UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ADMINISTRAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO

Flávio de Oliveira Alves

SIMULAÇÃO MULTI-AGENTE EM GESTÃO DE PROJETOS DE *SOFTWARE* EM AMBIENTES DE PROGRAMAÇÃO EXTREMA

Porto Alegre
2009

Flávio de Oliveira Alves

SIMULAÇÃO MULTI-AGENTE EM GESTÃO DE PROJETOS DE *SOFTWARE* EM AMBIENTES DE PROGRAMAÇÃO EXTREMA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Administração da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do título de Mestre em Administração.

Orientador: Prof. Dr. Eduardo Ribas Santos

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

A474s Alves, Flávio de Oliveira
Simulação multi-agente em gestão de projetos de *software* em ambientes de programação extrema / Flávio de Oliveira Alves. - 2009.
127 f. : il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul, Escola de Administração, Programa de Pós-Graduação em Administração, 2009.
"Orientador: Prof. Dr. Eduardo Ribas Santos"

1. Sistemas multi-agente. 2. Desenvolvimento de *software*. 3. Gestão de projetos. 4. Gestão de pessoal. I. Título

CDU 681.3

Ficha elaborada pela Biblioteca da Escola de Administração – UFRGS.

Flávio de Oliveira Alves

SIMULAÇÃO MULTI-AGENTE EM GESTÃO DE PROJETOS DE *SOFTWARE* EM AMBIENTES DE PROGRAMAÇÃO EXTREMA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Administração da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do título de Mestre em Administração.

Orientador: Prof. Dr. Eduardo Ribas Santos

Conceito Final:
Aprovado em ____ de março de 2009.

BANCA EXAMINADORA

_____
Prof. Dr. Denis Borenstein - Universidade Federal do Rio Grande do Sul

_____
Prof. Dr. Flávio Sanson Fogliatto - Universidade Federal do Rio Grande do Sul

_____
Prof. Dr. João Luiz Becker - Universidade Federal do Rio Grande do Sul

_____
Orientador: Prof. Dr. Eduardo Ribas Santos - Universidade Federal do Rio Grande do Sul

Para minha amada esposa
Fabiane Epping
e para nosso amado filho
Arthur Epping Alves.

AGRADECIMENTOS

Agradeço à minha amada e sempre presente esposa, Fabiane Epping, pelo amor, pela paciência e pela compreensão sobre a vida que vivo: sempre entre livros e computadores. Também agradeço ao nosso amado filho, Arthur Epping Alves, pela infinita alegria que a presença dele me proporciona. Aproveito para pedir desculpas para eles, Fabiane e Arthur, pelos meus momentos de ausência ao longo deste Mestrado - tenha ela sido física ou mental. Minha esposa sabe - e meu filho aprenderá - que tudo que vale a pena, na vida, tem um preço.

Agradeço, de coração, aos meus familiares, amigos e amigas que sempre acreditaram em minha capacidade de cumprir mais esta missão. Aproveito para agradecer aos meus avós maternos, Adail de Oliveira e Clotilde Sena de Oliveira, que me criaram como um filho e que sempre me ofereceram todas as oportunidades para que eu crescesse intelectualmente.

Ao meu orientador, Prof. Dr. Eduardo Ribas Santos, agradeço pelas conversas esclarecedoras, pela paciência e, principalmente, pela incomparável liberdade oferecida para mim durante a construção deste trabalho.

Ao Prof. Dr. João Luiz Becker pela admirável atenção aos detalhes - tanto falados quanto escritos por mim - e pelas consequentes sugestões que contribuíram sensivelmente para a evolução da minha proposta de pesquisa.

Agradeço ao Prof. Dr. Flávio Sanson Fogliatto pela disposição em participar da Banca de Avaliação desta dissertação.

Para os funcionários e professores que representam a Escola de Administração da Universidade Federal do Rio Grande do Sul, agradeço pelo bom trabalho.

Agradeço à minha prezada colega e amiga Luisa Mariele Strauss pelos bate-papos inspiradores, por sua paciência em acompanhar os relatos dramáticos da minha jornada e por um valioso conselho que passei a levar sempre comigo.

Agradeço aos meus colegas e às minhas colegas do Programa de Pós-Graduação em Administração (PPGA) que, de uma forma ou de outra, tornaram minha passagem pelo Mestrado ainda mais tranquila.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (2009), que me concedeu uma Bolsa de Mestrado, agradeço pelo apoio recebido.

Gostaria de agradecer pelos momentos agradáveis proporcionados pelo senso de humor *nonsense* daqueles que igualam a economia com a Gravitação. Sir Isaac Newton deve estar revirando-se até agora, em seu túmulo.

Finalmente, aos *palpiteiros* que tenho encontrado - desde os meus tempos na Iniciação Científica - e aos que confundem boa postura com postura submissa, ofereço meus agradecimentos pelo seu desperdício de energia.

Os arrogantes são como os balões:
basta uma picadela de sátira ou de dor para dar cabo deles.
Anne Louise Germaine de Staël

O sucesso é um péssimo professor.
Ele seduz as pessoas inteligentes e as faz pensarem que não podem perder.
William Henry Gates III

RESUMO

Nesta dissertação, o autor aborda a dificuldade de prever-se o desempenho dos recursos humanos em um processo de desenvolvimento de *software* em um ambiente de Programação Extrema (XP) (BECK, 2000) e propõe uma solução com potencial para minimizar esse problema. Especificamente, o problema, a ser tratado neste trabalho, consiste em melhorar as previsões dos gerentes de projeto – no âmbito do ambiente mencionado - com relação ao desempenho dos recursos humanos na geração de valor para o negócio. Tal valor para o negócio é alcançado através da implementação, por parte dos programadores, das diversas funcionalidades de um sistema de *software*. Para a construção da solução proposta neste trabalho, o autor analisou um sistema XP de desenvolvimento de *software* (composto por ambiente, pessoas e processo), conforme o processo de modelagem proposto por Streit (2006) e apoiado na revisão da literatura relevante. Em seguida, o autor estruturou esse sistema em um modelo conceitual para, finalmente, desenvolver um modelo computacional do sistema analisado, baseado em múltiplos agentes inteligentes modelados conforme a arquitetura *Beliefs-Desires-Intentions* (BDI), ou Crenças-Desejos-Intenções. O modelo computacional da simulação multi-agente foi desenvolvido com o apoio da ferramenta SeSAm (KLÜGL, 2006). Testado através da experimentação estatística 2^k Fatorial (LAW e KELTON, 2000), o modelo de simulação multi-agente de processos de desenvolvimento de *software*, para ambientes de Programação Extrema, demonstrou eficácia e aplicabilidade prática sobre o problema em questão.

Palavras-chave: sistemas multi-agente, desenvolvimento de *software*, programação extrema, simulação.

ABSTRACT

In this research, the author adresses the difficulty to forecast the performance of the human resources in a *software* development process in an Extreme Programming (XP) (BECK, 2000) environment and proposes a solution that may be suitable to minimize this problem. Specifically, the main problem consists on how to improve the assumptions of the project managers - in the aforementioned environment - related with the human resources performance in generating value for the business. This value generation is reached through the implementation, by programmers, of the various functionalities of a *software* system. To build the solution proposed in this research, the author analysed a XP *software* development system (composed of environment, people and process) considering the modeling process proposed by Streit (2006) and also the relevant related works. This system was later structured in a conceptual model and, in sequence, in a computational model based on the Beliefs-Desires-Intentions (BDI) architecture of intelligent agents. The computational model of the multi-agent simulation was build with the support of the SeSAm (KLÜGL, 2006) tool. The tests of the multi-agent simulation of XP *software* develoment process model used the 2^k Factorial statistical experimentation (LAW e KELTON, 2000) and their results demonstrated the effectiveness and practical applicability of the model for the research problem.

Keywords: multi-agent systems, *software* development, extreme programming, simulation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANOVA - *ANalysis Of VAriance* (Análise de Variância).

BDI - *Beliefs-Desires-Intentions* (Crenças-Desejos-Intenções).

CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico.

GNU - *GNU is Not Unix* (sigla recursiva que significa “GNU não é Unix”).

IA - Inteligência Artificial.

IAD - Inteligência Artificial Distribuída.

LGPL - *Lesser General Public License* (Licença Geral Pública Inferior).

SeSAM - *Shell for Simulated Agent Systems* (Interface para Simulação de Sistemas de Agentes).

UML - *Unified Modeling Language* (Linguagem Unificada de Modelagem).

XP - *eXtreme Programming* (Programação Extrema).

SUMÁRIO

1 INTRODUÇÃO

Nesta dissertação, o autor aborda a dificuldade de prever-se o desempenho dos recursos humanos em um processo de desenvolvimento de *software* em um ambiente de Programação Extrema (BECK, 2000) e propõe uma solução com potencial para minimizar esse problema.

Neste primeiro capítulo, serão apresentados o tema, o problema que esta pesquisa aborda, os objetivos e a justificativa para este trabalho.

O Capítulo 2 apresentará os Fundamentos da Programação Extrema. Em seguida, o Capítulo 3 apresentará os Fundamentos de Agentes Inteligentes.

No Capítulo 4, o autor apresentará a Revisão da Literatura recente, onde serão abordados: os processos de desenvolvimento de *software*; os principais modelos de simulação desses processos; alguns dos principais trabalhos envolvendo a simulação multi-agente de processos de desenvolvimento de *software* e uma comparação sucinta entre a simulação multi-agente e outros métodos de simulação.

O Capítulo 5 apresentará o Método da Pesquisa desenvolvida, através da descrição das etapas do desenvolvimento deste trabalho e do processo de modelagem de processos de desenvolvimento de *software*.

A descrição detalhada do modelo de simulação será apresentada no Capítulo 6. Nesse capítulo, serão apresentados o sistema analisado, o modelo conceitual desenvolvido e o modelo computacional projetado, juntamente com o processo de modelagem utilizado para a construção do modelo de simulação proposto.

No Capítulo 7, será apresentada a abordagem escolhida para os Experimentos com o modelo de simulação projetado, juntamente com os testes e os resultados alcançados.

O Capítulo 8 apresentará algumas considerações finais sobre esta pesquisa. Em seguida, apresentará algumas limitações deste trabalho e apontará alguns possíveis rumos de investigação futura.

Finalmente, são apresentadas as Referências consultadas ao longo da elaboração desta dissertação.

1.1 TEMA E PROBLEMA DE PESQUISA

Gerentes de projetos, entre suas diversas atribuições, devem preocupar-se com o gerenciamento dos recursos humanos em projetos (PMBOK, 2000).

O problema, a ser tratado neste trabalho, consiste em melhorar as previsões dos gerentes de projeto – especificamente, no âmbito de um projeto de desenvolvimento de *software* em um Ambiente de Programação Extrema - com relação ao desempenho dos recursos humanos na geração de valor para o negócio. Tal valor para o negócio é alcançado através da implementação, por parte dos programadores, das diversas funcionalidades de um sistema de *software*.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Auxiliar o gerente de projeto em seu processo de tomada de decisão com relação ao desempenho dos Recursos Humanos na geração de valor para o negócio. Tal valor para o negócio é alcançado através da implementação, por parte dos desenvolvedores (programadores), das diversas funcionalidades de um sistema de *software* em um ambiente de Programação Extrema.

1.2.2 Objetivos Específicos

Os Objetivos Específicos deste trabalho são:

a) elaborar um modelo de simulação, para apoio às decisões da gerência, que incorpore aspectos de Recursos Humanos e de processos de desenvolvimento de *software* em um ambiente de Programação Extrema;

b) configurar um modelo computacional para execução da simulação proposta.

1.3 JUSTIFICATIVA

A indústria de desenvolvimento de *software* possui um longo histórico de fracassos relacionados com os seus respectivos custos, qualidade do produto final e prazo de execução. Esses fracassos têm impulsionado pesquisas que visam desde a criação de ferramentas de apoio ao desenvolvimento (SILVA et al., 1999; WICKENBERG e DAVIDSSON, 2002; MORENO et al., 2003) até a alteração dos processos de desenvolvimento de *software* (BECK, 2000; HUMPHREY, 2005) para que os mesmos proporcionem produtos de maior qualidade, dentro do prazo e a custos menores.

No entanto, deve-se observar que imposição de mudanças a um processo de desenvolvimento de *software* consome tempo e não traz garantias de que tais mudanças resultarão em melhorias no processo (WICKENBERG e DAVIDSSON, 2002). Além disso, os desenvolvedores adicionam complexidade à rede de interdependências criadas pelo processo através de suas características individuais (p. ex.: experiência profissional, afinidade com outras pessoas) (WICKENBERG e DAVIDSSON, 2002; MORENO et al., 2003). Dessa forma, com freqüência, torna-se difícil prever como determinadas decisões sobre o projeto, relacionadas aos Recursos Humanos e às metodologias de desenvolvimento de *software*, afetarão o resultado do processo como um todo (WICKENBERG e DAVIDSSON, 2002).

A simulação de um processo de desenvolvimento de *software* oferece uma solução para essas limitações, permitindo que gerentes (normalmente hesitantes em experimentar mudanças no processo real, devido aos altos custos em tempo e recursos) e desenvolvedores elaborem várias configurações do processo e possam compreender melhor os efeitos de várias políticas (WICKENBERG e DAVIDSSON, 2002).

Especificamente, a simulação multi-agente de um processo de desenvolvimento de *software* demonstra ser a abordagem adequada para o problema deste trabalho pois, segundo Wickenberg e Davidsson (2002), a simulação multi-agente torna-se apropriada quando:

a) o resultado de um processo de desenvolvimento de *software* sob estudo é determinado por decisões discretas feitas por indivíduos interagindo uns com os outros e com seu ambiente;
b) existe um alto grau de localidade, seja em termos das características do ambiente, das características dos indivíduos, ou da informação disponível;
c) queremos simular uma organização, ou projeto específico, onde as características de cada um dos desenvolvedores de *software* são conhecidas;
d) queremos estudar a sensibilidade do processo de desenvolvimento de *software* à características individuais – p.ex.: medir o impacto de mudanças no comportamento de indivíduos;
e) quando políticas de gerenciamento e outras descrições de comportamentos individuais precisam ser capturadas no modelo ou traduzidas de volta para a organização.

Além disso, considerando-se que a imprevisibilidade relacionada com o gerenciamento de recursos humanos causa um grande impacto sobre os resultados finais de um projeto (p. ex.: custo, qualidade e prazo), em especial nos processos de desenvolvimento de *software*, pode-se observar que a busca por soluções para o problema de melhorar as previsões dos gerentes de projeto com relação ao desempenho dos recursos humanos pode ter conseqüências positivas tanto para a multibilionária indústria do *software* (projeção de

faturamento mundial de US$ 190.7 bilhões para 2008) (BRODKIN, 2008) quanto para a gestão de projetos em geral.

Finalmente, o autor julga o tema desta dissertação relevante no contexto atual da sociedade brasileira, pois o desenvolvimento de *software* tem demonstrado ser um importante fator gerador de riqueza para o Brasil. Alguns números do mercado de desenvolvimento de *software* podem ser observados na figura abaixo.

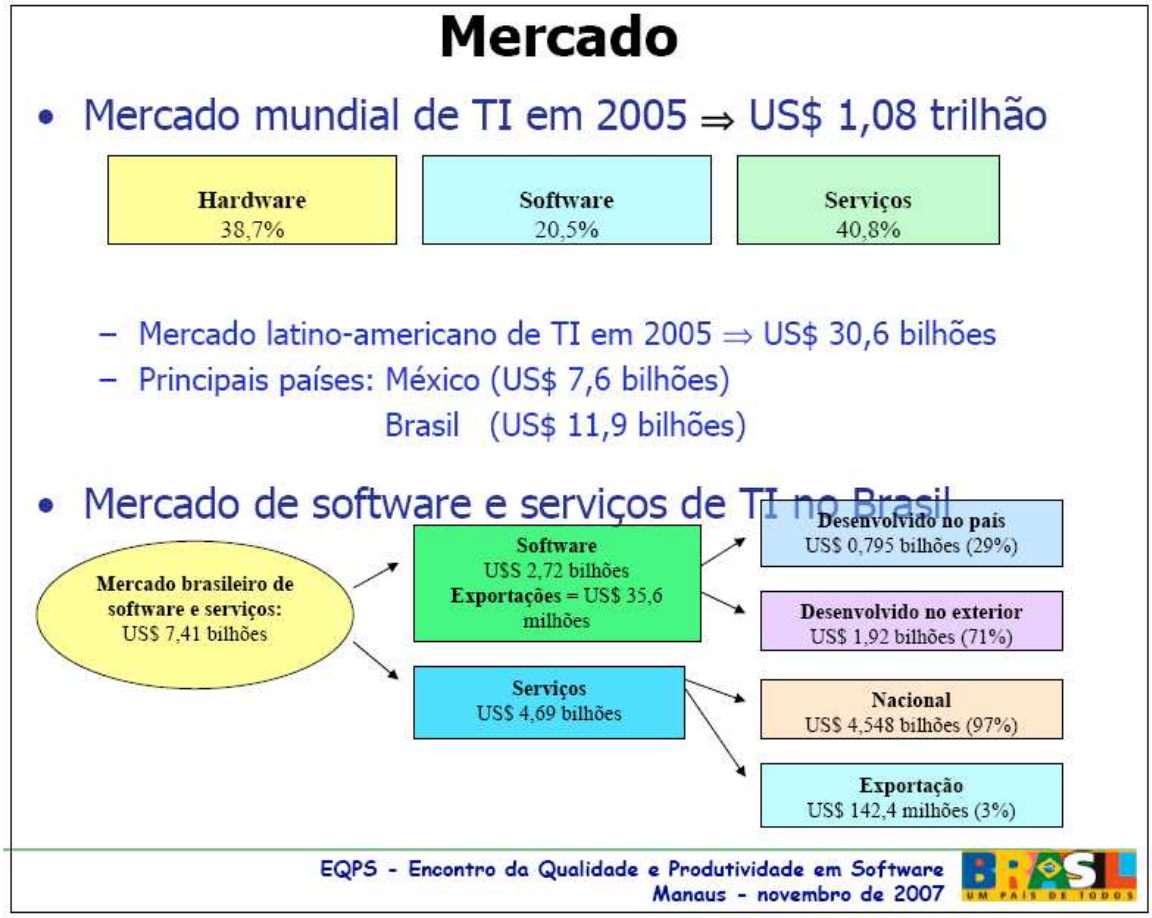


Figura 1 - Cenário do Mercado de Tecnologia da Informação
Fonte: Corrêa (2007).

2 FUNDAMENTOS DA PROGRAMAÇÃO EXTREMA

Conforme Beck (2000), **criador da Programação Extrema**, no desenvolvimento de *software* o risco é o problema básico. Tal risco pode manifestar-se de diversas maneiras, sendo que o autor acima destaca as seguintes:

a) deslizes no Cronograma – o dia da entrega chega e o *software* não está pronto;
b) projeto Cancelado – depois de vários deslizes, o projeto é cancelado;
c) sistema "Azeda" – com o tempo, o custo das modificações torna a manutenção do *software* inviável;
d) taxa de Erros – após o *software* ser colocado em produção, uma grande taxa de erros força o abandono do *software*;
e) negócio Mal Compreendido – o *software* não resolve o problema;
f) modificações nos Negócios – após o *software* ser colocado em produção, o problema de negócio que deu origem ao *software* foi substituído por outro problema, mais urgente;
g) falsa Riqueza de Funções – o *software* possui muitas funções interessantes que, no entanto, não geram dinheiro para o cliente;
h) rotatividade da Equipe – após dois anos, todos os bons programadores começam a odiar o *software* e abandonam a equipe.

Muitos desses riscos originam-se no método tradicional de desenvolvimento de *software* (BECK, 2000), em que um dos exemplos é o Método em Cascata, onde: (i) espera-se que os usuários digam, de uma vez só, como o *software* deverá operar; (ii) o sistema é projetado de acordo com o que os usuários disseram; (iii) o sistema é codificado de acordo com o projeto; (iv) o sistema é testado para garantir que os pedidos tenham sido atendidos. No entanto, o problema desse método está no fato de que os usuários, normalmente, não sabem explicar exatamente o que precisam, ou mudam de idéia após a sua explicação. Para piorar as coisas, após três quartos do projeto, os programadores, normalmente, descobrem que apenas executaram um terço do mesmo.

Em um método tradicional de desenvolvimento de *software*, o custo das modificações cresce exponencialmente ao longo do tempo (BECK, 2000).

Para minimizar os riscos no desenvolvimento de *software*, a Programação Extrema oferece um novo método, com ciclos de desenvolvimento mais curtos, testes contínuos e lançamentos frequentes do *software*. Esse método permite, entre outras vantagens, que o cliente obtenha retorno o mais rapidamente possível sobre seu investimento.

A Figura a seguir mostra a evolução dos métodos de desenvolvimento de *software*, desde o Método em Cascata (*Waterfall*) – com seus ciclos de desenvolvimento sequenciais de Análise, Projeto, Implementação e Teste (*Analysis, Design, Implementation, Test*) – até o Método da Programação Extrema (*eXtreme Programming* – XP) – que executa todos os ciclos de desenvolvimento do Método em Cascata de maneira iterativa (com tais iterações ocorrendo em intervalos pequenos e regulares ao longo de todo o tempo do projeto).

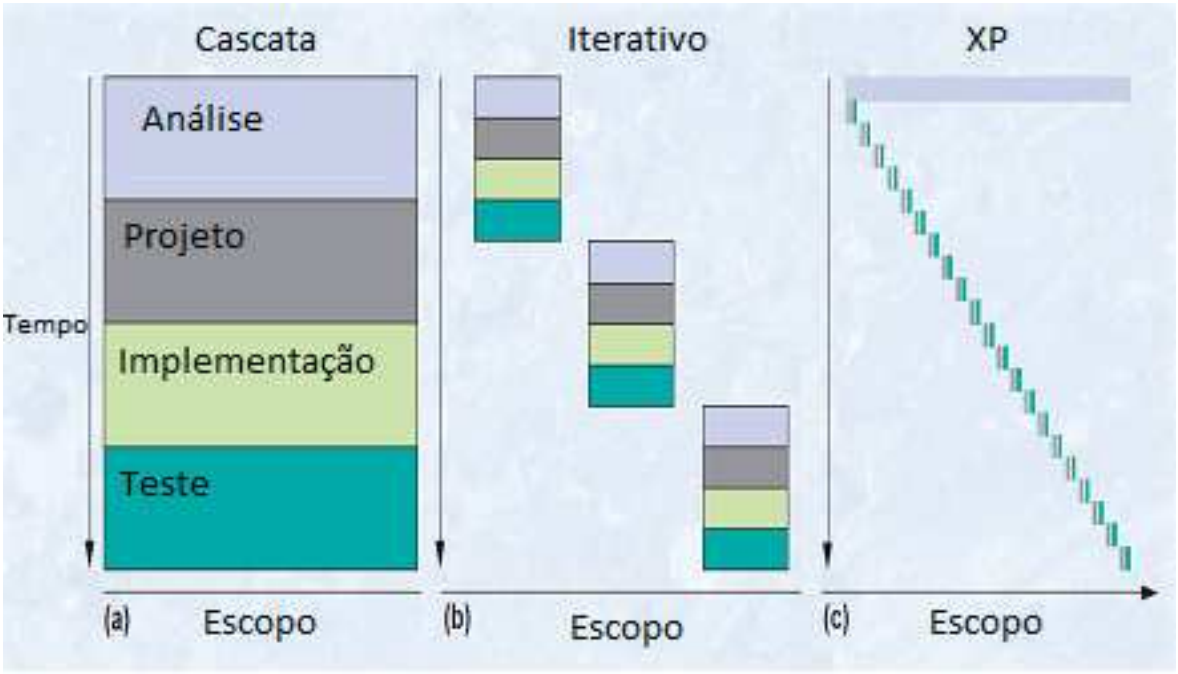


Figura 2 - Evolução dos Métodos de Desenvolvimento de *Software*

Fonte: figura adaptada de Beck (2000).

Como no Método da Programação Extrema todos os ciclos de desenvolvimento são executados de maneira aproximadamente paralela e iterativa (a intervalos de tempo pequenos e regulares), os custos provocados por mudanças no projeto tendem a permanecer pequenos se comparados com um método tradicional, uma vez que eventuais falhas (erros de programação ou atrasos) podem ser detectadas rapidamente e com maior facilidade – em contraste com o método tradicional, onde os erros de programação somente serão detectados nos períodos finais do cronograma do projeto.

A metáfora utilizada pelo Método XP, para ilustrar as frequentes iterações ao longo do desenvolvimento do projeto, é a de dirigir um veículo ao longo de uma estrada. Assim, o Método XP pode ser compreendido como um condutor (a equipe de desenvolvimento) que efetua pequenos e constantes ajustes na trajetória de seu veículo (o projeto) de acordo com as mudanças em sua situação (necessidades do negócio), para mantê-lo no rumo correto (dentro do orçamento, do prazo, da qualidade e do escopo) (BECK, 2000).

Uma visão geral do Método da Programação Extrema pode ser observada na Figura 3.

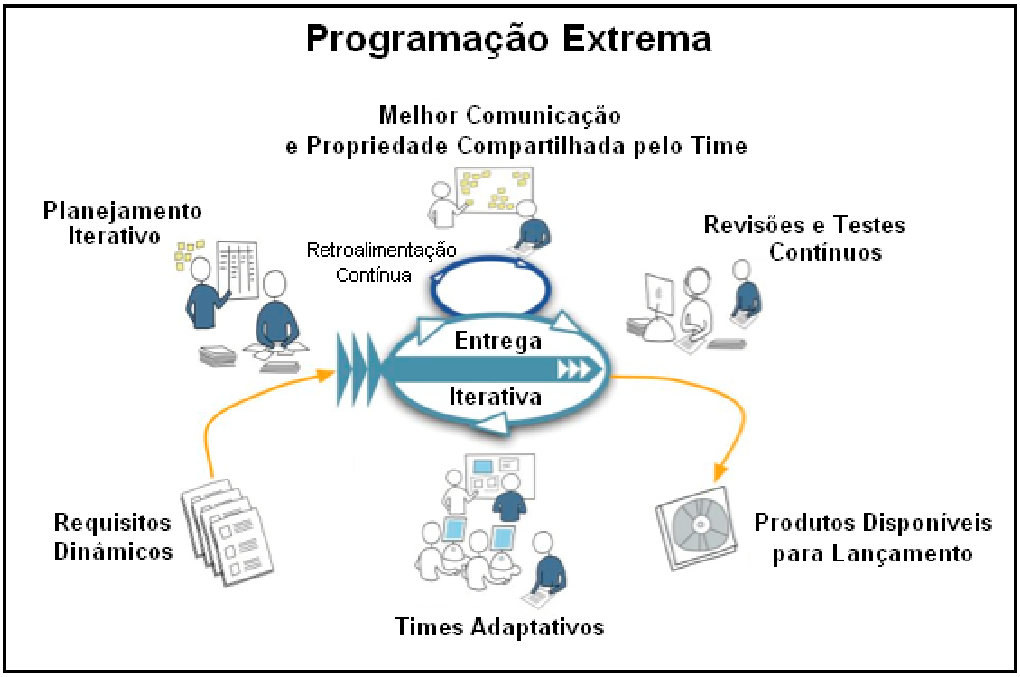


Figura 3 - Visão Geral da Programação Extrema

Fonte: adaptado de Electroglide.biz (2008).

2.1 CONCEITOS BÁSICOS, AMBIENTE E FLUXOGRAMA DA PROGRAMAÇÃO EXTREMA

A seguir, serão apresentados os conceitos básicos da Programação Extrema, seguidos por ilustrações do ambiente de desenvolvimento típico desse método e, finalmente, pelo seu fluxograma de atividades:

a) **tempo ideal de programação** - período de trabalho produtivo e sem distrações;

b) **iteração** - período com duração de uma a quatro semanas, durante o qual uma certa quantidade de histórias é implementada;

c) **história** - algo que o cliente quer que o sistema faça; deve ser algo testável; deve requerer de uma a quatro semanas de programação ideal;

d) **história normal** (conceito criado pelo autor deste trabalho apenas para o Fluxograma da Figura 7. Em XP, há apenas um conceito para História, já visto acima.) - história prevista para implementação em determinada iteração; agrega 100% do valor, caso seja implementada; se implementada com atraso, não agrega valor algum; desconta 50% do valor acumulado, caso não seja implementada;

e) **história extra** (conceito criado pelo autor deste trabalho apenas para o Fluxograma da Figura 7. Em XP, há apenas um conceito para História, já visto acima.) - história requisitada para implementação **muito cedo** (agrega apenas 50% do valor previsto, quando implementada na mesma iteração em que foi requisitada; não implica penalidade por não implementação nessa iteração). Se os desenvolvedores estão adiantados e implementam uma história antes do prazo, essa história agregará ao negócio apenas **metade** de seu valor. Isso ocorre porque a empresa (através de seu departamento de marketing) não anunciou a funcionalidade correspondente - logo, **poucos** comprarão o produto por causa dessa funcionalidade;

f) **história implementada com atraso** - se os desenvolvedores não conseguem entregar uma história no prazo, **nenhum valor** correspondente à essa história é agregado ao negócio (penalidade sobre o faturamento e a reputação da empresa);

g) **história não implementada** - todas as histórias não implementadas terão a metade de seus respectivos valores **descontados do valor total** obtido com as histórias implementadas (penalidade sobre o faturamento e a reputação da empresa);

h) **teste de aceitação** - teste definido pelo cliente e pelos desenvolvedores para a aceitação da implementação de cada uma das histórias;

i) **estimativa** - desenvolvedores analisam as histórias e estimam o tempo necessário para implementar cada uma (**esforço**). Os desenvolvedores também estimam o **esforço máximo** para a iteração (para a primeira iteração, a estimativa é, normalmente, inconsistente);

j) **velocidade** - razão entre a quantidade de esforço gasta até a iteração atual (com a implementação de todas as histórias completadas) e o número de iterações finalizadas. A velocidade representa com maior consistência (a partir da segunda iteração) **o esforço máximo** de desenvolvimento que a equipe pode oferecer em uma determinada iteração;

k) **planejamento** - o cliente recebe as histórias com os tempos estimados e ordena as mesmas de acordo com algum critério, que pode ser: (i) história geradora do maior valor para o negócio; (ii) história que requer o maior ou o menor tempo; (iii) história que possua a **maior** taxa de valor/esforço; (iv) história destinada para o cliente mais importante. Após ordenar as histórias, o cliente escolhe aquelas que deverão ser implementadas durante a iteração, respeitando o **esforço máximo** (**velocidade**) da equipe de desenvolvimento;

l) **cliente** - aquele que escreve as histórias e decide a ordem de implementação das mesmas. O cliente também define testes que verifiquem o funcionamento das histórias implementadas. O cliente, normalmente, é externo (ou seja, é o futuro usuário final do produto);

m) **desenvolvedor** (ou **Programador**) - aquele que analisa, projeta, testa, programa e integra o *software* desenvolvido a partir das histórias;

n) **programação em pares** - técnica de programação em que duas pessoas programam em apenas um computador; normalmente, os pares mudam duas vezes por dia;

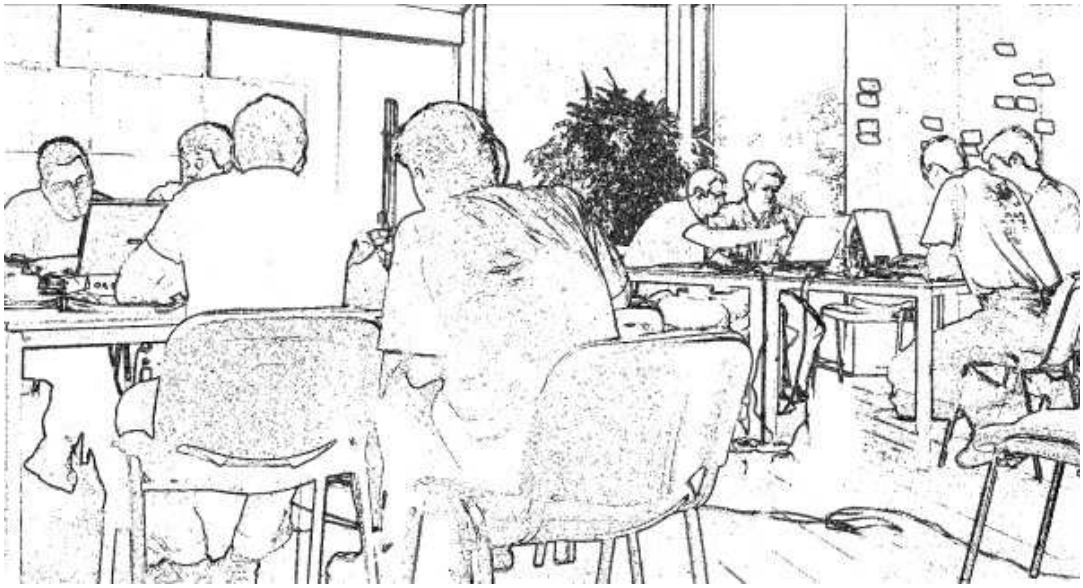


Figura 4 - Programação em Pares
Fonte: Agile Workshops (2009).

o) **Ambiente** (ou **Laboratório**) – Ambientes XP são otimizados para a **produtividade**. Programadores e gerentes de produto acomodam-se em uma grande sala. Programadores formam **pares** em computadores no centro da sala e integram o código em um computador próximo, enquanto os gerentes de produto (**clientes**) localizam-se em mesas na periferia da sala. Isso permite que os programadores façam perguntas pessoalmente aos gerentes de produto. Tais questões freqüentemente são respondidas em segundos, permitindo que um par rapidamente continue sua programação produtiva.

Figura 5 - Exemplo de Ambiente de Programação Extrema com Descrições dos Objetos
Fonte: adaptado de Scissor.com (2009).

Figura 6 - Exemplo de Ambiente de Programação Extrema
Fonte: Agile Workshops (2009).

Na Figura 7, a seguir, onde “C” indica uma tarefa do Cliente e “D” indica uma tarefa do Desenvolvedor, podemos observar o fluxograma do Processo de Desenvolvimento baseado no método da Programação Extrema.

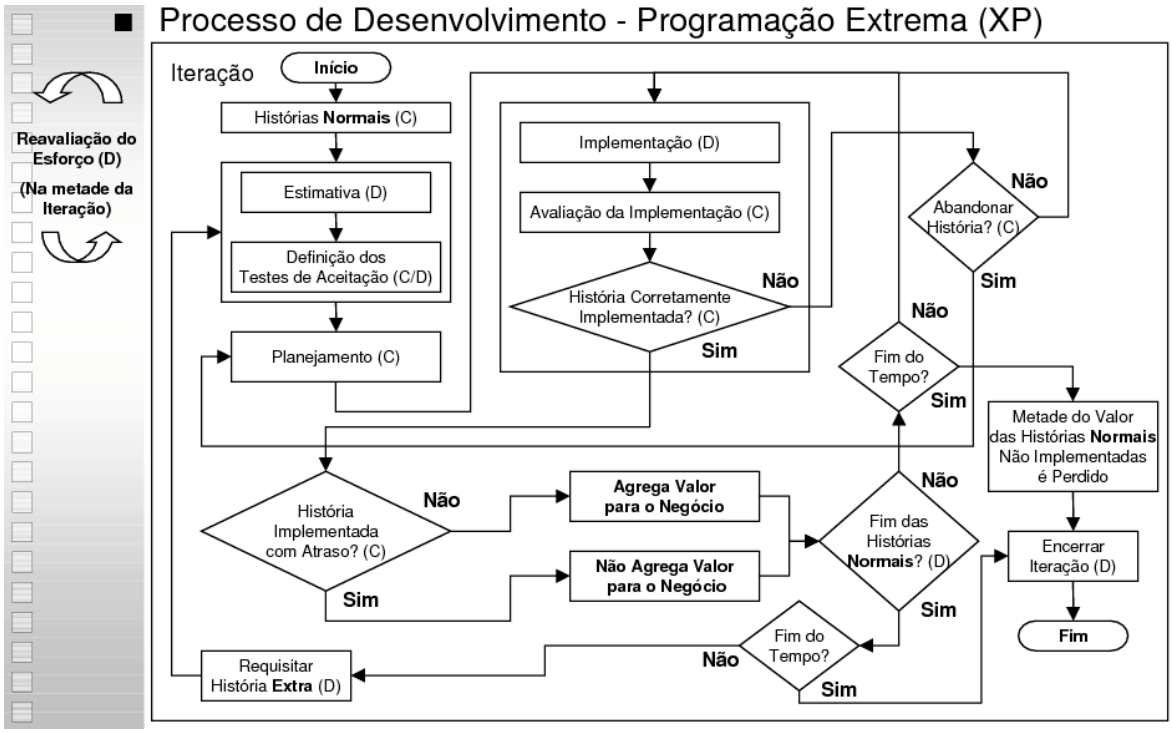


Figura 7 - Processo de Desenvolvimento do método da Programação Extrema

Fonte: figura elaborada pelo autor a partir de Beck (2000).

3 FUNDAMENTOS DE AGENTES INTELIGENTES

3.1 DEFINIÇÃO DE AGENTES INTELIGENTES

Na literatura (WOOLDRIDGE e JENNINGS, 1995; WEISS, 1999) há a definição de agentes (ou agentes inteligentes) como entidades (lógicas ou físicas - p.ex.: *software*s ou robôs) capazes de perceberem os estímulos do ambiente e de agirem sobre esse ambiente. Adicionalmente, os agentes também são entidades autônomas, uma vez que seus comportamentos dependem - ao menos parcialmente - de suas próprias experiências. Finalmente, sendo entidades interativas, os agentes podem ter o cumprimento de suas tarefas afetado pela interação com outros agentes, ou mesmo com seres humanos - e tal interação pode ser tanto de natureza cooperativa quanto de natureza competitiva.

Embora não exista um consenso com relação à definição de agentes inteligentes, existem mais pontos em comum do que diferenças entre as definições dos diversos pesquisadores da área de Inteligência Artificial Distribuída (MACAL e NORTH, 2005). O ponto fundamental sobre os agentes inteligentes, segundo Macal e North (2005), está na capacidade dos mesmos tomarem decisões independentes – o que requer que esses sejam entidades ativas em seu ambiente.

3.2 CARACTERÍSTICAS DE AGENTES INTELIGENTES

Macal e North (2005) enumeram as seguintes características comuns relativas à modelagem de agentes inteligentes:

a) um agente é identificável, isto é, é uma entidade discreta (logo, com uma fronteira que o separa do ambiente, permitindo discernir o que faz e o que não faz parte do agente) com um conjunto de características e regras que governam o seu comportamento e capacidade de tomada de decisão;

b) um agente está situado em um determinado ambiente, com o qual ele interage e no qual ele interage com outros agentes através de protocolos de comunicação;

c) um agente possui a capacidade de identificar outros agentes;

d) um agente é uma entidade direcionada, com relação aos seus comportamentos, para o cumprimento de objetivos (os quais não são, necessariamente, de maximização de resultados);

e) um agente é autônomo e auto-dirigido, isto é, pode operar independentemente no ambiente onde está inserido;

f) um agente é flexível e possui as habilidades de aprendizado e de adaptação de seus comportamentos, com o passar do tempo, de acordo com a experiência adquirida – o que requer uma forma de memória interna.

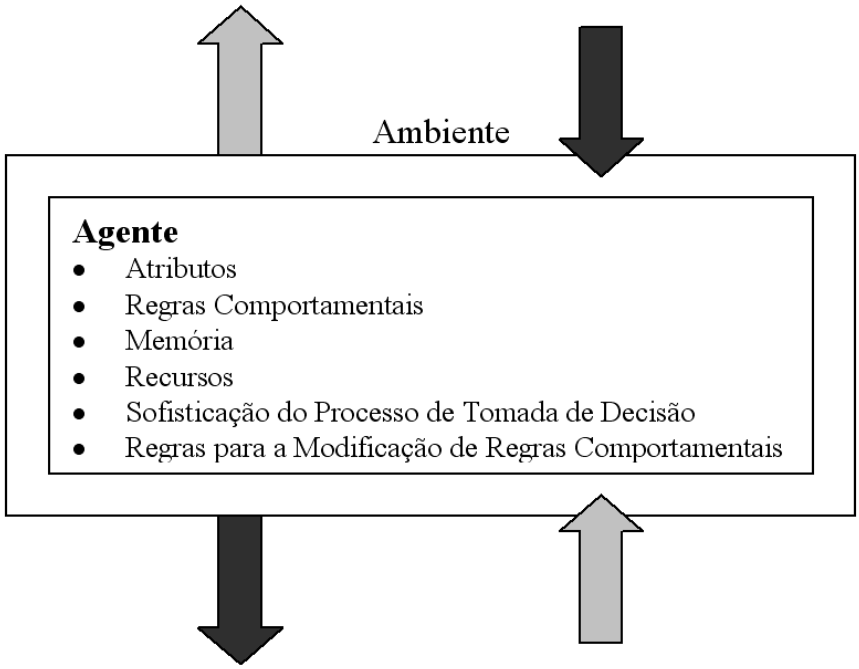


Figura 8 - Um agente inteligente
Fonte: Macal e North (2005).

3.3 ARQUITETURAS DE AGENTES INTELIGENTES

Segundo Müller (1998) e López (2004), as arquiteturas de agentes inteligentes podem ser classificadas em quatro tipos:

a) **deliberativa** - os agentes modelados com esta arquitetura são aqueles cujo comportamento envolve diferentes processos de "raciocínio" antes de uma decisão. Normalmente, esta arquitetura contém uma representação simbólica do ambiente externo;

b) **interativa** - esta arquitetura inclui mecanismos e elementos mentais adequados para lidar com a presença de outros agentes. Existem também protocolos de cooperação que irão permitir a comunicação com outros agentes;

c) **reativa** - agentes modelados com a arquitetura reativa respondem imediatamente à estímulos que ocorrem no ambiente. As principais características desta arquitetura são a desconsideração pelas conseqüências futuras de ações presentes e a falta de planejamento para coordenação em atividades com outros agentes;

d) **híbrida** - a arquitetura híbrida, por sua vez, tenta combinar as vantagens das arquiteturas já mencionadas.

3.4 SISTEMAS MULTI-AGENTE

Sistemas multi-agente derivam diretamente da área de estudos da Inteligência Artificial Distribuída (IAD), a qual origina-se na disciplina de Inteligência Artificial.

3.4.1 A Inteligência Artificial Distribuída

Enquanto a Inteligência Artificial (IA) clássica utiliza como modelo, normalmente, o comportamento individual humano, a Inteligência Artificial Distribuída (IAD) baseia-se no comportamento social, isto é, no comportamento que emerge a partir da interação de vários indivíduos com características e com objetivos próprios (RUSSEL e NORVIG, 1995; WEISS, 1999; MORENO et al., 2003).

Na literatura (WEISS, 1999), uma das definições possíveis para a IAD é a seguinte: "IAD é o estudo, construção e aplicação de sistemas multi-agentes, isto é, sistemas nos quais vários agentes inteligentes e interativos perseguem alguns objetivos ou desempenham algum conjunto de tarefas."

3.4.2 Classificação de Sistemas Multi-Agente

Os sistemas multi-agente são classificados em sistemas com agentes cognitivos e sistemas com agentes reativos (embora possam ser implementados sistemas com ambos os tipos de agentes) (SILVA et al., 1999).

Nos sistemas multi-agente cognitivos, os agentes mantêm uma representação explícita do ambiente, da memória de suas respectivas ações e da presença dos outros agentes. Nos sistemas multi-agente cognitivos, a comunicação entre os agentes é direta (através de mensagens) e seu mecanismo de controle é deliberativo (os próprios agentes decidem sobre seus objetivos, planos e ações). A organização é sociológica e, em geral, existem poucos agentes na sociedade.

Nos sistemas multi-agente reativos, não existe uma representação explícita do conhecimento, nem do ambiente e nem da memória (das ações) dos agentes. A organização é etológica - similar à organização de insetos como formigas, cupins e abelhas - e existe um grande número de agentes no ambiente.

3.4.3 Oportunidades para Sistemas Multi-Agente

Macal e North (2005) apresentam algumas oportunidades para a aplicação de sistemas multi-agente, mostradas a seguir:

a) problemas cuja complexidade emerge da interdependência de diversos fatores;

b) problemas cuja modelagem mostra-se complexa e, por vezes, inadequada - tal como o caso da modelagem de mercados, para a qual uma modelagem tradicional deve apoiar-se em noções tais como: mercados perfeitos; agentes homogêneos (cujas características são definidas a partir de médias populacionais) e equilíbrio de longo prazo;

c) problemas que envolvam grandes quantidades de dados e que exijam grande poder computacional – pois o nível de detalhamento das informações armazenadas em bancos de dados passou a permitir simulações mais detalhadas (em nível “micro”, ao contrário das técnicas de simulação tradicionais, de nível “macro”) e o poder computacional tem avançado significativamente ao longo das últimas décadas, permitindo a execução dos modelos sobre uma maior quantidade de informações em um curto período de tempo.

Segundo os autores acima, o modelador de um sistema poderá aproveitar melhor as vantagens da modelagem multi-agente quando:

a) os agentes representam naturalmente o sistema;

b) existem decisões e comportamentos que podem ser definidos discretamente (dentro de determinadas fronteiras);
c) os agentes devem se adaptar e mudar seus respectivos comportamentos;
d) é importante que agentes aprendam e assumam comportamentos estratégicos dinâmicos;
e) os agentes devam estabelecer relações dinâmicas com outros agentes e que tais relações formem-se e desfaçam-se dinamicamente;
f) os agentes devam formar organizações e que a adaptação e o aprendizado seja importante no nível organizacional;
g) é importante que os agentes possuam um componente espacial que oriente seus comportamentos e interações;
h) o passado não for determinante do futuro;
i) a escala para níveis superiores arbitrários (p.ex.: quantidade de agentes na organização) for importante;
j) uma mudança estrutural do processo precisa ser um resultado do próprio modelo, ao invés de uma alteração externa sobre o modelo.

4 REVISÃO DA LITERATURA

4.1 PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Um projeto de desenvolvimento de *software* é uma instância de um processo executável que, por sua vez, é uma instância de um modelo de processo de desenvolvimento de *software* (SILVA et al., 1999). Segundo Sikka (2005), um processo de desenvolvimento de *software* é um conjunto de atividades desempenhadas por um conjunto de desenvolvedores. Tais atividades - que podem ser, por exemplo, de definição de requisitos, de análise de arquitetura, de especificação de testes unitários, de programação, ou de design de casos de uso - resultam em um conjunto de artefatos que, por sua vez, apoiarão as etapas seguintes do desenvolvimento.

Neste trabalho, será utilizada a expressão "Processo de Desenvolvimento de *Software*" (*Software* Development Process) para referir-se à instância de um processo executável (logo, para referir-se a um determinado projeto). Um processo de desenvolvimento de *software* envolve objetivos, restrições, recursos, desenvolvedores, prazos e orçamentos (SILVA et al., 1999).

Os processos de desenvolvimento de *software* podem ser executados através de várias metodologias, cada uma com suas respectivas vantagens, desvantagens e grau de complexidade. As metodologias clássicas para processos de desenvolvimento de *software* são: Cascata; Incremental; Espiral; Prototipação; Sala Limpa; Orientada a Objetos; além de variações da Metodologia Ágil (SIKKA, 2005).

4.2 SIMULAÇÃO DE PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Uma das mais populares técnicas da Pesquisa Operacional utilizada para validação de modelos é a técnica de Simulação, que utiliza recursos computacionais para imitar o comportamento de um sistema ou de um processo real (HILLIER, 2005).

Embora com a ressalva de que uma simulação não oferece resultados otimizados – a simulação apenas mostra qual será o resultado de acordo com uma certa alternativa de ação (ou seja, um determinado conjunto de valores iniciais para os parâmetros da simulação) - a utilização da Simulação apresenta diversas vantagens, sendo alguma delas (PRAÇA e RAMOS, 1999):

a) aplicação em problemas muito difíceis ou complexos para serem resolvidos matematicamente (simulações podem levar a soluções quase ótimas, através de tentativa e erro);
b) utilização como uma ferramenta de treinamento;
c) estudo de um sistema sob condições controladas;
d) conforme Mayo (*apud* MACEFIELD, 2007), um modelo não sofre o **Efeito Hawthorne**, em que o comportamento das pessoas é modificado em conseqüência de estarem sendo estudadas;
e) uma simulação pode projetar condições que poderão manifestar-se no futuro;
f) o tempo de simulação pode ser expandido ou comprimido, para analisar-se com maior detalhe determinados eventos;
g) a Simulação é menos cara e envolve menores riscos do que um experimento real (por exemplo, com pessoas reais em um processo real).

A seguir, serão apresentados alguns exemplos de modelos de simulação de processos de desenvolvimento de *software*, encontrados na literatura recente. Cada um desses modelos exemplifica diferentes técnicas e abordagens para a construção da simulação em questão.

4.2.1 Modelo de Redes Bayesianas

Settas (2006) propõe em seu trabalho um modelo de gerenciamento de processo de desenvolvimento de *software*, baseado no método XP, que utiliza uma abordagem em Redes Bayesianas para modelar anti-padrões (soluções normalmente propostas para consertar comportamentos incomuns de gerentes ou práticas gerenciais que inibem o sucesso do projeto). Essa abordagem oferece um arcabouço (framework) aos gerentes que desejam modelar as relações de causa e efeito que estão por trás de um anti-padrão, considerando as incertezas inerentes a um projeto de *software*. O anti-padrão é modelado a partir de resultados empíricos (obtidos em um experimento controlado de desenvolvimento pelo método XP) que investigaram o impacto das personalidades e temperamentos dos desenvolvedores em sua comunicação e em seu trabalho de programação em pares.

4.2.2 Modelo de Sistemas Dinâmicos

Cao (2004) propõe um modelo de simulação baseado em sistemas dinâmicos (FORRESTER, 1991) para investicar a aplicabilidade dos métodos e impacto das práticas ágeis com relação aos custos, qualidade, prazo e satisfação do cliente.

Misic (2004) investiga a possibilidade de utilização da simulação de sistemas dinâmicos para modelar, simular e analisar o processo de desenvolvimento de *software* XP. O trabalho focaliza nos aspectos de qualidade do método XP, uma vez que o principal objetivo desse método é o de criar valor para o cliente através de produtos de alta qualidade.

Merrill e Collofello (1997) propõem um modelo de simulação, baseado em sistemas dinâmicos, do processo de desenvolvimento de *software* incremental. Tal modelo tem como objetivo auxiliar no treinamento de gerentes de processos.

Kuppuswami (2003 “a”) propõe um modelo de simulação, baseado em sistemas dinâmicos, do processo de desenvolvimento XP para mostrar a natureza constante da curva do custo de alterações versus tempo nos projetos desenvolvidos com o método XP.

Kuppuswami (2003 “b”), neste trabalho, desenvolve um modelo de simulação, baseado em sistemas dinâmicos, para analisar os efeitos das práticas do método XP no esforço de desenvolvimento de *software*. Tal modelo foi simulado para um projeto XP típico e os efeitos das práticas individuais do método XP também foram considerados.

4.2.3 Modelo em Lógica Fuzzy

Sanal (2000) apresenta um sistema de apoio à decisão para o acompanhamento (agendamento) de tarefas em um processo de desenvolvimento de *software*. Tal sistema - que tem por objetivo alocar engenheiros com diferentes níveis de experiência para tarefas de diferentes complexidades - utiliza um modelo em que as durações das tarefas são modeladas através de valores fuzzy (incertos ou nebulosos) dados em função do nível de experiência dos engenheiros. A experiência e a complexidade das tarefas também são modeladas através de valores fuzzy. O objetivo do acompanhamento de tarefas é o de minimizar o tempo de finalização do projeto.

4.2.4 Modelo Híbrido

Donzelli (2006) apresenta um sistema de apoio à decisão que utiliza um modelo híbrido de simulação do processo de desenvolvimento de *software*. Tal modelo utiliza as vantagens de três métodos tradicionais: o método analítico e as simulações contínua e discreta. Em um nível mais alto de abstração, o processo de desenvolvimento de *software* é modelado como uma rede de filas de eventos discretos – isto é, um conjunto de estações de serviço que processam clientes. Um sistema de filas oferece uma maneira natural de

representar a estrutura de um processo, suas atividades, interações e os artefatos que são trocados. Finalmente, em um nível mais baixo de abstração, modelos analíticos e simulação contínua são utilizados para representar o comportamento dinâmico de cada uma das estações de serviço do nível mais alto.

4.2.5 Modelo Orientado a Eventos

Mauerkirchner (1997) apresenta um modelo de simulação, baseado em eventos, para a criação de um método geral de alocação de recursos humanos. Tal modelo é composto por dois subsistemas: (i) o subsistema dedicado ao planejamento – com o objetivo principal de fornecer tabelas de tempos otimizados para o fluxo do projeto, através de simulação baseada em eventos; (ii) o subsistema de controle, para a emulação do projeto real.

4.2.6 Modelo Orientado a Objetos

Na pesquisa de Cau (2005) foi desenvolvido um modelo de simulação, orientado a objetos, para avaliar a aplicabilidade e a eficiência do processo de desenvolvimento XP, além dos efeitos de suas práticas individuais nos resultados dos projetos.

4.2.7 Modelo Probabilístico

Padberg (1999) propõe um modelo probabilístico de simulação do processo de desenvolvimento de *software* para tentar responder sobre as chances de um determinado projeto ser bem-sucedido dentro de um determinado prazo de desenvolvimento. Tal

resposta depende do caminho que o projeto segue em determinados momentos, fato que não permite prever o curso completo do projeto e que, consequentemente, reflete a incerteza e o risco inerentes para cada projeto. Para tentar atingir essa resposta, o modelo computa as probabilidades de cada caminho que o projeto poderia seguir ao longo de sua execução.

4.2.8 Modelo Qualitativo

Zhang (2006) apresenta um modelo qualitativo de simulação da alocação de pessoal para um processo de desenvolvimento de *software*. Tal modelo, ao contrário dos modelos puramente quantitativos de sistemas dinâmicos, não requer conhecimentos detalhados dos processos a serem simulados. A simulação qualitativa utiliza conhecimentos incompletos para gerar descrições qualitativas de todos os comportamentos possíveis para o sistema. Essas descrições são apresentadas na forma de comportamentos e estados consistentes com equações diferenciais qualitativas que, por sua vez, representam um conjunto maior de possíveis equações diferenciais ordinárias (equações que envolvem as derivadas das funções quantitativas das variáveis do modelo).

4.2.9 Modelo Multi-Agente

Como o foco deste trabalho está voltado para o modelo de simulação multi-agente, algumas pesquisas relacionadas a esse modelo serão apresentadas, em maior detalhe, na próxima seção.

4.3 SIMULAÇÃO MULTI-AGENTE DE PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Wickenberg e Davidsson (2002) afirmam que uma visão alternativa do processo de desenvolvimento de *software,* em contraste com a visão tradicional (que prioriza o foco sobre as atividades do processo), é a de que o mesmo é executado por um conjunto de indivíduos, com características próprias, que interagem uns com os outros.

De acordo com essa visão, os autores acima também afirmam que os métodos tradicionais de simulação – voltados para abordagens abstratas baseadas em atividades - não conseguem capturar os aspectos característicos de um trabalho cooperativo desenvolvido por atores com características próprias. Ao contrário desses métodos tradicionais, a abordagem multi-agente permite focalizar nos membros individuais da organização, permitindo, desse modo, uma modelagem mais próxima da realidade do processo.

Wickenberg e Davidsson (2002) afirmam:

> Os indivíduos interagem tanto através da comunicação quanto através da manipulação de artefatos, tais como especificações de desenvolvimento ou unidades de código. Quando a simulação é executada, podemos observar características das interações entre os atores, e os efeitos resultantes. O comportamento de membros da equipe pode ser capturado em diferentes níveis de abstração, dependendo de qual aspecto do desenvolvimento deseja-se compreender ou prever. Em um nível maior de detalhe, poderíamos (em princípio) modelar a conversação entre desenvolvedores durante seu trabalho. No outro extremo da escala, poderíamos modelar o comportamento em termos da rotatividade de pessoal e efetuar sugestões para a melhoria do processo. No entanto, não estamos limitados à modelagem de pessoas que trabalham em uma organização, pois os agentes também são úteis para modelar diferentes departamentos em uma organização ou mesmo modelar as interações entre diferentes empresas de desenvolvimento e seus respectivos clientes, o que nos permite, assim, atingir níveis mais altos de abstração.

Esses autores lembram, também, de uma limitação relacionada com a aplicabilidade da simulação multi-agente. Tal limitação afirma que pouco tem-se a ganhar com a modelagem multi-agente de um sistema quando:

a) comportamento dos desenvolvedores é definido de uma forma tal que esse comportamento corresponde diretamente ao comportamento de um processo (um fluxo de atividades) predefinido;

b) conhecimento sobre o comportamento do desenvolvedor é limitado a medições coletivas sobre como eles desempenham suas tarefas durante atividades específicas.

Para ambas as condições acima, o resultado de uma simulação seria o mesmo, não importando se uma abordagem centralizadora ou individualista fosse utilizada. Se dois modelos diferentes (multi-agente ou voltado para tarefas) fornecem o mesmo resultado, o modelo a ser escolhido deverá ser o mais conveniente para criar-se. Nesse caso, uma melhor escolha seria um modelo baseado em atividades, uma vez que sua representação é análoga ao modo como o processo é compreendido.

Dessa forma, uma importante vantagem da simulação multi-agente de processos de desenvolvimento de *software*, segundo Wickenberg e Davidsson (2002), consiste na habilidade de modelar-se comportamentos que não correspondem exatamente a um fluxo de atividades predefinido. Segundo esses autores, a modelagem de indivíduos tem o potencial de permitir que sejam feitas melhores previsões sobre o processo de desenvolvimento. Previsões mais precisas podem ser obtidas em casos onde o conhecimento do desempenho dos desenvolvedores já for conhecido, ou quanto esse conhecimento puder ser coletado durante o projeto, através do uso de métodos de diagnóstico tais como o *Personal Software Process* (HUMPHREY, 2000).

As próximas seções apresentarão alguns exemplos de modelos de simulação multi-agente de processos de desenvolvimento de *software* encontrados na literatura recente.

4.3.1 Modelo Baseado em Casos

Gabineski e Lorenzi (2007) propõem um sistema multiagente, baseado em casos, com foco na gerência de projetos. O sistema proposto controla a alocação de recursos do projeto e as atividades a serem realizadas, de forma dinâmica e interativa. Agentes com funções específicas monitoram o banco de casos (que armazena as melhores práticas de um determinado projeto) e, quando há necessidade, sugerem ações para o gerente do projeto.

4.3.2 Modelo com Vários Times de Agentes

Moreno (2003) propõe a utilização de um sistema multi-agente para simular o comportamento de diferentes equipes encarregadas do desenvolvimento de um projeto complexo.

Partindo da observação de que, devido a diversos fatores, é difícil encontrar um grupo adequado para finalizar com sucesso um projeto complexo, Moreno (2003) lista os fatores mais relevantes a serem considerados durante a montagem de uma equipe:

a) características individuais (nível profissional; grau de experiência; tempo para execução de tarefas; custo econômico individual);
b) características sociais (disposição para trabalhar com outras pessoas);
c) custos temporais e econômicos.

Assim, esse autor apresenta um sistema multi-agente que simula o comportamento de diferentes equipes e fornece resultados estatísticos que visam auxiliar no descobrimento de quais equipes (ou seja, combinações de indivíduos) podem, potencialmente, executar o projeto com os menores custos financeiros e temporais.

4.3.3 Modelo de Agentes Cooperativos

Silva (1999) propõe um modelo de simulação de processos que oferece uma forma de testar modelos de processos de *software* usando agentes inteligentes (atuando no papel de desenvolvedores). A partir de uma instância de um modelo de processo (com atividades definidas, desenvolvedores e recursos alocados para as atividades e cronograma inicial), o modelo de simulação disponibiliza atividades para os agentes desenvolvedores (através de uma agenda do agente) e estes utilizam seu conhecimento armazenado para executar as atividades nas quais eles estão envolvidos. Durante a simulação, são levadas em consideração as habilidades dos agentes desenvolvedores, as afinidades entre os desenvolvedores e os recursos disponíveis.

Lorenzi (2007) propõe um modelo de agentes cooperativos baseados em conhecimento (e capazes de acessar uma base de conhecimentos distribuída) para o apoio de agentes de viagem (humanos) na recomendação de pacotes turísticos. Tais agentes cooperativos trabalham como especialistas, cooperando e comunicando-se uns com os outros durante o processo de recomendação.

4.3.4 Modelo de Agentes e Programação Linear Fuzzy

Huang (2001) apresenta uma abordagem de programação linear fuzzy aplicada sobre um arcabouço (framework) de agentes inteligentes (também modelados com a abordagem fuzzy) voltados para o desenvolvimento de produtos modulares. Essa abordagem tem por objetivo investigar como um projeto modular (com módulos provenientes de diferentes fornecedores, separados geograficamente e operando diferentes plataformas computacionais) poderia ser executado por agentes inteligentes para atender às expectativas nebulosas (fuzzy) dos clientes.

4.3.5 Modelo de Agentes Rastreadores de Métricas

Paes e Almeida (2005) apresentam como agentes de *software* podem auxiliar na utilização de métricas no desenvolvimento colaborativo de aplicações. O sistema multi-agente tem por objetivo prover flexibilidade na alteração das métricas, levando em conta a natureza distribuída da equipe de desenvolvimento e a integração com ambientes de desenvolvimento já existentes.

4.3.6 Modelo em Predicados Lógicos

Murta e Werner (2000) propõem um modelo de desenvolvimento de *software*, baseado em agentes inteligentes, que procura aumentar a possibilidade do processo ser seguido corretamente pelos desenvolvedores. A abordagem proposta consiste em mapear o modelo do processo em predicados *Prolog* (POOLE et al., 1998) que, por sua vez, formarão a base de conhecimento do processo. Os agentes inteligentes utilizariam essa base de conhecimentos (predicados) para o controle e execução do processo de desenvolvimento de *software*.

4.3.7 Modelo de Agentes Reativos

Berger (2008) apresenta um modelo de simulação de agentes, puramente reativos, que busca analisar qualquer lei de natureza penal, não importanto suas características ou a quem se destina. Tal modelo utiliza um construto Cidadão-Estado-Oportunidade-Ambiente, cujas partes, modeladas no sistema multi-agente, definem os contornos de qualquer ação criminosa.

4.4 COMPARAÇÃO DA SIMULAÇÃO MULTI-AGENTE COM TÉCNICAS DE SIMULAÇÃO TRADICIONAIS

4.4.1 Simulação Orientada a Objetos

Com relação à modelagem de uma simulação baseada em agentes, Macal e North (2005) fazem a ressalva de que, embora o paradigma de programação orientado a objetos constitua-se em uma base útil para a modelagem de agentes inteligentes, uma simulação baseada em multi-agentes não é o mesmo que uma simulação orientada a objetos.

Davidsson (2000) afirma que, embora não exista uma clara distinção entre uma simulação multi-agente e uma simulação orientada a objetos, em uma simulação orientada a objetos (em contraste com a simulação multi-agente) as entidades simuladas são “tipicamente puramente reativas, não utilizam qualquer linguagem de comunicação, são estacionárias, são estáticas e não são modeladas através de modelos mentais.”

4.4.2 Simulação de Eventos Discretos

Na simulação de eventos discretos, um sistema é especificado em termos de estados e de eventos. Durante a execução de uma simulação de eventos discretos, a ocorrência de eventos afetará o estado corrente do sistema. Uma simulação contínua, onde eventos ocorrem continuamente com o passar do tempo, pode ser facilmente convertida para uma simulação discreta (considerando-se apenas o instante inicial e final de cada evento). Por essa razão, a comparação com a simulação multi-agente será feita apenas com a simulação de eventos discretos (DAVIDSSON, 2000).

A simulação de eventos discretos, em determinados casos, apresenta vantagens sobre a simulação multi-agente. Tais vantagens são: a simulação multi-agente consome

mais recursos computacionais e de comunicação do que a simulação de eventos discretos; ao contrário da simulação multi-agente, a simulação de eventos discretos é mais adequada para simulações baseadas em eventos (quando o tempo simulado avança de acordo com a ocorrência de eventos, ao invés de avançar em passos sucessivos e constantes (DAVIDSSON, 2000).

No entanto, ainda conforme o autor acima, a simulação multi-agente apresenta notáveis vantagens sobre a simulação de eventos discretos, tais como:

a) permitir a modelagem e a implementação de comportamento pró-ativo, o qual é importante quando deseja-se simular seres humanos ou animais que sejam capazes de agir com iniciativa (sem estímulos externos) – ou seja, a simulação multi-agente é uma opção mais natural para a implementação de humanos como agentes;

b) suportar a computação distribuída de maneira natural, uma vez que os agentes podem ser implementados logicamente para, de maneira individual, corresponderem à um processo computacional, ou à uma thread (uma thread é um “subprocesso”, relativamente independente, que opera no “interior” – no mesmo espaço de endereçamento lógico - de um processo computacional, podendo operar sozinha ou na presença de outras threads. Threads também são chamadas de “processos leves” (SILBERSCHATZ e GALVIN, 2005)), capaz de operar em diferentes equipamentos (p.ex.: computadores, celulares). Adicionalmente, a computação distribuída também permite um melhor desempenho e uma maior escalabilidade (possibilidade de inclusão de muitos agentes na simulação);

c) como cada agente é implementado na forma de um processo computacional distinto e tem a capacidade de comunicar-se com os demais agentes através de uma linguagem (um protocolo de comunicação) único, também é possível remover ou adicionar agentes durante a execução da simulação sem que a mesma precise ser interrompida ou que a estrutura que mapeia a correspondência da simulação com a realidade seja afetada. Tal característica também abre a possibilidade de que agentes inteligentes possam ser

substituídos por seres humanos durante a execução da simulação – permitindo o surgimento de cenários simulados extremamente dinâmicos;

d) finalmente, é possível que modelos de simulação multi-agente sejam especificados - através de aplicativos especializados – em um alto nível (com relação à proximidade com a linguagem humana, em contraste com o baixo nível representado pelos códigos de programação executados pelos computadores), permitindo que não-programadores de computador possam participar do processo de desenvolvimento de um simulador.

4.4.3 Microsimulação Dinâmica

Segundo Davidsson (2000), “a Microsimulação Dinâmica tem por objetivo simular o efeito da passagem do tempo em indivíduos. Dados de uma grande amostra aleatória de alguma população é usado para caracterizar, inicialmente, os indivíduos simulados. Algumas características da amostra podem ser, por exemplo, idade, sexo e situação de emprego. Um conjunto de probabilidades de transição é usado para simular como essas características evoluem durante um determinado período.”

As duas principais desvantagens desta técnica de simulação diante da simulação multi-agente são, primeiro, as limitações impostas pela modelagem dos indivíduos em termos de probabilidades, sem considerar as preferências individuais, as decisões, os planos e outras características únicas. Ou seja, desconsidera-se a heterogeneidade presente em uma sociedade composta por agentes independentes. Em segundo lugar, cada indivíduo simulado é considerado individualmente (ainda que não possua características únicas), sem levar-se em conta as interações que deveriam existir entre os mesmos.

5 MÉTODO DA PESQUISA

Este trabalho é de natureza exploratória, pois busca a melhor compreensão de um problema através da modelagem de um sistema real (TRIPODI et al., 1975). Para a sua elaboração, está sendo utilizada uma abordagem multimétodo (MINGERS, 2001), pois são combinados métodos qualitativos (no levantamento de informações para a modelagem) e quantitativos (para construção da simulação computacional).

A primeira etapa deste trabalho consiste em uma pesquisa bibliográfica e documental. Segundo Barbosa (2001), na pesquisa bibliográfica efetua-se o exame de referências teóricas publicadas em artigos científicos ou livros, para levantamento do que já foi produzido sobre determinado assunto. A pesquisa documental, segundo Fachin (2001), consiste "na coleta, classificação, seleção difusa e na utilização de toda espécie de informações, compreendendo também as técnicas e métodos que facilitam a sua busca e identificação."

A segunda etapa deste trabalho foi efetuada em laboratório, durante a configuração da simulação computacional do modelo desenvolvido. O trabalho em laboratório permite a manipulação de variáveis independentes e a observação de suas relações (FACHIN, 2001).

5.1 ETAPAS DO DESENVOLVIMENTO

A seguir, será apresentada a seqüência de etapas utilizada para o desenvolvimento deste trabalho.

a) caracterização do projeto a ser simulado;
b) definição do Modelo de Simulação Multi-Agente;
c) escolha do Ambiente de Desenvolvimento para Simulação Multi-Agente;

d) configuração da Ferramenta de Simulação Multi-Agente;
e) testes do modelo de simulação, utilizando a ferramenta acima;
f) validação do modelo desenvolvido, através da aplicação da Ferramenta de Simulação Multi-Agente sobre um determinado cenário.

Neste trabalho, com relação à Definição do Modelo de Simulação Multi-Agente, foi seguida a modelagem de um Sistema Multi-Agente proposta por Macal e North (2005):

a) a identificação dos agentes e a escolha de uma teoria comportamental para os agentes;
b) a identificação dos relacionamentos entre os agentes e a escolha de uma teoria para a interação entre os agentes – os relacionamentos entre os agentes baseiam-se nas premissas do processo de desenvolvimento de *software* em um ambiente de programação extrema. A programação extrema foi escolhida para a modelagem apresentada neste trabalho pois, além de ser voltada para equipes pequenas de desenvolvedores (p.ex.: 10 desenvolvedores), ela especifica claramente tanto os processos quanto as características do ambiente de desenvolvimento;
c) a escolha de uma plataforma de desenvolvimento (uma ferramenta) e de uma estratégia de modelagem;
d) a obtenção dos dados necessários para a modelagem dos agentes;
e) a validação dos modelos de comportamento dos agentes (além da validação do modelo como um todo);
f) a execução do modelo e a análise dos resultados.

5.2 MODELAGEM DE PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Um dos fatores mais importantes no processo de desenvolvimento dos modelos multi-agente é a definição de seus modelos internos, pois esses modelos internos definirão o comportamento dos agentes inteligentes diante do ambiente e de outros agentes (STREIT, 2006).

Neste trabalho, foi adotada uma arquitetura deliberativa para os agentes inteligentes envolvidos. Provavelmente, a arquitetura deliberativa mais conhecida é a Beliefs-Desires-Intentions (BDI) (ou Crenças, Desejos e Intenções) (LÓPEZ, 2004).

Segundo López (2004):

> Um modelo de agente BDI baseia-se na teoria do raciocínio prático, que afirma que o comportamento de um agente é impulsionado por seus objetivos. Esses agentes são inteiramente definidos através de três atitudes mentais, descritas a seguir. Crenças são a representação que o agente possui de seu mundo, Desejos (ou Objetivos) são os estados que o agente deseja obter e as Intenções representam os meios que o agente dispõe para atingir seus objetivos.

As Crenças são constantemente revisadas para detectar-se mudanças no ambiente e, logo, para tornar o processo decisório mais preciso. O processo de raciocínio prático divide-se em dois subprocessos: deliberação (o que se quer fazer) e raciocínio meios-fins (como fazer).

Na figura abaixo pode ser observada uma ilustração genérica da arquitetura BDI. As elipses representam as atitudes mentais dos agentes (crenças, desejos e intenções), enquanto as caixas representam os processos de decisão. As linhas tracejadas indicam o fluxo de informações, enquanto as linhas contínuas representam o fluxo de controle.

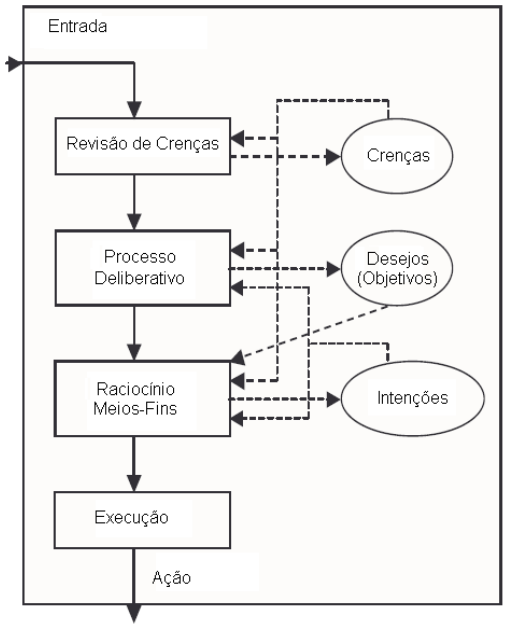


Figura 9 - Um modelo de agente inteligente com arquitetura BDI
Fonte: López (2004).

Em primeiro lugar, deve-se procurar incorporar ao modelo de simulação multi-agente os aspectos relevantes do sistema que estamos estudando. O modelo de simulação corresponde à representação formal de um modelo conceitual. O modelo conceitual, por sua vez, será uma abstração do mundo real (STREIT, 2006).

Na Figura 10, a seguir, são apresentados os três níveis do Processo de Modelagem para uma simulação multi-agente (STREIT, 2006), que foram adotados para o processo de modelagem neste trabalho.

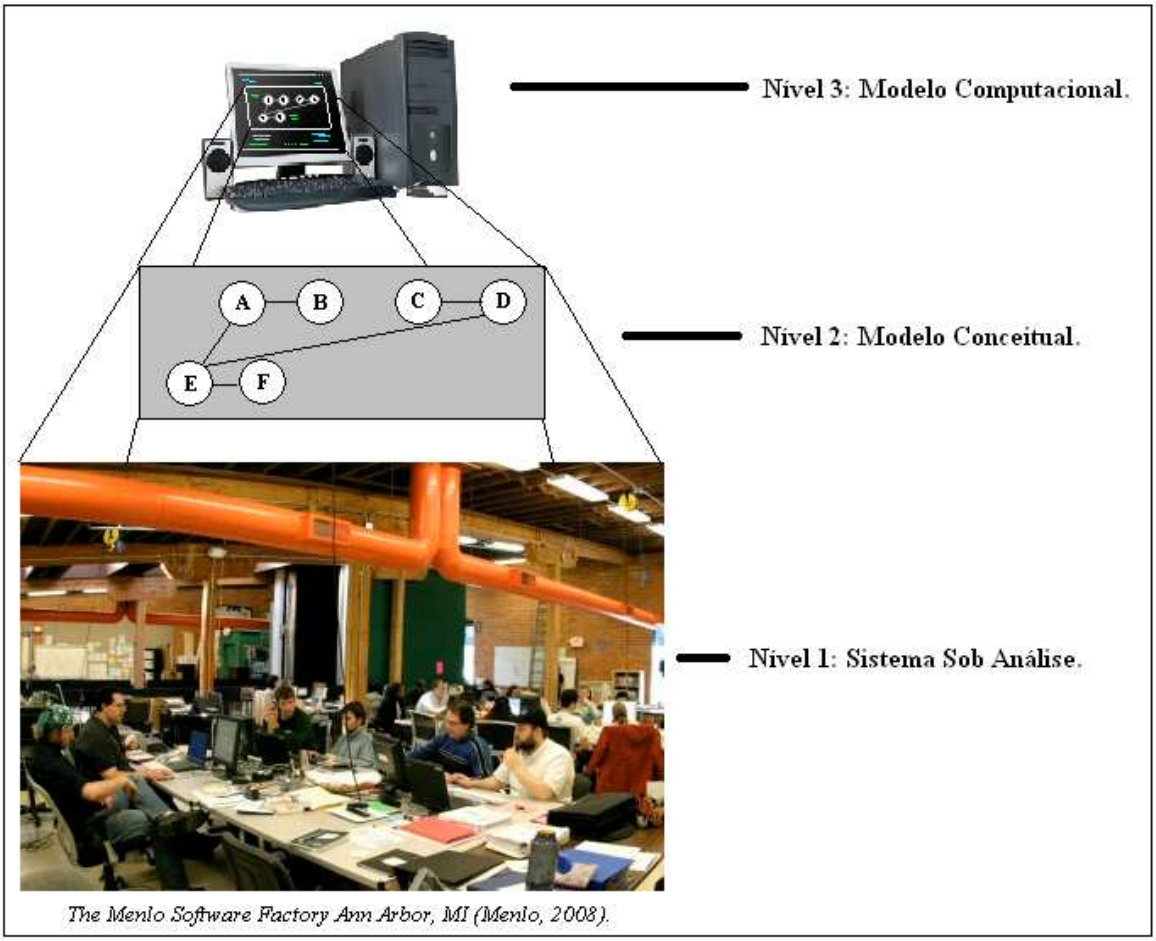


Figura 10 - O Processo de Modelagem
Fonte: adaptado de Streit (2006).

5.3 FERRAMENTAS COMPUTACIONAIS PARA SIMULAÇÃO MULTI-AGENTE

Existem diversas ferramentas específicas para a simulação multi-agente. Várias delas são citadas na Tabela 1.

Tabela 1 - Ferramentas para a Elaboração de Simulações Multi-Agente

Ferramenta	Desenvolvedor
AgentSheets	AgentSheets, Inc. (2009)
BREVE	Klein (2002), Spiderland (2009)
CORMAS	Cormas (2006)
JADE	Jade (2001), Bellifemine (2003)
MADKIT	MadKit (2007)
MASON	Mason (2003)
NetLogo	Wilensky (1999)
RePast	RePast (2007)
SeSAm	Klügl (2006), SeSAm (2009)
SIM_AGENT	Sloman (2005)
SimCog	SimCog (2001)
SIMULA	Birtwistle (1973), Simula (2009)
StarLogo	StarLogo (2000)
SWARM	Swarm (1999)

Fonte: tabela elaborada pelo autor.

As ferramentas citadas acima foram avaliadas de acordo com as necessidades do sistema multi-agente a ser desenvolvido. Em uma análise final, a ferramenta adotada para a configuração do Modelo Computacional da simulação multi-agente foi a SeSAm - *Shell for Simulated Agent Systems*.

5.3.1 SeSAm - Shell for Simulated Agent Systems

SeSAm (KLÜGL, 2006), como mencionado na seção anterior, significa “Shell for Simulated Agent Systems”, ou Interface para Simulação de Sistemas de Agentes. O *software* SeSAm está disponível para download gratuito, na internet, sujeito à licença LGPL (GNU.ORG, 2007). Esse *software* oferece um ambiente genérico para a modelagem

e a experimentação com simulação baseada em agentes. O objetivo de seus criadores é o de oferecer uma ferramenta que facilite a construção de modelos complexos.

No SeSAm, os agentes consistem em um corpo que contém um conjunto de variáveis de estado e um comportamento, implementado na forma de diagramas de atividades semelhantes à diagramas UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) (ESMIN, 1999).

A ferramenta também permite a construção de funções personalizadas, atributos (conjuntos de variáveis e funções) e a configuração de tipos de dados definidos pelo usuário. Tal capacidade possibilita a construção de blocos de níveis mais elevados.

O SeSAm é implementado visualmente na forma de descrições semi-declarativas, que são executadas no mesmo ambiente. A dinâmica da simulação pode ser observada na forma de uma animação ou na forma configurada pelo usuário, com curvas de análise que são atualizadas automaticamente.

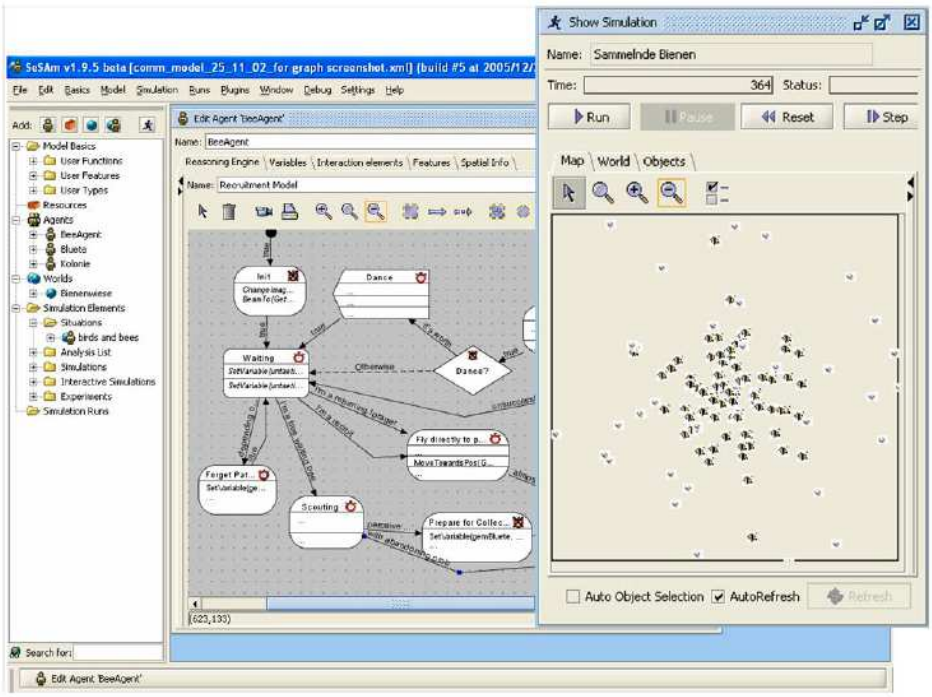


Figura 11 - O *Software* SeSAm - Grafo de Atividades e Janela de Animação

Fonte: Klügl (2006).

6 CONSTRUÇÃO DO MODELO DE SIMULAÇÃO

Para a construção do modelo de simulação do processo de desenvolvimento de *software* em um ambiente de Programação Extrema (BECK, 2000), este trabalho seguiu o Processo de Modelagem proposto por Streit (2006), apresentado na Figura 10 (Capítulo 5).

6.1 O SISTEMA SOB ANÁLISE

Neste trabalho, foi considerado como Sistema sob Análise (STREIT, 2006) o sistema composto pelo **ambiente**, pelas **pessoas** e **pelo processo de desenvolvimento em Programação Extrema**. (Figura 12).

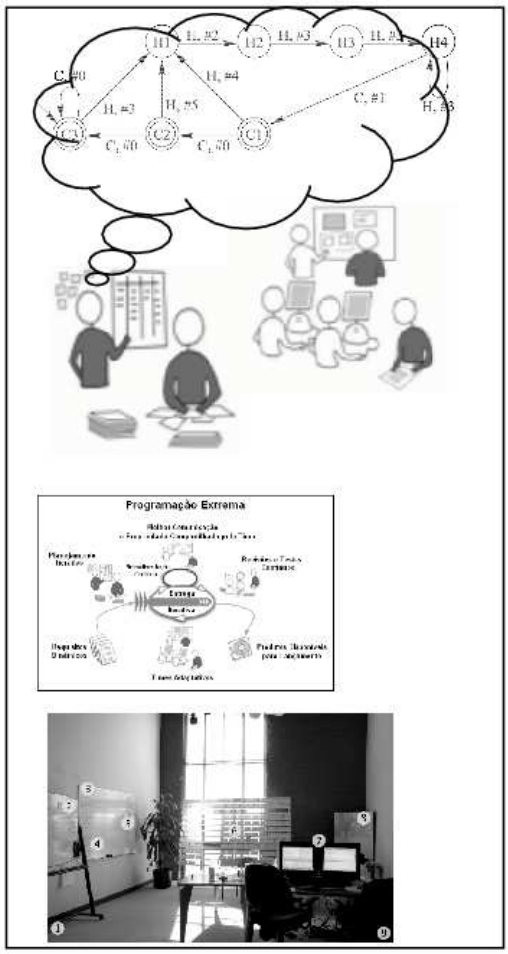


Figura 12 - O Sistema sob Análise
Fonte: figura elaborada pelo autor.

O ambiente foi analisado de acordo com as necessidades do método da Programação Extrema (AGILE WORKSHOPS, 2009; SCISSOR.COM, 2009; BECK, 2000).

O processo de desenvolvimento da Programação Extrema (Capítulo 2, Figura 3) foi traduzido em um fluxograma (Capítulo 2, Figura 7) de acordo com Beck (2000).

Finalmente, as pessoas foram analisadas relativamente aos seus papéis no processo de desenvolvimento da Programação Extrema (BECK, 2000), principalmente no aspecto da programação em pares.

6.2 O MODELO CONCEITUAL

Após a análise do sistema, foi elaborado o Modelo Conceitual do mesmo. O Modelo Conceitual, ilustrado na Figura 13, representa uma abstração do mundo real (STREIT, 2006). Essa abstração, no presente trabalho, foi elaborada na forma dos quatro níveis mostrados na Figura 13 - dois níveis físicos (Laboratório e Agentes) e dois níveis abstratos (Processo de Desenvolvimento e Estados Internos e Variáveis).

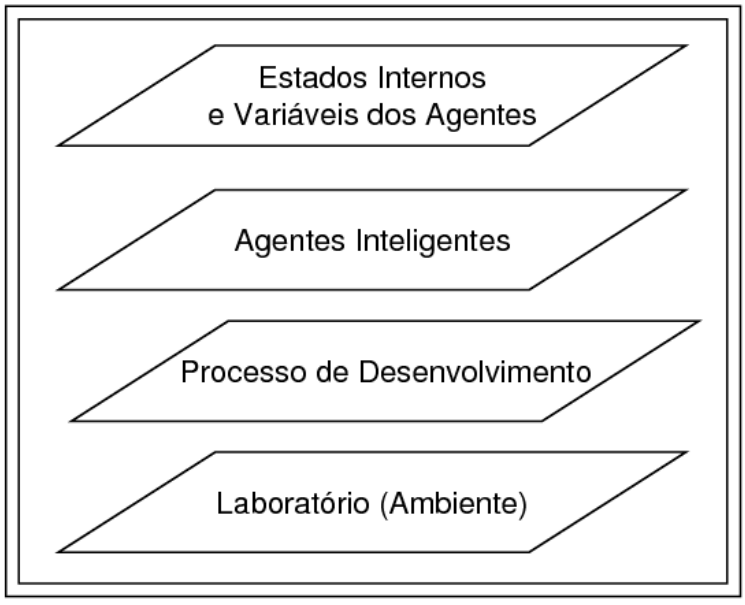


Figura 13 - O Modelo Conceitual
Fonte: figura elaborada pelo autor.

O ordenamento dos níveis, na Figura 13, não é casual. Esse ordenamento, conforme a abordagem deste trabalho, indica que o nível mais básico é o do Laboratório (nível inferior, físico), projetado para que o próximo nível (do Processo de Desenvolvimento, abstrato) seja desempenhado da melhor forma possível (BECK, 2000). No contexto desses dois primeiros níveis, o próximo nível (físico) é o dos Agentes Inteligentes (WOOLDRIDGE e JENNINGS, 1995; WEISS, 1999), os quais serão inseridos no ambiente (nível do Laboratório) e deverão agir conforme o Processo de Desenvolvimento (BECK, 2000). Finalmente, o nível mais alto é o dos Estados Internos e Variáveis dos Agentes, que corresponde ao comportamento inteligente dessas entidades. Tal comportamento inteligente permitirá a execução das tarefas designadas no modelo de simulação.

6.3 O MODELO COMPUTACIONAL

No SeSAm, cada agente possui seu diagrama de atividades, que corresponde ao seu "motor de raciocínio". Os nodos desse diagrama correspondem às atividades que o agente poderá desempenhar (apenas uma de cada vez), enquanto os arcos orientados correspondem às regras de passagem de uma atividade (nodo) para outra. Cada regra de saída de uma atividade (um arco que tem sua origem no nodo) é constantemente verificada (em sequencia, se há mais de uma regra) pelo simulador. No momento em que a resposta à condição de uma regra for verdadeira (a condição retorna um valor-verdade lógico *true*), o agente passa da atividade (ou estado) atual para a próxima atividade (estado), apontada pela regra (arco orientado).

As atividades, por sua vez, são roteiros simples que são iniciados e finalizados baseados em regras. As ações e condições, tanto dos roteiros quanto das regras, podem ser compostas por um extenso número de blocos básicos de primitivas. O SeSAm oferece um grande número de primitivas. Alguns exemplos dos tipos de primitivas oferecidas são: primitivas lógicas (por exemplo: `And`, `Or`, `Not`), de controle de fluxo de execução (por

exemplo: `If Then Else`; `While`), probabilísticas (por exemplo: `RandomBoolean`; `RandomNormal`), de interação com o ambiente (por exemplo: `Distance`; `PosGetDirection`) e aritméticas (por exemplo: +; -).

Embora o modelo computacional deste trabalho, configurado no SeSAm, possua quatro agentes distintos (conceitualmente, o ambiente também pode ser considerado um agente, mas sua presença pode ser abstraída, neste trabalho, sem prejuízo para o modelo desenvolvido), nosso foco principal é o Agente Programador. Isso decorre do fato de que, enquanto os demais agentes do modelo (Histórias, Computadores e Servidor de Integração) são considerados recursos materiais, o Agente Programador (em suas várias instâncias possíveis) é o único ator do sistema, ou seja, o único tipo de agente que desempenha um papel ativo no ambiente simulado.

Cabe destacar que, ao contrário do Agente Programador, os demais agentes não possuem atividades especificadas em seu diagrama, pois não necessitam das mesmas no modelo de simulação desenvolvido. Assim, estes últimos agentes possuem apenas variáveis e constantes, que serão apresentadas na subseção 6.4.2.1.

Sob esse aspecto, o modelo do Agente Programador confunde-se com o próprio modelo de simulação desenvolvido, uma vez que a interação das instâncias desse agente umas com as outras, com o ambiente e com os recursos materiais disponíveis é exatamente o que anima o sistema. No entanto, deve-se manter em mente que o modelo do Agente Programador é apenas uma parte - ainda que seja essencial - do modelo de simulação proposto neste trabalho.

O diagrama de atividades do Agente Programador procura representar uma parcela significativa das atividades que um determinado programador exerce em um processo de desenvolvimento de *software* em ambiente de Programação Extrema, levando-se em consideração as Premissas do Modelo (subseção 6.4.1). Esse diagrama é replicado em cada um dos agentes programadores instanciados em uma determinada situação de simulação. Por exemplo, na situação de simulação ilustrada na Figura 15, temos seis instâncias do Agente Programador. Todas as instâncias do Agente Programador iniciam sua

movimentação, efetivamente, a partir do estado funcional `Caminhando sem História` (A).

Embora todos os agentes instanciados possuam a mesma estrutura de atividades (representada pelo diagrama de atividades), funções que retornam valores aleatórios, inseridas em diversas atividades e regras do modelo, impedem que todos os agentes programadores comportem-se de maneira síncrona durante a simulação. Se essa sincronia ocorresse, os resultados seriam sempre iguais em diferentes execuções da simulação. Um caso simples da criação de um valor aleatório é dado pela função `RandomBoolean(x)`, do SeSAm, que retorna um valor-verdade *true* (verdadeiro) com uma probabilidade dada pelo parâmetro `x` fornecido pelo projetista do sistema. Assim, por exemplo, se usarmos apenas a função `RandomBoolean(0.3)`em uma regra, o agente somente passará de um estado funcional para o próximo em 30% das vezes que a regra for testada pelo simulador.

Todos os Estados Funcionais e Regras (que formam o diagrama de atividades) do modelo do Agente Programador estão ilustradas nas Figuras 14, 20, 21 e 22. Os Estados Funcionais (atividades, representadas pelos nodos do diagrama) são identificados por letras (de A a Z, além de AA e AB) e as Regras foram identificadas através de números, em sequencia arbitrária, de 1 a 51.

Os detalhes lógicos e matemáticos da modelagem do Agente Programador são apresentados no Apêndice A (Especificação Lógica do Modelo - Agente Programador) - **recomenda-se, antes da consulta ao referido Apêndice, a leitura das subseções 6.4.1, 6.4.2, 6.4.3 e 6.4.4** que apresentam, respectivamente, as Premissas do Modelo, seus Agentes, a explicação sobre os Estados Funcionais e as Regras do modelo e, finalmente, uma breve explicação sobre como ler a Especificação Lógica do Modelo do Agente Programador.

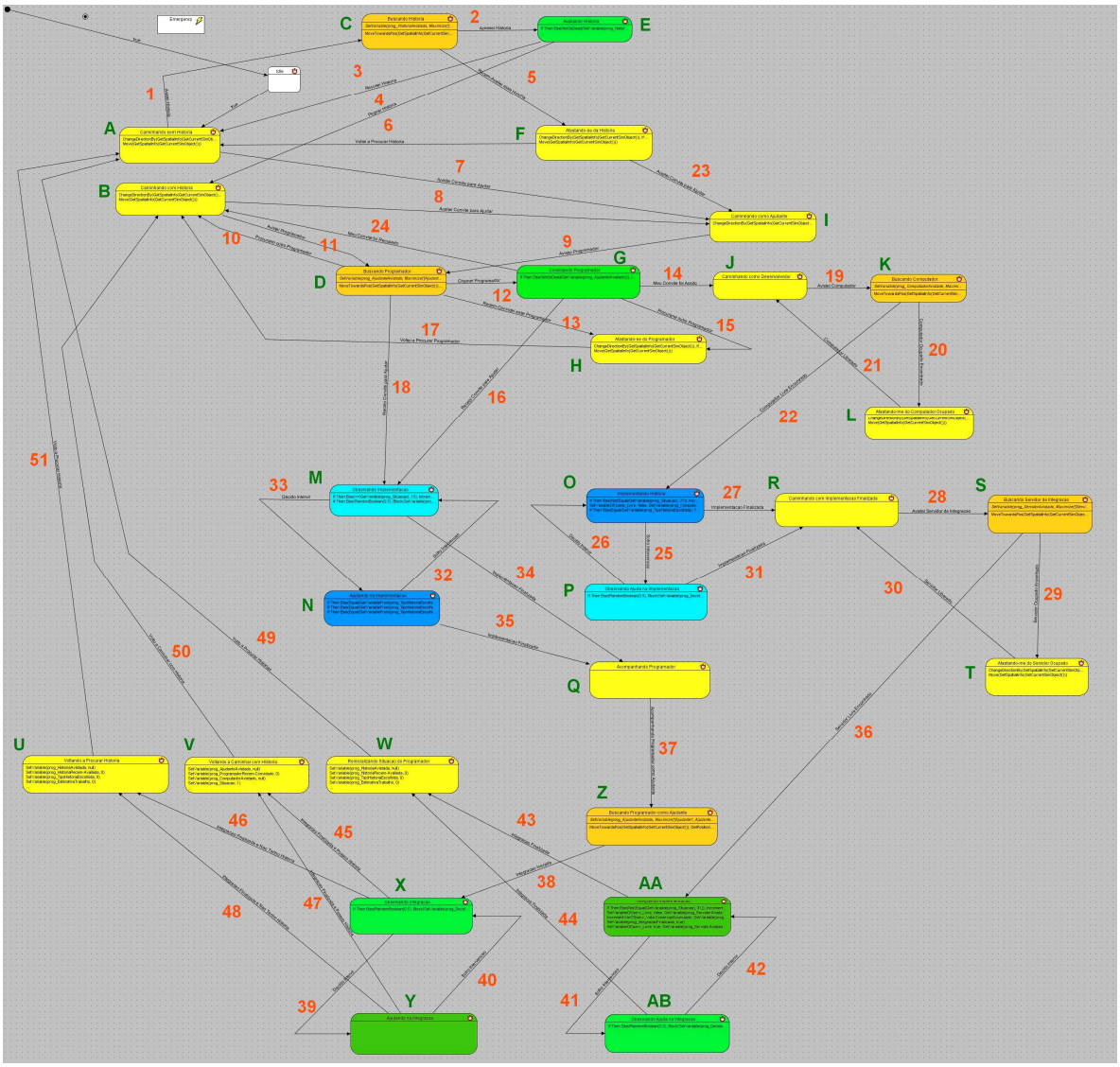


Figura 14 - O Modelo Computacional do Agente Programador
Fonte: figura elaborada pelo autor.

Figura 15 - Ambiente de Simulação Estabelecido no SeSAm
Fonte: figura elaborada pelo autor.

6.4 O PROCESSO DE MODELAGEM NESTE TRABALHO

A Figura 16, a seguir, apresenta o processo de modelagem elaborado, neste trabalho, para a simulação multi-agente do processo de desenvolvimento de software em ambiente de programação extrema.

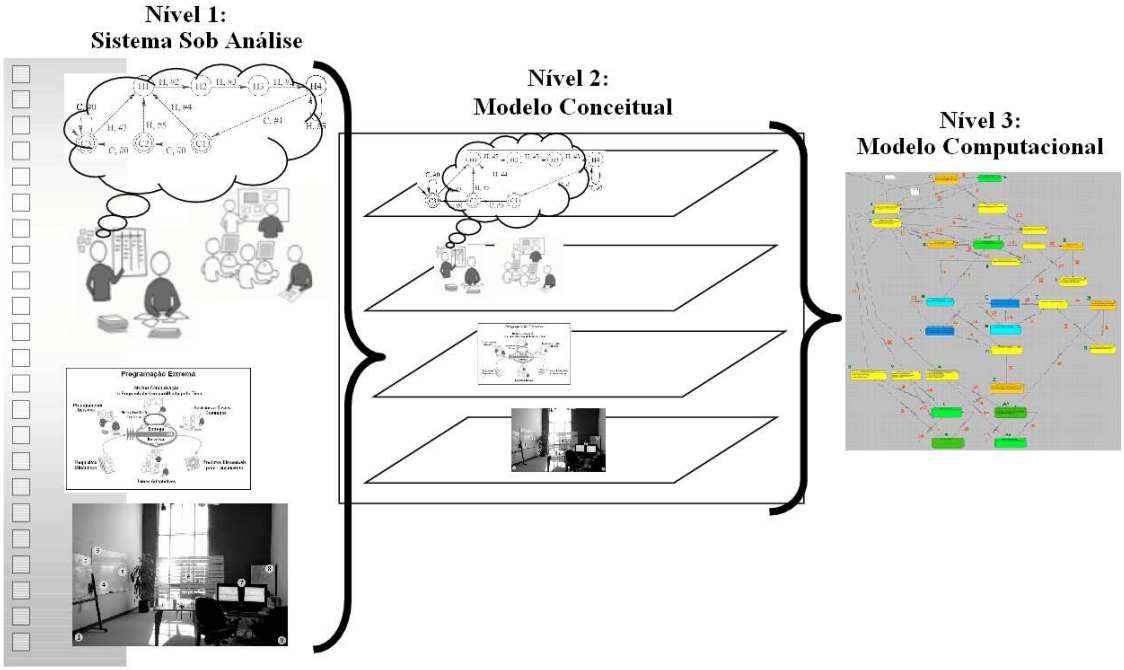


Figura 16 - O Processo de Modelagem neste Trabalho
Fonte: figura elaborada pelo autor.

6.4.1 Premissas do Modelo

a) não há cancelamentos de implementação ou de integração;
b) não é tratado, no Modelo de Simulação, o conceito de múltiplas iterações da Programação Extrema. O Modelo de Simulação foi construído para simular apenas uma iteração de desenvolvimento;
c) cada programador escolhe uma História de cada vez, para implementar;
d) não há trocas de pares pré-programadas;
e) não há pares de programadores predefinidos;
f) não há alocação de recursos (Histórias, Computadores ou Servidor) predefinidos;

g) a experiência de cada programador afeta a escolha da História e a velocidade de implementação da mesma;
h) um maior valor comercial aumenta a prioridade de implementação de uma História;
i) o Agente Programador não possui equações que simulem emoções ou quaisquer outros fatores psicológicos;
j) não foram modelados fatores de influência sobre o relacionamento entre componentes de um par de programadores durante sua cooperação – ou mesmo entre os agentes programadores em geral. Esta premissa baseia-se em Chao e Atli (2006) que, em sua pesquisa, não conseguiram demonstrar, de maneira estatisticamente significante, que determinados perfis de personalidade influenciam na qualidade do resultado final da programação. De acordo com Chong e Hurlbutt (2007), existe uma limitada influência da experiência profissional de cada componente do par sobre o trabalho da dupla, mas essa influência não foi considerada no Modelo do Agente Programador elaborado neste trabalho;
k) todas as probabilidades utilizadas no Modelo do Agente Programador são arbitrárias;
l) “piloto” e “co-piloto” são denominações criadas pelo autor deste trabalho para designar, respectivamente, um programador que já conseguiu a ajuda de outro programador para implementar a sua História escolhida e um programador que está auxiliando o Piloto (ambos formam o par de programadores da Programação Extrema);
m) as situações (não confundir com os Estados Funcionais, que são, por sua vez, os estados durante os quais os agentes desempenham suas atividades) possíveis para cada Agente Programador, dessa forma, são as seguintes:

- `prog_Situacao = 0 (Programador sem História)`
- `prog_Situacao = 1 (Programador com História)`
- `prog_Situacao = 5 ou 15 (Co-piloto sem História)`
- `prog_Situacao = 6 ou 16 (Co-piloto com História)`
- `prog_Situacao = 11 (Piloto (sempre com História)`
- `prog_Situacao = 21 (Piloto implementando História)`
- `prog_Situacao = 31 (Piloto integrando História)`

n) o modelo do Agente Programador permite que esse agente acumule experiência em, no máximo, quatro tipos de Histórias;
o) os tipos escolhidos para as Histórias são arbitrários, limitados a quatro tipos distintos e apenas influenciam as escolhas dos agentes pelas Histórias e as experiências acumuladas pela implementação destas;
p) a experiência é acumulada somente quanto o agente está implementando ou ajudando na implementação ou integrando ou ajudanto na integração – nunca durante os estados de observação da implementação ou da integração;
q) não foram modeladas interações de caráter social entre os agentes (Piloto e Co-piloto, que formam o par da Programação Extrema) durante a implementação da História ou durante a integração dessa implementação. A única influência mútua é a possibilidade de um agente intervir sobre o trabalho do outro, baseada em uma condição puramente probabilística;
r) a experiência do Agente Programador é calculada apenas para o agente que finaliza a tarefa, ainda que ambos tenham participado da implementação da História;
s) de acordo com o Processo de Desenvolvimento de *Software* da Programação Extrema, deve existir pelo menos um Computador para cada par de Programadores. Também deve existir no ambiente, no mínimo, um Servidor de Integração (computador onde as Histórias implementadas são integradas para formar o produto final);
t) a integração das Histórias implementadas consiste apenas na soma do valor comercial de cada uma das Histórias, efetuada no momento em que o Agente Programador (Piloto) encontra-se ocupando o Servidor de Integração. Assim, não foram modelados quaisquer aspectos relacionados com a integração prática de um sistema computacional (como, por exemplo: precedência de tarefas; dificuldades de integração; surgimento de erros; testes de integração);
u) alguns aspectos específicos, previstos na Programação Extrema, não foram modelados, tais como, por exemplo: testes (durante a implementação e a integração) e refatoração de código fonte;
v) os termos “Co-Piloto” e “Ajudante”, no âmbito deste trabalho, são intercambiáveis.

w) não existem relações de precedência entre as Histórias, isto é, as mesmas podem ser implementadas em qualquer ordem;
x) embora o processo de desenvolvimento da Programação Extrema preveja a interação entre executivos (a equipe de negócios) e programadores (a equipe de desenvolvimento), tal interação não foi modelada;
y) não foi modelada a influência do cliente sobre o processo de desenvolvimento;
z) o modelo corresponde a uma equipe de desenvolvimento trabalhando em apenas um projeto;
aa) não foi modelada uma penalidade para o negócio com relação à Histórias não implementadas ou implementadas com atraso;
bb) um “tick” do simulador equivale a um minuto de tempo simulado;
cc) o Valor Acumulado para o Negócio (`serv_ValorComercialAcumulado`) representa uma quantidade abstrata, que pode ser tanto um valor monetário quanto um valor qualquer percebido. A quantidade máxima do Valor Acumulado para o Negócio é de 3000 unidades, que foi distribuída entre: três histórias de valor 100; três histórias de valor 200; três histórias de valor 300; e, finalmente, três histórias de valor 400.

6.4.2 Agentes do Modelo

Todos os nomes de Variáveis, Estados Funcionais e Regras foram propositalmente criados sem acentuação ou “ç”, para evitar problemas durante a execução computacional da simulação.

6.4.2.1 Agentes Computador, Servidor de Integração e História (Recursos)

Variáveis	
Nome (visibilidade*, tipo)	**Descrição**
`comp_Livre (externa, lógica)`	Sinaliza se o Computador está livre para uso.
`serv_Livre (externa, lógica)`	Sinaliza se o Servidor de Integração está livre para uso.
`serv_ValorComercialAcumulado (externa, real)`	Valor Comercial total das Histórias implementadas e integradas no Servidor.
Constantes	
Nome	**Descrição**
`hist_Tipo (externa, real)`	Tipo da História (no máximo quatro tipos).
`hist_ValorComercial (externa, real)`	Valor comercial da História.
`hist_Identificacao (externa, real)`	Identificação da História.

*A visibilidade refere-se à capacidade de outros agentes consultarem o valor da variável do agente em questão. Variáveis "externas" podem ser consultadas.

Figura 17 - Variáveis dos Agentes Recursos

Fonte: figura elaborada pelo autor.

6.4.2.2 Agente Programador (Ator)

Variáveis	
Nome (visibilidade*, tipo)	**Descrição**
`prog_HistoriaAvistada (interna, objeto)`	História avistada pelo agente.
`prog_HistoriaRecemAvaliada (interna, objeto)`	História recém-avaliada pelo agente.
`prog_TipoHistoriaEscolhida (externa, real)`	Tipo da história escolhida pelo agente.
`prog_EstimativaTrabalho (externa, real)`	Estimativa de trabalho calculada pelo agente.
`prog_AjudanteAvistado (interna, objeto)`	Programador avistado pelo agente.
`prog_ProgramadorRecem-Convidado (interna, real)`	Identificação do programador recém-convidado.
`prog_ComputadorAvistado`	Computador avistado pelo agente.

(interna, objeto)	
prog_ValorComercialHistoriaEscolhida (externa, real)	Valor comercial da história escolhida pelo agente.
prog_Situacao (externa, real)	Código da situação atual do agente. 0: Agente sem História. 1: Agente com História. 5 ou 15: Agente co-piloto (ajudante) sem História. 6 ou 16: Agente co-piloto (ajudante) com História. 11: Agente piloto (implementador) com História. 21: Agente piloto executando implementação. 31: Agente piloto executando integração.
prog_ExpTipo1 (externa, real)	Valor da experiência em histórias do tipo 1.
prog_ExpTipo2 (externa, real)	Valor da experiência em histórias do tipo 2.
prog_ExpTipo3 (externa, real)	Valor da experiência em histórias do tipo 3.
prog_ExpTipo4 (externa, real)	Valor da experiência em histórias do tipo 4.
prog_Passo (interna, real)	Passos de implementação, conforme experiência.
prog_ServidorAvistado (interna, objeto)	Servidor de Integração avistado pelo agente.
prog_ImplementacaoFinalizada (externa, lógica)	Sinaliza finalização da implementação.
prog_IntegracaoFinalizada (externa, lógica)	Sinaliza finalização da integração.
prog_DecidoIntervir (externa, lógica)	Sinaliza intervenção do agente sobre outro.
prog_EstouSofrendoIntervencao (externa, lógica)	Sinaliza intervenção sofrida pelo agente.
prog_ExpTipo1_Anterior (externa, real)	Experiência anterior em histórias do tipo 1.
prog_ExpTipo2_Anterior (externa, real)	Experiência anterior em histórias do tipo 2.
prog_ExpTipo3_Anterior (externa, real)	Experiência anterior em histórias do tipo 3.
prog_ExpTipo4_Anterior (externa, real)	Experiência anterior em histórias do tipo 4.
prog_HistAcumuladasTipo1 (externa, real)	Quantidade de histórias do tipo 1 implementadas.
prog_HistAcumuladasTipo2	Quantidade de histórias do tipo 2

(externa, real)	implementadas.
prog_HistAcumuladasTipo3 (externa, real)	Quantidade de histórias do tipo 3 implementadas.
prog_HistAcumuladasTipo4 (externa, real)	Quantidade de histórias do tipo 4 implementadas.
prog_HistAcumuladasTotal (externa, real)	Quantidade total de histórias implementadas.
Constantes	
Nome	**Descrição**
prog_RaioPercepcao (interna, real)	Visibilidade, em pixels, do agente no ambiente.
prog_Identificacao (externa, real)	Identificação do agente.

*A visibilidade refere-se à capacidade de outros agentes consultarem o valor da variável do agente em questão. Variáveis "externas" podem ser consultadas.

Figura 18 - Variáveis do Agente Programador

Fonte: figura elaborada pelo autor.

6.4.3 Estados Funcionais e Regras

Os Estados Funcionais são situações em que o agente está executando um conjunto de uma ou mais ações – por isso, os nomes desses estados sempre utilizam verbos no gerúndio (p.ex.: caminhando, acompanhando).

Cada Regra é um conjunto com uma ou mais condições lógicas que devem ser completamente atendidas (retornarem um valor lógico *true*, ou "verdade") para que o agente passe de um Estado Funcional para outro Estado Funcional.

As Regras de Saída de um Estado Funcional são Regras que, se atendidas, permitem que o agente passe para um próximo Estado Funcional.

As Regras de Entrada, por sua vez, são Regras com origem em um ou mais Estados Funcionais anteriores ao atual – assim, as mesmas estão descritas como Regras de Saída em seus respectivos Estados Funcionais.

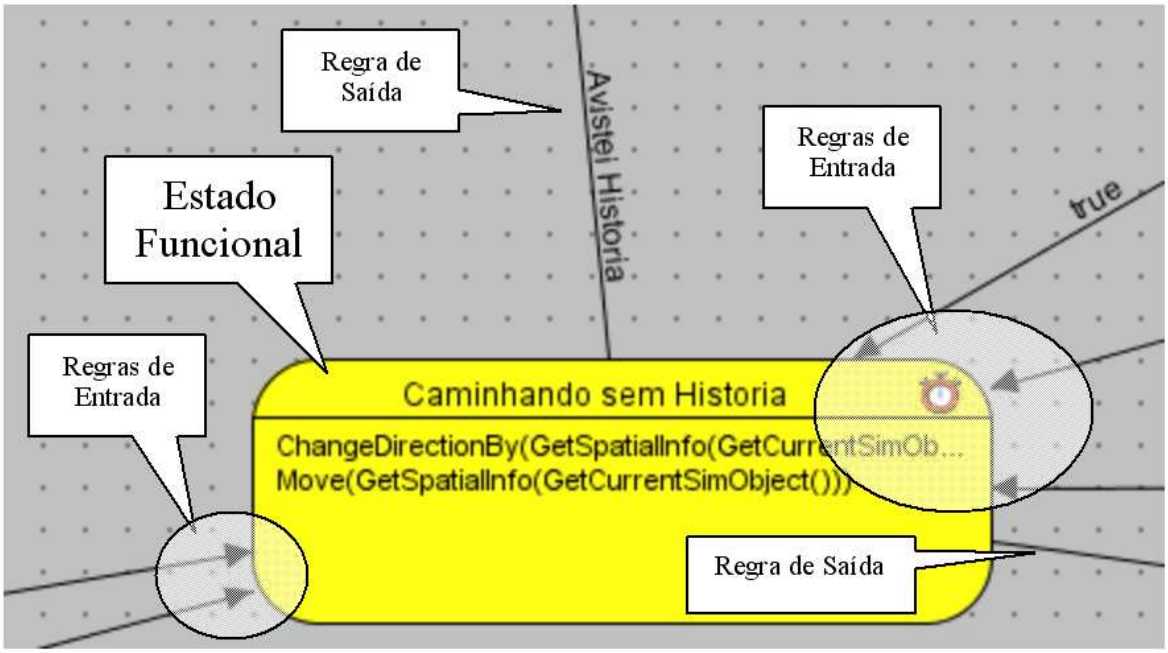


Figura 19 - Estado Funcional e suas Regras de Entrada e Saída
Fonte: figura elaborada pelo autor.

Conforme já mencionado na subseção 6.1.3, todos os Estados Funcionais e Regras do modelo computacional do Agente Programador receberam um código de identificação. Os Estados Funcionais foram identificados, em sequencia arbitrária, com as letras A a Z, além de AA e AB. As Regras foram identificadas através de números, em sequencia arbitrária, de 1 a 51. Nas páginas seguintes, será apresentado o diagrama do modelo computacional (configurado no *software* SeSAm) do Agente Programador. Para tentar facilitar a visualização desse modelo, o mesmo foi separado em três figuras - Figuras 18, 19 e 20.

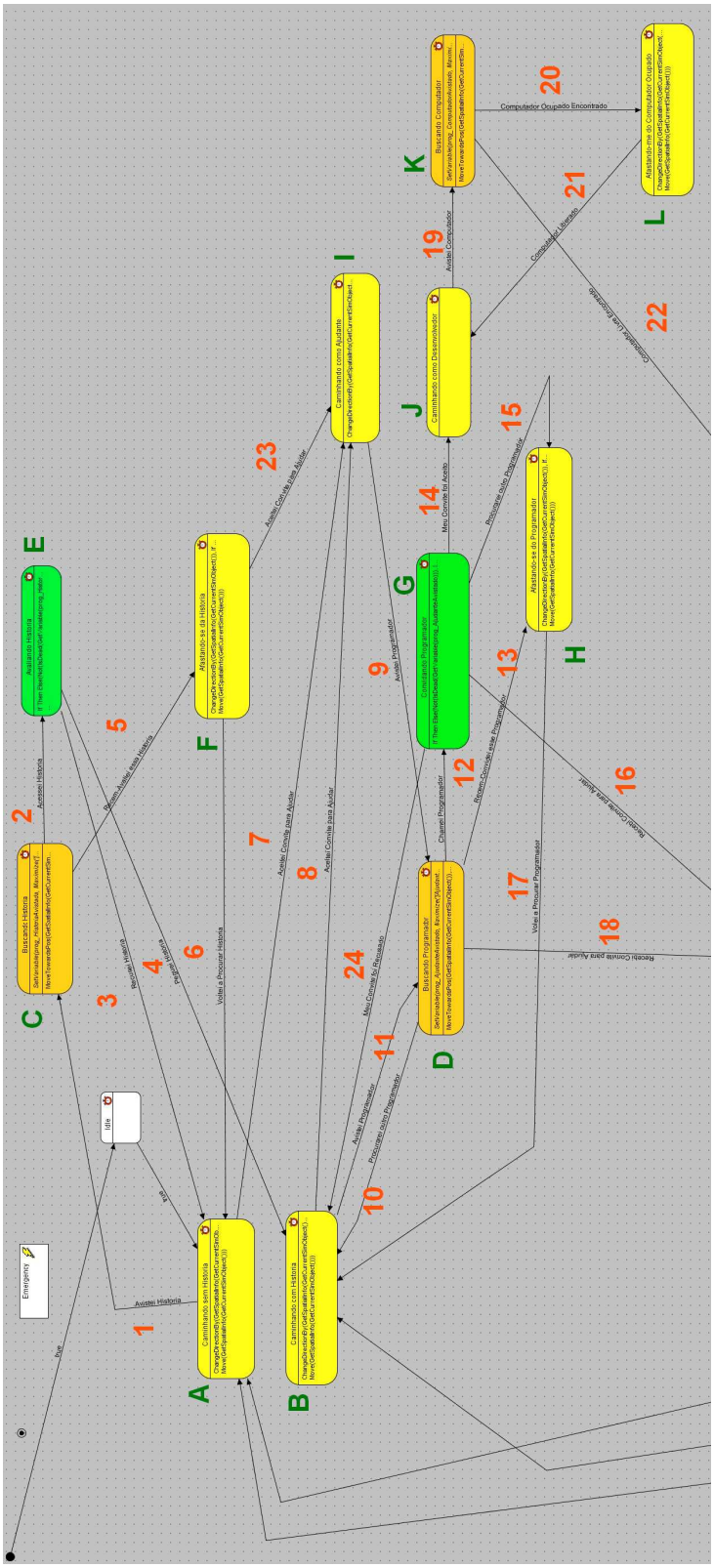


Figura 20 - O Modelo Computacional do Agente Programador (Parte 1 de 3)
Fonte: figura elaborada pelo autor.

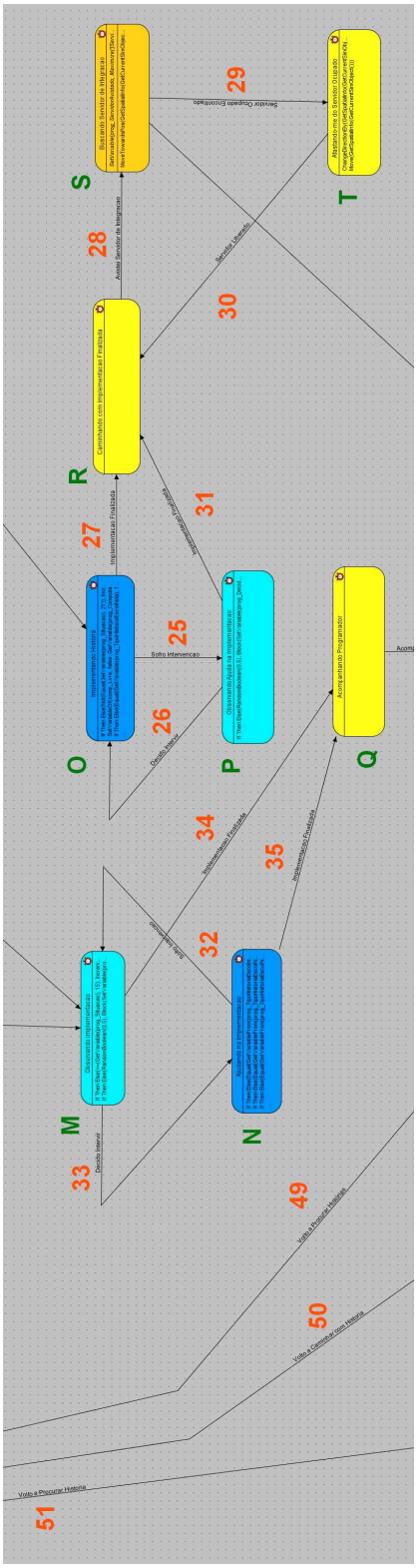


Figura 21 - O Modelo Computacional do Agente Programador (Parte 2 de 3)
Fonte: figura elaborada pelo autor.

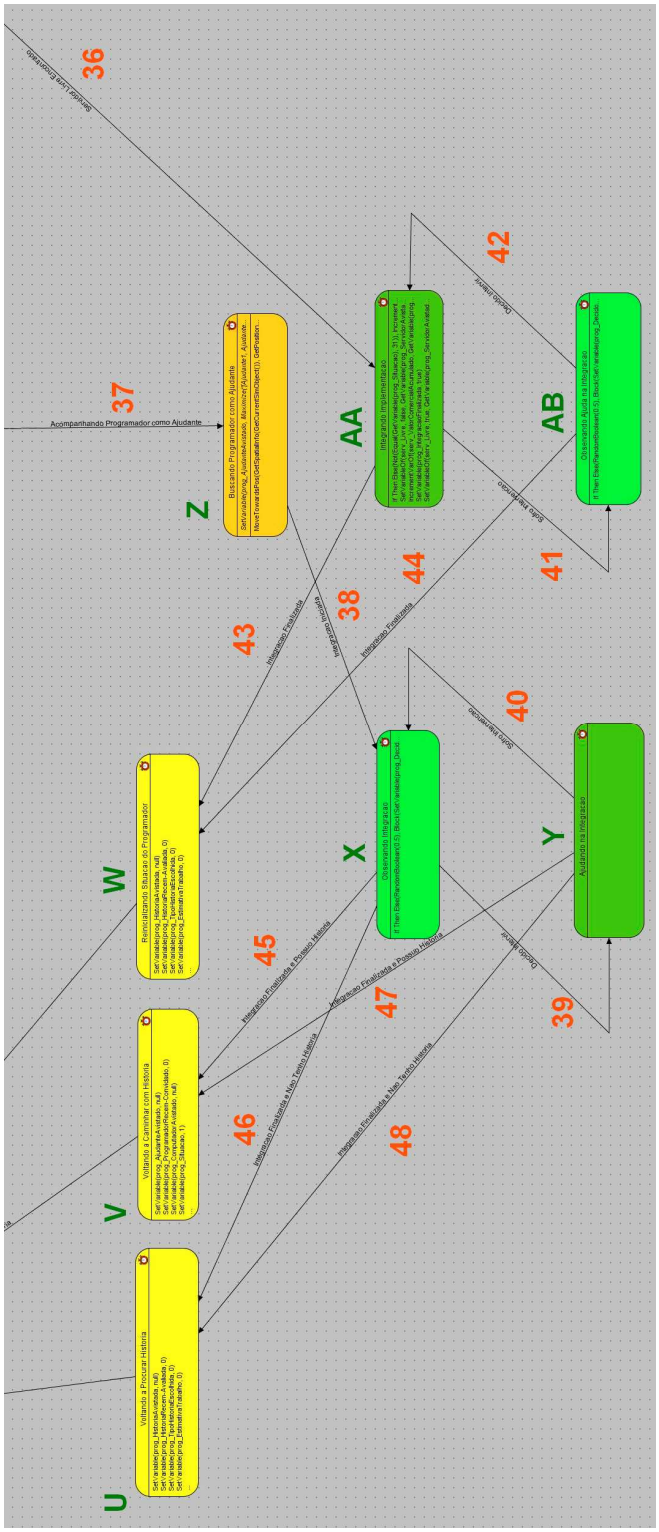


Figura 22 - O Modelo Computacional do Agente Programador (Parte 3 de 3)
Fonte: figura elaborada pelo autor.

6.4.4 Especificação Lógica do Modelo - Agente Programador

A seguir, será apresentado um exemplo da Especificação Lógica do Modelo - Agente Programador.

A especificação lógica foi construída seguindo a sequencia estabelecida, no diagrama de atividades (Figura 14 ou Figuras 20, 21 e 22), para os Estados Funcionais (de A a Z, além de AA e AB). Cada Estado Funcional é apresentado através de diversas tabelas divididas em três conjuntos: (i) tabela do Estado Funcional; (ii) tabelas com Regras de Saída e (iii) tabelas com Regras de Entrada.

Em todas as tabelas, as **descrições** encontram-se nas células com **fundo branco** e os **valores** correspondentes encontram-se nas células de **fundo sombreado**, ao lado da respectiva descrição. Excepcionalmente, optou-se por colocar, nas primeiras linhas de cada tabela, o valor da identificação do Estado Funcional, ou da Regra (no caso das tabelas de regras), **na coluna da esquerda** e com uma fonte de tamanho maior. Tal exceção na disposição de valores em uma tabela foi adotada para tentar facilitar a busca, na especificação, por um determinado Estado Funcional ou por uma determinada Regra.

Após a tabela que identifica o **Estado Funcional**, temos os dois conjuntos de tabelas das Regras.

O primeiro conjunto de Regras é o das **Regras de Saída**, pois as mesmas determinam quais serão os **próximos** Estados Funcionais que o programador poderá alcançar, dependendo de qual regra do conjunto resultar em um valor-verdade *true* (verdadeiro). Como já foi mencionado, cada regra é analisada sequencialmente pelo SeSAm. As condições de cada uma das Regras de Saída, assim como suas identificações e Estados Funcionais de origem e destino são apresentadas.

O segundo conjunto de Regras é o das **Regras de Entrada**. Essas são as regras que têm como destino o Estado Funcional atual. Para essas regras, em suas respectivas tabelas, há apenas a identificação das mesmas e a identificação de seus respectivos **Estados**

Funcionais de Origem. Para consultar-se a especificação completa de uma determinada Regra de Entrada, deve-se buscar, na especificação, pelo Estado Funcional de origem da mesma (onde a mesma estará listada no conjunto de Regras de Saída).

Exemplo:

*** Estado Funcional `Caminhando sem Historia`**********

A	Identificação do Estado Funcional
`Caminhando sem Historia` (Estado Inicial do Agente Programador)	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção inicial do agente, durante a qual o mesmo ainda não avistou uma História.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

1	Identificação da Regra de Saída
`Avistei Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando sem Historia`
Nome do Estado Funcional de Destino	`Buscando Historia`
Identificação do Estado Funcional de Destino	C
Condição da Regra	Encontrar uma História no ambiente, dentro do raio de percepção do agente.

7	Identificação da Regra de Saída
`Aceitei Convite para Ajudar`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando sem Historia`
Nome do Estado Funcional de Destino	`Caminhando como Ajudante`
Identificação do Estado Funcional de Destino	I
Condição da Regra	Se o agente tiver sido chamado para ajudar, ou seja, se `prog_Situacao` for igual a 5 ou 6.

Regras de Entrada

3	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	E
Nome do Estado Funcional de Origem	`Avaliando Historia`

6	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	F
Nome do Estado Funcional de Origem	`Afastando-se da Historia`

49	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	W
Nome do Estado Funcional de Origem	`Reinicializando Situacao do Programador`

51	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	U
Nome do Estado Funcional de Origem	`Voltando a Procurar Historia`

Para ver os demais Estados Funcionais do Agente Programador, consultar o Apêndice A – Especificação Lógica do Modelo - Agente Programador. **A leitura do Apêndice A é imprescindível para a correta compreensão da simulação multi-agente deste trabalho.**

6.4.5 Capacidades e Comportamentos do Agente Programador

Conforme já mencionado, López (2004) afirma:

> Um modelo de agente BDI baseia-se na teoria do raciocínio prático, que afirma que o comportamento de um agente é impulsionado por seus objetivos. Esses agentes são inteiramente definidos através de três atitudes mentais, descritas a seguir. Crenças são a representação que o agente possui de seu mundo, Desejos (ou Objetivos) são os estados que o agente deseja obter e as Intenções representam os meios que o agente dispõe para atingir seus objetivos.

O Agente Programador, proposto neste trabalho, foi modelado de acordo com o modelo BDI. Tal fato torna-se evidente ao perceber-se que:

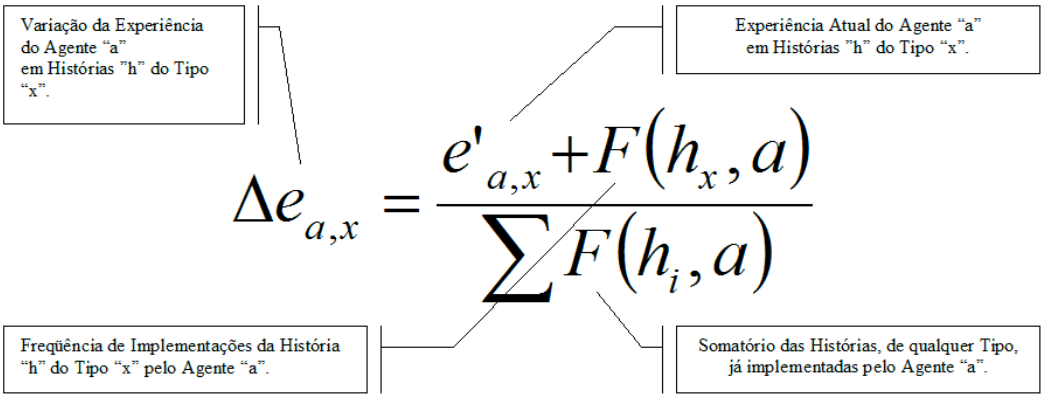
a) a representação que o agente possui do seu mundo – ou sua **crença** – é evidenciada pela capacidade de avistar outros objetos (no caso, histórias ou outros agentes), de memorizar suas posições (através de variáveis implícitas à própria ferramenta de simulação) e de ir de encontro aos mesmos para efetuar as interações necessárias;

b) os **desejos** do Agente Programador podem ser exemplificados pela necessidade de entrar em contato com outros agentes para obter ajuda, a fim de implementar uma determinada história ou de realizar alguma outra tarefa prevista no modelo;

c) finalmente, as **intenções** do Agente Programador são evidenciadas pelos meios que o mesmo dispõe para cumprir suas tarefas – possuir uma história para implementar, recrutar outro agente para ajudá-lo e ocupar um computador.

Ainda que os Agente Programadores presentes na simulação sejam instâncias de uma mesma classe de agente, cada um deles age de maneira independente na simulação multi-agente, **sem seguir algoritmos procedurais**. Além disso, cada um dos Agentes Programadores, ao longo de uma execução da simulação, age de acordo com o conteúdo de sua memória (representada pelas suas respectivas variáveis e constantes) e de acordo com a evolução de sua experiência em implementar cada um dos quatro tipos de histórias presentes no modelo.

A experiência adquirida por cada um dos Agentes Programadores, a qual influencia tanto a escolha das histórias quanto a velocidade de implementação das mesmas, foi modelada através da equação 1.


Variação da Experiência do Agente "a" em Histórias "h" do Tipo "x".
Experiência Atual do Agente "a" em Histórias "h" do Tipo "x".
$\Delta e_{a,x} = \frac{e'_{a,x} + F(h_x, a)}{\sum F(h_i, a)}$
Freqüência de Implementações da História "h" do Tipo "x" pelo Agente "a".
Somatório das Histórias, de qualquer Tipo, já implementadas pelo Agente "a".

(1)

7 RESULTADOS EXPERIMENTAIS

7.1 ABORDAGEM ESCOLHIDA PARA O EXPERIMENTO COM O MODELO DE SIMULAÇÃO

Como foi mencionado no início deste trabalho, o objetivo do mesmo é o de auxiliar o gerente de projeto, em seu processo de tomada de decisão, através de um modelo de simulação do processo de desenvolvimento de *software* em um ambiente de Programação Extrema.

Assim, a abordagem escolhida para os testes sobre o modelo desenvolvido foi a de um exemplo prático de uso do mesmo. Tal exemplo de uso tem por objetivo demonstrar, de maneira sucinta, como um gerente de projeto **poderia** utilizar o modelo de simulação para avaliar uma determinada situação no ambiente de desenvolvimento. Cabe lembrar que **tal demonstração explora apenas uma das muitas possibilidades de experimentação disponibilizadas pelo modelo**. Outras possibilidades de experimentação serão brevemente apresentadas no próximo capítulo.

Utilizando o modelo e os procedimentos que serão apresentados a seguir, o gerente de projeto terá, à sua disposição, mais uma ferramenta de apoio para a sua tomada de decisão.

7.2 FATORES ANALISADOS

Law e Kelton (2000) afirmam:

> Experimentos cuidadosamente projetados são muito mais eficientes do que uma sequencia de execuções por “tentativa e erro”, durante as quais simplesmente testamos, sem sistemática, um determinado número de configurações alternativas para ver o que acontece.

Para a experimentação com o modelo desenvolvido, o autor adotou o formato de experimentação estatística 2^k Fatorial. Tal formato é adequado para testar modelos computacionais de simulação - particularmente nos primeiros estágios de sua experimentação (quando ainda não há certeza sobre como os parâmetros de entrada e as estruturas do modelo afetam as respostas do mesmo) (LAW e KELTON, 2000). Adicionalmente, a revisão da literatura indica que a pesquisa de Berger (2008) - que também envolve um modelo de simulação multi-agente - também aplicou o formato de experimentação estatística 2^k Fatorial aos testes de seu modelo.

Na terminologia do formato de experimentação estatística 2^k Fatorial, os **fatores** correspondem aos parâmetros de entrada e às estruturas do modelo, enquanto as **respostas** correspondem às saídas fornecidas pelo modelo de simulação (LAW e KELTON, 2000).

Segundo Law e Kelton (2000), o formato 2^k Fatorial requer que sejam escolhidos apenas dois níveis para cada um dos fatores a serem estudados. Após essa escolha, devem ser executadas simulações para cada uma das 2^k combinações de fatores e níveis. Embora seja arbitrário, normalmente um sinal positivo (+) é associado a um dos níveis do fator (ao valor mais alto, caso seja quantitativo) e um sinal negativo (-) é associado ao outro nível do fator (ao valor mais baixo, caso seja quantitativo). A descrição completa do método pode ser encontrada em Law e Kelton (2000).

No caso deste trabalho, baseando-se na situação ilustrada na Figura 15 (Capítulo 6) para a programação em pares do método XP, foram especificados os dois fatores - logo, $k=2$ - e os quatro níveis apresentados na Tabela 2.

Tabela 2 - Fatores e Níveis

Fatores	Níveis
1) Computadores	(+) 3 Computadores (quantidade adequada) (-) 1 Computador (quantidade insuficiente)
2) Programadores	(+) 6 Programadores (quantidade adequada) (-) 3 Programadores (quantidade insuficiente)

Fonte: tabela elaborada pelo autor.

De acordo com o formato 2^k Fatorial, devemos construir uma matriz de experimentação. Essa matriz organiza as 2^k combinações de fatores e níveis (Tabela 2) especificados para as simulações. A Matriz de Experimentação é apresentada a seguir.

Tabela 3 - Matriz de Experimentação 2^k Fatorial

Identificação da Combinação de Fatores	Fator 1 (Computadores)	Fator 2 (Programadores)	Resposta (Média do Valor de Negócio Acumulado)
1	-	-	R_1
2	+	-	R_2
3	-	+	R_3
4	+	+	R_4

Fonte: tabela elaborada pelo autor.

Para cada uma das quatro combinações de fatores (linhas 1, 2, 3 e 4 da Tabela 3), a resposta R_i , onde *i* é o número da combinação, corresponde à média aritmética de uma amostra de 40 execuções do modelo de simulação.

As simulações foram executadas utilizando-se como critério de parada o tempo de 960 minutos (960 “ticks” do simulador). Conforme uma das premissas do modelo, estabeleceu-se que um “tick” do simulador equivale a um minuto de tempo simulado, o que totaliza um período de 16 horas simuladas, ou dois dias úteis simulados - desconsiderando-se, assim, o intervalo de tempo entre os dois dias.

Essa quantidade de minutos foi determinada a partir do tempo necessário para a implementação de uma História (sujeito à experiência do programador), conforme a especificação modelo de simulação do Agente Programador (Apêndice A). Esse tempo corresponde a 100 minutos, mais ou menos 20 minutos, e foi definido arbitrariamente pelo autor.

Como, na situação original proposta (Capítulo 6, Figura 15), existem seis programadores e doze Histórias, seriam consumidos 200 minutos, mais ou menos 40 minutos, por programador, para a implementação de todas as Histórias. A esse tempo, foram adicionados 720 minutos, que corresponderiam à movimentação pelo ambiente e demais interações que os programadores precisam manter para executarem suas tarefas. Com isso, chegou-se ao valor de 960 minutos ("ticks") de duração para cada simulação executada.

No *software* SeSAm (executado por um microprocessador Intel® Core™2 CPU 6600@2.40 GHz), o tempo de cada execução foi de aproximadamente 35 segundos. No entanto, as simulações não foram executadas na velocidade máxima permitida pelo SeSAm. Tal restrição de velocidade foi imposta para que a progressão da quantidade de "ticks" especificada pudesse ser acompanhada visualmente pelo autor e todas as simulações fossem interrompidas precisamente no instante 960.

As tabelas, apresentadas na próxima subseção, apresentam os resultados das simulações para cada uma das combinações de fatores e níveis da Tabela 3.

7.3 EXPERIMENTOS E RESULTADOS

Tabela 4 - Simulações da Combinação de Fatores 1

Combinação de Fatores 1	
Fator 1:	(-) 1 Computador
Fator 2:	(-) 3 Programadores
Simulação (960 ticks)	Valor de Negócio Acumulado (máx.=3000)
1	900
2	800
3	900
4	200
5	400
6	500
7	900
8	100
9	300
10	800
11	700
12	800
13	600
14	700
15	200
16	700
17	900
18	400
19	1100
20	500
21	900
22	900
23	700
24	500
25	200
26	900
27	600
28	800
29	100
30	900
31	400
32	600
33	800
34	500
35	100
36	600
37	900
38	200
39	700
40	600
Média:	**607,5**
Desvio Padrão:	**274,924232**

Fonte: tabela elaborada pelo autor.

Tabela 5 - Simulações da Combinação de Fatores 2

Combinação de Fatores 2	
Fator 1:	(+) 3 Computadores
Fator 2:	(-) 3 Programadores
Simulação (960 ticks)	Valor de Negócio Acumulado (máx.=3000)
1	1100
2	1500
3	700
4	700
5	500
6	700
7	400
8	800
9	900
10	600
11	800
12	900
13	500
14	800
15	400
16	1000
17	600
18	1600
19	1000
20	400
21	1000
22	700
23	500
24	600
25	800
26	800
27	1000
28	400
29	1500
30	500
31	800
32	700
33	500
34	400
35	800
36	800
37	700
38	600
39	1100
40	600
Média:	**767,5**
Desvio Padrão:	**297,3278429**

Fonte: tabela elaborada pelo autor.

Tabela 6 - Simulações da Combinação de Fatores 3

Combinação de Fatores 3	
Fator 1:	(-) 1 Computador
Fator 2:	(+) 6 Programadores
Simulação (960 ticks)	Valor de Negócio Acumulado (máx.=3000)
1	400
2	400
3	800
4	1300
5	800
6	900
7	900
8	1600
9	500
10	400
11	900
12	800
13	900
14	300
15	300
16	1300
17	600
18	600
19	1200
20	1000
21	300
22	400
23	1000
24	1100
25	400
26	1300
27	600
28	1100
29	500
30	800
31	900
32	400
33	1000
34	600
35	1200
36	800
37	1000
38	900
39	1100
40	800
Média:	**802,5**
Desvio Padrão:	**330,1029365**

Fonte: tabela elaborada pelo autor.

Tabela 7 - Simulações da Combinação de Fatores 4

Combinação de Fatores 4	
Fator 1:	(+) 3 Computadores
Fator 2:	(+) 6 Programadores
Simulação (960 ticks)	Valor de Negócio Acumulado (máx.=3000)
1	1300
2	900
3	700
4	900
5	900
6	1900
7	800
8	1300
9	800
10	900
11	1300
12	1500
13	800
14	2100
15	900
16	2800
17	1500
18	700
19	1000
20	2400
21	2300
22	1100
23	900
24	1300
25	1300
26	1800
27	1500
28	1000
29	1900
30	1100
31	800
32	1300
33	1800
34	1400
35	700
36	1500
37	1000
38	1200
39	900
40	1500
Média:	**1292,5**
Desvio Padrão:	**508,5865279**

Fonte: tabela elaborada pelo autor.

Uma vez executadas as 40 simulações para cada uma das combinações, passa-se para o cálculo das médias aritméticas dos Valores Acumulados para o Negócio, novamente para cada uma das quatro combinações. Assim, obtemos os valores R_i (média do Valor Acumulado para o Negócio) de cada combinação, mostrados abaixo.

Tabela 8 - Matriz de Experimentação 2^k Fatorial com Respostas e Desvios Padrão

Identificação da Combinação de Fatores	Fator 1 (Computadores)	Fator 2 (Programadores)	Resposta (R_i) ou Média do Valor de Negócio Acumulado	Desvio Padrão Amostral (*s*)
1	-	-	607,5	274,92
2	+	-	767,5	297,33
3	-	+	802,5	330,10
4	+	+	1292,5	508,59

Fonte: tabela elaborada pelo autor.

Para verificarmos a significância estatística dos experimentos, utilizamos a Análise de Variância (ANOVA) por fator único (HAIR, 2006) com alfa igual a 0,05. A tabela 9 apresenta um resumo das informações coletadas e a tabela 10 apresenta os resultados da ANOVA.

Tabela 9 - Resumo das Informações Coletadas nos Experimentos

Grupo	*Contagem*	*Soma*	*Média*	*Variância*
Combinação 1	40	24300	607,5	75583,33333
Combinação 2	40	30700	767,5	88403,84615
Combinação 3	40	32100	802,5	108967,9487
Combinação 4	40	51700	1292,5	258660,2564

Fonte: tabela elaborada pelo autor.

Tabela 10 - Cálculo da ANOVA

Fonte da variação	*Soma dos Quadrados*	*graus de liberdade*	*Média Quadrada*
Entre grupos	10498000	3	3499333,333
Dentro dos grupos	20733000	156	132903,8462
Total	31231000	159	

F	*valor-P (Probabilidade)*	*F crítico*
26,32981238	7,78253E-14	2,662567056

Fonte: tabela elaborada pelo autor.

Hipótese Nula: as médias populacionais das quatro combinações, das quais foram colhidas as amostras, são iguais.

Calculada a ANOVA, descobrimos que o valor de **F**, ou **a variância entre as quatro Combinações**, é igual a 26,33. A probabilidade possui valor inferior a 0,05 e mostra a que nível os resultados são estatisticamente significativos. Como **F** é maior do que $\mathbf{F_{crítico}}$ (26,33 > 2,66), podemos concluir que os resultados são significativos. Em outras palavras: é verdade que existe uma diferença, entre todas as quatro combinações, para o valor acumulado para o negócio. Logo, **rejeitamos** a Hipótese Nula.

Uma vez cientes da significância estatística dos resultados, aplicamos o cálculo dos *efeitos principais dos fatores (e).* O efeito principal de um fator é a mudança média na resposta (R) devido à mudança de nível do fator enquanto os demais fatores permanecem fixos, calculada sobre todas as possíveis combinações dos outros *k-1* fatores (LAW e KELTON, 2000).

Para calcular os efeitos principais de cada fator utilizamos as equações 2 e 3:

$$e_1 = \frac{(R_2 - R_1) + (R_4 - R_3)}{2^{k-1}} \quad (2)$$

$$e_2 = \frac{(R_3 - R_1) + (R_4 - R_2)}{2^{k-1}} \quad (3)$$

Com os valores das respostas R_i (tabela 8) e sabendo que k =2, aplicando as equações 2 e 3 obtemos o valor de 325 para e_1 e o valor de 360 para e_2.

O efeito principal do primeiro fator (e_1) indica que o efeito médio de aumentar-se o número de computadores de **um** (quantidade insuficiente) para **três** (quantidade adequada) aumenta o Valor Acumulado para o Negócio em 325.

Da mesma forma, O efeito principal do segundo fator (e_2) indica que o efeito médio de aumentar-se o número de programadores de **três** (quantidade insuficiente) para **seis** (quantidade adequada) aumenta o Valor Acumulado para o Negócio em 360.

Assim, os testes efetuados sobre o modelo de simulação desenvolvido (apresentados neste Capítulo), utilizando o formato de experimentação estatística 2^k Fatorial, demonstraram a eficácia e a aplicabilidade prática do modelo proposto neste trabalho.

8 CONSIDERAÇÕES FINAIS

8.1 RESULTADOS ALCANÇADOS

O objetivo deste trabalho é o de auxiliar o gerente de projeto em seu processo de tomada de decisão com relação ao desempenho dos Recursos Humanos na geração de valor para o negócio. Tal valor para o negócio, no âmbito da Programação Extrema, é alcançado através da implementação, por parte dos desenvolvedores (programadores), das diversas funcionalidades de um sistema de *software*.

Para o cumprimento de tal objetivo, o autor, conforme o processo de modelagem proposto por Streit (2006), analisou um sistema (composto por ambiente, pessoas e processo) de desenvolvimento de *software* segundo o método da Programação Extrema e apoiado na revisão da literatura relevante. Em seguida, o autor estruturou esse sistema em um modelo conceitual para, finalmente, desenvolver um modelo computacional do sistema analisado, baseado em múltiplos agentes inteligentes.

O modelo computacional da simulação multi-agente do processo de desenvolvimento de *software* em ambiente de Programação Extrema foi desenvolvido com o apoio da ferramenta SeSAm (KLÜGL, 2006). Os detalhes da construção desse modelo foram apresentados no Capítulo 6.

No Capítulo 7, foi apresentado um exemplo de aplicação do modelo computacional para o apoio do gerente de projeto em seu processo de tomada de decisão. Testado através do formato de experimentação estatística 2^k Fatorial, tal exemplo demonstrou a eficácia e a aplicabilidade prática do modelo proposto neste trabalho.

8.2 LIMITAÇÕES DESTE ESTUDO E RECOMENDAÇÕES PARA ESTUDOS FUTUROS

Na opinião do autor, uma releitura da subseção 6.4.1, que apresenta as Premissas do Modelo, oferece um panorama inicial das limitações que envolvem o presente estudo.

Tanto o sistema analisado quanto o tipo de modelagem escolhida para tentar modelar o mesmo envolvem um grande número de aspectos qualitativos e quantitativos. Enquanto alguém poderia considerar isso uma barreira intransponível para o estudo adequado do problema, o autor considera esse grande número de aspectos uma valiosa oportunidade para novas abordagens sobre a simulação multi-agente de processos de desenvolvimento de *software*.

Como um pequeno exemplo de recomendação para estudos futuros, podemos perceber que os testes sobre o modelo desenvolvido envolveram apenas uma **mudança situacional** - isto é, apenas foi efetuada uma alteração na quantidade de computadores ou de programadores disponíveis, para o estudo das respectivas respostas do modelo.

Testes ainda mais interessantes poderiam ser efetuados, para o estudo do problema, envolvendo **mudanças estruturais** no próprio modelo de simulação **ou mudanças numéricas** em uma ou mais das diversas variáveis e constantes presentes no modelo.

Enfim, em um trabalho como o apresentado nesta dissertação, a grande quantidade de aspectos que poderiam ser abordados em estudos futuros fazem o autor lembrar de uma famosa afirmação de Einstein (1931):

> Eu acredito na intuição e na inspiração. A imaginação é mais importante do que o conhecimento. Porque o conhecimento é limitado, enquanto a imaginação abraça o mundo inteiro, estimulando o progresso, criando a evolução. É, estritamente falando, um fator real na pesquisa científica.

REFERÊNCIAS BIBLIOGRÁFICAS

AGENTSHEETS, Inc. Disponível em: <http://www.agentsheets.com/> Acesso em: 2 fev. 2009.

AGILE WORKSHOPS. 2009. Disponível em: <http://agileworkshops.com/FrontPage.html>. Acesso em: 22 jan. 2009.

BARBOSA apud LEMENHE, Flávio et al., op. cit., p.3.

BECK, Kent. **Extreme programming explained: embrace change**. Upper Saddle River: Addison-Wesley, 2000. 190 p.

BELLIFEMINE, Fabio, CAIRE, Giovanni, POGGI, Agostino, RIMASSA, Giovanni. **JADE – A White Paper**. exp - Volume 3 - n. 3 - September 2003. Disponível em: <http://exp.telecomitalialab.com>. Acesso em: 21 fev. 2008.

BERGER, Luiz Marcelo. **Um modelo baseado em agentes para estudo das propriedades emergentes decorrentes da aplicação da lei penal**. Dissertação (Mestrado em Administração) – Programa de Pós-Graduação em Administração, Escola de Administração, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2008.

BIRTWISTLE, G.M. (Graham M.) 1973. **SIMULA begin**. Philadelphia, Auerbach.

BRODKIN, Jon. *Software* revenue to rise 8% in 2008, Gartner predicts. **Network World**, 14.fev.2008. Disponível em:<http://www.networkworld.com/news/2008/021408-gartner-*software*-revenue-2008.html>. Acesso em: 21 fev. 2008.

CAO, L. 2004. Modeling dynamics of agile *software* development. In: **Companion To the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications** (Vancouver, BC, CANADA, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, 46-47.

CAU, A., CONCAS, G. MELIS, M. and TURNU, I. Evaluate XP Effectiveness Using Simulation Modeling. In: **Proc. Sixth Int'l Conf. Extreme Programming and Agile Processes in Software Eng.**, June 2005.

CHAO, J. ATLI, G. Critical personality traits in successful pair programming. In: **Agile Conference**, 2006. Volume 1. 23-28 July 2006 Page(s):5 pp. - 93.

CHONG, Jan, HURLBUTT, T. The Social Dynamics of Pair Programming. In: **Software Engineering, 2007. ICSE 2007. 29th International Conference on**. Center for Work, Technol. e Organ., Stanford Univ., Stanford, CA; Publication Date: 20-26 May 2007. Page(s): 354-363. Location: Minneapolis, MN.

CNPq - **Conselho Nacional de Desenvolvimento Científico e Tecnológico**. Ministério da Ciência e Tecnologia. Brasil. 2009. Disponível em: <http://www.cnpq.br/>. Acesso em: 2 fev. 2009.

CORRÊA, Antenor. **Política de *Software* e Serviços: Ações MCT 2007-2010**. Ministério da Ciência e Tecnologia - Brasil. Manaus - Novembro de 2007. Disponível em: <http://www.mct.gov.br/upd_blob/0019/19587.pdf>. Acesso em: 21 fev. 2008.

CORMAS. **Natural Resources and Agent-Based Simulation**. Disponível em: <http://www.spiderland.org/>. Acesso em: 2 fev. 2009.

DAVIDSSON, Paul. Multi Agent Based Simulation: Beyond Social Simulation. Book Series Lecture Notes in Computer Science. Publisher Springer Berlin / Heidelberg. ISSN 0302-9743 (Print) 1611-3349 (Online). Volume 1979/2000. In: **Multi-Agent-Based Simulation: Second International Workshop, MABS 2000**, Boston, MA, USA, July, 2000. Pages 141-155.

DONZELLI, P. Decision support system for *software* project management. **Software, IEEE**. Volume 23, Issue 4, Page(s):67 – 75, July-Aug. 2006.

EINSTEIN, Albert. **Cosmic Religion : With Other Opinions and Aphorisms**. Publisher: Covici-Friede. 109 p. 1931.

ELECTROGLIDE.BIZ. **Iterative Delivery**. Disponível em: <http://electroglide.biz/images/IteativeDelivery.png>. Acesso em: 11 nov. 2008.

ESMIN, A.A.A. Modelando com UML - Unified Modeling Language. In: **II SECICOM - Universidade Federal de Lavras - UFLA**, Lavras , Novembro, 1999.

FACHIN apud LEMENHE, Flávio et al., op. cit., p.3.

FORRESTER, Jay W. System Dynamics and the Lessons of 35 Years. In: **The Systemic Basis of Policy Making in the 1990s**, De Greene, Kenyon B. ed., 1991.

GABINESKI, R. ; LORENZI, F. . Sistema Multiagente Baseado em Casos de Apoio a Gerência de Projetos. In: **VIII Salão de Iniciação Científica e Trabalhos Acadêmicos**, 2007, Guaiba. VIII Salão de Iniciação Científica e Trabalhos Acadêmicos. Guaiba : Ulbra, 2007.

GNU.ORG. **GNU Lesser General Public License**, 2007. Disponível em: <http://www.gnu.org/licenses/lgpl-3.0.txt>. Acesso em: 27 jan. 2009.

HAIR, Joseph F. **Análise Multivariada de Dados**. 5. ed. Porto Alegre: Bookman, 2006. 593 p.

HILLIER, Frederick S.; LIEBERMAN, Gerald J. **Introduction to operations research**. 8. ed. Boston: McGraw-Hill, 2005. 1061 p.

HUANG, Chun-Che. Using intelligent agents to manage fuzzy business processes**. IEEE Transactions on Systems, Man, and Cybernetics**, Part A 31(6): 508-523 (2001).

HUMPHREY, Watts S. The Personal *Software* Process. **Technical Report CMU/SEI-2000-TR-022 ESC-TR-2000-022**, Carnegie Mellon University, Pittsburgh, Pennsylvania US 2000.).

HUMPHREY, Watts S. **Managing the software process**. Boston: Addison-Wesley, 2004-2005. 494 p. (The SEI series in *software* engineering).

JADE. **Java Agent DEvelopment Framework**. *2001*. Disponível em: <http://jade.tilab.com/>. Acesso em: 21 fev. 2008.

KLEIN, J. 2002. breve: a 3D simulation environment for the simulation of decentralized systems and artificial life. In: **Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems**. The MIT Press.

KLÜGL, F.; HERRLER, R.; FEHLER, M. SeSAm: Implementation of Agent-Based Simulation Using Visual Programming. In: **AAMAS 2006**, May 8-12, 2006, Hakodate, ACM Press, p 1439f, 2006.

KUPPUSWAMI, S., VIVEKANANDAN, K., RODRIGUES, P.: A system dynamics simulation model to find the effects of xp on cost of change curve. In: **Marchesi, M., Succi, G., eds.: XP2003, Conference proceedings**, Springer (2003) 54–62) (2003 “a”).

KUPPUSWAMI, S., VIVEKANANDAN, K., RODRIGUES, P.:: The effects of individual xp practices on *software* development effort. In: **SIGSOFT Softw. Eng. Notes** 28 (2003) 6–6. (2003 “b”).

LAW, Averill M.; KELTON, W. David. **Simulation modeling and analysis**. 3. ed. Boston: McGraw-Hill, 2000. 760 p.

LEMENHE, Flávio. JÚNIOR, Emílio Capelo. ROCHA, Carlos Artur Sobreira. ALEXANDRE, João Welliandre Carneiro. CIARLINI, Aíla de Fátima Silva. O método de Simulação de Monte Carlo para precificação de planos de saúde: uma abordagem didática. In: **XXVI ENEGEP** - Fortaleza, CE, Brasil, 9 a 11 de Outubro de 2006.

LÓPEZ, F. L. MÁRQUEZ, A.A. An architecture for autonomous normative agents. In: **Proceedings of the Fifth Mexican International Conference in Computer Science**, 2004. ENC 2004. 2004 Page(s):96 - 103.

LORENZI, F. . A Multiagent Knowledge-Based Recommender Approach with Truth Maintenance. In: **Doctotal Symposium - ACM Conference on Recommender Systems**,

2007, Minneapolis. Proceedings of the 2007 ACM Conference on Recommender Systems. New York : ACM. p. 195-198.

MACAL, Charles M., NORTH, Michael J. Tutorial on agent-based modeling and simulation. In: **Proceedings of the 37th conference on Winter simulation WSC '05**. December 2005. Publisher: Winter Simulation Conference.

MACEFIELD, Ritch. Usability Studies and the Hawthorne Effect**. Journal of Usability Studies - JUS**. Vol. 2, Issue 3, May 2007, pp. 145-154. Disponível em: <http://www.upassoc.org/upa_publications/jus/2007may/hawthorne-effect.pdf>. Acesso em: 27 jan. 2009.

MADKIT. 2007. Disponível em: <http://www.madkit.org/>. Acesso em: 2 fev. 2009.

MASON. **Multi-Agent Simulator Of Neighborhoods**. 2003. Disponível em: <http://cs.gmu.edu/~eclab/projects/mason/> Acesso em: 2 fev. 2009.

MAUERKIRCHNER, M. Event based modelling and control of *software* development processes. In: **Engineering of Computer-Based Systems, 1997. Proceedings., International Conference and Workshop on**. Publication Date: 24-28 Mar 1997 page(s): 149-156. Meeting Date: 03/24/1997 - 03/28/1997 Location: Monterey, CA, USA.

MERRILL, D.; COLLOFELLO, J.S. Improving *software* project management skills using a *software* project simulator. In: **Frontiers in Education Conference**, 1997. 27th Annual Conference. Teaching and Learning in an Era of Changeapos;. Proceedings.Volume 3, Issue , 5-8 Nov 1997 Page(s):1361 - 1366 vol.3.

MINGERS, John. Combining IS research methods: Towards a pluralist methodology. **Information Systems Research**, vol. 12, n. 3, p. 240 – 259, 2001.

MISIC, V.B., GEVAERT, H., RENNIE, M.: Extreme dynamics: Modeling the extreme programming *software* development process. In: **Proceedings of ProSim04 workshop on Software Process Simulation and Modeling**. (2004) 237–242)

MORENO, Antonio. VALLS, Aïda. MARÍN, Marta. Multi-agent Simulation of Work Teams. Book Series Lecture Notes in Computer Science. Volume 2691/2003. In: ***Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003***, Prague, Czech Republic, June 16-18, 2003. Proceedings.

MÜLLER, J. P. Architectures and applications of intelligent agents: A survey. **The Knowledge Engineering Review**, 13(4):353–380, 1998.

MURTA, L. G. P. ; WERNER, C. M. L. . Uma Máquina de Processo de Desenvolvimento de *Software* Baseada em Agentes Inteligentes. In: **Simpósio Brasileiro de Engenharia de Software (SBES)**, Workshop de Teses e Dissertações (WTES), 2000, João Pessoa. Anais..., 2000. p. 413-416.

PADBERG, F. 1999. A probabilistic model for *software* projects. In: **Proceedings of the 7th European Software Engineering Conference held jointly with the 7th ACM SIGSOFT international Symposium on Foundations of Software Engineering** (Toulouse, France, September 06 - 10, 1999). Foundations of *Software* Engineering. Springer-Verlag, London, 109-126.

PAES, R, ALMEIDA, H. Agentes e Métricas no Processo de Desenvolvimento de Software em Equipes Distribuídas. In: **INFOCOMP - Journal of Computer Science** - Federal University of Lavras (Brazil) - Dept. Computer Science - VOL.4, N.2, June, 2005. p.53-62.

PMBOK (Project Management Body of Knowledge). **Project Management Institute** – Brazil, Minas Gerais Chapter. 2000.

POOLE, D., MACKWORTH, A., GOEBEL, R. **Computational Intelligence: A Logical Approach**. Oxford University Press, Nova York, 1998.

PRAÇA, Isabel C., RAMOS, Carlos. Multi-agent simulation for balancing of assembly lines. In: **Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning**, 1999. (ISATP '99) 21-24 July 1999 Page(s): 459 – 464.

REPAST Simphony. 2007. Disponível em <http://repast.sourceforge.net/>. Acesso em: 21 fev. 2008.

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial intelligence: a modern approach**. Upper Saddle River: Prentice-Hall, 1995. 932 p. ISBN 0-13-103805-2.

SANAL, U. Z. A decision support system for fuzzy scheduling of *software* projects. In: **AUTOTESTCON Proceedings**, 2000 IEEE. Volume , Issue , 2000 Page(s):263 - 272.

SCISSOR.COM. **Basic Team Room**. 2009. Disponível em: <http://www.scissor.com/resources/teamroom/main_room.jpg>. Acesso em: 21 jan. 2009.

SESAM. **Shell for Simulated Agent Systems**. 2009. Disponível em: <http://www.simsesam.de/>. Acesso 3 de fev. 2009.

SETTAS, D., BIBI, S., SFETSOS, P., STAMELOS, I., and GEROGIANNIS, V. 2006. Using Bayesian Belief Networks to Model *Software* Project Management Antipatterns. In: **Proceedings of the Fourth international Conference on Software Engineering Research, Management and Applications**. August 9 nov. 2006.

SIKKA, Vijay. **Maximizing ROI on software development**. Boca Raton: Auerbach, 2005. 253 p.

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg**. Operating system concepts**. 7. ed. New York: John Wiley e Sons, c2005. 921 p.

SILVA, Fábio Augusto, REIS, Rodrigo, et al. Um Modelo de Simulação de Processos de *Software* baseado em Agentes Cooperativos. UFRGS/UFPA. In: **XIII Simpósio Brasileiro de Engenharia de Software**. Florianópolis, 1999.

SIMCOG. **Simulation of Cognitive Agents**. 2001. Disponível em: <http://www.lti.pcs.usp.br/SimCog/>. Acesso em: 2 de fev. 2009.

SIMULA. **The Simula Programming Language**. 2009. Disponível em: <http://www.engin.umd.umich.edu/CIS/course.des/cis400/simula/simula.html>. Acesso em: 2 de fev. 2009.

SLOMAN, Aaron. **The SimAgent TOOLKIT - for Philosophers and Engineers**. 2005. Disponível em:
<http://www.cs.bham.ac.uk/research/projects/poplog/packages/simagent.html>.
Acesso em: 2 fev. 2009.

SPIDERLAND. Disponível em: <http://www.spiderland.org/>. Acesso em: 2 fev. 2009.

STARLOGO. 2000. Disponível em: <http://education.mit.edu/starlogo/>. Acesso em: 2 fev. 2009.

STREIT, Rosalvo Ermes. **Um modelo baseado em agentes para o estudo regulamentar de governança do sistema financeiro**. 2006. Tese (Doutorado em Administração) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006.

SWARM. 1999. Disponível em: <http://www.swarm.org/index.php/Main_Page>. Acesso em: 2 fev. 2009.

TRIPODI apud STREIT, Rosalvo Ermes. op. cit., p.63.

WEISS, Gerhard (Ed.). **Multiagent Systems: a modern approach to distributed artificial intelligence**. Cambridge: MIT, 1999. 619 p.

WICKENBERG, Tham. DAVIDSSON, Paul. On Multi Agent Based Simulation of *Software* Development Processes (Book Chapter). Book Series Lecture Notes in Computer Science Volume 2581/2003. In: **Multi-Agent-Based Simulation II: Third International Workshop, MABS 2002**, Bologna, Italy, July 15-16, 2002.

WILENSKY, Uri. **NetLogo**. 1999. Disponível em <http://ccl.northwestern.edu/netlogo/docs/>. Acesso em: 2 fev. 2009.

WOOLDRIDGE, M. e JENNINGS, N. Intelligent agents: Theory and Practice. **The Knowledge Engineering Review**, 10(2):115–152, 1995.

ZHANG, H., HUO, M., KITCHENHAM, B., and JEFFERY, R. Qualitative Simulation Model for *Software* Engineering Process. In: **Proceedings of the Australian Software Engineering Conference**. April 18 - 21, 2006.

APÊNDICE A – ESPECIFICAÇÃO LÓGICA DO MODELO – AGENTE PROGRAMADOR

Estados Funcionais do Agente Programador

*** Estado Funcional `Caminhando sem Historia`

A	Identificação do Estado Funcional
`Caminhando sem Historia` (Estado Inicial do Agente Programador)	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção inicial do agente, durante a qual o mesmo ainda não avistou uma História.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

1	Identificação da Regra de Saída
`Avistei Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando sem Historia`
Nome do Estado Funcional de Destino	`Buscando Historia`
Identificação do Estado Funcional de Destino	C
Condição da Regra	Encontrar uma História no ambiente, dentro do raio de percepção do agente.

7	Identificação da Regra de Saída
`Aceitei Convite para Ajudar`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando sem Historia`
Nome do Estado Funcional de Destino	`Caminhando como Ajudante`
Identificação do Estado Funcional de Destino	I
Condição da Regra	Se o agente tiver sido chamado para ajudar, ou seja, se `prog_Situacao` for igual a 5 ou 6.

Regras de Entrada

3	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	E
Nome do Estado Funcional de Origem	`Avaliando Historia`

6	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	F
Nome do Estado Funcional de Origem	`Afastando-se da Historia`

49	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	W
Nome do Estado Funcional de Origem	`Reinicializando Situacao do Programador`

51	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	U
Nome do Estado Funcional de Origem	`Voltando a Procurar Historia`

*** Estado Funcional `Caminhando com Historia`

B	Identificação do Estado Funcional
`Caminhando com Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente, durante a qual o mesmo já escolheu uma História para implementar.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

8	Identificação da Regra de Saída
`Aceitei Convite para Ajudar`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando com Historia`
Nome do Estado Funcional de Destino	`Caminhando como Ajudante`
Identificação do Estado Funcional de Destino	I
Condição da Regra	Se o agente tiver sido chamado para ajudar, ou seja, se `prog_Situacao` for igual a 5 ou 6.

11	Identificação da Regra de Saída
`Avistei Programador`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando com Historia`
Nome do Estado Funcional de Destino	`Buscando Programador`
Identificação do Estado Funcional de Destino	D
Condição da Regra	Em 50% das vezes que encontrar um Programador dentro do raio de percepção do agente.

Regras de Entrada

4	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	E
Nome do Estado Funcional de Origem	`Avaliando Historia`

24	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	G
Nome do Estado Funcional de Origem	`Convidando Programador`

10	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	D
Nome do Estado Funcional de Origem	`Buscando Programador`

17	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	H
Nome do Estado Funcional de Origem	`Afastando-se do Programador`

50	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	V
Nome do Estado Funcional de Origem	`Voltando a Caminhar com Historia`

*** Estado Funcional `Buscando Historia`

C	Identificação do Estado Funcional
`Buscando Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Move-se em direção à História avistada.
Ações	Move-se em direção à História (avistada) mais próxima.

Regras de Saída

2	Identificação da Regra de Saída
`Acessei Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Historia`
Nome do Estado Funcional de Destino	`Avaliando Historia`
Identificação do Estado Funcional de Destino	E
Condição da Regra	Se a posição do Agente Programador for igual à posição da História avistada.

5	Identificação da Regra de Saída
`Recem-Avaliei essa Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Historia`
Nome do Estado Funcional de Destino	`Afastando-se da Historia`
Identificação do Estado Funcional de Destino	F
Condição da Regra	Se a identificação da História avistada (`hist_Identificacao`) for igual ao valor da História recém-avaliada (`prog_HistoriaRecem-Avaliada`).

Regras de Entrada

1	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	A
Nome do Estado Funcional de Origem	`Caminhando sem Historia`

*** Estado Funcional `Buscando Programador`

D	Identificação do Estado Funcional
`Buscando Programador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Move-se em direção ao Agente Programador mais próximo.
Ações	Move-se em direção ao Agente Programador (avistado) mais próximo.

Regras de Saída

10	Identificação da Regra de Saída
`Procurarei outro Programador`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Programador`
Nome do Estado Funcional de Destino	`Caminhando com Historia`
Identificação do Estado Funcional de Destino	B
Condição da Regra	Em 30% das ocasiões, o agente desiste do Programador Avistado e decide procurar outro.

12	Identificação da Regra de Saída
Chamei Programador	Nome da Regra
Nome do Estado Funcional de Origem	Buscando Programador
Nome do Estado Funcional de Destino	Convidando Programador
Identificação do Estado Funcional de Destino	G
Condição da Regra	Se o agente ainda não possui ou já possui uma História (prog_Situacao = 0 ou prog_Situacao = 1) e a posição do agente for igual à posição do Programador avistado.

13	Identificação da Regra de Saída
Recem-Convidei esse Programador	Nome da Regra
Nome do Estado Funcional de Origem	Buscando Programador
Nome do Estado Funcional de Destino	Afastando-se do Programador
Identificação do Estado Funcional de Destino	H
Condição da Regra	Se o agente ainda não possui ou já possui uma História (prog_Situacao = 0 ou prog_Situacao = 1) e a identificação do Programador avistado (prog_Identificacao) for igual ao valor do Programador recém-convidado (prog_ProgramadorRecem-Convidado).

18	Identificação da Regra de Saída
Recebi Convite para Ajudar	Nome da Regra
Nome do Estado Funcional de Origem	Buscando Programador
Nome do Estado Funcional de Destino	Observando Implementacao
Identificação do Estado Funcional de Destino	M
Condição da Regra	[Se o Programador avistado começou a implementar a História (prog_Situacao = 21) e a posição do agente for igual à posição do Programador avistado] e [o agente já estiver agindo como co-piloto sem História ou co-piloto com História (prog_Situacao = 5 ou 15 ou 6 ou 16)].

Regras de Entrada

11	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	B
Nome do Estado Funcional de Origem	Caminhando com História

9	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	I
Nome do Estado Funcional de Origem	Caminhando como Ajudante

*** Estado Funcional `Avaliando Historia`

E	Identificação do Estado Funcional
`Avaliando Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	O agente avalia a História encontrada e decide, de acordo com a sua experiência prévia, se a escolhe para implementar ou não. Se a experiência do desenvolvedor for a mais alta para o tipo de História avistada, a História é escolhida, ou em 30% das ocasiões o desenvolvedor escolherá a História mesmo que sua experiência com o tipo da mesma não seja a mais alta. Após escolher a História, o desenvolvedor define sua estimativa de trabalho para a mesma e a retira do quadro de histórias. Caso o desenvolvedor não tenha aceito a História, a mesma é marcada como recém-avaliada pelo agente.
Ações	Se a História ainda não foi pega por outro programador, então o agente avalia se o tipo da mesma corresponde à maior experiência pessoal para aquele tipo, ou em 30% das ocasiões, pega a História mesmo que não corresponda à maior experiência pessoal para o tipo da História escolhida. Se a História for aceita (por um dos motivos acima), então o agente memoriza o tipo da história (`prog_TipoHistoriaEscolhida`) e o valor comercial da mesma (`prog_ValorComercialHistoriaEscolhida`). Em seguida, define a sua estimativa de trabalho (`prog_EstimativaTrabalho`), calculando tal estimativa como sendo um valor aleatório de uma distribuição Normal com Média `100` e Desvio Padrão igual a `20/prog_ExpTipoX`, onde prog_ExpTipoX refere-se à experiência do programador para o tipo da História escolhida **(tal cálculo simula a precisão da estimativa, que torna-se maior à medida que o Desvio Padrão da mesma torna-se menor)**. Finalmente, caso tenha escolhido uma História, sua situação passa a ser de programador com História (`prog_Situacao = 1`) e a História é retirada do quadro de histórias. Caso a História avaliada não tenha sido escolhida, a mesma é marcada como recém-avaliada.

Regras de Saída

3	Identificação da Regra de Saída
`Recusei Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Avaliando Historia`
Nome do Estado Funcional de Destino	`Caminhando sem Historia`
Identificação do Estado Funcional de Destino	A
Condição da Regra	O agente continua sem História (`prog_Situacao = 0`).

4	Identificação da Regra de Saída
`Peguei Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Avaliando Historia`
Nome do Estado Funcional de Destino	`Caminhando com Historia`
Identificação do Estado Funcional de Destino	B
Condição da Regra	O agente escolheu uma História (`prog_Situacao = 1`).

Regras de Entrada

2	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	C
Nome do Estado Funcional de Origem	`Buscando Historia`

*** Estado Funcional `Afastando-se da Historia`

F	Identificação do Estado Funcional
`Afastando-se da Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente para longe da História avistada.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

6	Identificação da Regra de Saída
`Voltei a Procurar Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Afastando-se da Historia`
Nome do Estado Funcional de Destino	`Caminhando sem Historia`
Identificação do Estado Funcional de Destino	A
Condição da Regra	Há uma probabilidade de 50% de que esta Regra seja acionada.

23	Identificação da Regra de Saída
`Aceitei Convite para Ajudar`	Nome da Regra
Nome do Estado Funcional de Origem	`Afastando-se da Historia`
Nome do Estado Funcional de Destino	`Caminhando como Ajudante`
Identificação do Estado Funcional de Destino	I
Condição da Regra	Se a situação do agente mudou para a de Co-piloto sem História (`prog_Situacao = 5`) ou Co-piloto com História (`prog_Situacao = 6`).

Regras de Entrada

5	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	C
Nome do Estado Funcional de Origem	`Buscando Historia`

*** Estado Funcional `Convidando Programador`

G	Identificação do Estado Funcional
`Convidando Programador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Se o Programador avistado ainda está ao alcance, convida o mesmo para ajudar na implementação da História. Caso contrário, sinaliza o Programador avistado como recém-convidado.
Ações	Se o Programador avistado ainda está ao alcance, então verifica se o mesmo já não estaria implementando uma História (`prog_Situacao = 21`). Há uma probabilidade de 10% de que, mesmo que o Programador avistado não esteja implementando uma História, o mesmo não seja convidado. Caso o Programador avistado seja convidado, sua situação passa a ser a de Co-piloto sem História ou de Co-piloto com história (`prog_Situacao = 5 ou 6`); a situação do agente passa para a de Piloto (um Piloto sempre possuirá uma História) (`prog_Situacao = 11`). Caso o convite não seja bem-sucedido, o Programador avistado é marcado como recém-convidado.

Regras de Saída

14	Identificação da Regra de Saída
`Meu Convite foi Aceito`	Nome da Regra
Nome do Estado Funcional de Origem	`Convidando Programador`
Nome do Estado Funcional de Destino	`Caminhando como Desenvolvedor`
Identificação do Estado Funcional de Destino	J
Condição da Regra	Se o agente agora é um Piloto (um Programador com uma História), ou seja, se `prog_Situacao = 11`.

15	Identificação da Regra de Saída
`Procurarei outro Programador`	Nome da Regra
Nome do Estado Funcional de Origem	`Convidando Programador`
Nome do Estado Funcional de Destino	`Afastando-se do Programador`
Identificação do Estado Funcional de Destino	H
Condição da Regra	Em 30% das ocasiões, o agente decide procurar outro Programador para tentar convidar.

16	Identificação da Regra de Saída
`Recebi Convite para Ajudar`	Nome da Regra
Nome do Estado Funcional de Origem	`Convidando Programador`
Nome do Estado Funcional de Destino	`Observando Implementacao`
Identificação do Estado Funcional de Destino	M
Condição da Regra	Se a situação do agente passar a ser de Co-piloto sem História ou Co-piloto com História (`prog_Situacao = 5 ou 15 ou 6 ou 16`).

24	Identificação da Regra de Saída
`Meu Convite foi Recusado`	Nome da Regra
Nome do Estado Funcional de Origem	`Convidando Programador`
Nome do Estado Funcional de Destino	`Caminhando com Historia`
Identificação do Estado Funcional de Destino	B
Condição da Regra	Se a situação do agente continuar sendo a de Programador com História (`prog_Situacao = 1`).

Regras de Entrada

12	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	D
Nome do Estado Funcional de Origem	`Buscando Programador`

*** Estado Funcional `Afastando-se do Programador`

H	Identificação do Estado Funcional
`Afastando-se do Programador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente para longe do Programador avistado.
Ações	Muda de direção em +30º ou –30º, em 360º possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

17	Identificação da Regra de Saída
`Voltei a Procurar Programador`	Nome da Regra
Nome do Estado Funcional de Origem	`Afastando-se do Programador`
Nome do Estado Funcional de Destino	`Caminhando com Historia`
Identificação do Estado Funcional de Destino	B
Condição da Regra	Há uma probabilidade de 50% de que esta Regra seja acionada.

Regras de Entrada

13	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	D
Nome do Estado Funcional de Origem	`Buscando Programador`

15	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	G
Nome do Estado Funcional de Origem	`Convidando Programador`

*** Estado Funcional `Caminhando como Ajudante`

I	Identificação do Estado Funcional
`Caminhando como Ajudante`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente no ambiente.
Ações	Muda de direção em +30º ou –30º, em 360º possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

9	Identificação da Regra de Saída
`Avistei Programador`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando como Ajudante`
Nome do Estado Funcional de Destino	`Buscando Programador`
Identificação do Estado Funcional de Destino	D
Condição da Regra	Se um Programador for avistado, esta Regra tem a probabilidade de 50% de ser acionada.

Regras de Entrada

23	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	F
Nome do Estado Funcional de Origem	`Afastando-se da Historia`

7	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	A
Nome do Estado Funcional de Origem	`Caminhando sem Historia`

8	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	B
Nome do Estado Funcional de Origem	`Caminhando com Historia`

*** Estado Funcional `Caminhando como Desenvolvedor`

J	Identificação do Estado Funcional
`Caminhando como Desenvolvedor`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional transitório.
Ações	Nenhuma.

Regras de Saída

19	Identificação da Regra de Saída
`Avistei Computador`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando como Desenvolvedor`
Nome do Estado Funcional de Destino	`Buscando Computador`
Identificação do Estado Funcional de Destino	K
Condição da Regra	Se um Computador for avistado, esta Regra tem a probabilidade de 50% de ser acionada.

Regras de Entrada

14	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	G
Nome do Estado Funcional de Origem	`Convidando Programador`

21	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	L
Nome do Estado Funcional de Origem	`Afastando-me do Computador Ocupado`

*** Estado Funcional `Buscando Computador`

K	Identificação do Estado Funcional
`Buscando Computador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Move-se em direção ao Computador mais próximo.
Ações	Move-se em direção ao Computador (avistado) mais próximo.

Regras de Saída

20	Identificação da Regra de Saída
`Computador Ocupado Encontrado`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Computador`
Nome do Estado Funcional de Destino	`Afastando-me do Computador Ocupado`
Identificação do Estado Funcional de Destino	L
Condição da Regra	Se a posição do agente for igual à do Computador avistado e se o Computador avistado estiver ocupado.

22	Identificação da Regra de Saída
`Computador Livre Encontrado`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Computador`
Nome do Estado Funcional de Destino	`Implementando Historia`
Identificação do Estado Funcional de Destino	O
Condição da Regra	Se a posição do agente for igual à do Computador avistado e se o Computador avistado estiver livre.

Regras de Entrada

19	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	J
Nome do Estado Funcional de Origem	`Caminhando como Desenvolvedor`

*** Estado Funcional `Afastando-me do Computador Ocupado`

L	Identificação do Estado Funcional
`Afastando-me do Computador Ocupado`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente para longe do Computador avistado.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

21	Identificação da Regra de Saída
`Computador Liberado`	Nome da Regra
Nome do Estado Funcional de Origem	`Afastando-me do Computador Ocupado`
Nome do Estado Funcional de Destino	`Caminhando como Desenvolvedor`
Identificação do Estado Funcional de Destino	J
Condição da Regra	Se o Computador avistado está livre.

Regras de Entrada

20	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	K
Nome do Estado Funcional de Origem	`Buscando Computador`

*** Estado Funcional `Observando Implementacao`

M	Identificação do Estado Funcional
`Observando Implementacao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Observa andamento da implementação, com uma probabilidade de 50% de intervir na mesma.
Ações	Com uma probabilidade de 50%, o agente intervém na implementação.

Regras de Saída

33	Identificação da Regra de Saída
`Decido Intervir`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Implementacao`
Nome do Estado Funcional de Destino	`Ajudando na Implementacao`
Identificação do Estado Funcional de Destino	N
Condição da Regra	Se agente decidiu intervir na implementação.

34	Identificação da Regra de Saída
`Implementacao Finalizada`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Implementacao`
Nome do Estado Funcional de Destino	`Acompanhando Programador`
Identificação do Estado Funcional de Destino	Q
Condição da Regra	Se a implementação foi finalizada.

Regras de Entrada

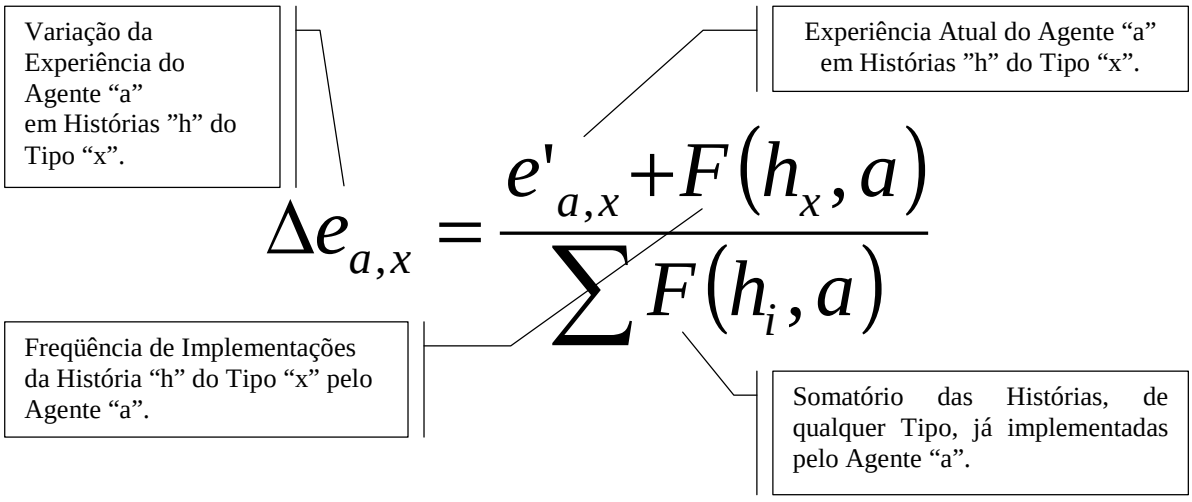
18	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	D
Nome do Estado Funcional de Origem	`Buscando Programador`

16	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	G
Nome do Estado Funcional de Origem	`Convidando Programador`

32	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	N
Nome do Estado Funcional de Origem	`Ajudando na Implementacao`

*** Estado Funcional `Ajudando na Implementacao`

N	Identificação do Estado Funcional
`Ajudando na Implementacao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	O agente (no papel de Co-Piloto) ajuda na implementação da História, passando a efetivamente realizar o trabalho. A História é implementada com uma velocidade que depende da experiência do agente para o tipo de História em questão. Tendo finalizado a tarefa, o agente tem sua própria experiência incrementada. A História em questão pertence ao Agente Programador Piloto do par.
Ações	De acordo com o tipo da História a ser implementada, o agente calcula um passo de execução da implementação (`prog_Passo`), que corresponderá à uma quantidade de trabalho a ser executada a cada "tick" (ou ciclo) da simulação. O valor calculado de `prog_Passo` é igual ao produto entre a experiência na História em questão e um valor aleatório real obtido no intervalo fechado entre zero e um (isto é, o intervalo inclui tais valores). Essa quantidade de trabalho será subtraída da Estimativa de Trabalho para a História que está sendo implementada (cujo "proprietário" é o Agente Programador Piloto). (Obs.: quando a Estimativa de Trabalho atinge um valor igual ou menor do que zero, considera-se a implementação finalizada.) Finalmente, caso o agente tenha finalizado a implementação, este avisa o Agente Programador Piloto e, em seguida, atualiza a própria experiência de acordo com a equação abaixo.



Regras de Saída

32	Identificação da Regra de Saída
Sofro Intervencao	Nome da Regra
Nome do Estado Funcional de Origem	Ajudando na Implementacao
Nome do Estado Funcional de Destino	Observando Implementacao
Identificação do Estado Funcional de Destino	M
Condição da Regra	Agente sofre intervenção do Agente Programador (Piloto).

35	Identificação da Regra de Saída
Implementacao Finalizada	Nome da Regra
Nome do Estado Funcional de Origem	Ajudando na Implementacao
Nome do Estado Funcional de Destino	Acompanhando Programador
Identificação do Estado Funcional de Destino	Q
Condição da Regra	A implementação foi finalizada.

Regras de Entrada

33	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	M
Nome do Estado Funcional de Origem	Observando Implementacao

*** Estado Funcional `Implementando Historia`

O	Identificação do Estado Funcional
`Implementando Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	O agente (no papel de Piloto) implementa a História, passando a efetivamente realizar o trabalho. A História é implementada com uma velocidade que depende da experiência do agente para o tipo de História em questão. Tendo finalizado a tarefa, o agente tem sua própria experiência incrementada.
Ações	O agente passa para a situação de Programador com História Implementando (`prog_Situacao = 21`). Ao acessar um Computador, o agente o sinaliza como ocupado. De acordo com o tipo da História a ser implementada, o agente calcula um passo de execução da implementação (`prog_Passo`), que corresponderá à uma quantidade de trabalho a ser executada a cada "tick" (ou ciclo) da simulação. O valor calculado de `prog_Passo` é igual ao produto entre a experiência na História em questão e um valor aleatório real obtido no intervalo fechado entre zero e um (isto é, o intervalo inclui tais valores). Essa quantidade de trabalho será subtraída da Estimativa de Trabalho para a História que está sendo implementada. (Obs.: quando a Estimativa de Trabalho atinge um valor igual ou menor do que zero, considera-se a implementação finalizada.) Finalmente, caso o agente tenha finalizado a implementação, este libera o Computador que estava sendo utilizado e, em seguida, atualiza a própria experiência de acordo com a equação mostrada no Estado Funcional "N" (`Ajudando na Implementacao`), descrito acima.

Regras de Saída

25	Identificação da Regra de Saída
`Sofro Intervencao`	Nome da Regra
Nome do Estado Funcional de Origem	`Implementando Historia`
Nome do Estado Funcional de Destino	`Observando Ajuda na Implementacao`
Identificação do Estado Funcional de Destino	P
Condição da Regra	Agente sofre intervenção do Agente Programador (Co-Piloto).

27	Identificação da Regra de Saída
`Implementacao Finalizada`	Nome da Regra
Nome do Estado Funcional de Origem	`Implementando Historia`

Nome do Estado Funcional de Destino	`Caminhando com Implementacao Finalizada`
Identificação do Estado Funcional de Destino	R
Condição da Regra	A implementação foi finalizada.

Regras de Entrada

22	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	K
Nome do Estado Funcional de Origem	`Buscando Computador`

26	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	P
Nome do Estado Funcional de Origem	`Observando Ajuda na Implementacao`

*** Estado Funcional `Observando Ajuda na Implementacao`

P	Identificação do Estado Funcional
`Observando Ajuda na Implementacao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Observa andamento da implementação, com uma probabilidade de 50% de intervir na mesma.
Ações	Com uma probabilidade de 50%, o agente intervém na implementação.

Regras de Saída

26	Identificação da Regra de Saída
`Decido Intervir`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Ajuda na Implementacao`
Nome do Estado Funcional de Destino	`Implementando Historia`
Identificação do Estado Funcional de Destino	O
Condição da Regra	Se agente decidiu intervir na implementação.

31	Identificação da Regra de Saída
`Implementacao Finalizada`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Ajuda na Implementacao`
Nome do Estado Funcional de Destino	`Caminhando com Implementacao Finalizada`
Identificação do Estado Funcional de Destino	R
Condição da Regra	A implementação foi finalizada.

Regras de Entrada

25	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	O
Nome do Estado Funcional de Origem	`Implementando Historia`

*** Estado Funcional `Acompanhando Programador`

Q	Identificação do Estado Funcional
`Acompanhando Programador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional transitório.
Ações	Nenhuma.

Regras de Saída

37	Identificação da Regra de Saída
`Acompanhando Programador como Ajudante`	Nome da Regra
Nome do Estado Funcional de Origem	`Acompanhando Programador`
Nome do Estado Funcional de Destino	`Buscando Programador como Ajudante`
Identificação do Estado Funcional de Destino	Z
Condição da Regra	Existe um Agente Programador dentro do raio de percepção do agente.

Regras de Entrada

34	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	M
Nome do Estado Funcional de Origem	`Observando Implementacao`

35	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	N
Nome do Estado Funcional de Origem	`Ajudando na Implementacao`

*** Estado Funcional `Caminhando com Implementacao Finalizada`

R	Identificação do Estado Funcional
`Caminhando com Implementacao Finalizada`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional transitório.
Ações	Nenhuma.

Regras de Saída

28	Identificação da Regra de Saída
`Avistei Servidor de Integracao`	Nome da Regra
Nome do Estado Funcional de Origem	`Caminhando com Implementacao Finalizada`
Nome do Estado Funcional de Destino	`Buscando Servidor de Integracao`
Identificação do Estado Funcional de Destino	S
Condição da Regra	Se o Servidor de Integração for avistado, esta Regra tem a probabilidade de 50% de ser acionada.

Regras de Entrada

27	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	O
Nome do Estado Funcional de Origem	`Implementando Historia`

31	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	P
Nome do Estado Funcional de Origem	`Observando Ajuda na Implementacao`

30	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	T
Nome do Estado Funcional de Origem	`Afastando-me do Servidor Ocupado`

*** Estado Funcional `Buscando Servidor de Integracao`

S	Identificação do Estado Funcional
`Buscando Servidor de Integracao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Move-se em direção ao Servidor de Integração mais próximo.
Ações	Move-se em direção ao Servidor de Integração (avistado) mais próximo.

Regras de Saída

29	Identificação da Regra de Saída
`Servidor Ocupado Encontrado`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Servidor de Integracao`
Nome do Estado Funcional de Destino	`Afastando-me do Servidor Ocupado`
Identificação do Estado Funcional de Destino	T
Condição da Regra	Agente está na mesma posição do Servidor de Integração e o mesmo está ocupado.

36	Identificação da Regra de Saída
`Servidor Livre Encontrado`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Servidor de Integracao`
Nome do Estado Funcional de Destino	`Integrando Implementacao`
Identificação do Estado Funcional de Destino	AA
Condição da Regra	Agente está na mesma posição do Servidor de Integração e o mesmo está livre.

Regras de Entrada

29	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	R
Nome do Estado Funcional de Origem	`Caminhando com Implementacao Finalizada`

*** Estado Funcional `Afastando-me do Servidor Ocupado`

T	Identificação do Estado Funcional
`Afastando-me do Servidor Ocupado`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Locomoção do agente para longe do Servidor de Integração avistado.
Ações	Muda de direção em $+30^o$ ou -30^o, em 360^o possíveis, com uma probabilidade de 50%. Avança na direção escolhida.

Regras de Saída

30	Identificação da Regra de Saída
Servidor Liberado	Nome da Regra
Nome do Estado Funcional de Origem	Afastando-me do Servidor Ocupado
Nome do Estado Funcional de Destino	Caminhando com Implementacao Finalizada
Identificação do Estado Funcional de Destino	R
Condição da Regra	O Servidor de Integração está livre.

Regras de Entrada

29	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	S
Nome do Estado Funcional de Origem	Buscando Servidor de Integracao

*** Estado Funcional Voltando a Procurar Historia

U	Identificação do Estado Funcional
Voltando a Procurar Historia	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional utilizado para reinicialização de variáveis.
Ações	Reinicialização de variáveis.

Regras de Saída

51	Identificação da Regra de Saída
Volto a Procurar Historia	Nome da Regra
Nome do Estado Funcional de Origem	Voltando a Procurar Historia
Nome do Estado Funcional de Destino	Caminhando sem Historia
Identificação do Estado Funcional de Destino	A
Condição da Regra	Nenhuma.

Regras de Entrada

46	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	X
Nome do Estado Funcional de Origem	`Observando Integracao`

48	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	Y
Nome do Estado Funcional de Origem	`Ajudando na Integracao`

*** Estado Funcional `Voltando a Caminhar com Historia`

V	Identificação do Estado Funcional
`Voltando a Caminhar com Historia`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional utilizado para reinicialização de variáveis.
Ações	Reinicialização de variáveis.

Regras de Saída

50	Identificação da Regra de Saída
`Volto a Caminhar com Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Voltando a Caminhar com Historia`
Nome do Estado Funcional de Destino	`Caminhando com Historia`
Identificação do Estado Funcional de Destino	B
Condição da Regra	Nenhuma.

Regras de Entrada

45	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	X
Nome do Estado Funcional de Origem	`Observando Integracao`

47	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	Y
Nome do Estado Funcional de Origem	`Ajudando na Integracao`

*** Estado Funcional `Reinicializando Situacao do Programador`

W	Identificação do Estado Funcional
`Reinicializando Situacao do Programador`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Estado Funcional utilizado para reinicialização de variáveis.
Ações	Reinicialização de variáveis.

Regras de Saída

49	Identificação da Regra de Saída
`Volto a Procurar Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Reinicializando Situacao do Programador`
Nome do Estado Funcional de Destino	`Caminhando sem Historia`
Identificação do Estado Funcional de Destino	A
Condição da Regra	Nenhuma.

Regras de Entrada

43	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	AA
Nome do Estado Funcional de Origem	`Integrando Implementacao`

44	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	AB
Nome do Estado Funcional de Origem	`Observando Ajuda na Integracao`

*** Estado Funcional `Observando Integracao`

X	Identificação do Estado Funcional
`Observando Integracao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Observa andamento da integração, com uma probabilidade de 50% de intervir na mesma.
Ações	Com uma probabilidade de 50%, o agente intervém na integração.

Regras de Saída

45	Identificação da Regra de Saída
`Integracao Finalizada e Possuo Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Integracao`
Nome do Estado Funcional de Destino	`Voltando a Caminhar com Historia`
Identificação do Estado Funcional de Destino	V
Condição da Regra	Se a Integração foi finalizada e o agente possui uma História (`prog_TipoHistoriaEscolhida > 0`).

46	Identificação da Regra de Saída
`Integracao Finalizada e Nao Tenho Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Integracao`
Nome do Estado Funcional de Destino	`Voltando a Procurar Historia`
Identificação do Estado Funcional de Destino	U
Condição da Regra	Se a Integração foi finalizada e o agente não possui uma História (`prog_TipoHistoriaEscolhida = 0`).

39	Identificação da Regra de Saída
`Decido Intervir`	Nome da Regra
Nome do Estado Funcional de Origem	`Observando Integracao`
Nome do Estado Funcional de Destino	`Ajudando na Integracao`
Identificação do Estado Funcional de Destino	Y
Condição da Regra	Se agente decidiu intervir na Integração.

Regras de Entrada

38	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	Z
Nome do Estado Funcional de Origem	`Buscando Programador como Ajudante`

40	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	Y
Nome do Estado Funcional de Origem	`Ajudando na Integracao`

*** Estado Funcional `Ajudando na Integracao`

Y	Identificação do Estado Funcional
`Ajudando na Integracao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	O agente ajuda na Integração da História implementada.
Ações	Nenhuma.

Regras de Saída

40	Identificação da Regra de Saída
`Sofro Intervencao`	Nome da Regra
Nome do Estado Funcional de Origem	`Ajudando na Integracao`
Nome do Estado Funcional de Destino	`Observando Integracao`
Identificação do Estado Funcional de Destino	X
Condição da Regra	Agente sofre intervenção do Agente Programador.

47	Identificação da Regra de Saída
`Integracao Finalizada e Possuo Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Ajudando na Integracao`
Nome do Estado Funcional de Destino	`Voltando a Caminhar com Historia`
Identificação do Estado Funcional de Destino	V
Condição da Regra	Se a Integração foi finalizada e o agente possui uma História (`prog_TipoHistoriaEscolhida > 0`).

48	Identificação da Regra de Saída
`Integracao Finalizada e Nao Possuo Historia`	Nome da Regra
Nome do Estado Funcional de Origem	`Ajudando na Integracao`
Nome do Estado Funcional de Destino	`Voltando a Procurar Historia`
Identificação do Estado Funcional de Destino	U
Condição da Regra	Se a Integração foi finalizada e o agente não possui uma História (`prog_TipoHistoriaEscolhida = 0`).

Regras de Entrada

39	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	X
Nome do Estado Funcional de Origem	`Observando Integracao`

*** Estado Funcional `Buscando Programador como Ajudante`

Z	Identificação do Estado Funcional
`Buscando Programador como Ajudante`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Move-se em direção ao Agente Programador mais próximo.
Ações	Move-se em direção ao Agente Programador (avistado) mais próximo.

Regras de Saída

38	Identificação da Regra de Saída
`Integracao Iniciada`	Nome da Regra
Nome do Estado Funcional de Origem	`Buscando Programador como Ajudante`
Nome do Estado Funcional de Destino	`Observando Integracao`
Identificação do Estado Funcional de Destino	X
Condição da Regra	Se o Agente Programador iniciou a Integração (`prog_Situacao = 31`).

Regras de Entrada

37	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	Q
Nome do Estado Funcional de Origem	`Acompanhando Programador`

*** Estado Funcional `Integrando Implementacao`

AA	Identificação do Estado Funcional
`Integrando Implementacao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	O agente, tendo acessado o Servidor de Integração livre, inicia a Integração das Histórias implementadas. A integração é efetuada através da adição do valor comercial da História recém implementada à variável `serv_ValorComercialAcumulado`. Em seguida, a integração é finalizada e o Servidor é liberado.
Ações	O Agente Programador assume a situação de integrador (`prog_Situacao = 31`). Sinaliza o Servidor de Integração como ocupado e adiciona o valor comercial da História recém implementada à variável `serv_ValorComercialAcumulado`. Finalmente, sinaliza o final da Integração e libera o Servidor de Integração.

Regras de Saída

43	Identificação da Regra de Saída
`Integracao Finalizada`	Nome da Regra
Nome do Estado Funcional de Origem	`Integrando Implementacao`
Nome do Estado Funcional de Destino	`Reinicializando Situacao do Programador`
Identificação do Estado Funcional de Destino	W
Condição da Regra	Se a integração for finalizada.

41	Identificação da Regra de Saída
`Sofro Intervencao`	Nome da Regra
Nome do Estado Funcional de Origem	`Integrando Implementacao`
Nome do Estado Funcional de Destino	`Observando Ajuda na Integracao`
Identificação do Estado Funcional de Destino	AB
Condição da Regra	Se o agente sofreu intervenção.

Regras de Entrada

36	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	S
Nome do Estado Funcional de Origem	`Buscando Servidor de Integracao`

42	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	AB
Nome do Estado Funcional de Origem	`Observando Ajuda na Integracao`

*** Estado Funcional `Observando Ajuda na Integracao`

AB	Identificação do Estado Funcional
`Observando Ajuda na Integracao`	**Nome do Estado Funcional**
Descrição do Estado Funcional	Observa andamento da integração, com uma probabilidade de 50% de intervir na mesma.
Ações	Com uma probabilidade de 50%, o agente intervém na integração.

Regras de Saída

42	Identificação da Regra de Saída
Decido Intervir	Nome da Regra
Nome do Estado Funcional de Origem	Observando Ajuda na Integracao
Nome do Estado Funcional de Destino	Integrando Implementacao
Identificação do Estado Funcional de Destino	AA
Condição da Regra	Se agente decidiu intervir na integracao.

44	Identificação da Regra de Saída
Integracao Finalizada	Nome da Regra
Nome do Estado Funcional de Origem	Observando Ajuda na Integracao
Nome do Estado Funcional de Destino	Reinicializando Situacao do Programador
Identificação do Estado Funcional de Destino	W
Condição da Regra	Se a integração for finalizada.

Regras de Entrada

41	Identificação da Regra de Entrada
Identificação do Estado Funcional de Origem	AA
Nome do Estado Funcional de Origem	Integrando Implementacao