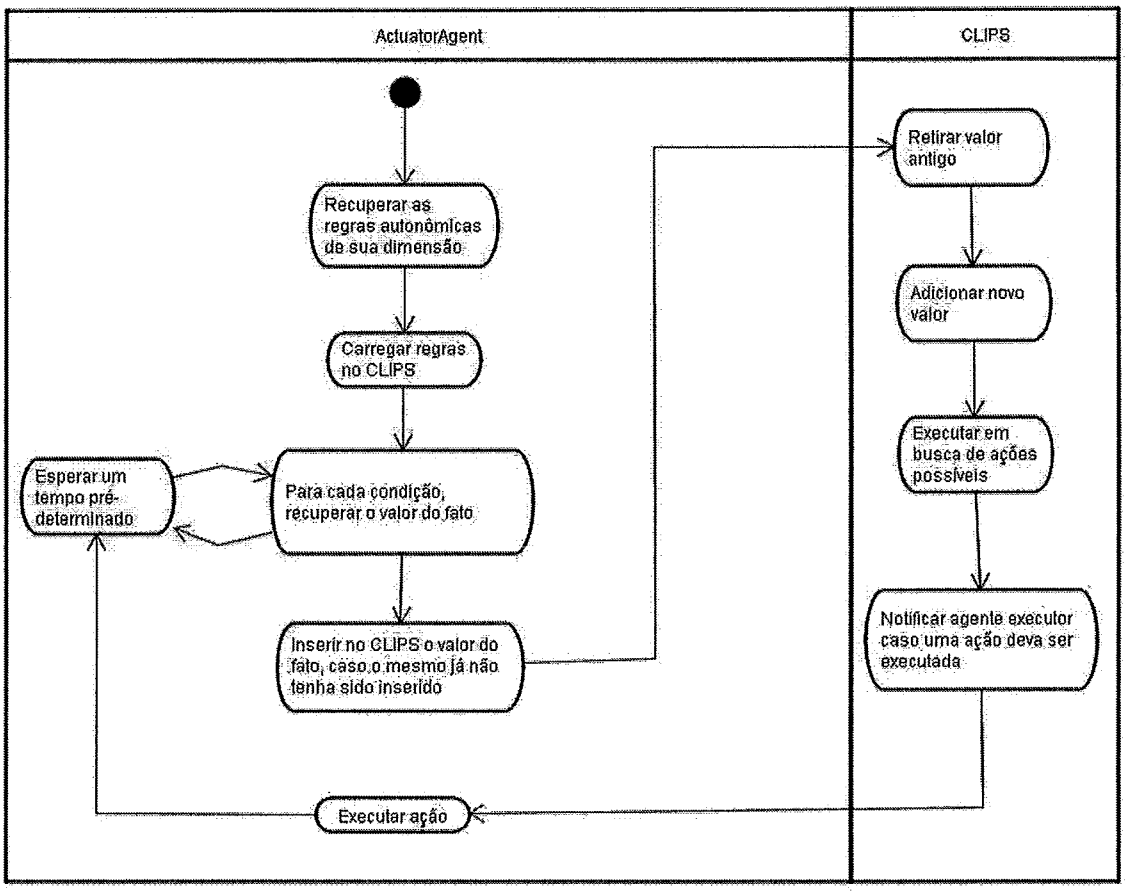


Figura 29 - Diagrama de atividades do agente atuador

Quando o CLIPS percebe que uma regra tem sua condição válida, é disparada uma seqüência de atividades que estão dispostas na Figura 30. Primeiramente, o CLIPS informa a todos os seus observadores, ou seja, todos os agentes atuadores. Esses agentes atuadores verificam se a regra pertence à dimensão autonômica e se a tarefa (ou processo) também está sob sua responsabilidade. Caso ambos sejam verdadeiras, ele recupera os parâmetros que devem ser passados e instancia a classe referente à ação. Após isso, finalmente, é executada a ação devida.

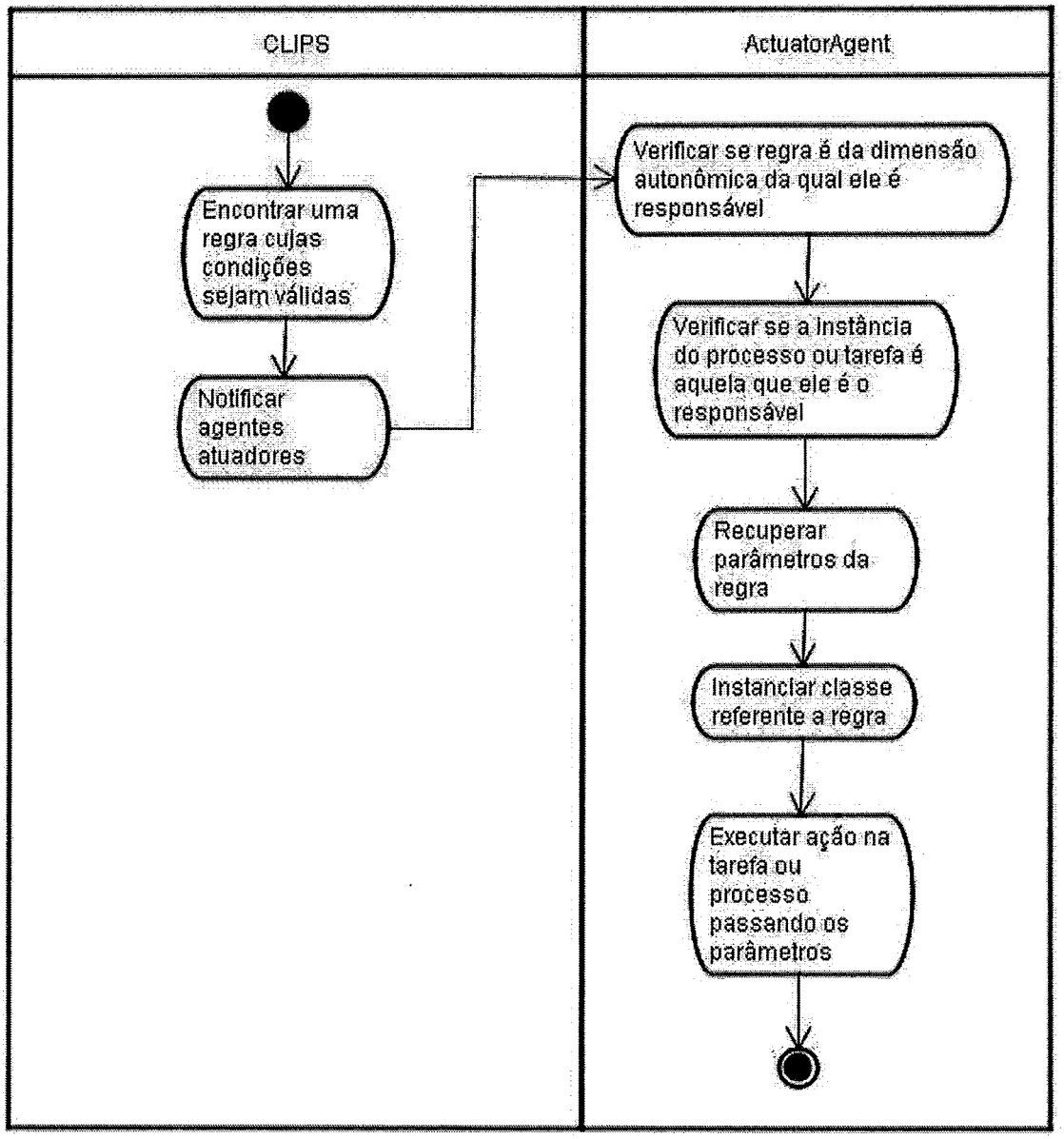


Figura 30 - Fluxo de execução de uma ação

4.3.3 – Diagramas de Classes dos Agentes

Nesta seção são apresentados os diagramas de classe dos agentes. O primeiro deles será o diagrama dos agentes das instâncias dos processos e das tarefas. Posteriormente, o diagrama dos agentes monitores seguido pelo diagrama dos agentes atuadores juntamente com as classes das ações.

O agente "ProcessDefinitionAgent" guarda a definição do processo pela qual é responsável e pode ter vários agentes do tipo "ProcessInstanceAgent", que armazenam a

instância do processo e só podem ter um "ProcessDefinitionAgent" associado, porém podem ter vários "TaskInstanceAgent" os quais possuem um único "ProcessDefinitionAgent" associado e a instância da tarefa relacionada. Tanto o "ProcessInstanceAgent" quanto o "TaskInstanceAgent", possuem zero ou mais "MonitorAgent" e "ActuatorAgent", enquanto que esses últimos devem estar associados a pelo menos um dos agentes anteriores, respectivamente. Essas associações podem ser visualizadas na Figura 31.

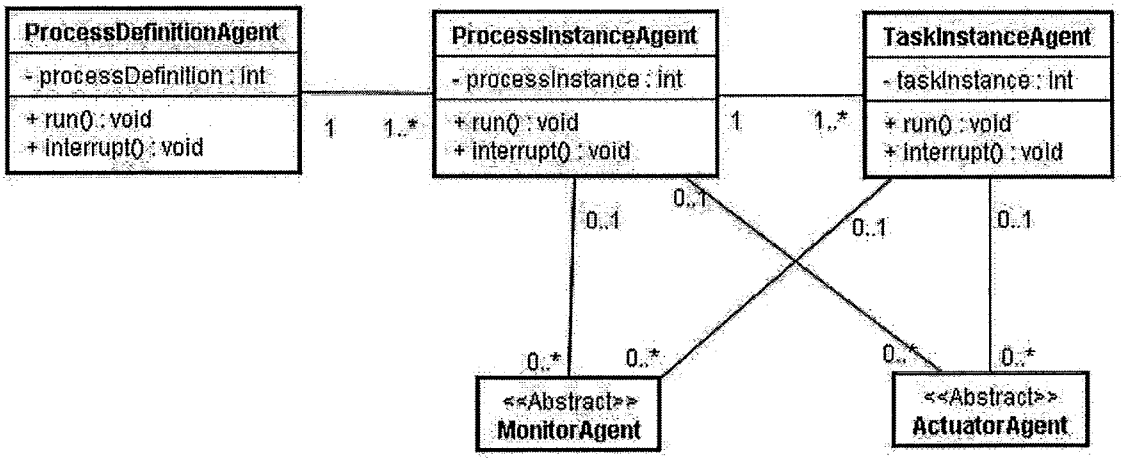


Figura 31 - Diagrama de classe diagrama dos agentes das instâncias dos processos e das tarefas

No diagrama dos agentes monitores da Figura 32, pode-se observar que o pai na hierarquia é o "MonitorAgent", porém ele tem o estereótipo abstrato, que não o permite ser instanciado. Nele, estão definidos os métodos e atributos que serão sobrescritos ou definidos pelos seus filhos.

Os filhos diretos de "MonitorAgent" também são abstratos e representam os tipos de recursos: "FinancialMonitorAgent", "InfrastructureMonitorAgent", "HumanMonitorAgent", "MaterialMonitorAgent" e "TimeMonitorAgent".

Cada um deles, também possui filhos, e esses que serão instanciados e representam cada tipo de atributo: "CostMonitorAgent", "DocumentMonitorAgent", "RawMaterialMonitorAgent", "ServerMonitorAgent", "SystemMonitorAgent",

"DurationMonitorAgent", "FinalDateMonitorAgent" e "PersonBySkillMonitorAgent". Os atributos dessas classes são:

- *resource*: o tipo de recurso (humano, material, financeiro, entre outros);

- *attribute*: o tipo de atributo (custo, servidor, documento, etc.);

- *bbValue*: o último valor do atributo colocado no quadro negro; e

- *attributeName*: o nome do atributo para aqueles que necessitam (por exemplo, para o atributo Servidor pode-se ter o nome Banco de Dados).

Além disso, possuem mais dois atributos que indicam a instância da tarefa ou processo, chamando respectivamente de *taskInstance* ou *processInstance*, sendo que quando um estiver preenchido o outro deve possuir valor nulo.

Com relação aos métodos, o primeiro, nomeado de *getFactotyInstance*, que utiliza o padrão de projeto *Factory Method* (Gamma *et al*, 1995), e que recebe o atributo como parâmetro, é aquele usado para instanciar o "MonitorAgent" com a classe correta. Depois vem o que executa o fluxo do agente, em um *loop* infinito, chamado de *run*. Nesse *loop*, primeiramente é calculado o valor atual do atributo através do método *calculateValue*. Esse método é abstrato. Logo o seu algoritmo é descrito em cada uma das classes não abstratas. Após calcular o valor, esse método verifica se o valor é diferente do armazenado no quadro negro, e caso seja, esse novo valor é adicionado no quadro negro através do método *addValueInBB*. Caso seja a primeira inserção, é necessário iniciar o quadro negro com o *initializeBB*. Nesses dois últimos métodos, é usado um método, *getBBName*, para definir o nome do quadro negro. Um último método, o *getType*, é usado para definir se trata-se de uma instância de processo ou de tarefa, e pode ser necessário tanto no cálculo do valor atual quanto na definição do nome do quadro negro. Por questões de simplificação, os *getters* e *setters* foram omitidos.

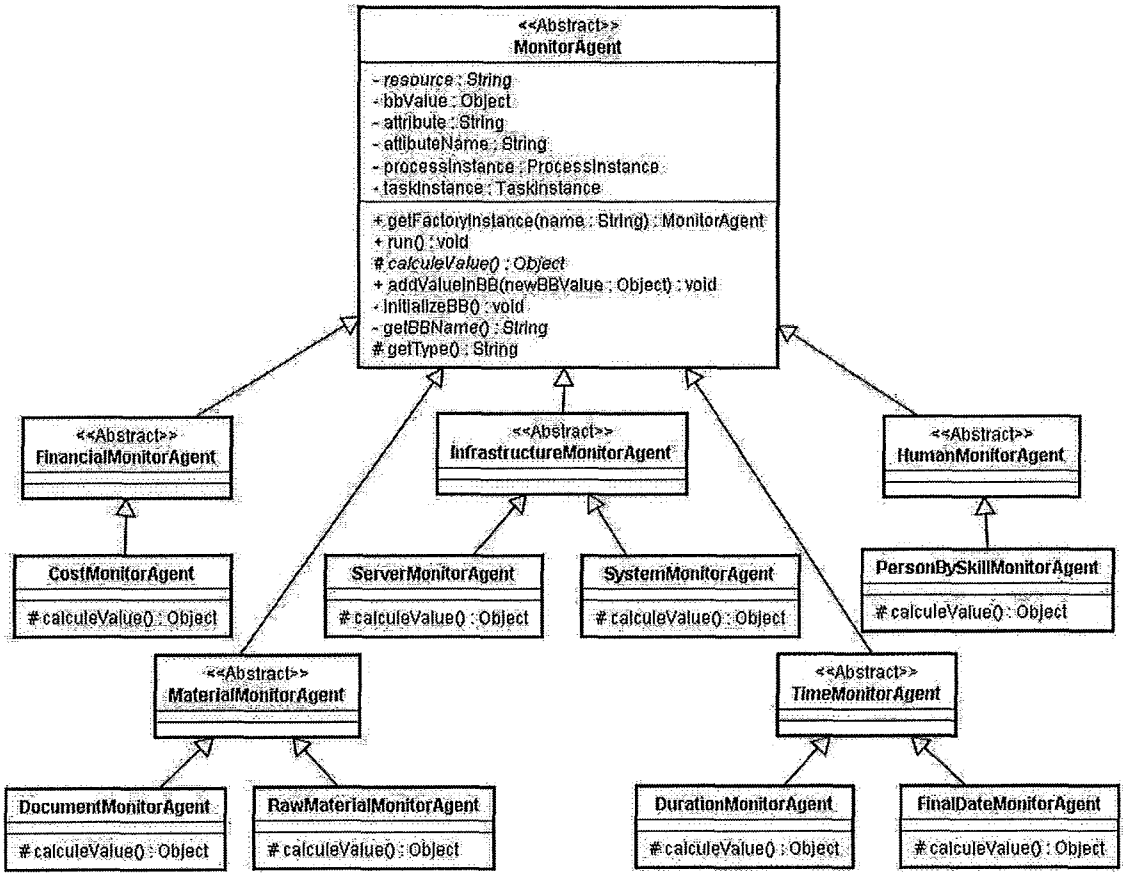


Figura 32 - Diagrama de classes dos agentes monitores

No que diz respeito aos agentes atuadores, vê-se na Figura 33, o "ActuatorAgent" como classe abstrata, com filhos representando cada dimensão autonômica: "ConfigurationActuatorAgent", "HealingActuatorAgent" e "OptimizationActuatorAgent".

Esses filhos definem o valor do atributo *dimension* com o nome da dimensão autonômica pela qual são responsáveis e podem sobrescrever os métodos da classe abstrata superior. Dois atributos que indicam a instância da tarefa ou processo são chamandos respectivamente de *taskInstance* ou *processInstance*, sendo que quando um tiver preenchido o outro deve possui valor nulo. Existe mais uma associação, representada pelo atributo *jClipsFacade*, à classe JClipsFacade, que interage diretamente com o CLIPS.

O primeiro método de um "MonitorAgent" definido é o *getFactoryInstance* que também utiliza o padrão de projeto *Factory Method* (Gamma *et al*, 1995) para retornar o tipo de agente não abstrato que será iniciado. A seguir, tem-se o método *run*, que é responsável pelo monitoramento do agente no quadro negro e pela inserção dos valores no CLIPS. Há ainda os métodos:

- *getRulesFromDimension*: para recuperar as regras que foram definidas para a tarefa (ou processo);

- *loadRuleInJClips*: carregar essas regras no CLIPS;

- *getValueFormBB*: buscar valor do atributo no quadro negro;

- *getBBName*: buscar o nome do quadro negro;

- *loadAttributeInJClips*: carregar os atributos no CLIPS; e

- *verifyIfRuleApplied*: verificar se a regra recebida pelo CLIPS se aplica a ele.

No método, *doAction*, de executar a ação, o agente instancia a classe correspondente à ação e chama o método *doAction* dessa classe, passando o tipo (tarefa ou processo), o identificador instância da tarefa (ou processo) e os parâmetros da regra como parâmetros para esse método.

As classes de ação, por sua vez, têm o pai abstrato chamado de "Action" com os filhos "CancelTask", "ReturnPreviousActivity", "CancelProcess", "AddNewExecutor", "RemoveExecutor", "NotifySpecialist", "InformUpLevel", "UseAnotherSystem", "ReplaceExecutor", e "LooKingForBetterFlow". Na classe "Action", há ainda o método getF*actoryInstance* para retornar qual desses filhos será instanciado.

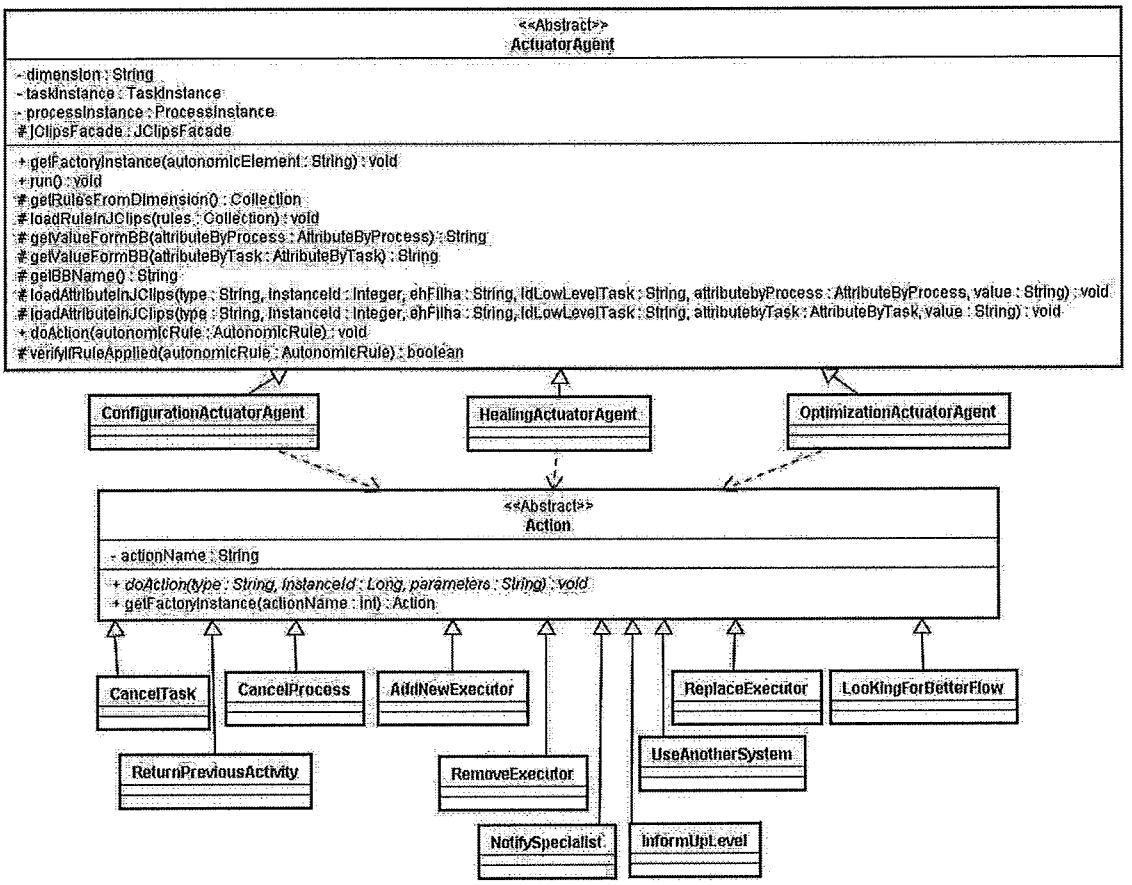


Figura 33 - Diagrama de classe dos agentes atuadores e das ações

4.4 – Procedimentos de monitoramento e da ação

É apresentado abaixo como os agentes monitores calculam os valores para preenchimento dos quadros negros e como as ações são executadas no sistema.

4.4.1 – Procedimento de cálculo dos valores do quadro negro

Cada agente monitor é responsável por um tipo de atributo, assim cada um calcula seu valor de forma diferente.

Os agentes responsáveis por cada um dos atributos Servidor, Sistema, Documento e Material são os mais simples. Eles simplesmente verificam no banco de dados, na tabela "Monitor Atributo", o estado do atributo em questão, que é atualizado via serviços *web*.

O agente de Pessoas por Capacidade recupera todos os executores da tarefa e, para cada um deles, o agente requisita, via serviços *web*, os seus conhecimentos, para o sistema responsável por gerenciar de recursos humanos. A quantidade de recurso humano com cada conhecimento é totalizada e, ao final, é escrita no quadro negro a quantidade de cada conhecimento.

No JBPM Web Console, cada tarefa executada possui um formulário de execução e os campos desse formulário são definidos pelo modelador como variável de controle. Sempre que o atributo custo for monitorado, é criado um campo Custo no formulário de execução da tarefa e o usuário deve preencher seu valor. Logo é representado no JBPM como uma variável de controle com nome *custo*. O agente responsável percorre todas as variáveis de controle da tarefa até que encontre aquela que tenha nome *custo*. Ao encontrá-la, recupera o seu valor, que é o custo da tarefa. Para um processo, o custo é a soma do custo de todas as suas tarefas.

O agente responsável pela Data Final escreve no quadro negro o valor da data atual. Assim, o CLIPS, para verificar se a data final já foi ultrapassada, compara a data atual com a data final esperada.

Para a Duração, o valor escrito no quadro negro é a diferença entre o início da tarefa e a data e a hora atual.

4.4.2 – Procedimento das ações executadas

Para as ações "Cancelar Tarefa" e "Cancelar Processo", é chamado um método cujo nome é *end* e já é definido, tanto para tarefa quanto para processo, pela máquina de workflow JBPM. Após isso, é escrito no quadro negro superior que a tarefa foi cancelada.

"Notificar especialista via correio eletrônico" recupera os parâmetros com o endereço e com o texto a ser enviado, e envia a mensagem para o destinatário.

Para "Retornar a uma Atividade Anterior", o sistema, primeiramente, recupera o parâmetro com o nome da atividade anterior a ser executada. Depois, procura a instância já terminada com o nome da tarefa anterior e termina a tarefa que estava sendo executada, através do método *end*. Posteriormente, reinicia novamente a tarefa anterior, apagando o campo correspondente a data de fim da tarefa e executa o método *resume*, já definido pela máquina de workflow.

Como esta dissertação se propõe a tratar tarefas que são executadas por um ser humano, e não por um computador ou uma máquina, a ação Usar outro Sistema apenas recomenda o uso de outro sistema para o usuário, para que o mesmo não fique parado quando o sistema oficial estiver fora do ar.

Para Adicionar um Novo Executor, são pedidos ao sistema gerenciador de recursos humanos os executores que não excedam a quantidade máxima esperada pela tarefa em ordem de propriedade, e o primeiro que não tiver alocado a outra tarefa em execução no sistema, é alocado.

A Remoção de um Executor é feita de forma que seja retirado o primeiro executor que não deixe em falta a quantidade mínima esperada para a tarefa.

Já para a Substituição de Executor, primeiro é feita uma remoção, e posteriormente uma adição de executor.

4.5 – CLIPS

Esta seção apresenta como o sistema desenvolvido utiliza o CLIPS para tratamento das regras. Primeiramente, é demonstrado como os fatos são inseridos no CLIPS, e, posteriormente, é explicado a inserção das regras.

4.5.1 – Inserindo fatos

A inserção de fatos no CLIPS é feita de forma muito parecida para todos os atributos. A estrutura é a seguinte:

(assert (Tipo IdInstancia ehFilha idFilha Atributo Nome Valor))

Onde "Tipo" é um valor que identifica o nível do processo, com as únicas opções de *processo* ou *tarefa* e "IdInstancia" é o identificador da instância na base de dados onde está configurado a máquina de workflow. Para "ehFilha" é preenchido ou *sim* ou *não*, indicando se esse valor do atributo se refere a alguma tarefa de nível inferior. Enquanto que "idFilha" é o identificador dessa tarefa, sendo "null" quando não houver. "Atributo" se refere às já conhecidas opções vistas na coluna da direita da Tabela 2, tendo o valor *null* quando se referir a uma tarefa filha.

Para o campo "Nome", nos atributos Servidor, Sistema, Documento e Material, é o nome definido para os atributos, como visto na Seção 4.1.1. No atributo Duração, tem-se o texto *Duração*. Em Data Final, é colocado *DataAtual*. No atributo Custo, corresponde a unidade monetária utilizada. E, finalmente, no atributo Pessoas por Capacidade, é o nome do conhecimento. Quando se referir a uma tarefa filha, o valor é *null*.

E para o campo "Valor", primeiramente para os atributos Sistema, Documento e Material são as opções "Disponível" ou "Indisponível". Já para o atributo Servidor, é Ativo ou Inativo. Para custo é o valor monetário e para Pessoas por Capacidade é a quantidade de pessoas que possuem aquele determinado conhecimento executando a tarefa. Em Duração, tem-se a quantidade de minutos que já decorridos da tarefa e em Data Final é posto o valor em *timestamp* da data atual. Quando se referir a uma tarefa filha, o valor é Iniciado, Cancelado (Finalizado sem sucesso), Finalizado com sucesso ou o valor textual escrito no quadro negro definido na modelagem.

Na Tabela 4, abaixo, será visto um exemplo para cada tipo de atributo:

Tabela 4 - Exemplo de fato inserido no CLIPS

Atributo	Exemplo de fato inserido no CLIPS	Explicação
Custo	(assert (processo 01 não null Custo Dolares 3000))	Custo atual é 3000 dólares.
Pessoas por Capacidade	(assert (tarefa 02 não null PessoasPorCapacidade Java 2))	2 pessoas com conhecimento em Java estão executando o processo.
Servidor	(assert (processo 03 não null Servidor JBoss Ativo))	Servido JBoss se encontra ativo.
Duração	(assert (processo 07 não null Duracao Duracao 3600))	A duração atual é 3600 minutos.
Data final	(assert (tarefa 08 não null DataFinal DataAtual 1230868118))	A data atual é 1230868118 em *timestamp*, ou seja, 2 de janeiro de 2009 as 3:48 (GMT)

4.5.2 – Inserindo regras

Esta subseção é dividida em condições e ações. Na inserção da regra, no CLIPS, não há essa separação.

4.5.2.1 – Inserindo condições

Para uma regra ter sua ação disparada, o CLIPS examina a condição da regra verificando se é verdadeira. Porém, antes de analisar a regra, é necessário que o agente atuador do AWE transforme as condições de uma regra em algo que o CLIPS entenda. A condição transformada é dividida em duas partes: recuperação de valores e teste.

Para a recuperação de valores, é feita uma listagem de todos os atributos que são necessários. Esses são individualmente colocados no início da condição, substituindo o valor por uma variável, como "?Variavel". O caractere "?" no CLIPS indica que os

caracteres que vierem depois, até um espaço em branco, formam um nome de uma variável. Cada condição tem a seguinte forma:

Tipo IdInstancia ehFilha idFilha Atributo Nome ?Variavel

Onde Tipo, IdInstancia, ehFilha, idFilha, Atributo e Nome têm a mesma definição feita na inserção de fatos e "?Variavel" é o nome a variável que armazenará o valor do atributo.

Após a recuperação do valor de todos os atributos necessários, é feita a comparação deles, para verificar se atendem à condição definida pelo usuário no modelador. Para isso, o CLIPS possui uma função chamada *test* que examina o fato passado a ela como parâmetro e informa se é verdadeiro ou falso. Por exemplo: (test (= 1 1)), que verifica que 1 é igual a 1 e retorna verdadeiro.

A comparação numérica é feita como no exemplo acima. Porém, para comparação de valor textual, é usada a função *str-compare* com dois parâmetros, sendo que quando um parâmetro é uma variável, é necessário concatenar essa variável com um texto vazio, usando assim a função *sym-cat*.

A função *str-compare* não retorna verdadeiro ou falso, mas sim -1, 0, ou 1 indicando respectivamente se o primeiro parâmetro é maior que o segundo, se são iguais ou o segundo é o maior, em termos de ASCII. Para contornar esse problema, basta executar *test* com o comparador (=, <>, <, >,>=, <=) necessário a esses valores. Por exemplo, para verificar se a variável "?x" é igual ao texto "Inativo", tem-se: (test (= (str-compare "Inativo" (sym-cat ?x "")) 0)).

Outra função útil do CLIPS é a função *or* que pode ser passada como parâmetro para *test* e, facilitando assim, fazer a disjunção. Para formar a conjunção, basta inserir mais uma linha na função *test*. Por exemplo, para verificar se a variável "?x" é igual ao

texto "Inativo" ou se "?y" é igual a "Ativo", tem-se a seguinte expressão: (test (or (= (str-compare "Inativo" (sym-cat ?x "")) 0) (= (str-compare "Ativo" (sym-cat ?y ""))0))).

Para definir as prioridades, na definição da regra, deve-se definir a importância da regra através de um valor numérico, e aquela regra que possuir maior número, tem maior importância. Para definição dessa importância, no CLIPS, usa-se a função (declare (salience VALOR_NUMERICO)) e ela deve ser colocada logo abaixo da linha que faz a declaração do nome da regra.

Na Tabela 5, é apresentado um exemplo para cada tipo de atributo. Para simplificação, o nome da regra é apenas o nome do atributo, e a ação é a exibição desse nome. No sistema desenvolvido, o nome da regra é formado pela palavra "regra" concatenada com o identificador da mesma. O nome da ação é explicado a seguir na Seção 4.5.2.2.

Tabela 5 - Exemplo de condição inserida no CLIPS

Atributo	Exemplo de condição inserida no CLIPS	Explicação
Custo	(defrule custos (declare (salience 10)) (processo 01 no null Custo Dolares ?x) (test (> ?x 3)) => (printout t "custos" crlf))	Verifica se o custo é maior que 3 doláres.
Pessoas por Capacidade	(defrule conhecimento (declare (salience 10)) (tarefa 02 no null PessoasPorCapacidade Java ?x) (tarefa 02 no null PessoasPorCapacidade MySql ?y) (test (= ?x 3)) (test (= ?y 2)) => (printout t "conhecimento" crlf))	Verifica se existem 3 pessoas com conhecimento em Java e 2 em Banco de Dados.
Servidor	(defrule servidor (declare (salience 10)) (processo 03 no null Servidor JBoss ?x) (processo 03 no null Servidor FTP ?y) (test (or (= (str-compare "Inativo" (sym-cat ?x "")) 0) (= (str-compare "Inativo" (sym-cat ?y "")) 0)))	Verifica se o servidor JBoss ou o servidor FTP estão inativo.

	=> (printout t "servidor" crlf))	

4.5.2.2 – Inserindo ações

A ação no CLIPS não é a ação que efetivamente irá ser executada, mas sim um aviso ao AWE que uma determinada ação deve ser executada. A ação inserida no CLIPS, utiliza uma função chamada de *send-to-java* que retorna um conjunto de caracteres a todos os objetos Java observadores. A ação inserida no CLIPS tem a seguinte forma:

(send-to-java tipo=Tipo&idInstancia=IdInstancia&idRule=IdRule)

Onde Tipo indica se é uma tarefa ou um processo, IdInstancia é o identificador da instância do mesmo e IdRule o identificador da regra que foi encontrada. É possível reparar que esses três valores possuem o par "nome=valor" e são separados pelo caractere "&", assim como em HTML.

Com esses valores recebidos, o agente atuador, no AWE, pode recuperar a ação juntamente com seus parâmetros para, assim, efetuar a sua execução.

Capítulo 5 – Estudo de caso

Com o intuito de validar a idéia e as ferramentas desenvolvidas, foi realizado um estudo de caso. Este capítulo apresenta a definição e os objetivos desse estudo de caso. Posteriormente, foram feitas algumas análises dos resultados obtidos. Finalmente, é mostrado a execução de um processo autonômico .

5.1 – Definição e objetivos do estudo de caso

O estudo de caso foi realizado com alunos de graduação, na disciplina de Banco de Dados II, do curso de Ciência da Computação da Universidade Federal do Rio de Janeiro. Esses alunos desenvolveram duas ontologias (problemas e ações) e modelaram um processo de negócio. Ao final, foram feitas críticas para melhorias do ABPM.

Para a modelagem, foi usado um texto sobre um processo fictício de reservas e vendas de passagem, de uma companhia imaginária de linhas aéreas, a Cia. Atlântica. A descrição do processo se encontra no Anexo A.

Os alunos foram divididos em grupos de dois tipos: um tipo usou o JBPM e o outro utilizou o ABPM. O objetivo principal era verificar a dificuldade em modelar certas exceções no fluxo normal e, também, verificar a cobertura e precisão dos modelos, pois com o ABPM são gerados processos visualmente mais limpos. Um objetivo secundário era testar e evoluir a ferramenta. Eles também tiveram que executar o processo modelado no estudo de caso simulando algumas exceções, com objetivo obter um *feedback* sobre a modelagem, e assim corrigí-la se necessário.

5.2 – Análise do Estudo

A variável **tempo** não foi trabalhada nesse estudo de caso, porque a execução dos workflows tratados nesta dissertação é dependente de intervenção do usuário, o que demanda uma simulação complexa. No caso do ABPM, o sistema notifica sobre a ocorrência de um problema. Enquanto que sem o ABPM, tudo dependeria da atenção e da freqüência em que o usuário verifica o seu processo.

A construção de ontologias pelos alunos não tinha como objetivo o desenvolvimento de novas ontologias, nem suas avaliações no estudo. Visava apenas aumentar o conhecimento dos alunos no domínio da aplicação, para que assim pudessem opinar e questionar os atributos e as regras autonômicas do sistema, o que contribui positivamente para qualidade de seus modelos.

A execução do processo modelado também não visava à avaliação, como citado anteriormente. O modelo original feito com o ABPM pode ser visualizado na Figura 34, enquanto que as regras de negócio se encontram na Tabela 6. Na Figura 35 pode ser visto o modelo original sem o ABPM, com a inserção manual do reconhecimento das falhas. A descrição do processo da Cia. Atlântica encontra-se no Anexo A.

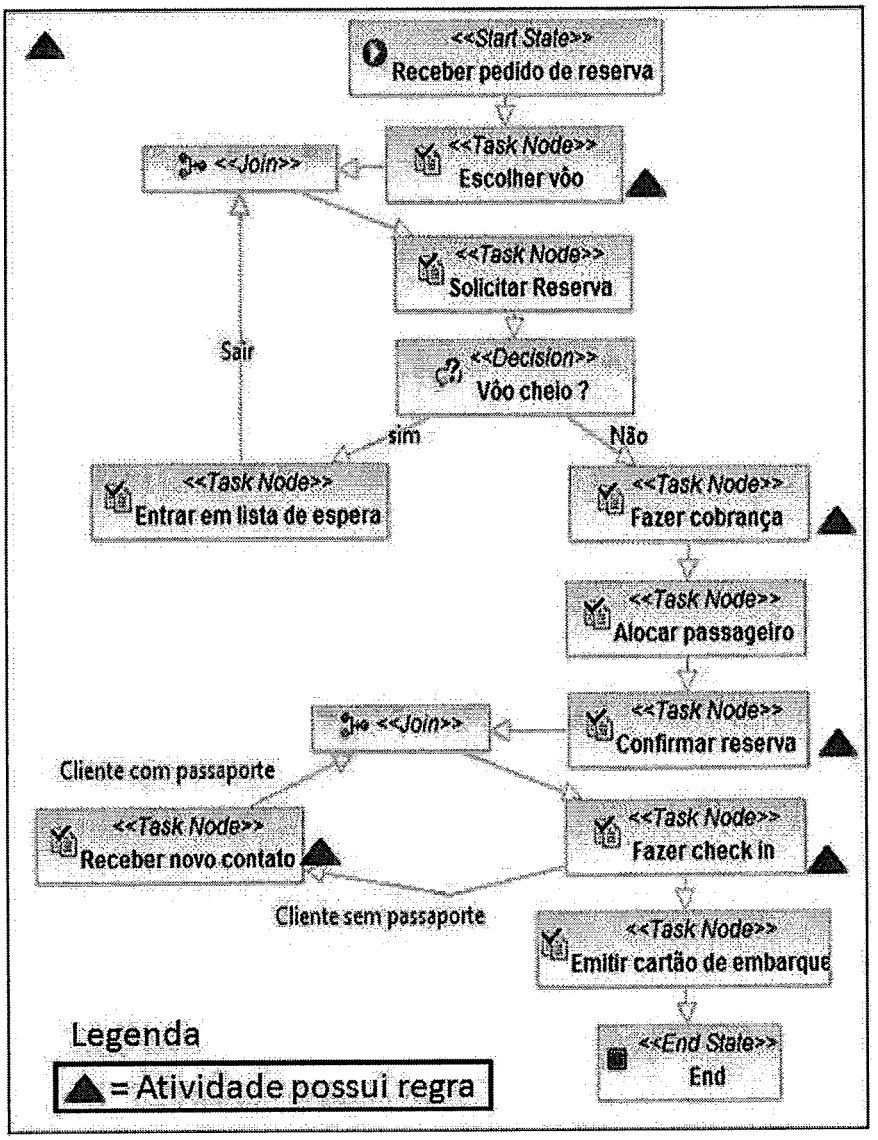


Figura 34 - Modelo original do processo da Cia. Atlântica feito com o ABPM

Tabela 6 - Gabarito das regras do estudo de caso

Atividade	Regra
PROCESSO	**se** atividade Fazer *Check-in* ou Receber novo contato estiver cancelada, **então** Cancelar Processo
ESCOLHER VÔO	**se** Sistema de Reservas estiver indisponível, **então** usar sistema Central Telefônica
FAZER COBRANÇA	**se** conhecimento em cobrança estiver indisponível, **então** substituir executor
CONFIRMAR RESERVA	**se** duração > 10 minutos, **então** notificar especialista
FAZER *CHECK-IN*	**se** a data atual for maior que a data do vôo, **então** cancelar tarefa
RECEBER NOVO CONTATO	**se** a data atual for maior que a data do vôo, **então** cancelar tarefa

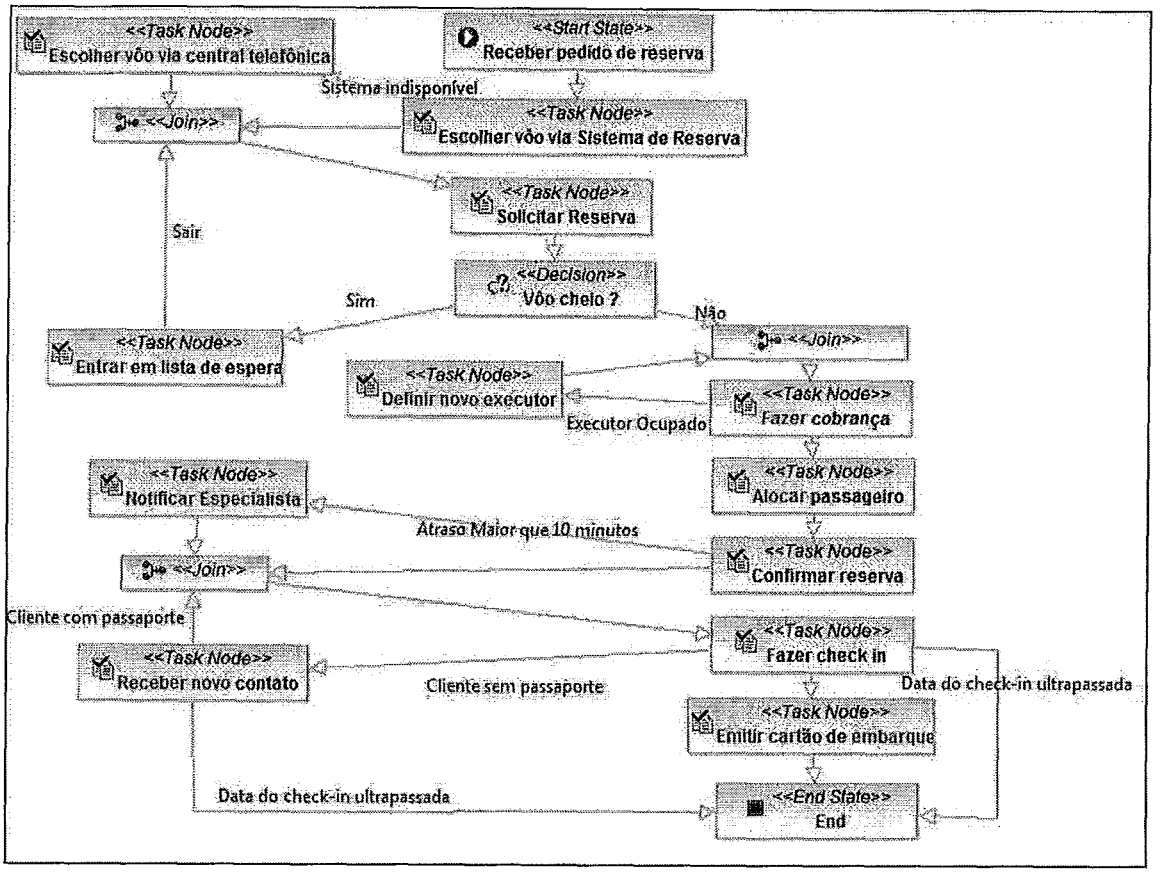


Figura 35 - Modelo original do processo da Cia. Atlântica, feito sem o gestor autonômico

Algumas métricas foram medidas com o intuito de se avaliar o uso da ferramenta, o resumo delas pode ser visto na Tabela 7.

Tabela 7 - Comparação da modelagem do ABPM com o JBPM

	JBPM	ABPM
COBERTURA DAS ATIVIDADES	85%	80%
COBERTURA DAS REGRAS	40%	90%
NÚMERO DE ATIVIDADES	15,7 atividades	11 atividades

A primeira das métricas é a **Cobertura das Atividades**, ou seja, se todas as atividades esperadas para o processo foram modeladas. Pode ser visto, como já era previsto, que não há grandes diferenças entre a cobertura de ambas, já que o ABPM não se propõe a atuar nesse quesito. A cobertura de atividades não foi 100% porque os

alunos não são especialistas no domínio da aplicação, e, assim, acabam cometendo erros de modelagem. Isso não inviabilizou o estudo de caso, pois todos os grupos possuíam a mesma experiência.

Quanto à **Cobertura das Regras**, a diferença entre ambas é visível. Pode-se observar que, devido a complexidade da modelagem do fluxo, no caso do tratamento de uma exceção manualmente, várias regras foram deixadas de lado. Sem o ABPM, apenas 40% das regras esperadas foram mapeadas, enquanto que com o ABPM foram 90%. Uma regra foi considerada mapeada, quando sem o gestor autonômico, sempre que há no fluxo a possibilidade de inserção, manualmente, das exceções geradas, gerando novos fluxos, como na Figura 36. Acreditamos que essa diferença no resultado se deve ao fato de que seja mais intuitivo ao gerente modelar o tratamento das exceções como regras, ao invés de modelá-las como fluxo, visto que no seu dia-a-dia ele costuma seguir regras, e não fluxos. Segundo Consentino (2006), o cérebro trata as atividades organizadas e repetitivas como scripts, tornando, portanto, tal enfoque mais intuitivo.

O **Número de Atividades** avalia a diferença na quantidades de atividades que foram necessárias para mapear o processo com relação ao número esperado. O objetivo da métrica é verificar se com a ferramenta desenvolvida, o número de tarefas diminui. Esse indicador não visa a afirmar que o ABPM é melhor ou pior nesse quesito, mas sim nos levar questão interessante, como, já que enquanto usando o ABPM são modelados processos criando menos atividades e, por isso, visualmente mais limpos, sem ele tem-se um maior nível de detalhe visualmente acessível no processo.

Como a criação de regras por meio de inserção de decisões e atividades é uma tarefa árdua, acaba-se multiplicando o número de atividades. Como exemplo, pode-se citar uma atividade que utilize o Sistema de Reserva. Pode acontecer do mesmo estar fora do ar, sendo substituído pelo uso do telefone através da Central de Atendimento.

Nesse caso, como visto na Figura 36, sem o ABPM, é verificado antes de começar a tarefa se o sistema está ou não disponível, criando-se uma atividade para cada sistema. Caso o sistema falhe antes do início da atividade, essa opção parece plausível. Entretanto, se o sistema falhar durante a execução da tarefa, o mecanismo executor não saberá o que fazer. Para conseguir capturar uma exceção, sem o ABPM, durante uma atividade, teria que existir uma opção na tarefa, para incluir manualmente uma exceção, e assim, terminar essa tarefa, para o processo seguir outro fluxo. Logicamente, todos esses fluxos devem estar devidamente mapeados. Isso ainda gera outro problema, pois supondo que uma série de tarefas utilize o mesmo sistema, o modelador tem que criar uma decisão em cada uma delas, verificando se o sistema está disponível, poluindo demais a visualização do processo, e, conseqüentemente, prejudicando a sua legibilidade e a sua inteligibilidade. Para esse caso, no ABPM, bastaria criar um atributo confirmando que o valor esperado do sistema é disponível, e uma regra informando que se o sistema cair, deve-se utilizar outro. Isso foi feito pelos grupos que utilizam a ferramenta desenvolvida.

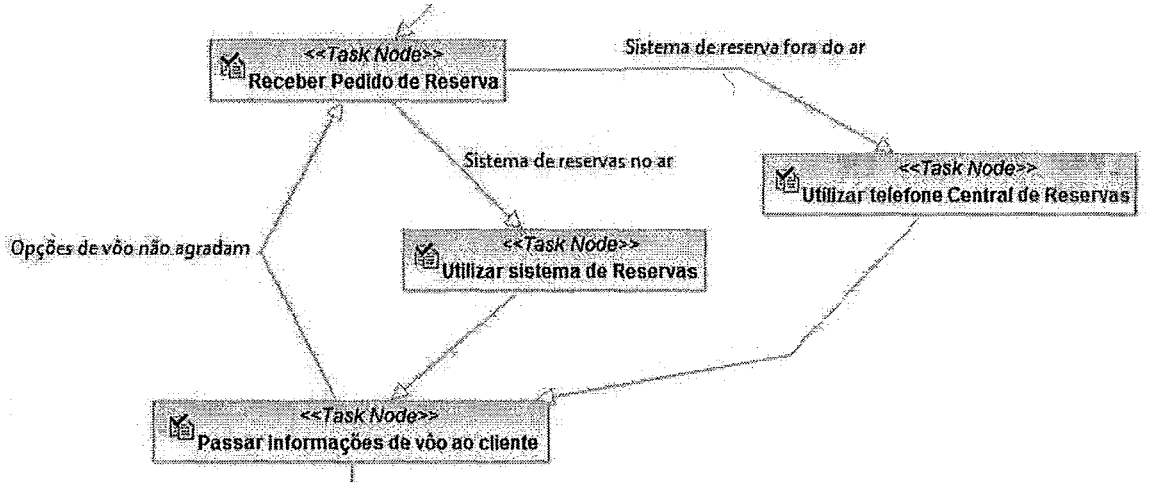


Figura 36 - Opção para criação de tarefas quando um sistema falhar sem o ABPM

5.3 – Execução de um processo autonômico

Esta subseção tem como objetivo mostrar o passo a passo da execução de um processo autonômico. Foi escolhido como exemplo o processo fictício de reservas e vendas de passagem, de uma companhia imaginária de linhas aéreas, modelado no estudo de caso. A seguir serão listados cada um dos passos:

- **Passo 0: Início do workflow.**

É onde se inicia o processo, o cliente procura a empresa. O atendente inicia o workflow, através da interface vista na Figura 37, e executa a tarefa de recebimento do pedido de reserva, como na Figura 39.

Ao se iniciar uma instância do processo, é iniciado um agente Gerente do Processo, além de criar os agentes Monitores e Atuadores para essa atividade, ele monitora o fluxo para iniciar um agente Gerente da Tarefa sempre que uma tarefa for criada. Nesse exemplo, é instanciado o agente atuador da dimensão auto-configuração que contém uma regra para cancelar o processo quando a atividade "Fazer *Check-in*" ou "Receber novo contato" for cancelado. Não há criação de Monitores porque a única regra tem sua condição dependente exclusivamente de atributos de atividades de níveis inferiores. Também é criado o Gerente da Tarefa da atividade "Receber pedido de reserva". Pode ser vista na Figura 38 uma interface para acompanhamento do Agente Gerente do Processo.

Como na atividade "Receber pedido de reserva" não existe regras autonômicas, o agente Gerente da Tarefa não inicia nenhum agente Monitor e nem Atuador.

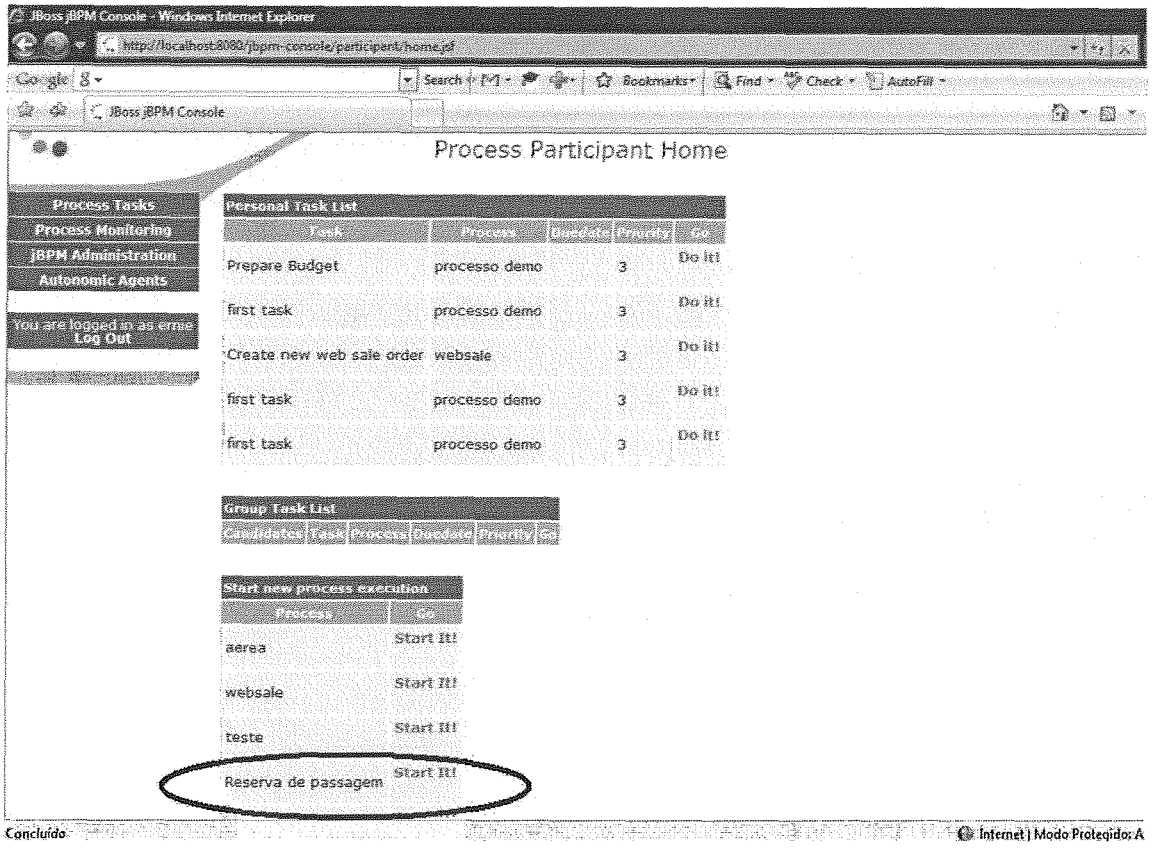


Figura 37 - Início Processo

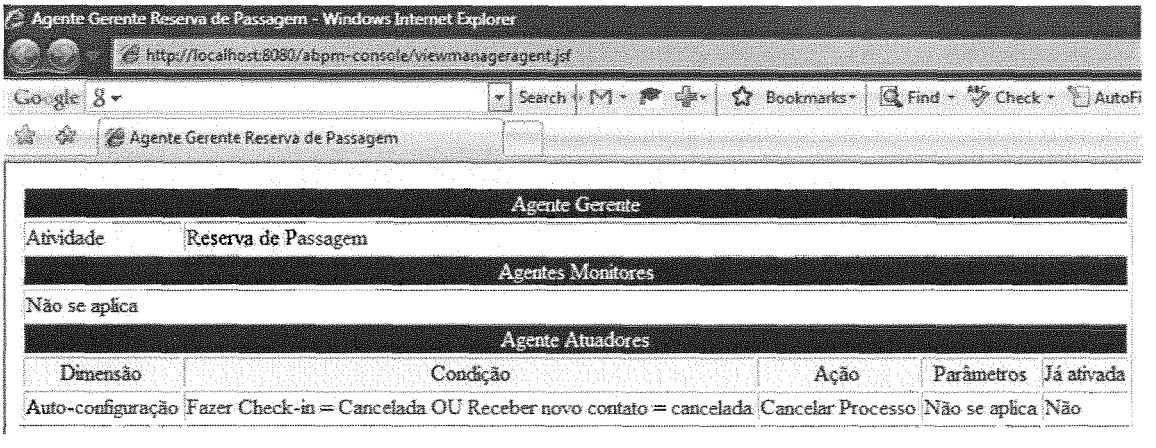


Figura 38 – Interface de acompanhamento do Agente Gerente do Processo

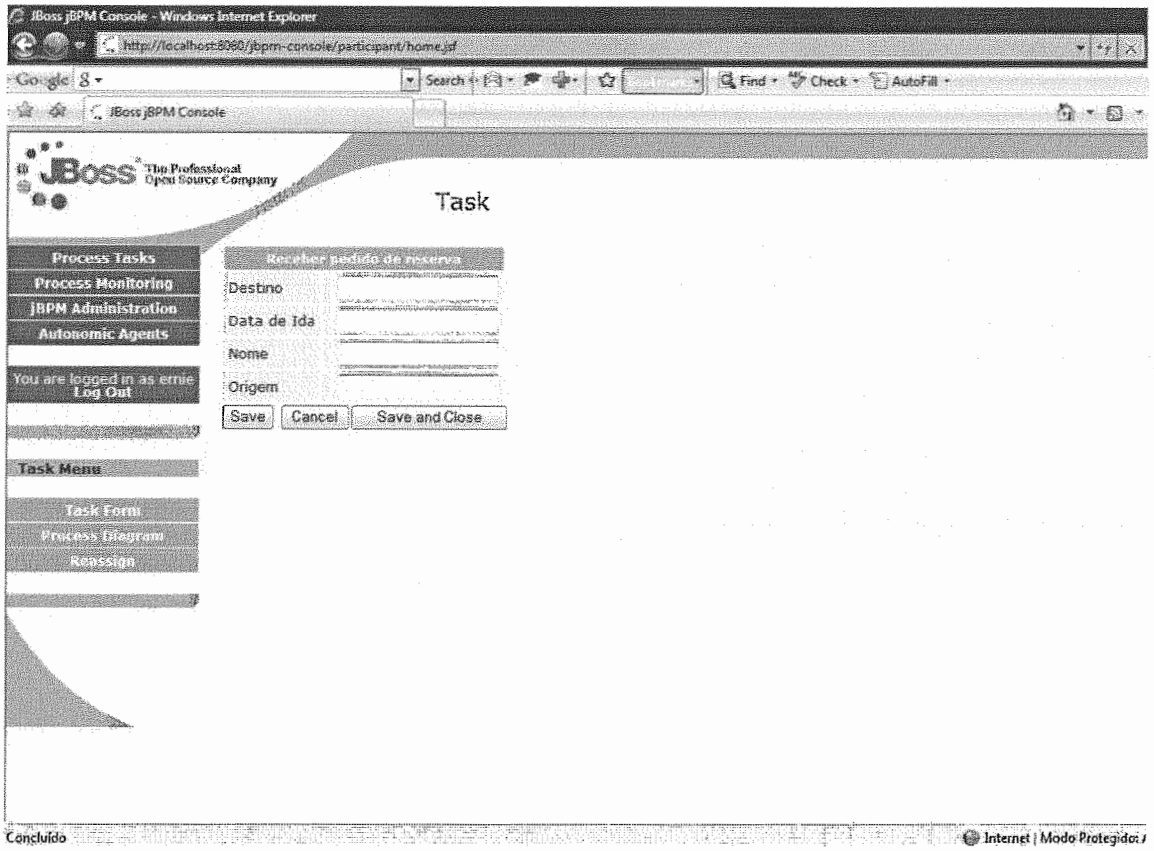


Figura 39 - Execução da atividade Receber pedido de reserva

- **Passo 1: Início da tarefa Escolher Vôo.**

O atendente conclui a atividade "Receber pedido de reserva" e a máquina de workflow disponibiliza a tarefa "Escolher vôo" (Figura 40). O Gerente do Processo instancia o agente gerente para tal tarefa. Esse agente, como pode ser visto na Figura 41, cria um Monitor para o Sistema de Reservas e um atuador referente à dimensão de auto-cura com uma regra para usar a Central Telefônica caso o Sistema de Reserva esteja indisponível.

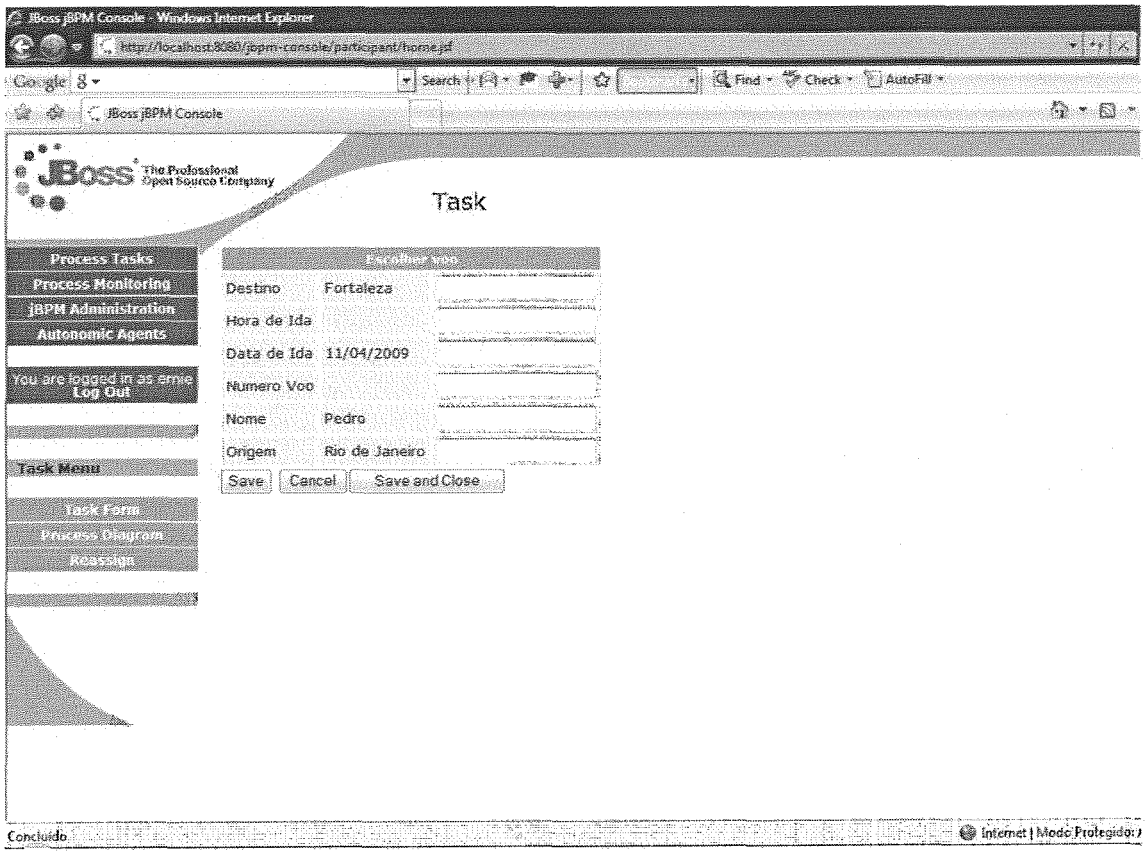


Figura 40 - Execução da atividade Escolher vôo

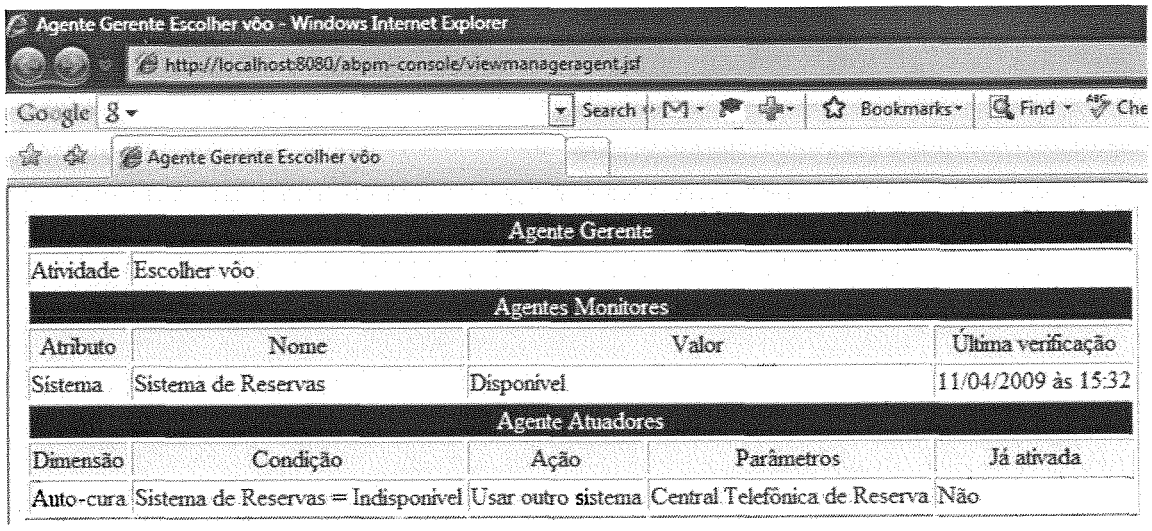


Figura 41 - Interface de acompanhamento do Agente Gerente da tarefa Escolher vôo

- **Passo 2: Sistema de Reserva se encontra indisponível.**

O gerenciador de infra-estrutura, sistema externo, no caso o Tivoli (IBM Tivoli, 2008), avisa via serviços *web* ao ABPM que o Sistema de Reserva está indisponível e o agente monitor lê essa informação e a escreve no quadro-negro. Assim que o agente

Atuador verifica o quadro negro, ele compara, via CLIPS, as informações com a condição de sua regra e ao receber a informação, novamente do CLIPS, que sua regra foi acionada, ele executa a ação informando ao atendente que deve ser usado a Central de Reservas, não deixando assim o atendente ocioso. A interface de acompanhamento pode ser vista na Figura 42.

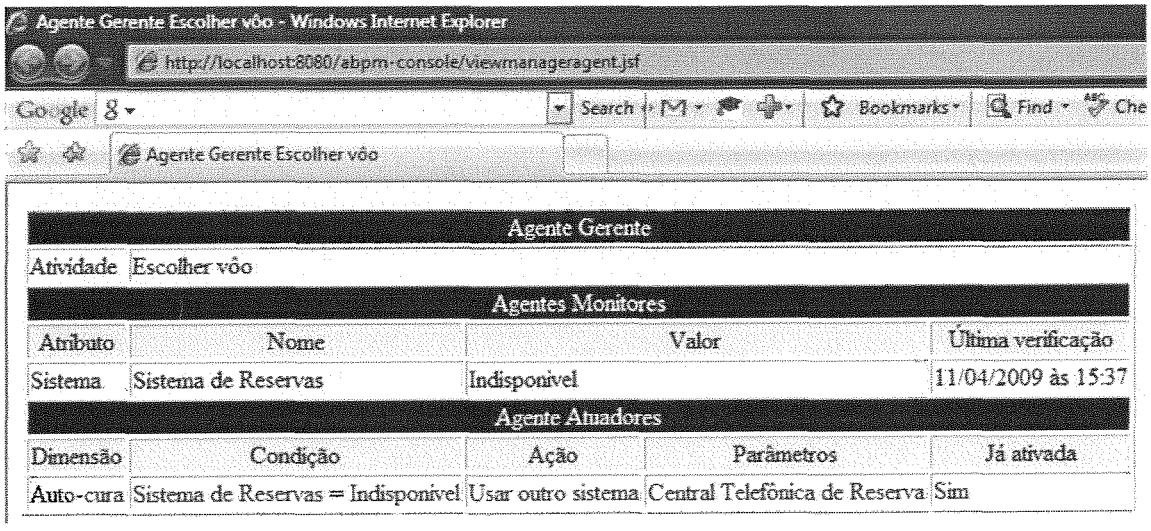


Figura 42 – Agente Gerente Monitor do Sistema de Reservas atualiza seu valor e aciona uma regra

- **Passo 3: Execução da tarefa Solicitar Reserva.**

Neste momento, é iniciada a tarefa "Solicitar Reserva" (Figura 43) juntamente com seu gerente. Como nessa atividade não existe regras autonômicas, o agente Gerente da Tarefa não inicia nenhum agente Monitor e nem Atuador. Para facilitar, o vôo não está cheio, logo a máquina de workflow leva o fluxo para a atividade "Fazer Cobrança".

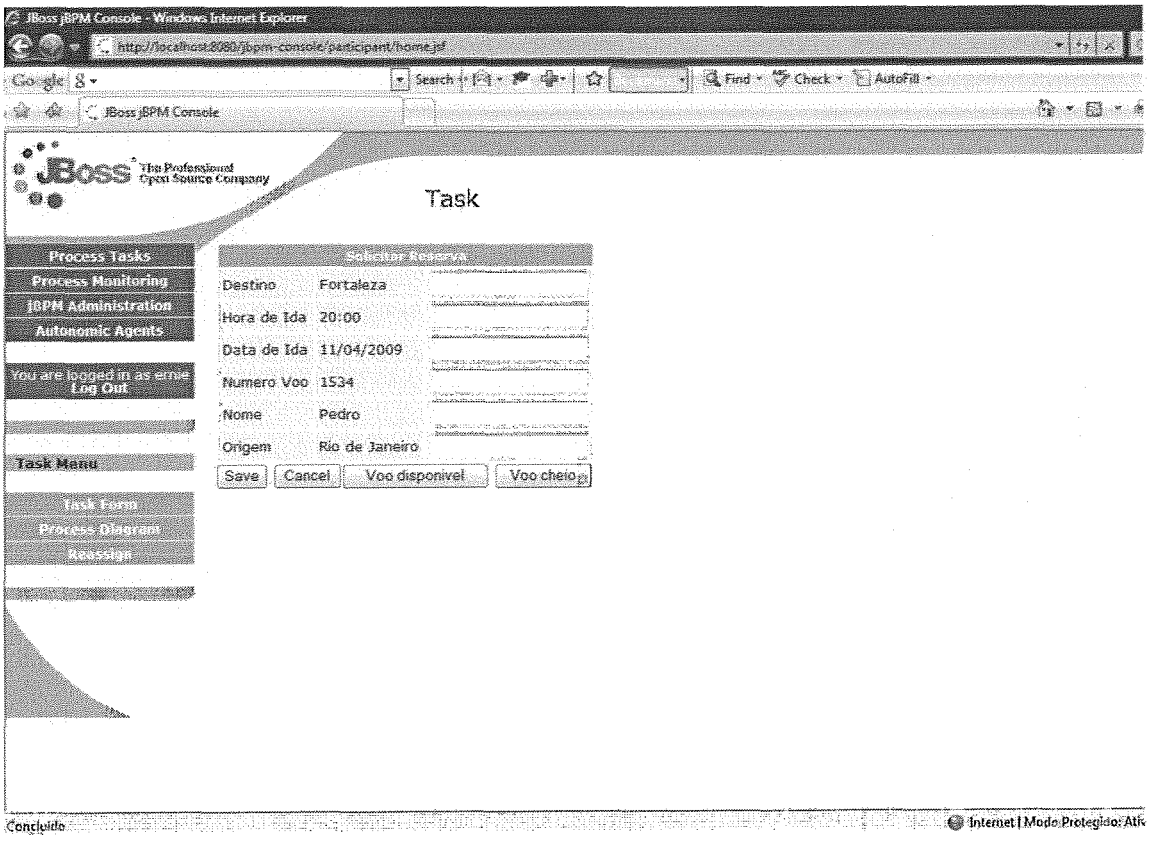


Figura 43 - Execução da atividade Solicitar reserva

- **Passo 4: Início da tarefa Fazer Cobrança.**

O Gerente do Processo instancia o Gerente da Tarefa (acompanhamento na Figura 45) para "Fazer Cobrança" e a execução da atividade deve ser feita na interface correspondente a Figura 44. O Monitor instanciado é responsável pelos conhecimentos que estão disponíveis e o Atuador contém uma regra substituir o executor caso a tarefa não tenha pelo menos um executor com conhecimento em cobrança e pertence à dimensão auto-cura.

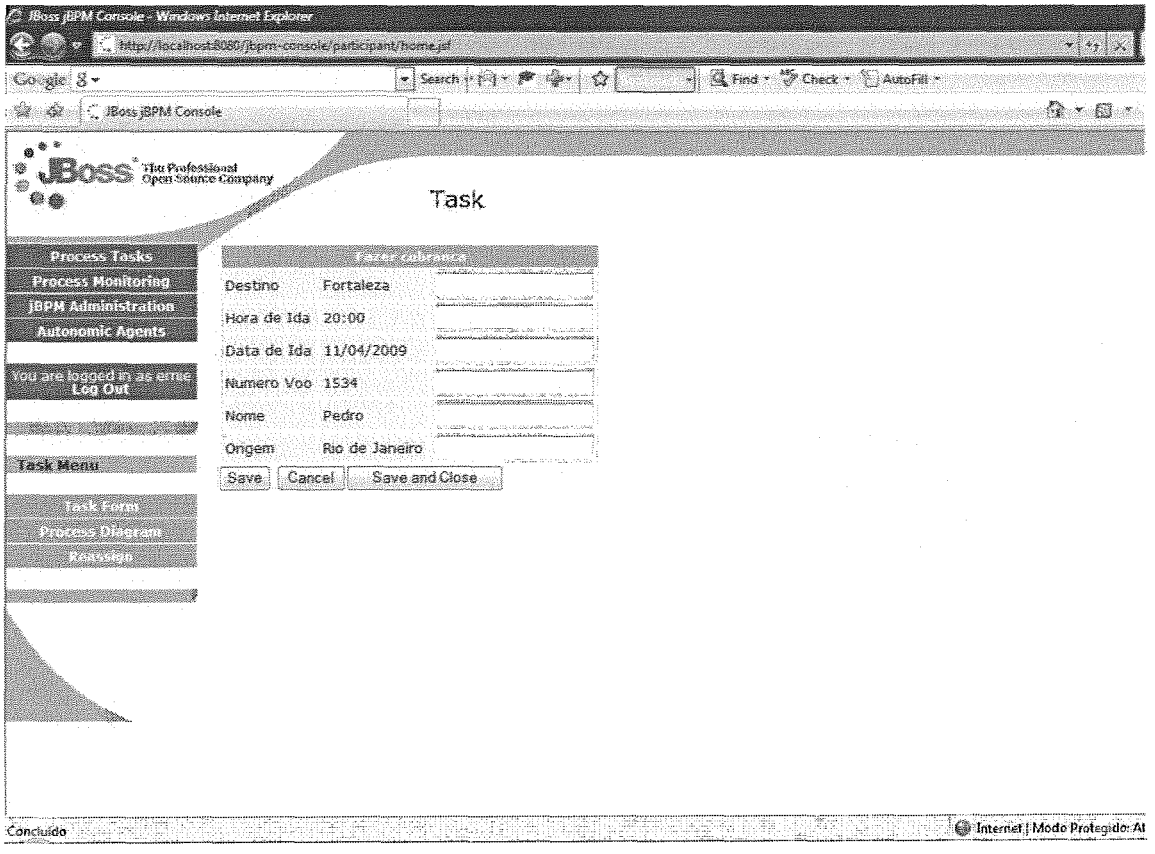
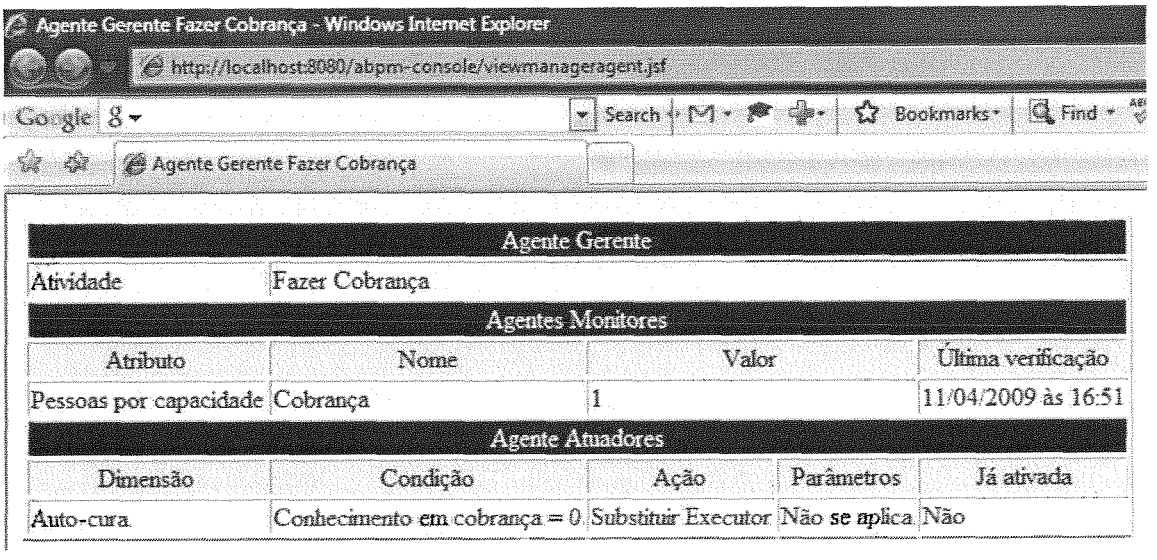


Figura 44 - Execução da atividade Fazer cobrança



Agente Gerente	
Atividade	Fazer Cobrança

Agentes Monitores			
Atributo	Nome	Valor	Última verificação
Pessoas por capacidade	Cobrança	1	11/04/2009 às 16:51

Agente Amadores				
Dimensão	Condição	Ação	Parâmetros	Já ativada
Auto-cura	Conhecimento em cobrança = 0	Substituir Executor	Não se aplica	Não

Figura 45 - Interface de acompanhamento do Agente Gerente da tarefa Fazer Cobrança

- **Passo 5: A tarefa Fazer Cobrança não possui conhecimento em cobrança.**

O agente Monitor dos conhecimentos calcula os conhecimentos disponíveis na tarefa e escreve no quadro negro. No momento existe somente conhecimento

atendimento ao cliente. O agente Atuador verifica o quadro negro e compara, via CLIPS, as informações com a condição de sua regra. Recebe do CLIPS a informação que sua regra foi acionada e, assim, requisita ao gerenciador de recursos humanos- GCC (Oliveira, Souza e Perazolo, 2006) – as pessoas que possuem tal conhecimento. Após retirar da listagem recebida aqueles já estão executando alguma tarefa, o Atuador escolhe um executor e substitui o atual. O novo executor faz a cobrança e termina a tarefa. O acompanhamento do agente gerente da tarefa pode ser visualizado na Figura 46.

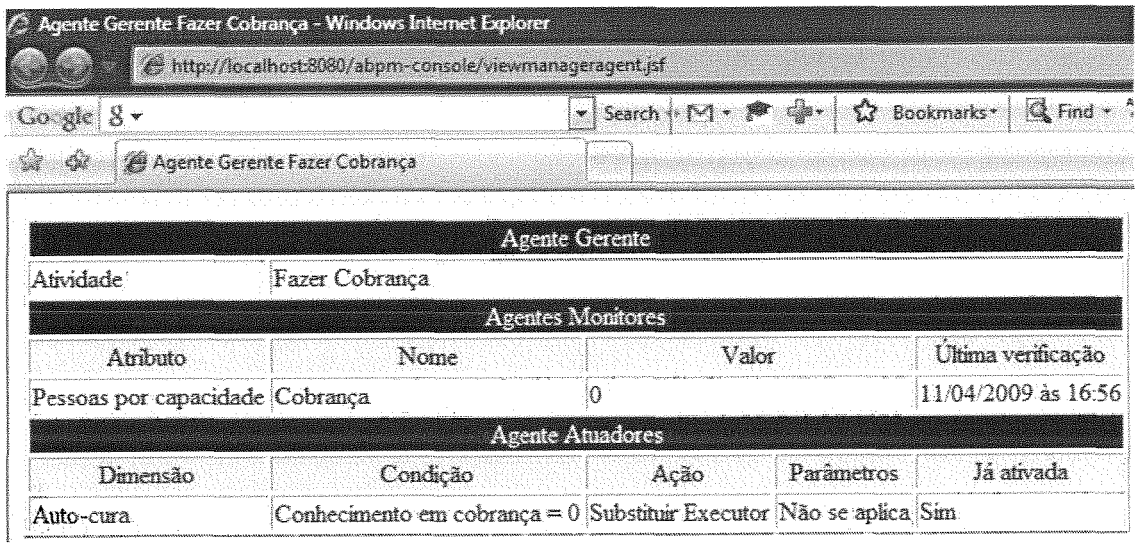


Figura 46 - Agente Gerente Monitor do Conhecimento em Cobrança atualiza seu valor e aciona uma regra

- **Passo 6: Execução da tarefa Alocar Passageiro.**

A tarefa "Alocar Passageiro" é iniciada (Figura 47) juntamente com seu agente gerente. Como nessa atividade não existe regras autonômicas, o agente Gerente da Tarefa não inicia nenhum agente Monitor e nem Atuador. A máquina de workflow guia o fluxo para a atividade "Confirmar Reserva".

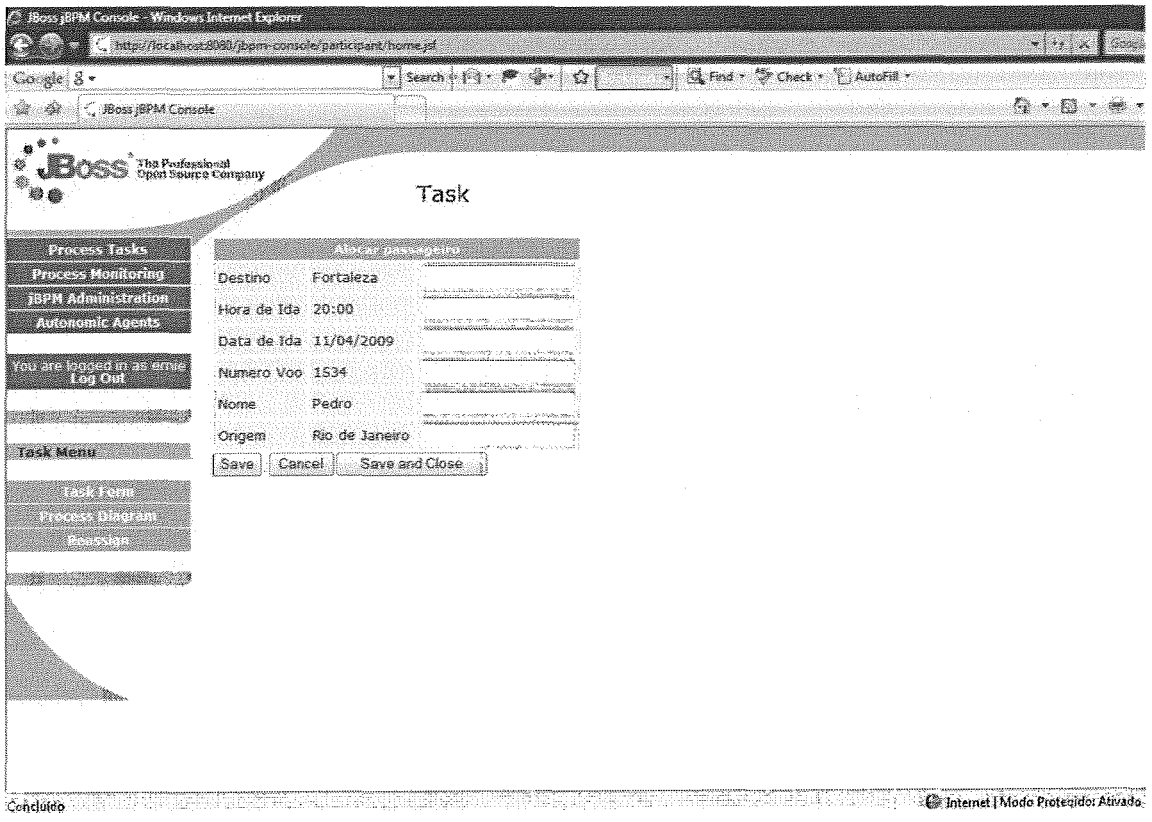


Figura 47 - Execução da atividade Alocar Passageiro

- **Passo 7: Início da tarefa Confirmar Reserva.**

O agente Gerente do Processo cria o gerente para "Confirmar Reserva", que pode ser acompanhado na interface correspondente a Figura 49, e esse último cria um Monitor para a duração e um Atuador de auto-cura que irá notificar um especialista caso a duração seja maior que 10 minutos. A atividade é executada na interface correspondente a Figura 48.

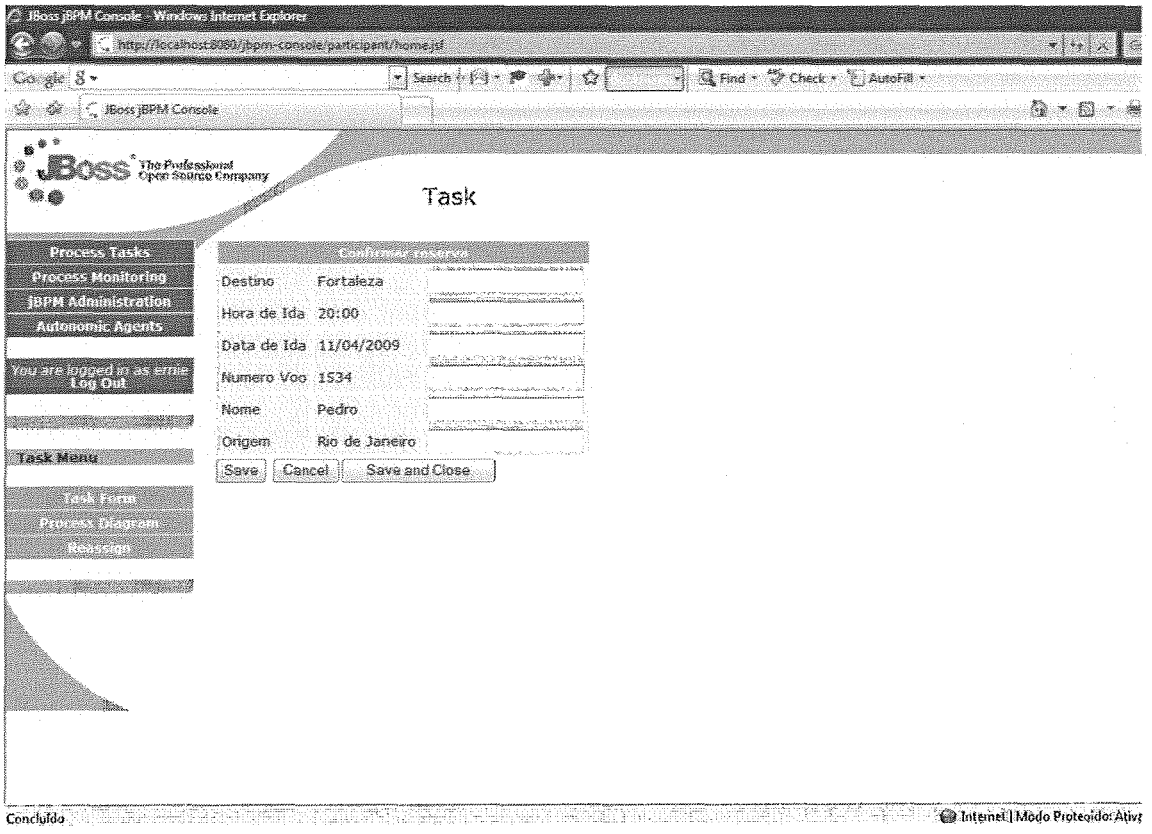


Figura 48 - Execução da atividade Confirmar reserva

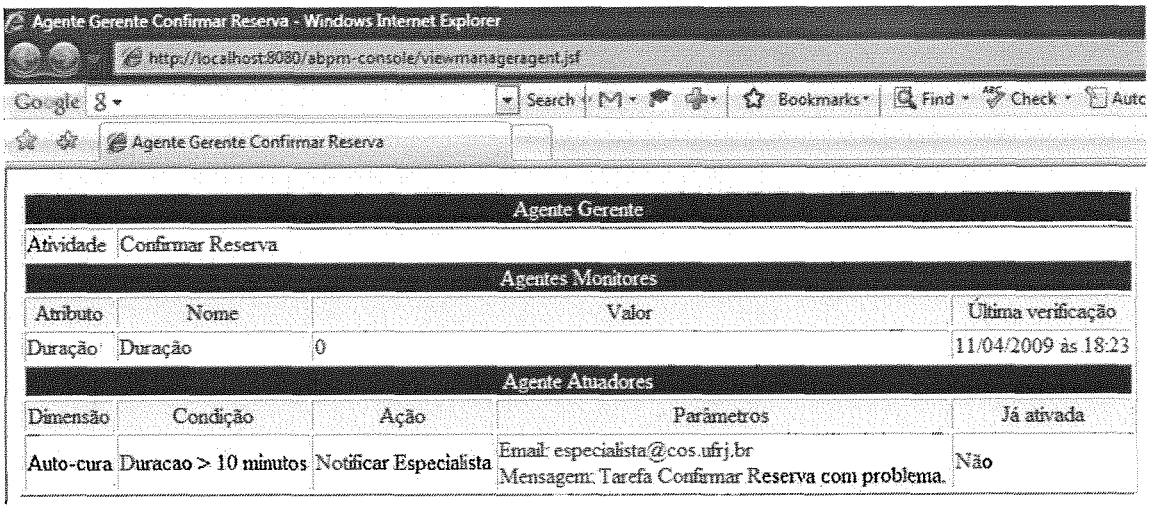


Figura 49 - Interface de acompanhamento do Agente Gerente da tarefa Confirmar Reserva

- **Passo 8: Duração da tarefa Confirmar Reserva está maior que 10 minutos.**

Após o Monitor escrever no quadro negro a duração um valor maior que 10 minutos, exemplo na Figura 50, o Atuador irá verificar, via CLIPS, a condição da regra e executará a ação para notificar um especialista via correio eletrônico. O especialista

avisado irá ajudar o executor da tarefa e assim ela é finalizada e o fluxo segue para "Fazer *Check-in*" (Figura 52).

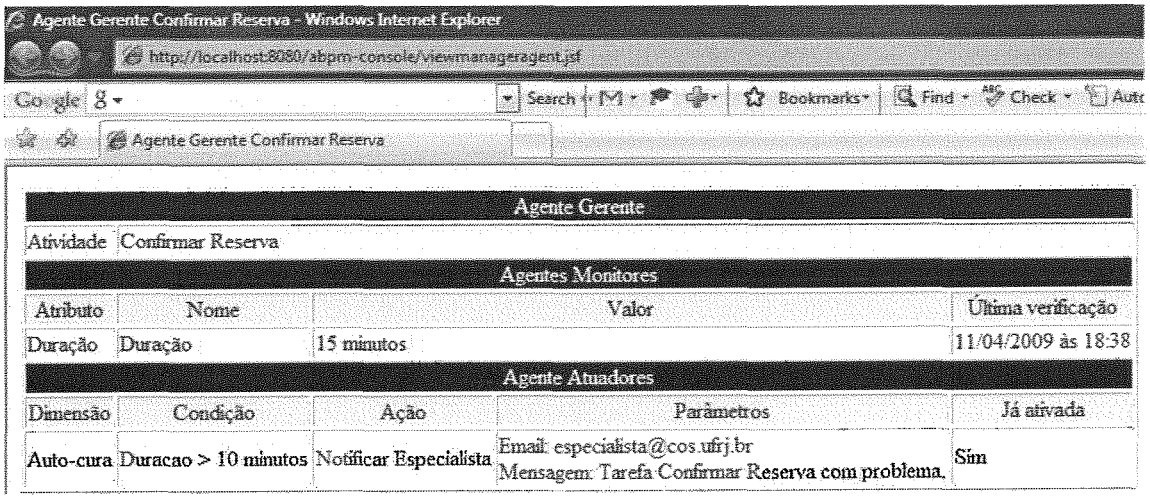


Figura 50 - Agente Gerente Monitor de Duração atualiza seu valor e aciona uma regra

- **Passo 9: Execução da tarefa Fazer *Check-in*.**

O agente Gerente da Tarefa (visto na Figura 51) "Fazer *Check-in*" cria um Monitor para a data atual e um Atuador (auto-configuração) que verifica se a data atual é maior que a data e horário do vôo. Essa regra não foi acionada e como o passageiro esqueceu seu passaporte, o workflow foi para tarefa "Receber Novo Contato".

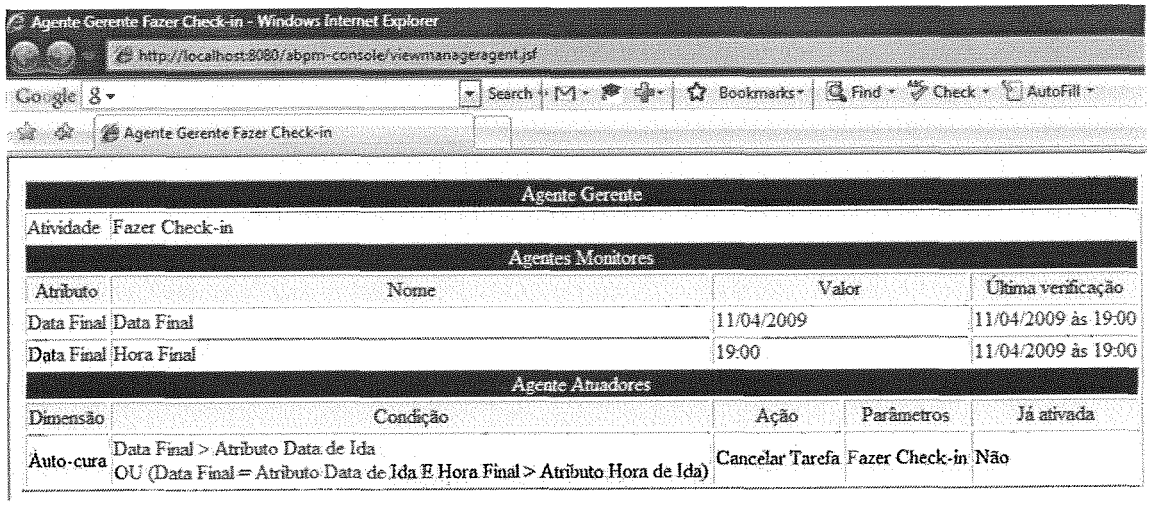


Figura 51 - Interface de acompanhamento do Agente Gerente da tarefa Fazer *Check-in*

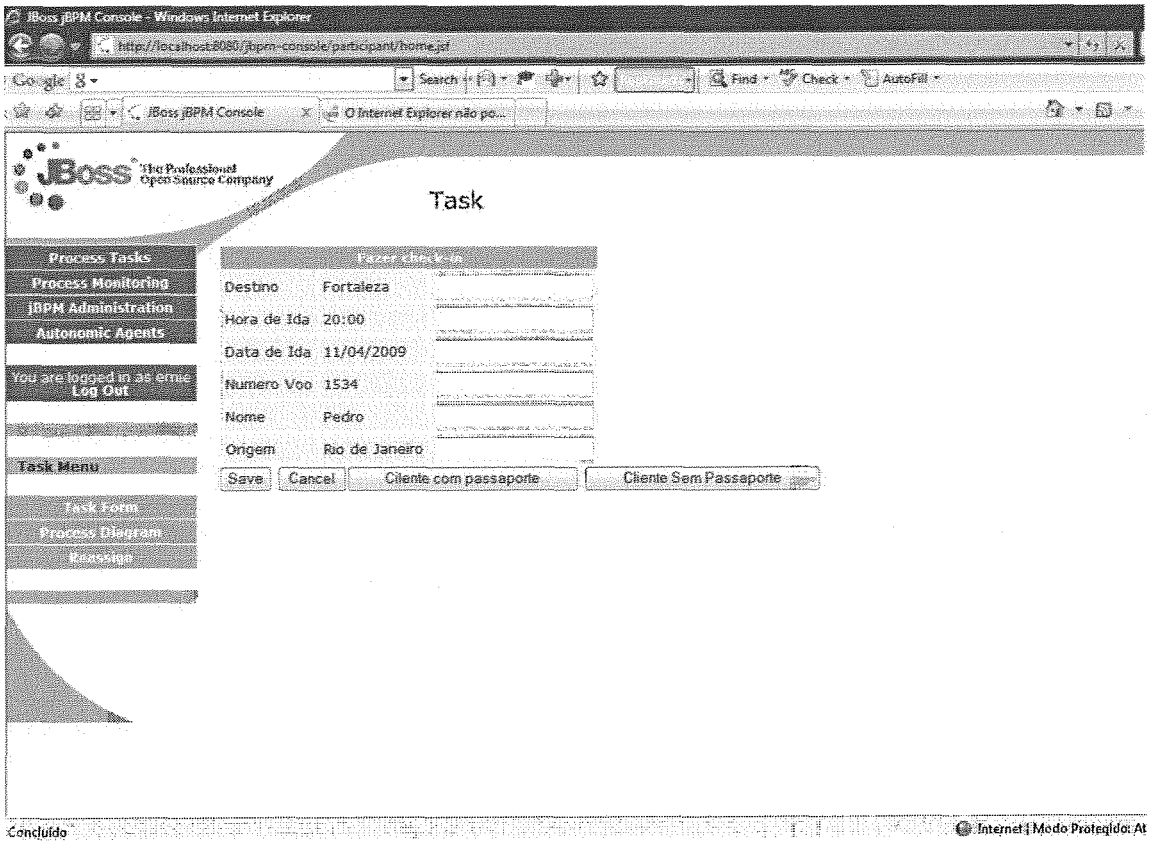


Figura 52 - Execução da atividade Fazer *check-in*

- **Passo 10: Execução da tarefa Receber Novo Contato.**

Os agentes Monitores e Atuadores têm são do mesmo tipo da tarefa anterior ("Fazer *Check-in*") já que a definição da regra é idêntica. Agora com o passaporte em mãos, o passageiro deverá fazer o *check-in* novamente.

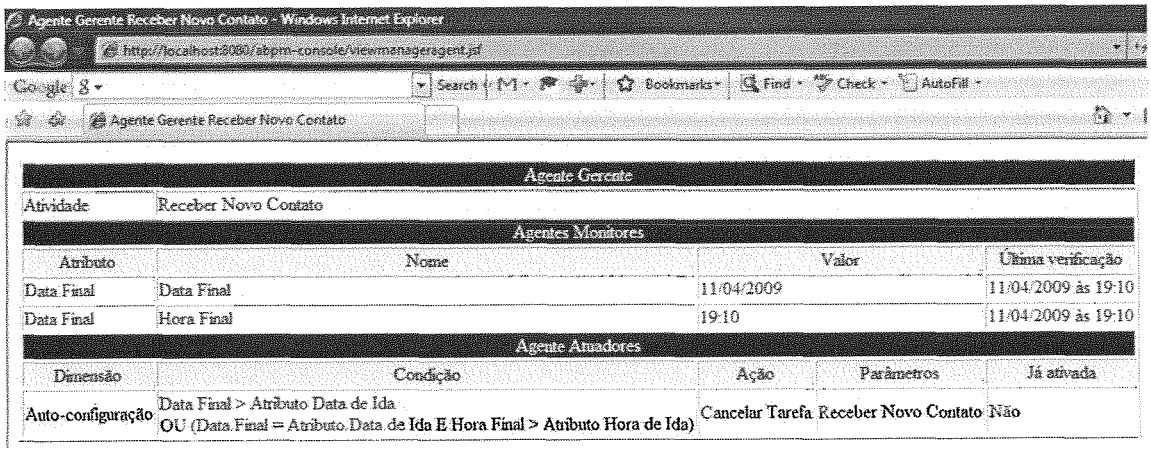


Figura 53 - Interface de acompanhamento do Agente Gerente da tarefa Receber Novo Contato

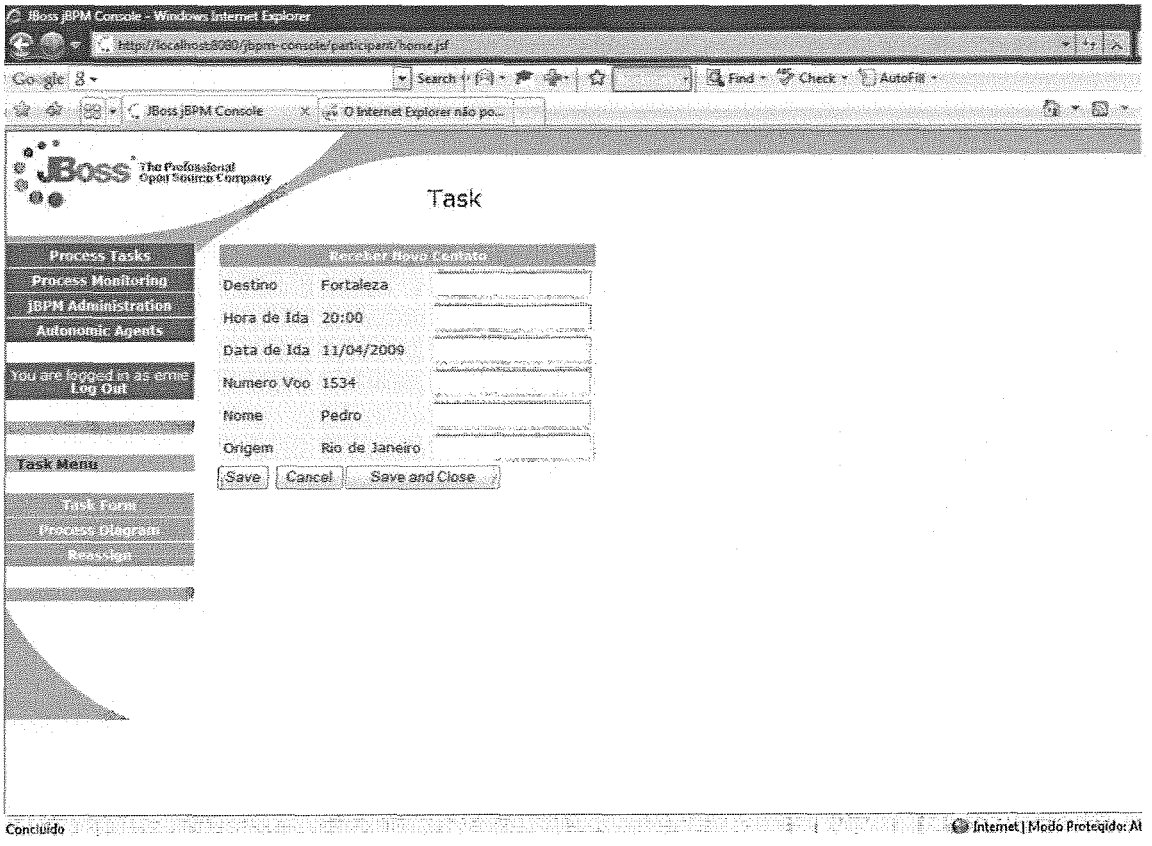


Figura 54 - Execução da atividade Receber novo contato

- **Passo 11: Re-execução da tarefa Fazer *Check-in*.**

Os agentes criados são os mesmo da primeira execução dessa tarefa. Como o passageiro está com o passaporte, a próxima tarefa a ser executada é "Emitir Cartão de Embarque".

- **Passo 12: Execução da tarefa Emitir Cartão do Embarque.**

O único agente criado é o Gerente da Tarefa, ele não cria nenhum Monitor e nem nenhum Atuador porque essa tarefa não tem regras definidas. O atendente emite o cartão de embarque, entrega ao passageiro e encerra a atividade. A interface para execução da atividade é vista na Figura 55.

Não há mais tarefas a serem executadas e o workflow é encerrado.

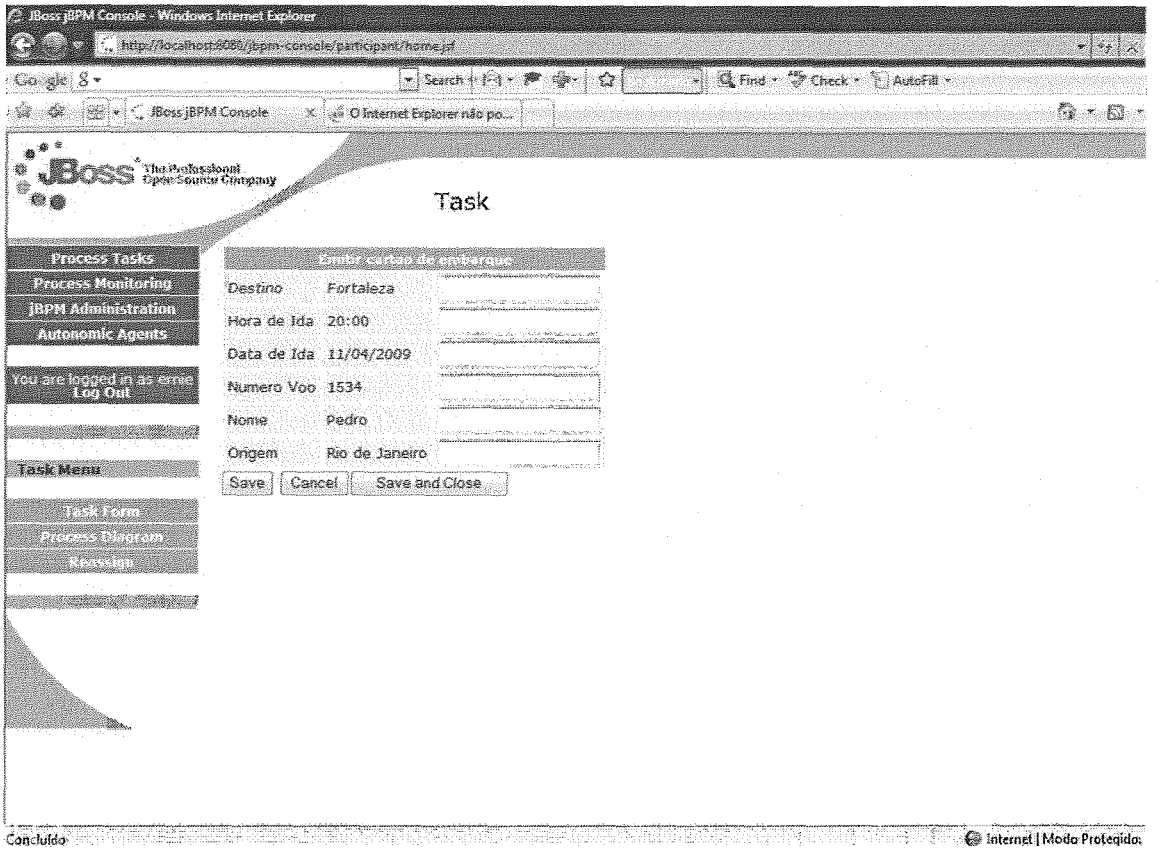


Figura 55 - Execução da atividade Emitir cartão de embarque

Capítulo 6 – Conclusão

Este capítulo apresenta as considerações finais, relembrando o problema, mostrando a solução proposta juntamente com as contribuições. Posteriormente, são listados alguns trabalhos futuros para uma possível continuação.

6.1 – Resultados Obtidos

Para tratar com o aumento da complexidade dos sistemas computacionais, surgiu um modelo denominado de Computação Autonômica. O termo autonômico se origina na medicina, onde o sistema autonômico humano é definido como aquela parte do corpo responsável pela respiração, digestão e defesa de germes e vírus, sem que o ser humano tenha consciência do que está fazendo, ou seja, sem executar ordens. Então, aplicando-se o termo autonômico à computação, tem-se um paradigma onde os sistemas se auto-gerenciam, ou seja, executam sem a constante intervenção humana. Esse gerenciamento é dividido em quatro dimensões: auto-configuração, auto-cura, auto-otimização e auto-proteção.

Nos dias atuais, a economia, assim como os sistemas computacionais, é baseada em conhecimento e, ainda, o dinamismo do processo é alto. Desse modo, é inevitável que exista um impacto no gerenciamento.

Considerando a necessidade de respostas cada vez mais instantâneas, os sistemas de workflow atuais apresentam um problema. Eles são feitos de maneira que suas tarefas estão encadeadas e, assim, não tratam exceções que possam ocorrer no desenvolvimento de uma atividade.

Foi abordado neste trabalho o problema do tratamento de exceções em processos de negócio, ou workflows. O objetivo principal é contribuir para a melhoria do

tratamento das exceções lançadas em um workflow através de sua reconfiguração, tratando os impactos negativos gerados pelas exceções que venham a ocorrer, e assegurando que os recursos do workflow sejam disponibilizados de forma a minimizá-las, ou seja, usando conceitos da Computação Autonômica.

Para suportar essa gestão autonômica, foi proposto o sistema ABPM. Esse sistema tem um modelador e uma máquina de workflow (AWE), e possui uma arquitetura multi-agentes, baseada em regras, provendo a gerência autonômica de processos, para assim diminuir a necessidade de intervenção humana. O modelador é um *plug-in* da plataforma Eclipse, baseado em diagramas de atividades UML. Adicionalmente, a arquitetura provê flexibilidade à solução, pois ela pode ser ampliada futuramente, para atender a níveis mais altos de abstração, mais próximo do nível estratégico, como no caso do *Balanced Scorecard* (Kaplan e Norton, 1996).

Com intuito de validar as ferramentas desenvolvidas, foi conduzido um estudo de caso, onde as ferramentas desenvolvidas foram utilizadas. Os participantes avaliaram a utilização das ferramentas e fizeram críticas para a melhoria do ABPM. Tal estudo mostrou que com o uso do ABPM houve um ganho de precisão e eficácia, e, ainda, uma diminuição do número de atividades modeladas. Até onde fomos capazes de investigar, este trabalho demonstrou a viabilidade da utilização dos conceitos de computação autonômica na gestão de processos de negócio e um sensível ganho com tal utilização.

Além do ABPM e AWE, outros artefatos foram gerados como contribuição deste trabalho: uma ontologia sobre os problemas que podem ocorrer do decorrer da execução de um processo de negócio, uma ontologia sobre ações a serem executadas para tratamento de problemas no workflow e a arquitetura de um sistema para gestão autonômica de processos (incluindo o diagrama de classes da aplicação, diagramas de classes e atividade dos agentes).

6.2 – Trabalhos futuros

Como uma primeira proposta de trabalho futuro, seria necessário um estudo de caso mais detalhado, em uma organização real, envolvendo um workflow bastante usado e em contextos diferentes de tal forma que gerem exceções diferentes, e, assim, se conseguir mais métricas dos ganhos obtidos com da utilização da ferramenta em um ambiente real.

Outro trabalho futuro direto seria a definição de um algoritmo para a redefinição de workflow. Para a implementação do algoritmo de otimização, pode-se usar como ponto inicial o trabalho de Pankratius e Stucky (2005), que formaliza mecanismos para verificação de similaridade e alteração de workflows, assim como a proposta de Muller (2002) que propõe uma adaptação dinâmica de workflow.

Já está em andamento a integração do AWE com o ABSC (*Autonomic Balanced Scoredcard*) (Rodrigues Nt., 2007), para que se possa alcançar níveis mais estratégicos de uma organização. Também pode ser estudada a análise de riscos para integração com a dimensão da auto-cura, com prevenção e mitigação de riscos, contribuindo assim para uma criação de empresa autonômica.

Um outro trabalho poderia utilizar arquivos de *log* como *feedback* aos gerentes, avaliando as ações tomadas, para que assim o sistema seja capaz de avaliar e propor novas ações do mesmo tipo que as avaliadas. Nesse caso, técnicas de aprendizado de máquina, como CBR (*Case Based Reasoning*) (Russel e Norvig, 1995) podem ser utilizadas.

Referencias bibliográficas

AALST, W.M.P. VAN DER, HOFSTEDE, A.H.M. TER, WESKE, M., 2003, "Business Process Management: A Survey", **In: Proceedings of the 1st International Conference on Business Process Management**, volume 2678 , pp. 1–12 , Eindhoven, Holanda, Junho.

AIBER, S., GILAT, D., LANDAU, A., RAZINKOV, N., SELA, A., WASSERKRUG, S., 2004, "Autonomic Self-Optimization According to Business Objectives", **In: Proceedings of the International Conference on Autonomic Computing**, pp. 206- 213, Nova Iorque, USA, Maio.

ANDRZEJAK, A., HERMAN, U., SAHAI, A, 2005, "FEEDBACKFLOW - An Adaptive Workflow .Generator for System Management". **In: Proceedings of the 2nd International Conference on Autonomic Computing**, pp. 335-336 , Seattle, USA, Junho.

BABAOGLU, O., JELASITY, M. , MONTRESOR, A. , FETZER, C. , LEONARDI, S. , MOORSEL, A. VAN, STEEN, M. VAN, 2005, **Self-Star Properties in Complex Information Systems**, 1st edition, Nova Iorque, Springer.

BUDINSKY, F., STEINBERG, D., MERKS, E., ELLERSICK, R., GROSE, T. J., 2003, **Eclipse Modeling Framework**, 1st edition, USA, Ed. Pearson.

BUHLER, A. P., VIDAL, J. M., VERHAGEN, H., 2003, "Adaptive Workflow = Web Services + Agents", **In: Proceedings of the International Conference on Web Services**, vol. 2853, pp. 131-137, Las Vegas, Nevada, USA, Junho.

CASATI, F., CERI, S., PARABOSCHI, S., POZZI, G., 1999, "Specification and implementation of exceptions in workflow management systems", **In: ACM Transactions on Database Systems**, Volume 24, Número. 3, pp. 405-451.

GIARRATAN, J. C., 2007, **Clips User guide**, Disponível em http://clipsrules.sourceforge.net/documentation/v630/ug.htm, acessado em 25 março de 2009.

CONSENTINO, S., CHUTE, D., LIBON, D., MOORE, P., GROSSMAN, M., 2006, "How Does the Brain Support Script Comprehension? A Study of Executive Processes and Semantic Knowledge in Dementia", **In: Neuropsychology**, Volume 20, Número 20, pp. 307-318.

CORKILL, D., 1991, "Blackboard Systems". **In: AI Expert**, Volume 6, Número, 9, pp. 40-47.

CORKILL, D., 2003, "Collaborating Software", **In: Proceedings of International Lisp Conference**, Nova Iorque, USA, Outubro.

DAVENPORT, T. H., 1994, **Reengenharia de processos: Como inovar na empresa através da tecnologia da informação**. 4ª Edição, Rio de Janeiro, Editora Campus.

FLEURY, M., REVERBEL, F., 2003, "The JBoss Extensible Server". **In: Middleware 2003**, Volume 2672, pp 344—373.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J.,1995, **Design Patterns: Elements of Reusable Object-Oriented Software**. 18ª edição, USA, Addison-Wesley.

HAN, M., THIERY, T., SONG, X., 2006, "Managing exceptions in the medical workflow systems". **In: Proceeding of the 28th International Conference on Software Engineering,** pp. 741 – 750, Shanghai, China, Maio.

HEINIS, T., PAUTASSO, C., ALONSO, G., 2005, "Design and Evaluation of an Autonomic Workflow Engine", **In: Proceedings of the 2nd International Conference on Autonomic Computing**, pp. 27-38 , Seattle, USA, Junho.

HOLLINGSWORTH, D., 1995, **The Workflow Reference Model**. In: Workflow Management Coalition, Número do documento: TC00-1003, Inglaterra.

HORN, P., 2001, **Autonomic Computing**, In: IBM Manifesto, EUA.

IBM Tivoli, 2008, **User guide**. IBM Corporation, Disponível em http://www.ibm.com/tivoli, acessado em 22 agosto de 2008.

JBPM, 2006, **User guide**. JBoss Corporation, Disponível em http://docs.jboss.com/jbpm/v3/userguide/, acessado em 13 junho de 2008.

KAPLAN, R., NORTON, D., 1996, **The Balanced Scorecard: Translating Strategy into Action**, 1a Edição, USA, Harvard Business School Press.

KEPHART, J. O. E CHESS, D. M., 2003, "The Vision of Autonomic Computing", **In: IEEE Computer Society**, volume 36, número. 1, pp. 41-50.

KHARGHARIA, B., HARIRI, S., YOUSIF, M.. 2007, "Autonomic power and performance management for computing systems", **Cluster Computing**, Volume 11, Number 2 , pp. 167-181.

LEE, K., SAKELLARIOU, R., PATON, N. W. E FERNANDES, A. A. A., 2007, "Workflow Adaptation as an Autonomic Computing Problem", **In: Proceedings of the 2nd workshop on Workflows in support of large-scale science,** pp. 29-34, Califórnia, USA, Junho.

MANGAN, P., SADIQ, S., 2002,"On building workflow models for flexible processes", **In: Proceedings of the 13th Australasian Database Conference**. vol. 24., pp. 103-109, Darlinghurst, Austrália,.Fevereiro.

MCCANN, J., HUEBSCHER, M., 2008, "A survey of Autonomic Computing - degrees, models and applications", **In: ACM Computing Surveys**, Volume 40, número. 7, Nova Iorque, USA.

MELHORAMENTOS, 2004, **Michaelis Moderno Dicionário da Língua Portuguesa**. 1ª Edição, São Paulo, Editora Melhoramentos.

MONTEIRO JR., P. C. L., RODRIGUES NT., J.A. ; SAMPAIO, J. O. ; SOUZA, J. M., 2008, "ABPM: Autonomic Business Process Manager", **In: Proceedings of the 3th Latin American Autonomic Computing Symposium**, p. 61-70, Gramado, Brasil, Outubro.

MOURÃO, H., ANTUNES, P., 2007, "Supporting effective unexpected exceptions handling in workflow management systems". **In: Proceedings of the 2007 ACM Symposium on Applied Computing**, pp. 1242 - 1249 , Seoul, Korea,.Março.

MULLER, R., 2002, **Event-Oriented Dynamic Adaptation of Workflows: Model, Architecture, and Implementation**. Tese de D. Sc., Universidade de Leipzig, Alemanha.

MURCH, R., 2004, **Autonomic Computing**. 1ª edição, USA, IBM Press.

NAMI, M. R., BERTELS, K., 2007, "A Survey of Autonomic Computing Systems", **In: Proceedings of the 3rd International Conference on Autonomic and Autonomous Systems**, Volume 228/2007 , pp. 101-110 , Washington, DC, USA., Junho.

OLIVEIRA, J. ; MONTEIRO JR, P. C. L.; RODRIGUES NT, J. A.; SOUZA, J. M.; PERAZOLO, M., 2007, "The Autonomic Balanced Scorecard", **In: Proceedings of the 2nd Latin American Autonomic Computing Symposium**, Petropolis, Brasil, Setembro.

OLIVEIRA, J.; SOUZA, J. M.; PERAZOLO, M., 2006, "Methexis AC Problem Detection and Solution through Knowledge Management Principles". **In: Latin American Autonomic Computing Symposium**, Campo Grande. Brasil.

OMG, 2009, **Business Process Modeling Notation (BPMN)**, versão 1.2

PAIM, R., 2002, **Engenharia de Processos Análise do Referencial Teórico-Conceitual**. Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

PANKRATIUS, V., STUCKY, W., 2005, "A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets", **In: Proceedings. 2nd Asia-Pacific Conference on Conceptual Modelling**, Volume 43, pp. 79 – 88, Newcastle, Australia, Janeiro.

PARASHAR, M., HARIRI, S., 2006, **Autonomic Computing: Concepts, Infrastructure, and Applications**. 1ª Edição, USA, CRC Press.

PINHEIRO, W. A. ; VIVACQUA, A. ; BARROS, R. ; MATTOS, A.S. ; CIANNI, N.M. ; MONTEIRO JR., P.C.L. ; MARTINO, R. ; MARQUES, V. ; XEXEO, G. B. ; SOUZA, J. M., 2007, "Dynamic Workflow Management for P2P Environments using Agents", **In: Computational Science – ICCS 2007**, Volume 4489/2007, **Lecture Notes in Computer Science**, Springer Berlin / Heidelberg, pp. 253-256.

PINHEIRO, W. A. ; SILVA, M. C. O. ; BARROS, R. ; XEXÉO, G. ; SOUZA, J. M.. 2009, "Autonomic Collaborative RSS: an Implementation of Autonomic Data using Data Killing Patterns". **In: The 13th International Conference on Computer Supported Cooperative Work in Design**, Santiago, Chile.

RODRIGUES NT., J.A. ; MONTEIRO JR. ; SAMPAIO, J. O. ; SOUZA, J. M. ; SILVA, G. Z., 2007, "Autonomic Business Processes Scalable Architecture". **In: Business Process Management Workshops**, volume 4928, **Lecture Notes in Computer Science**, Springer Berlin / Heidelberg , pp. 78-83.

RODRIGUES NT., J.A. ; MONTEIRO JR. ; SAMPAIO, J. O. ; SOUZA, J. M. ; SILVA, G. Z., 2007, "Towards an Autonomic Enterprise: From Autonomic Business Processes to Autonomic Balanced Scorecard", **Brazilian Workshop on Business Process Management**, , Gramado. Brasil.

RUSSELL, N., AALST, W.M.P. VAN DER E HOFSTEDE, ARTHUR TER, 2006, "Workflow Exception Patterns", **In: Advanced Information Systems Engineering**, Volume 4001/2006, **Lecture Notes in Computer Science**, Springer Berlin / Heidelberg, pp 288–302.

RUSSELL, S., NORVIG, P., 1995, **Artificial Intelligence A Modern Approach**. 1ª Edição, Nova Jersey: Prentice Hall.

SANTOS, L. O., 2008,- **Identificação de Problemas em Workflows Autonômicos**. XXX Jornada Giulio Massarani de Iniciação Científica, Artística e Cultural - JIC 2008, UFRJ, Rio de Janeiro.

SAVARIMUTHU, B. T., PURVIS, M., FLEURKE, M., 2004, "Monitoring and controlling of a multi-agent based workflow system", **In: Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web intelligence, and Software Internationalization**, vol. 54., pp. 127-132, Darlinghurst, Austrália, Janeiro.

SERUGENDO, G. DI M., GLEIZES, M., KARAGEORGOS, A., 2006, "Self-Organisation and Emergence in MAS: An Overview", **Journal of Informatica**, vol. 30, n. 2, pp. 45-54.

SHAW, M., GARLAN, D., 1996, **Software Architecture – Perspectives on an Emerging Discipline**. 1ª Edição, Nova Jersey, Prentice Hall.

STEELS, L., 1990, "Cooperation between Distributed Agents through Self-Organization", **In: Proceedings Intelligent Robots and Systems**, pp. 8-14,Amsterdam, Junho.

STROHMAIER, M., YU, E., 2006, "Towards Autonomic Workflow Management Systems", **Journal of Information Technology and Management**, Volume 6, Number 1, pp. 61-87.

VERMA, K. E SHETH, A. P, 2005, "Autonomic Web Processes", **In: Service-Oriented Computing - ICSOC 2005**, Volume 3826/2005, **Lecture Notes in Computer Science**, Springer Berlin / Heidelberg, pp. 1-11.

VIVAQUA, A. ; PINHEIRO, W. A. ; BARROS, R. ; MATTOS, A.S. ; CIANNI, N.M. ; MONTEIRO JR., P.C.L. ; MARTINHO, R. ; MARQUES, V. ; XEXEO, G. B. ; SOUZA, J. M. ; SSCHNEIDER, D., 2007, "Dynaflow: Agent-Based Dynamic Workflow Management for P2P Environments",. **In: Computational Science – ICCS 2007,** Volume 4489/2007, **Lecture Notes in Computer Science**, Springer Berlin / Heidelberg, pp. 253-256.

WOOLDRIDGE, M. J., 2002, **Introduction to Multiagent Systems**. 1ª Edição, Nova Iorque, Ed. John Wiley & Sons Inc.

Anexo A – Processo modelado no estudo de caso

Este anexo apresenta o texto passado aos alunos que participaram do estudo de caso para que pudessem modelar o processo.

Processo de Reserva da Atlântica Linhas Aéreas

Ao viajar de avião, é costume o passageiro efetuar reserva em um vôo, antes da sua chegada ao aeroporto. Considere uma Cia. Aérea, denominada "Atlântica Linhas Aéreas" e um passageiro viajando de uma cidade para outra, sempre com vôos internacionais.

O processo se inicia com o assistente de viagens recebendo um pedido de reserva do cliente. Deve ser pedida a cidade de origem, destino, data de ida e data de volta, se aplicável. Isso feito, deve-se procurar no sistema os vôos disponíveis. Caso o sistema de reservas não esteja no ar, pode ser utilizado o telefone da Central de Reservas. As opções de vôo juntamente com os preços são informadas ao cliente. Caso elas não o agradem, uma nova consulta com uma nova data pode ser feita. Como tudo é feito instantaneamente, não existe um prazo determinado para término da tarefa.

Caso algum vôo o agrade, ele solicita a reserva. O assistente de viagens então faz a cobrança e aloca o passageiro em um assento. Será dada preferência ao assistente de viagens que atendeu o cliente na atividade de solicitação de reserva, porém caso ele não esteja disponível, ou seja, a tarefa está sem o recurso humano necessário (dimensão conhecimento) para cobrança, outro assistente é alocado. A reserva deve ser feita o mais rápido possível, para que a vaga não fique indisponível no sistema. Logo, se tiver a duração maior que 10 minutos, um especialista é avisado da expiração do tempo para conclusão da reserva.

Pode acontecer de o vôo escolhido pelo cliente estar cheio. Quando isso ocorrer, o cliente deve entrar em uma lista de espera, e ficar aguardando. Assim que uma vaga no vôo for disponibilizada, e ele for o primeiro da fila, a reserva dele é feita e o mesmo é avisado.

Finalmente, chega o dia do vôo, e o cliente vai ao aeroporto fazer o *check-in*. Um atendente de *check-in* o recebe e pede o seu passaporte. Caso esteja sem passaporte, o processo deverá esperar um novo contato do cliente com o passaporte, sendo cancelado após ser atingido o horário do vôo. No caso de cancelamento, a atividade gerará um aviso ao nível mais alto, de processo, e o processo irá verificar que a atividade de fazer *check-in* está cancelada e irá cancelar o processo. Após o *check-in*, o cartão de embarque é emitido.