SISTEMATIZAÇÃO DO DESENVOLVIMENTO DE JOGOS DE SIMULAÇÃO PARA TREINAMENTO

Gustavo Olanda Veronese

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Cláudia Maria Lima Werner, D.Sc.

Márcio de Oliveira Barros, D.Sc.

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Paulo Cesar Masiero, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 2004

VERONESE, GUSTAVO OLANDA

Sistematização do Desenvolvimento de Jogos De Simulação para Treinamento [Rio de Janeiro] 2003

XIII, 133 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Projeto de Jogos de Simulação para Treinamento

2. Dinâmica de Sistemas

I. COPPE/UFRJ II. Título (Série)

Agradecimentos

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

SISTEMATIZAÇÃO DO DESENVOLVIMENTO DE JOGOS DE SIMULAÇÃO PARA TREINAMENTO

Gustavo Olanda Veronese

Maio/2004

Orientadores: Cláudia Maria Lima Werner
Márcio de Oliveira Barros
Programa: Engenharia de Sistemas e Computação

Pesquisas recentes destacam os jogos de simulação como uma poderosa ferramenta de complemento aos métodos convencionais de ensino, que se baseiam em leituras e aulas expositivas. Isto pode ser verificado, especialmente, quando os jogos são aplicados em treinamento adulto, no qual enfatiza-se a experimentação como meio de fixação das teorias aprendidas pelos alunos. Em conseqüência, um mercado promissor de jogos emerge na indústria de treinamento.

No entanto, pouco existe na literatura sobre a organização do projeto de jogos de simulação para treinamento. Alguns dos trabalhos já desenvolvidos na área são específicos e pouco flexíveis.

Este trabalho propõe uma sistematização do desenvolvimento deste gênero de software que une a interatividade dos jogos à acurácia das simulações. A proposta consiste na definição de atividades de projeto e implementação, na descrição de notações de apoio à criação dos artefatos e na especificação de um ferramental de suporte ao desenvolvimento e execução dos jogos. O trabalho é direcionado ao desenvolvimento de um jogo de treinamento em gerência de projetos de software, que se baseia em um modelo de simulação previamente construído.

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

SISTEMATIZATION FOR THE DEVELOPMENT OF SIMULATION TRAINING GAMES

Gustavo Olanda Veronese

May/2004

Advisors: Cláudia Maria Lima Werner
Márcio de Oliveira Barros
Department: Computer and System Engineering

Use of simulation and games has been affirmed by several researches in adult training literature as an emerging technology that complement the conventional training methods (based on lectures and readings) in many disciplines.

However, in education technologies literature and in the industry there is a lack of well documented techniques aimed to organize development activities of training games based on simulations.

In this context we propose a systematization that empathizes separation of game concerns and the reuse of different aspects of simulation games. We also specified set of tools and notations that support development activities.

A training game for project managers was modeled as case study using this approach. The game is based on a simulation model and allows the students to test theirs management skills, like controlling and staffing.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto

As simulações e jogos têm sido utilizados como meios de transmissão do conhecimento organizacional na indústria. Em treinamento, auxiliam novos gerentes a absorverem conhecimento de maneira experimental, sem os riscos que uma pessoa inexperiente poderia causar em um ambiente real de operação.

O campo de jogos e simulações computacionais surgiu na década de 50 e tem sido desenvolvido e praticado por profissionais de uma variedade de disciplinas, como física, química, biologia e engenharia, especialmente em cibernética e em sistemas sociais. Em economia, a teoria matemática dos jogos tem atingido uma posição sólida entre muitas abordagens formais e empíricas (Klabbers, 2001). A simulação computacional tem raízes na matemática, teoria das probabilidades, teoria de jogos e em outras técnicas matemáticas associadas.

Muito têm sido pesquisado na área de treinamento baseado em jogos e simulações, tanto no campo de gerência de projetos de desenvolvimento de software (Drappa e Ludewig, 2000, Oh e van der Hoek, 2001, Pfahl e Ruhe, 2001), como na gerência em outras disciplinas (Maier e Stohhecker, 1996, Martin, 2000, Klabbers, 2002, Romme, 2002, P.E.D. *et al.*, 2002). O objetivo dessas pesquisas concentra-se na formação de profissionais capacitados a enfrentar as diferentes situações de tomada

de decisão com as quais um gerente pode se deparar durante o desenvolvimento de um projeto.

Os jogos são envolventes devido ao entretenimento e à interatividade. A atividade de jogar é definida como uma atividade não obrigatória, que tem o seu próprio espaço e tempo, é incerta quanto aos seus resultados, é governada por regras e apresenta elementos de mímica da realidade (Prensky, 2001).

O jogo The Sims (EA, 2004b) é considerado um dos jogos que envolvem simulação mais populares. Sua interface, antes apresentada nas simulações principalmente com números, relatórios e gráficos, é bastante intuitiva e engraçada. The Sims foi o sucessor do Sim City (EA, 2004a), para o gerenciamento de cidades. Sim City e os jogos que o sucederam incorporaram modelos de sistemas dinâmicos, incluindo equações lineares, equações diferenciais e autômatos, em que o comportamento de certos objetos originam das suas propriedades e regras de como estas propriedades interagem com seus vizinhos. Isto transmite maior realismo do que uma mera planilha de cálculos ou simulações simples (Prensky, 2001).

Prensky (2001) apresenta seis fatores estruturais que caracterizam um jogo: regras, metas e objetivos, resultados, conflito/competição/desafio/oposição, interação e enredo. Esses fatores estão presentes em quase todos os tipos de jogos.

- Regras: as regras diferenciam os jogos de outras formas de entretenimento. As regras impõem limites, que forçam os jogadores a seguirem caminhos específicos para atingir os objetivos;
- Metas e objetivos: As metas e objetivos são importantes, pois o ser humano é uma espécie direcionada a objetivos. Isto torna possível a concepção de estado futuro e elaboração de estratégias para atingi-lo. As regras, logicamente, tornam esse processo mais difícil, limitando as estratégias à disposição;
- Resultados e efeitos (*feedback*): É a maneira de medir o progresso em relação aos objetivos estabelecidos. O efeito aparece quando algo no jogo muda em resposta ao que o jogador faz. É o componente que insere a interatividade no jogo. É pelo *feedback* que o jogador é recompensado por algo, ou piora por ter

falhado em alguma atitude;

- Conflito, competição, desafio e oposição: Este conjunto de fatores estimula a criatividade do jogador. A manutenção do nível de conflito/competição/desafio em sincronia com as habilidades do jogador é chamada de "balanceamento" do jogo, e é fundamental para o sucesso de um projeto de jogo;

- Interação: A interação tem dois aspectos importantes. O primeiro é a interação do jogador com o computador, isto é, resultados são produzidos em função das ações do jogador e alteram o curso do jogo. O segundo é o aspecto social inerente aos jogos. Existe uma vasta quantidade de jogos no mercado que possibilitam sessões com múltiplos jogadores. Jogos deste tipo promovem a formação de grupos sociais;

- Enredo: O enredo faz o jogo representar algo e inclui os elementos de fantasia, como o espaço, épocas medievais, o mundo dos negócios ou guerras modernas.

O potencial de atração que os jogos exercem nas pessoas por meio dos fatores citados anteriormente e a precisão das simulações é uma combinação que vem sendo utilizada no ensino e apresenta um vasto campo de pesquisa (em usabilidade e cognição, técnicas de validação, ferramentas e metodologias de desenvolvimento, entre outros).

Apesar disso, existe um preconceito das pessoas que consideram os jogos como uma atividade trivial e sem importância, pois usualmente, os associam a uma atividade infantil. Na visão de alguns estudiosos, no entanto, o oposto é verdadeiro. O jogo tem um importância biológica, evolucionária, profunda, associada diretamente ao aprendizado. Tarefas interativas estimulam a criatividade. Quando essas tarefas são novas, muito esforço é dispensado no seu aprendizado e na sua exploração (Prensky, 2001).

1.2 Motivação

Pela pesquisa bibliográfica na área de jogos de simulação para treinamento pode-se perceber a falta de literatura voltada aos aspectos do desenvolvimento deste tipo de software. Especificamente, os protótipos dos trabalhos existentes apresentam forte acoplamento entre a máquina de execução do jogo e informações de cálculos do sistema modelado, dificultando a evolução tanto dos elementos que inserem no jogo fantasia (por exemplo, sua história e a a representação gráfica), como dos cálculos que regem o comportamento do sistema sob estudo. Por conseqüência, compromete-se a reutilização de parte destes jogos em problemas diferentes em um mesmo domínio de atuação e adaptação do jogo a diferentes platéias de aprendizes com diferentes níveis de conhecimento.

Parte deste problema pode estar associado à falta de suporte de ferramental e metodologias de desenvolvimento. No mercado de jogos de entretenimento, existem inúmeras ferramentas que auxiliam o desenvolvimento de alguns aspectos do jogo (como modelos gráficos, definição fases, etc), mas são proprietárias e de díficil integração com modelos de simulação. Une-se ao suporte inadequado de ferramentas, a falta de apoio na formalização e especificação dos aspectos que constituem os jogos de simulação.

1.3 Objetivos

O objetivo desta tese é propor uma sistematização para o desenvolvimento de jogos de simulação para treinamento em domínios complexos. A sistematização baseia-se na definição de atividades de desenvolvimento e artefatos produzidos nas atividades, na descrição de notações para representação dos artefatos e na implementação de um ferramental de apoio a execução dos jogos. Utiliza-se, como um estudo de caso, a modelagem e construção completa de um jogo de simulação para treinamento no domínio de Gerência de Projetos de Software.

A sistematização proposta fornece uma infra-estrutura para **desenvolvedores**

de jogos de treinamento. O desenvolvimento deve ocorrer com o auxílio de **especialistas do domínio** do jogo. Os jogos são destinados a **aprendizes** e as sessões de aprendizados devem ser acompanhadas por **facilitadores** com conhecimentos do domínio e do jogo.

Este trabalho não se propõe a abordar aspectos comerciais de jogos de simulação para treinamento. Prensky (2001) trata deste tema em detalhes. Além disso, não se busca apresentar uma taxonomia de jogos e simulações. Diversos outros trabalhos tratam deste tema com profundidade (Dempsey *et al.*, 1996, Schmucker, 1999, Prensky, 2001). Neste trabalho, busca-se utilizar os termos de consenso na literatura com as respectivas referências[1].

É importante destacar que os jogos de simulação devem executar um papel complementar no ensino. Sessões de treinamento correspondem, portanto, a uma parcela do processo de treinamento como um todo. Este trabalho preocupa-se em explorar os aspectos da tecnologia dos jogos de simulação e não discute todas as fases que precedem a decisão pelo desenvolvimento de um jogo para complementar o ensino. Discussões acerca deste tema podem ser encontradas na literatura da área pedagógica e de ensino mediado por computador (Greenblat, 1988, Murphy, 1997).

1.4 Organização

Esta monografia está organizada em 6 capítulos. Este capítulo apresentou o contexto e a motivação para o desenvolvimento desta pesquisa.

O capítulo 2 é uma revisão da literatura de jogos e simulações no ensino e discute com maior profundidade o contexto, os problemas atuais na área e alguns trabalhos relacionados.

O capítulo 3 aborda alguns conceitos importantes de simulações e questões do projeto de jogos, apresentando uma perspectiva de união dessas tecnologias em treinamento.

[1] Assunto abordado no capítulo 2

O capítulo 4 apresenta uma visão geral da sistematização do desenvolvimento de jogos de simulação, proposta neste trabalho.

O capítulo 5 detalha aspectos de implementação da abordagem e apresenta a aplicação da mesma em um estudo de caso: o desenvolvimento de um jogo gráfico de simulação para o treinamento em gerência de projetos.

Finalmente, o capítulo 6 discute as contribuições, limitações e possibilidades de trabalhos futuros.

Capítulo 2

Jogos e Simulações em Treinamento

2.1 Introdução

Este capítulo busca explorar o embasamento teórico que motiva o uso de jogos e simulações em treinamento. Esta visão acaba por conduzir à identificação de alguns problemas do uso deste tipo de técnica de ensino.

Além desta seção de introdução, este capítulo apresenta 6 seções. Na seção 2.2 são discutidos os termos utilizados na literatura de jogos e simulações. Na seção 2.3, é feito um estudo sobre as teorias educacionais de ensino adulto que justificam o uso de jogos e simulações como ferramentas de auxílio ao treinamento. Na seção 2.4, é feita uma análise sobre jogos e simulações como ferramentas de auxílio ao treinamento e o problema de avaliação da eficácia dessas ferramentas computacionais nesse contexto de uso. Na seção 2.5, são apresentados alguns casos de treinamento baseados em jogos e simulações. Finalmente, na seção 2.6, são apresentadas as conclusões do capítulo.

2.2 Jogos e Simulações

A simulação serve, usualmente, para duas propostas: científica e educacional (Rieber, 1996). As simulações científicas fornecem meios aos cientistas para estudar um sistema em particular. Por exemplo, um meteorologista pode estudar um tornado por simulações. Estas simulações auxiliam no estabelecimento e refinamento de uma teoria e no entendimento do sistema. As simulações educacionais são projetadas para ensinar alguém sobre um sistema pela observação do resultado de ações ou decisões, através do retorno fornecido pela simulação em tempo real, acelerado ou desacelerado (Rieber, 1996).

Greenblat (1988) considera que o termo jogo se aplica a simulações que funcionam inteira, ou parcialmente, com base nas decisões dos jogadores. Para Dempsey *et al.* (1996), jogos, diferentemente de simulação, são definidos como qualquer formato instrucional que envolve competição e é guiado por regras (*rule-guided*). Um formato competitivo não requer necessariamente dois ou mais participantes. Um aprendiz pode, por exemplo, estar competindo consigo mesmo, portanto jogando, se utilizar o resultado de uma simulação (*score*) como parâmetro para superação em diversas tentativas de simulação (Dempsey *et al.*, 1996).

Nos jogos de simulação, o ambiente e as atividades dos participantes têm características de jogos: jogadores possuem papéis para desempenhar, metas a serem atingidas, atividades para realizar, restrições e recompensas (positivas e negativas) como resultado de suas ações e das ações de outros elementos no sistema (Greenblat, 1988). Greenblat (1988) conclui que jogos de simulação, na forma híbrida, envolvem a realização de atividades de jogos em contextos simulados.

Outro termo encontrado na literatura de simulações é o "Simulador de Vôo de Gerência" (*Management Flight Simulator* - MFS). Segundo Saunders (2002), um MFS é a soma de uma interface computacional dinâmica baseada em quadros e uma máquina de simulação dinâmica. A essência da interface é o movimento e uso de símbolos: uma interface simples pode conter barras de controle como entradas e gráficos de duas dimensões como saída. A figura 2.1 apresenta um exemplo deste

tipo de interface. Interfaces mais complexas podem envolver áudio e vídeo, interação com reconhecimento de voz, luvas com sensores, entre outros recursos.

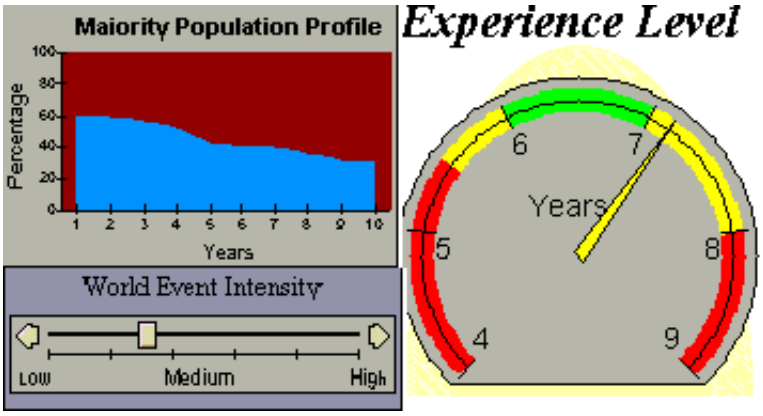

Figura 2.1: Controles de um Simulador de Vôo de Gerência

Maier e Größler (1998) criticam o emprego deste termo, que tenta buscar a analogia da aviação. Eles apontam duas desvantagens: (i) o termo não é adequado para aqueles que não têm o inglês como língua mãe; e (ii) simuladores de negócio não objetivam cobrir a realidade de forma tão congruente como simuladores de vôo reais. Enquanto simuladores de vôo tentam ser o mais realista possível, simuladores de negócio tentam abstrair detalhes, possibilitando a ênfase em importantes estruturas e comportamentos.

2.3 Jogos e Simulações no Contexto Educacional

A argumentação de que jogos e simulações podem ser instrumentos eficazes no processo educacional deve passar, obrigatoriamente, pela busca de teorias educacionais que reforcem tal idéia. As teorias educacionais tentam explicar como o aprendizado ocorre. Diversos trabalhos fazem a conexão da tecnologia de jogos com a sua justificativa de uso em educação (incluindo o treinamento adulto) baseados nestas teorias educacionais (Greenblat, 1988, Spector, 2000, Größler, 2000, Ahdell e Andresen, 2001).

As novas correntes consideram que uma abordagem mais experimental no aprendizado pode aumentar sua eficácia. As simulações computacionais podem oferecer

às pessoas a oportunidade de participar, cometer erros, ter chances, se desafiar e aprender (Ahdell e Andresen, 2001).

Greenblat (1988) destaca um conjunto de objetivos educacionais e de treinamento para os quais os jogos e simulações são projetados:

- aumento da motivação e interesse em um assunto abordado, no campo geral de estudo e na pesquisa sobre um tópico;
- ensino, transmitindo e reforçando a informação já fornecida em outro formato;
- construção de um entendimento sistêmico, capaz de relacionar o papel exercido por um elemento específico com o sistema como um todo;
- desenvolvimento de habilidades como a análise e o pensamento crítico, tomada de decisões, negociação, comunicação, preparação de orçamento, gerenciamento de pessoas em uma equipe e reação a emergências, e
- auto-avaliação pela percepção do próprio conhecimento, habilidades, suposições, atitudes, ou habilidades de liderança e, também, avaliação por terceiros pela avaliação por professores ou treinadores das características citadas.

O papel que o entretenimento exerce em relação à motivação tem dois lados. Primeiro, a motivação promove o desejo pela recorrência da experiência. Segundo, o entretenimento pode motivar aprendizes a se engajarem em atividades com as quais eles tem pouca ou nenhuma experiência anterior (Prensky, 2001).

Todos os ambientes de aprendizado motivantes possuem as seguintes características: desafio, curiosidade, fantasia e controle (Rieber, 1996). O jogo é a ferramenta instrucional que mais fornece esses componentes motivacionais (Rieber, 1996). Um jogo de simulação combina características de jogos (competição, cooperação, regras, participantes e papéis) e simulação (abstração da realidade por um modelo) (Basnet, 1996). Esta abordagem integra o conhecimento de várias disciplinas científicas e tenta tornar contextos de vivência complexos (a dinâmica de sistemas naturais, sociais, tecnológicos e econômicos e seus efeitos) mais fáceis de entender (Kriz e Rizzi, 1999).

A teoria mais referenciada na literatura que aborda o ensino mediado por computador é o Construtivismo. Nesta teoria, o aprendizado é um processo ativo no qual os aprendizes constroem novas idéias ou conceitos baseados no seu conhecimento passado. O aprendiz seleciona e transforma a informação, constrói hipóteses e toma decisões, contando com uma estrutura cognitiva para fazê-lo. A estrutura cognitiva (modelos mentais) provê significado e organização às experiências e possibilita ao indivíduo "ir além da informação dada" (Kearsley, 2001). A seguir, destacam-se os princípios e as características do construtivismo (Kearsley, 2001, Murphy, 1997):

- a instrução deve estar relacionada com as experiências e contextos que fazem o querer e estar apto a aprender;
- a instrução deve estar estruturada tal que ela seja facilmente captada pelo estudante;
- a instrução deve ser planejada para facilitar a extrapolação e/ou preencher lacunas (deve ir além da informação dada);
- o aprendizado adulto deve ser centrado no problema em vez de ser centrado no conteúdo;
- os professores devem exercer o papel de guias, monitores, técnicos, tutores e facilitadores. Atividades, oportunidades, ferramentas e ambientes devem ser providos para encorajar a metacognição ("aprender como aprender");
- o estudante tem o papel central em mediar e controlar o aprendizado;
- situações de aprendizado, ambientes, habilidades, conteúdo e tarefas devem ser relevantes, realistas, autênticos e representar a complexidade do mundo real;
- a construção do conhecimento (e não reprodução) deve ser enfatizada. Esta construção toma lugar em contextos individuais e através da negociação social, colaboração e experiência;
- as construções de conhecimento anterior do aprendiz, crenças e atitudes devem ser consideradas no processo de construção do conhecimento;

- a resolução de problemas, habilidades de pensamento de alta-ordem e conhecimento profundo devem ser enfatizados; e
- os erros provêem a oportunidade para o discernimento entre as construções de conhecimento anteriores do estudante.

O Construtivismo é uma teoria de conhecimento com raízes na filosofia, psicologia e cibernética. Um problema que surge é saber como esta teoria de conhecimento se traduz na prática. Argumenta-se que não há um método, abordagem ou uma pedagogia particular para sua operacionalização. Baseada nesta questão, Murphy (1997) desenvolveu uma lista de requisitos que caracterizam projetos, atividades e ambientes construtivistas. A lista apresentada sugere a adequação de jogos e simulações como ambientes de forte caráter construtivista.

O Construtivismo sugere que é vital reconhecer o papel ativo do indivíduo na construção de representações e interpretações da realidade. A promoção de processos ativos e construtivos no aprendiz tem sido uma parte vital do suporte ao aprendizado e do projeto instrucional, mas especificamente com relação ao aprendizado avançado sobre domínios complexos (Spector, 2000).

Outra teoria referenciada na literatura sobre aprendizado baseado em jogos e simulações é a Andragogia. A Andragogia é uma tentativa de desenvolver uma teoria específica para o aprendizado adulto. Nesta teoria, enfatiza-se que os adultos são auto-direcionados e esperam ter responsabilidade para decisões. A Andragogia faz as seguintes suposições sobre o planejamento do aprendizado (Kearsley, 2001):

- adultos necessitam saber porque eles necessitam aprender algo;
- adultos são mais interessados em assuntos de relevância imediata para o trabalho ou vida pessoal;
- adultos abordam o aprendizado como resolução de problemas;
- adultos necessitam aprender experimentalmente. A experiência (incluindo os erros) provê a base para as atividades de aprendizado;

- adultos necessitam ser envolvidos no planejamento e avaliação de sua instrução.

As simulações apresentam características que se mostram adequadas a esta teoria, pois provêem um ambiente experimental, muitas vezes dedicado à resolução de problemas (como o equilíbrio de algumas variáveis, por exemplo, tempo e custo em um projeto) e, se projetadas de maneira fiel ao domínio analisado, podem proporcionar uma reflexão e mudança de comportamento em relação a esse domínio.

A Andragogia estabelece que a instrução deve enfatizar mais o processo e menos o conteúdo sendo ensinado. Estratégias como estudos de caso, simulações e auto-avaliação são úteis neste modelo de ensino. Os instrutores adotam o papel de facilitador em vez de avaliador ou professor.

Greenblat (1988) identifica algumas desvantagens do ensino convencional baseado em aulas com discussão, quando comparado aos jogos de simulação. As principais desvantagens são exibidas na tabela 2.1.

Os participantes de jogos se tornam aprendizes ativos, devem tomar decisões. Além disso, os jogos são úteis para a transmissão de características de um sistema, pois se utilizam de modelos gráficos e diagramas que, em alguns casos, são mais eficazes que descrições verbais (Greenblat, 1988).

Desvantagens das aulas com discussão	**Solução por simuladores**
Aprendizes mais passivos	Aprendizes devem ser ativos
O material deve ser apresentado em uma ordem seqüencial determinada pelo professor	Apenas a tarefa determina a ordem do material que pode, portanto, ser selecionado pelo aprendiz
As discussões são caracterizadas por hierarquias sociais ou o que é socialmente desejável	As discussões são caracterizadas por experiências feitas no simulador
Pontos de vista holísticos e sistêmicos são mais difíceis de mediar	Os estudantes são induzidos a atingir uma visão sistêmica para poder gerenciar o simulador com eficácia
Descrições verbais são interpretadas diferentemente	Os termos são definidos pelo seu uso na simulação

Tabela 2.1: Comparação entre o ensino convencional e o uso de simuladores (Greenblat, 1988)

Para Forrester (1991), os métodos educacionais tradicionais tendem a desencorajar a síntese e o uso do conhecimento que um estudante tenha adquirido. A educação convencional ensina fatos em vez das dinâmicas de mudanças naturais e sociais. Apenas quando considerações dinâmicas forem introduzidas no processo educacional, os estudantes terão tempo para desenvolver melhores modelos mentais que guiem suas ações.

Hsu (1989) estabelece, em uma visão geral, quatro fases que norteiam o processo de aprendizado:

- **Retenção de informação**: envolve a recepção de informação apresentada por meio de aulas, leituras e algum meio de memorização da informação apresentada;
- **Organização do conhecimento**: envolve a participação ativa do estudante em discussões, workshops e apresentações de estudos de caso;
- **Experimentação**: possibilita aos estudantes praticar certos comportamentos em um ambiente monitorado e controlado pelo professor;
- **Afirmação**: Esta fase envolve observação, medição, discussão e outras formas de fornecer retorno aos estudantes por especialistas, com o objetivo de correção de procedimentos incorretos e reforço de hábitos corretos.

Uma das regras que governam o aprendizado é que as pessoas lembram-se melhor do que elas sentem, ou seja, fazer do aprendizado uma experiência emocional intensa pode torná-lo mais eficaz (Ahdell e Andresen, 2001). Assim, o uso de jogos pode fornecer este ingrediente ao aprendizado por ser uma tecnologia que utiliza o divertimento e engajamento do aprendiz.

Os participantes em jogos de simulações podem aprender na prática quais fatores e dinâmicas são efetivos nos vários subsistemas que são relevantes para eles. Desta maneira, podem ser desenvolvidas as habilidades para uma melhor resolução de problemas, a cooperação em uma equipe e comunicação eficiente, a liderança e as estruturas organizacionais (Kriz e Rizzi, 1999).

2.4 Eficácia do Aprendizado com Jogos e Simulações

Embora possa se alegar, de acordo com os relatos da seção anterior, que as ferramentas computacionais com base em simulação se constituem em ferramentas promissoras no auxílio ao treinamento, pouco se apresentou até o momento de evidência empírica de sua eficácia (Größler, 2000).

Spector (2000) afirma que a comunidade deve sempre cobrar fortes evidências em relação a melhora de aprendizado devido a alguma nova abordagem ou tecnologia, dado que os métodos de avaliação de aprendizado são bastante complexos. Spector e Davidsen (1998) destacam que a eficácia de ambientes de aprendizado não é bem documentada. Além disso, não existe uma metodologia bem estabelecida para guiar o projeto e implementação desses ambientes de aprendizado.

De fato, existe pouca evidência na literatura de que simuladores são tão eficazes quanto se diz que são. Muitos dos trabalhos não apresentam evidências empíricas ou, quando apresentam, freqüentemente mostram problemas metodológicos (Größler, 2000). Entre as possíveis causas para a falta de indícios da eficácia de ferramentas de simulação quando aplicadas ao ensino, destacam-se:

- **A eficácia da simulação depende da fidelidade do modelo ao mundo real:** a fidelidade é o nível de realismo em que a simulação se apresenta ao aprendiz. As simulações não precisam ser réplicas exatas de um sistema. As simulações simples podem auxiliar os gerentes pela ênfase em variáveis importantes, atuando de forma bastante positiva do ponto de vista do aprendizado, mas atuando de forma pobre como uma ferramenta de modelagem de fenômenos do mundo real. Feinstein e Cannon (2001) apresentam uma série de estudos que atestam que um alto nível de fidelidade não se traduz em um aprendizado mais efetivo. Um dos estudos afirma que alta fidelidade pode dificultar o treinamento e o aprendizado, porque superestimula aprendizes novatos;

- **A dificuldade de validação:** a validação do modelo de um sistema complexo não é trivial e requer análise cuidadosa. Um modelo é válido e realístico quando aceito pelas pessoas que são familiares com o sistema (por exemplo, gerentes de projeto experientes). Um modelo válido deve reagir apropriadamente a condições de teste extremas e deve refletir dados históricos (Spector, 1998). A validação de simuladores é ainda um tópico de pesquisa em aberto (Feinstein e Cannon, 2001);

- **A dificuldade de se avaliar os resultados da simulação:** baseado em alguns estudos, Basnet (1996) constatou que resultados de sessões de aprendizado com jogos de simulação possivelmente influenciam na opinião dos aprendizes sobre jogos. De acordo com o autor, existe a impressão de que apenas os vencedores do jogo apresentam opiniões positivas, dado o caráter competitivo dos jogos;

- **A inexistência de métodos e medições que avaliem alterações na curva de aprendizado:** pode-se argumentar a favor de um determinado método instrucional quando se verifica o efeito positivo do uso do método no aprendizado em relação a outros métodos. Isto ocorre pela extração e análise de medidas produzidas no aprendizado (Spector, 1998). A dificuldade está na extração e análise dessas medidas. Há consenso na literatura em relação à ausência de um método sistemático para determinação da eficácia de aprendizado por estratégias instrucionais alternativas (Spector, 2000, Größler, 2000, Cavaleri *et al.*, 2002). Cavaleri *et al.* (2002) apresentam algumas possíveis justificativas para tal. A mais significativa destaca que a atenção na avaliação dessas estratégias enfatiza a medição de mudanças em modelos mentais. Esta abordagem, como os autores colocam, se mostra bastante difícil de se por em prática pela falta de uma definição operacional que possibilite uma medição precisa. Isto se constitui em um fator limitante na pesquisa em modelos mentais.

2.5 Casos de Treinamento baseado em Jogos e Simulações

Nesta seção, são apresentados alguns casos de treinamento em gerência de projetos baseado em jogos de simulação. São ainda avaliados os pontos positivos e negativos de cada uma das abordagens, analisadas a seguir.

2.5.1 SimSE

Em (Oh *et al.*, 2004), é proposto um método de ensino de processos de desenvolvimento de software, tanto no nível individual quanto organizacional. O método baseia-se em um ambiente gráfico de jogos de simulação, denominado SimSE. Os autores apresentam a hipótese de que a abordagem proposta possibilita aos indivíduos um entendimento dos processos de software utilizados nas suas organizações, enquanto que as organizações podem explorar os diferentes aspectos de seus processos de desenvolvimento.

O jogo é monousuário e o jogador assume o papel de um gerente de projetos em uma equipe de desenvolvedores. O jogador deve gerenciar os desenvolvedores para completar algum aspecto particular de um projeto de engenharia de software. As atividades de gerenciamento incluem, entre outras, a contratação e demissão de desenvolvedores, alocação de tarefas, monitoramento do progresso e a compra de ferramentas.

A interface definida para o ambiente é totalmente gráfica. É exibido um escritório virtual no qual o processo de engenharia de software se desenvolve, incluindo todos os elementos que compõem este cenário de escritório (mesas, cadeiras, computadores, salas de reunião), desenvolvedores, clientes e informações do projeto (orçamento e tempo, por exemplo), bem como representação dos artefatos (documentos de análise, projeto e código, por exemplo) que incluem informações de corretude, completude e outras qualidades. O jogo também apresenta graficamente informações acerca do estado dos desenvolvedores (que indicam o grau de satisfação com o salário ou o

quanto ele está ocupado). As decisões do jogador devem se basear nestas informações. Os modelos de processo são adaptáveis aos diferentes objetivos de ensino. Para tornar isso possível, um construtor de modelos é disponibilizado para especificação de um modelo de processo.

Este trabalho ainda se encontra em andamento (Oh *et al.*, 2004). Um protótipo do ambiente gráfico foi desenvolvido com base em um modelo específico de processo e exibe as informações dos desenvolvedores, artefatos, projetos, ferramentas e clientes em tabelas e mensagens textuais.

2.5.2 Projeto SESAM

Drappa e Ludewig (2000) apresentam o projeto SESAM, utilizado na educação da engenharia de software pelo uso de simulações, em que o estudante atua como um gerente de projetos de software.

O simulador construído realiza a interação com o usuário por meio de textos. O jogador pode contratar ou despedir desenvolvedores para desempenhar funções específicas (como a preparação de um especificação, revisão de um documento de projeto ou testes). A figura 2.2 ilustra uma tela de interação do simulador.

Ao final do jogo, o jogador recebe seu resultado que pode ser processado por uma ferramenta de análise. A ferramenta exibe graficamente as variáveis internas que são omitidas durante a sessão de jogo. O modelo do projeto de software sobre o qual o jogador atua é implementado em separado do simulador, permitindo a utilização de diferentes modelos sem impacto na infra-estrutura.

O simulador atua tanto na forma contínua, quanto na discreta (dirigida a eventos). Os modelos de projeto de software são divididos em uma parte estática, que descreve a estrutura do processo (pessoas, documentos, relações, etc.), e outra dinâmica, que descreve as pré-condições e conseqüências de mudanças provocadas pelas ações do jogador.

No trabalho, são relatados resultados de experimentos baseados em um modelo

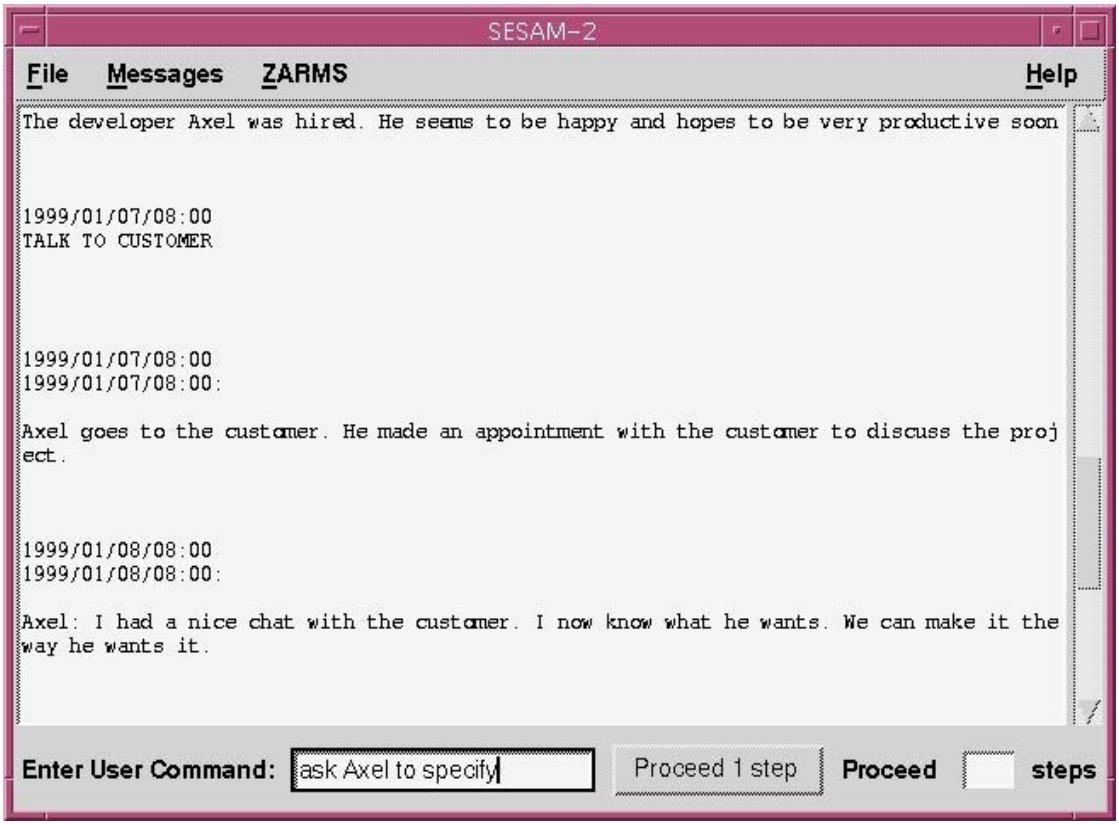

Figura 2.2: Interface do jogo de simulação SESAM (Drappa e Ludewig, 2000)

para o ensino de qualidade de software. Os estudos se mostraram inconclusivos quanto à eficácia do aprendizado com o uso do simulador. Algumas das conclusões relatadas pelos autores foram:

- os estudantes não gostam de planejar: a maioria dos participantes não planejou os projetos simulados; os que planejaram obtiveram melhores resultados;
- os estudantes cometem os mesmos erros: nos experimentos, observou-se que muitos erros cometidos no pré-teste foram repetidos no pós-teste;
- os estudantes não refletem sobre os detalhes de seus resultados;
- os estudantes não se esforçaram para analisar seus resultados. Este problema pode ter sido acarretado pela falta de clareza da exposição dos resultados pelo simulador.

Os autores concluem que apenas "jogar" não é suficiente, pois estudantes não

estão aptos a entender as razões pelas quais eles falharam.

2.5.3 *The Incredible Manager*

O jogo *The Incredible Manager* (Dantas, 2003) é um jogo de simulação baseado em modelos dinâmicos para aplicação em um contexto de treinamento de gerentes de projetos de software.

O estudante atua como gerente, planejando e controlando projetos de software. No contexto do jogo proposto, é considerado um projeto de sucesso aquele que é desenvolvido dentro das estimativas previstas para ele pelo próprio gerente. Este sucesso pode ser medido, quantitativamente, pelo tempo de conclusão e o custo do projeto.

A interação do estudante com o jogo, tendo em vista o treinamento aplicado no gerenciamento de projetos de software, aborda as principais etapas do gerenciamento de projetos, tais como planejamento, gerência de recursos humanos, direção e controle. É um jogo gráfico em tempo contínuo e apresenta como personagens desenvolvedores e o patrocinador. O jogo se baseia em um simulador, um modelo de simulação e uma máquina de jogo projetada específicamente para o modelo de simulação.

O jogo tem duas etapas principais. Uma etapa de planejamento na qual o jogador deve produzir o plano de projeto a ser executado durante o desenvolvimento do produto e uma etapa de execução do projeto, na qual ocorre a simulação do desenvolvimento do produto de software.

Um estudo de foco qualitativo foi executado em ambiente acadêmico com alunos de graduação e pós-graduação em Computação e indicou a viabilidade do uso de jogos de simulação no treinamento de gerentes de projetos. Conforme relatos dos participantes, o treinamento não foi considerado cansativo, e sim estimulante, prático e divertido. A interface foi considerada atrativa e amigável ao usuário, fazendo com que o tempo passasse despercebido durante o treinamento. O nível de dificuldade alto foi apontado como fator que aumentou a atenção e a competição na busca

por melhores resultados na execução do jogo de simulação. Como desvantagens, os participantes citaram as limitações do modelo de simulação em capturar todos os aspectos existentes nas dinâmicas de projetos reais (como, por exemplo, os aspectos de relacionamento inter-pessoal).

Pela disponibilidade do código fonte do jogo, pode-se verificar que, apesar do mesmo ter sido desenvolvido de acordo com princípios de projeto de software, alguns problemas que poderiam dificultar sua extensão. O principal está relacionado ao controle de estados dos elementos do jogo. Percebe-se que há uma mistura das dimensões de estados gráficos (como andando, sentado) e estados lógicos dos personagens (como cansado ou ocioso). Este controle é distribuído em complexas estruturas condicionais da linguagem de programação utilizada. Assim, criações de novos estados e personagens seriam dificultadas em função deste alto grau de acoplamento.

2.5.4 *Contract & Construct*

Martin (2000) descreve uma implementação chamada *Contract & Construct* que modela uma abordagem de gerência de contratos em um projeto. A gerência de contratos delega o controle de recursos a sub-contratantes.

O autor caracteriza a implementação como um jogo que utiliza simulação. Foi desenvolvida para auxiliar no ensino de gerência de projetos para alunos de Mestrado, MBA e graduação em uma escola de negócios da Inglaterra. Seu objetivo é prover uma experiência em gerenciamento de projetos que destaque:

- contrastes entre custo, qualidade (ou satisfação do cliente), tempo e segurança, para atingir os objetivos determinados;
- a prática de planejamento de projeto;
- a aplicação de técnicas de gerenciamento de projetos;
- alguns aspectos mais sutis da gerência de projeto, como gerenciamento de pessoal, e

- a necessidade de conviver com as conseqüências de estratégias e decisões.

Alguns elementos do jogo são simulados, como a negociação entre o gerente de projetos e o contratante. Os jogadores são divididos em grupos e competem com outras equipes. Um projeto de engenharia na área química foi modelado, com base na literatura e em discussões com especialistas.

O participante em uma simulação tem o papel do gerente do projeto. O jogo é dividido em duas fases. A primeira envolve a seleção de contratantes para as atividades do projeto. Na segunda fase, ocorre a execução do projeto. Durante a execução podem ocorrer diversos eventos, planejados ou não. As decisões tomadas em reação aos eventos podem afetar o custo, a duração, a qualidade ou segurança do projeto.

No trabalho descrito, o autor relata positivamente os resultados apresentados. A interface com o usuário, basicamente, apresenta os indicadores a serem regulados e a descrição textual dos eventos. A implementação de uma nova rede de atividades, diferente da planejada para o projeto de engenharia modelado, requer um certo esforço de programação (Martin, 2000).

2.6 Conclusões

Conforme o que foi apresentado neste capítulo, existe a percepção de que as simulações, isoladamente e também combinadas com os jogos, podem ser um instrumento poderoso e complementar ao ensino tradicional. Segundo Hsu (1989), no campo da educação gerencial, jogos de simulações devem ser pensados e projetados mais como um meio de prover aprendizado experimental do que um veículo de distribuição de conhecimento, e mais como um ambiente no qual professores e aprendizes trabalham juntos em vez de um pacote de auto-aprendizado.

Os jogos de simulação podem atuar provendo um contexto de experimentação propício para a exploração em profundidade de domínios complexos de aprendizado, como o da gerência de projetos, sistemas sociais, econômicos, entre outros.

Muito tem sido pesquisado, porém a área está em sua infância, sendo poucos os resultados palpáveis que indicam sua eficácia no treinamento. Há muito a ser feito na área de treinamento baseado em jogos e simulações. Pode-se destacar: a necessidade de construção de ambientes mais amigáveis para sessões de experimentação e treinamento, o estudo de formas de validação de modelos de simulação, a avaliação de eficácia do aprendizado e formas de organização do processo de desenvolvimento de jogos de simulação para treinamento. Esta pesquisa é concentrada neste último tópico.

O desenvolvimento de processos e infra-estruturas de suporte deve passar pelo estudo das técnicas de auxílio a construção tanto em simulação quanto em jogos de treinamento. O projeto de jogos e simulações é objeto do próximo capítulo.

Capítulo 3

Projeto de Jogos de Simulação

3.1 Introdução

O projeto de jogos de simulação para treinamento abrange duas áreas principais: a primeira trata do desenvolvimento de modelos de simulação, que representam um determinado problema ou fenômeno em um domínio alvo. O modelo fornece meios para compreender o problema ou fenômeno através da análise dos relacionamentos entre seus elementos. A segunda área referida está relacionada ao projeto de jogos especificamente, considerando a elaboração de histórias para contextualizar o jogo, mecanismos de interatividade empregados, tecnologias que facilitem o seu desenvolvimento, entre outros aspectos.

Embora o objetivo desta pesquisa seja a discussão sobre jogos de treinamento com fins didáticos, muito pode ser aproveitado do que já foi pesquisado no projeto de jogos de entretenimento. Isto porque os jogos, de maneira geral, compartilham um conjunto de características como a interatividade, a presença de elementos de ficção e a não-linearidade, conceitos discutidos mais adiante neste capítulo (seção 3.4). Desta maneira, é importante analisar o que existe no projeto de jogos que pode auxiliar o desenvolvimento de jogos de simulação específicos para treinamento. É importante discutir as particularidades dos jogos em relação a sistemas convencionais, tanto no que diz respeito ao produto quanto ao processo.

Este capítulo trata desses tópicos e apresenta uma perspectiva de unificação das duas áreas, identificando as necessidades tecnológicas que surgem dos jogos e simulações no ensino. As discussões são acompanhadas de exemplos específicos do domínio de Gerenciamento de Projetos de Software.

A próxima seção discute os conceitos de modelos mentais e modelos computacionais, para introduzir a importância dos modelos de simulação. A seção 3.3 apresenta a técnica de simulação chamada Dinâmica de Sistemas. A seção 3.4 aborda o projeto de jogos de simulação voltados para treinamento e a última seção apresenta as conclusões deste capítulo.

3.2 Modelos Mentais e Modelos Computacionais

Os seres humanos baseiam suas ações não no mundo real, mas em imagens mentais do mundo, dos relacionamentos entre suas partes e da influência que as ações exercem no mundo (Sterman, 1988). Um **modelo mental** é a percepção mental formada por um indivíduo sobre as interações que ocorrem em um sistema e do comportamento que estas interações produzem (Martin, 1997).

Todo modelo é uma representação de um sistema - um grupo de elementos funcionalmente inter-relacionados que formam um todo complexo. Este sistema pode ser um sistema ecológico, social, ou, por exemplo, um projeto de engenharia. A utilidade dos modelos está no fato deles simplificarem a realidade. Os modelos mentais apresentam algumas vantagens em relação aos modelos computacionais, que são (Sterman, 1988):

- podem levar em conta uma grande quantidade de informações, não apenas numéricas, e
- são flexíveis, podem ser adaptados a novas situações e modificados quando novas informações se tornam disponíveis.

Porém, esses modelos apresentam desvantagens (Sterman, 1988):

- não são facilmente compreendidos por outros;
- as interpretações de modelos mentais podem diferir, e
- as hipóteses nas quais eles são baseados são usualmente difíceis de examinar, de maneira que ambigüidades e contradições entre eles podem não ser detectadas ou resolvidas.

Os modelos mentais que o ser humano utiliza são simples e muitas vezes esses modelos são também falhos, pois erros são cometidos na dedução de conseqüências de hipóteses nas quais eles são baseados.

Os modelos de simulação pertencem à categoria dos modelos computacionais. Todo modelo de simulação inclui uma representação do mundo físico relevante ao problema sob estudo. Além disso, esses modelos devem representar o comportamento dos atores no sistema. O comportamento significa a maneira como as pessoas respondem a diferentes situações, ou seja, como elas tomam decisões. O componente comportamental é colocado no modelo na forma de regras de tomada de decisão, que geralmente são determinadas pela observação direta dos procedimentos reais de tomada de decisão no sistema (Sterman, 1988).

Os modelos computacionais, por sua vez, apresentam diversos problemas (Sterman, 1988). Primeiramente, são modelos "caixa preta", freqüentemente mal documentados e complexos. Além disso, são incapazes de lidar com relacionamentos e fatores qualitativos e descritivos, comuns no mundo real, para os quais não existem dados numéricos, ou estão além do conhecimento dos especialistas que construíram o modelo. Essas informações, chamadas de variáveis subjetivas, são difíceis de quantificar e costumam desempenhar papéis importantes nos modelos. Por exemplo, a produtividade de um desenvolvedor ou a qualidade de um produto são importantes em um modelo de simulação em gerência de projetos de software e, no entanto, são informações difíceis de quantificar.

Apesar dos problemas apresentados, os modelos computacionais oferecem vantagens em relação aos modelos mentais em diversos aspectos (Sterman, 1988): são explícitos, suas hipóteses são descritas em documentos; as conseqüências lógicas das

hipóteses do modelo são calculadas e podem inter-relacionar muitos fatores simultaneamente.

Considerando que as pessoas tomam decisões com base em seus modelos mentais, o aperfeiçoamento desses modelos por meio de treinamento pode constituir maior entendimento acerca de um problema ou fenômeno e da estrutura que produz o comportamento deste problema ou fenômeno (Martin, 1997). Modelos mentais incompletos ou incorretos induzem os aprendizes a cometer erros em situações reais, em que o conhecimento sobre um determinado sistema deve ser aplicado.

3.3 Dinâmica de Sistemas

A **Dinâmica de Sistemas** combina a teoria, os métodos e a filosofia necessária para analisar o comportamento de sistemas não apenas em gerência, mas também no impacto ambiental, na política, no comportamento econômico, na medicina, na engenharia e em outros campos. É uma técnica e uma linguagem para a descrição de sistemas complexos, focalizando seus aspectos estruturais (Forrester, 1961). Um modelo em Dinâmica de Sistemas é uma representação matemática da estrutura de um sistema.

O processo de modelagem separa a consideração de hipóteses existentes (estrutura, políticas e parâmetros) do comportamento implicado (Forrester, 1991). O comportamento se refere ao modo como os elementos ou variáveis que compõem o sistema variam ao longo do tempo. Uma vez construído o modelo e especificadas suas condições iniciais, um computador pode simular o comportamento (Martin, 1997).

Na Dinâmica de Sistemas, os limites do modelo são estabelecidos de maneira que as relações de causa e efeito estejam no interior do sistema. O **comportamento endógeno** do modelo é enfatizado, isto é, o comportamento do modelo é resultado de sua estrutura interna (Forrester, 1991).

No paradigma da Dinâmica de Sistemas, identificam-se e modelam-se relações de

causa e efeito e ciclos de realimentação presentes em um sistema. Estas relações são descritas em modelos de fluxos compostos de quatro elementos básicos: repositórios, fluxos, processos e conectores.

Um **repositório** representa um elemento que pode ser gradualmente acumulado ou consumido ao longo do tempo. Um repositório é descrito por seu nível, que indica o número de elementos nele contidos em um determinado instante.

Um **fluxo** representa a taxa de variação de um repositório ao longo do tempo. Um fluxo indica o número de elementos que ingressam ou são retirados de um repositório a cada instante de tempo e, matematicamente, é a derivada parcial do nível do repositório em relação ao tempo, representando os acréscimos e reduções do nível do repositório através de uma equação.

Um **processo** é utilizado para calcular informações a partir de um conjunto de parâmetros. Um parâmetro pode ser o valor de um fluxo, o nível de um repositório ou o resultado de outro processo. Estes parâmetros determinam uma equação que representa o processo. A informação resultante de um processo pode ser utilizada em outros processos, ou como parte da equação que irá determinar a taxa de variação de um ou mais fluxos.

Um **conector** é uma via de transmissão de informações no modelo. Assim como os fluxos transferem elementos entre repositórios ou entre repositórios e os limites do sistema, um conector transfere informações entre dois processos, entre um repositório e um processo, entre um processo e um fluxo, ou entre um fluxo e um processo.

A representação gráfica dos modelos em Dinâmica de Sistemas pode ser feita nos chamados diagramas de repositórios e fluxos. Estes diagramas apresentam os quatro blocos básicos da técnica: repositórios, fluxos, processos e conectores. Além disso, são definidos elementos produtores e consumidores infinitos, discutidos mais adiante nesta seção. A figura 3.1 apresenta um diagrama de repositórios e fluxos.

Um repositório é representado por um retângulo. Um fluxo é representado por uma seta, conectada a um ou dois repositórios. Nos fluxos conectados a um único repositório, a ponta livre é conectada a um produtor infinito ou a um consumidor

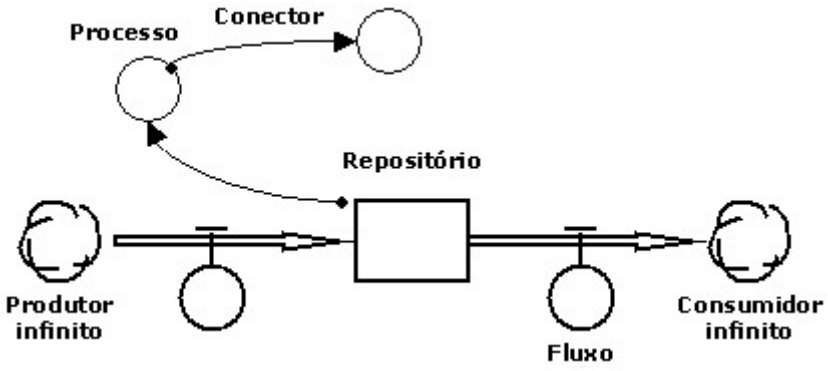

Figura 3.1: Representação diagramática dos construtores da Dinâmica de Sistemas

infinito. Um produtor infinito fornece uma quantidade ilimitada de unidades a um repositório por meio de um fluxo. Por outro lado, um consumidor infinito absorve uma quantidade ilimitada de unidades de um repositório através de um fluxo. Os produtores e consumidores infinitos determinam os limites do sistema e são representados por nuvens. Os processos são representados por círculos. Um conector é representado por uma seta que indica a origem e o destino da informação transmitida.

Para efeito de ilustração da técnica, considere um exemplo de um modelo simplificado de produção de software (Madachy, 2001). A figura 3.2 ilustra o diagrama de repositórios e fluxos desse modelo.

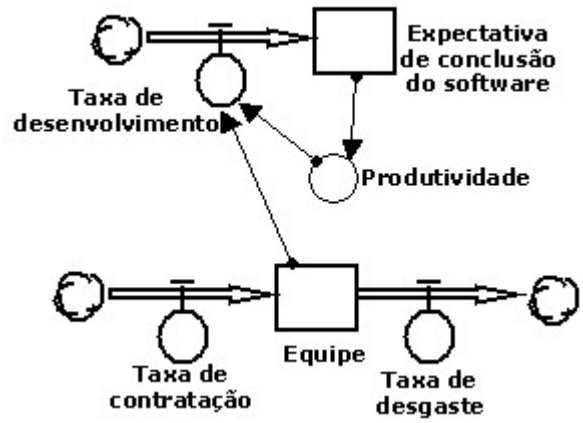

Figura 3.2: Diagrama de Repositórios e Fluxos de um modelo de produção de software (Madachy, 2001)

Neste modelo, a taxa de desenvolvimento de software contribui para que o pro-

duto de software seja concluído. A produtividade é influenciada pelo estágio em que se encontra o software e, por sua vez, influencia a taxa de desenvolvimento. A taxa de desenvolvimento é também influenciada pela quantidade de pessoas alocadas no projeto. A quantidade de pessoas alocadas é alimentada por uma taxa de contratação e decrementada por uma taxa de desgaste da equipe.

3.3.1 Metamodelo para a Dinâmica de Sistemas

Um problema da Dinâmica de Sistemas refere-se à representação do conhecimento no modelo. Embora os seus blocos básicos (fluxos, repositórios, processos e conectores) possibilitem a manipulação de equações de maneira flexível, a compreensão de modelos complexos e extensos pode se tornar uma tarefa árdua. Os elementos do mundo real não são facilmente reconhecidos em grandes modelos com centenas de construtores. Sua representação é, normalmente, espalhada entre várias equações, o que força os desenvolvedores a analisar todo o modelo para determinar um grupo preciso de equações que descrevem o comportamento de um elemento e seus relacionamentos com outros elementos.

As equações são necessárias para simulações e análise dos modelos, mas não são adequadas à sua descrição, pois representam conceitos distantes do domínio do problema de quem o utiliza, dificultando o entendimento do modelo. Barros (2001) propõe uma abordagem que busca contornar a complexidade dos modelos construídos segundo a Dinâmica de Sistemas. Essa abordagem, chamada de Metamodelo para Dinâmica de Sistemas, promove um aumento no nível de abstração dos modelos pela modelagem com construtores mais próximos de conceitos do mundo real (elementos do domínio do problema), ao invés de postulações matemáticas (elementos do domínio da solução).

A abordagem é ilustrada na figura 3.3. O processo de modelagem é dividido em três passos. Inicialmente, um especialista em um domínio desenvolve um modelo que contém os conceitos de mais alto nível da área do problema. Especialistas de domínio utilizam uma linguagem de alto nível de representação para desenvolver

estes modelos. Esta linguagem possibilita a descrição de classes, suas propriedades e comportamentos, e seus relacionamentos.

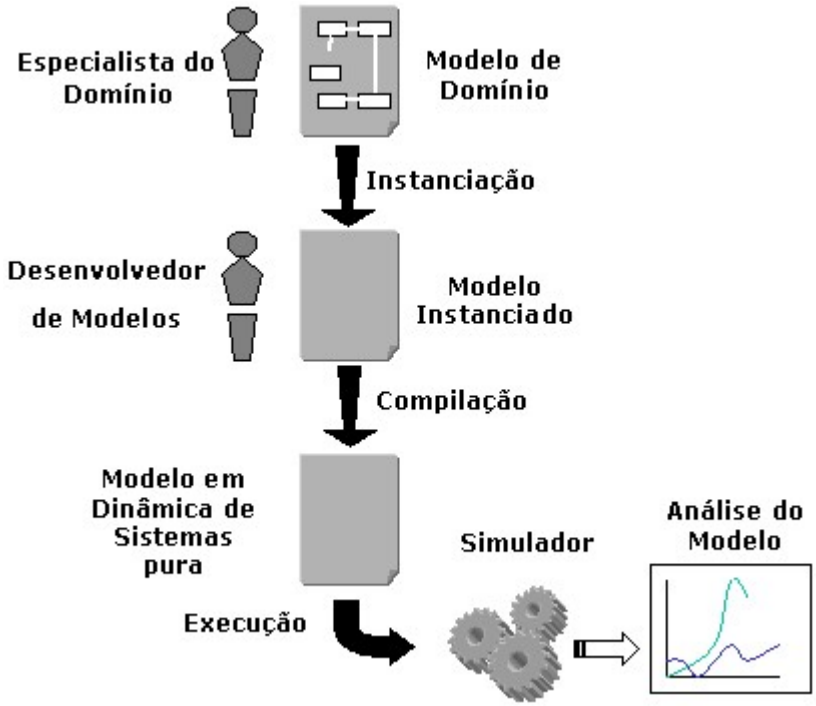

Figura 3.3: Processo de modelagem utilizando o Metamodelo para a Dinâmica de Sistemas

As classes são utilizadas como construtores para modelos específicos desenvolvidos no domínio. Uma **classe** representa um conjunto de elementos que podem ser descritos pelas mesmas propriedades e exibem comportamento similar. Uma **instância** de classe é um elemento no contexto de um modelo construído para o domínio ao qual pertence a classe. Por exemplo, uma classe que represente um desenvolvedor em um modelo de projeto de software terá tantas instâncias quantos forem os desenvolvedores alocados ao projeto a ser simulado.

Uma **propriedade** é uma informação relevante sobre uma classe e pode assumir valores distintos para cada instância. O **comportamento** da classe é uma formulação matemática das respostas produzidas por ela, decorrentes de alterações em outras instâncias ou no ambiente. O comportamento pode depender das propriedades, permitindo que diferentes instâncias de uma mesma classe reajam de forma distinta a uma mesma alteração. Formalmente, o comportamento é descrito por

repositórios, processos e taxas.

Um **relacionamento** é uma conexão estrutural entre duas ou mais instâncias de classes. Os relacionamentos podem ser simples, quando ligam uma instância a outra instância, ou podem ser múltiplos, quando ligam uma instância a várias outras instâncias. Além disso, um relacionamento pode ser unidirecional, quando apenas a instância origem de um relacionamento pode acessar as informações (propriedades e comportamentos) da instância destino, ou bidirecional, quando a visibilidade ocorre nos dois sentidos.

No segundo passo do processo de modelagem, um desenvolvedor de modelos constrói um modelo específico no domínio. Nesta etapa, o desenvolvedor especifica quantas instâncias de cada classe do modelo de domínio existem no modelo de interesse e os relacionamentos existentes entre elas, de acordo com os tipos de relacionamentos definidos entre as classes no modelo de domínio. Além disso, o desenvolvedor especifica os valores das propriedades das instâncias de classes. O modelo resultante contém informações acerca de suas instâncias, sem apresentar construtores da Dinâmica de Sistemas, que são descritos no modelo de domínio.

Finalmente, o modelo construído a partir do modelo de domínio é traduzido para construtores básicos da Dinâmica de Sistemas. O processo de tradução é automatizado por uma ferramenta. O modelo em Dinâmica de Sistemas tradicional pode ser, então, executado e analisado em um simulador.

Para exemplificar o metamodelo, é utilizado, neste trabalho, um modelo de projeto de software simplificado [1]. Os elementos envolvidos no projeto são ilustrados na figura 3.4. O projeto é organizado em três elementos que se relacionam: desenvolvedores, atividades e artefatos. Cada elemento é representado por uma classe e apresenta um conjunto de propriedades que o caracteriza. Existem dois desenvolvedores que possuem uma propriedade denominada *experiência*, que influencia a qualidade do seu trabalho. Os desenvolvedores estão alocados a atividades. A alocação de um desenvolvedor a uma atividade representa um relacionamento entre as instâncias destas classes. Os elementos que representam atividades estão relacio-

[1] A versão completa do modelo é descrita com detalhe em (Barros, 2001).

nados entre si. A seta da relação indica uma ordem de precedência na execução das atividades. Cada atividade apresenta uma propriedade denominada *duração*, que indica o tempo gasto, em dias, na sua execução.

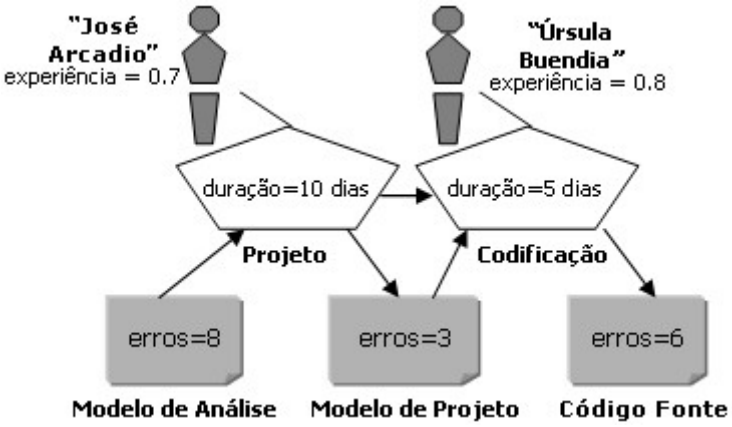

Figura 3.4: Elementos de um projeto simplificado

Os artefatos - modelo de análise, modelo de projeto e código - apresentam como propriedade o número de erros gerados durante a execução das atividades. Os artefatos estão relacionados com as atividades. As setas do relacionamento indicam o consumo ou a produção de artefato pelas atividades. A atividade de projeto utiliza o modelo de análise como insumo e produz o modelo de projeto. A atividade de codificação utiliza o modelo de projeto e produz o código fonte do módulo.

O trecho de código da figura 3.5 ilustra a definição de um modelo de domínio com os conceitos supracitados. Não entraremos em maiores detalhes sobre a notação utilizada. No entanto, é importante identificar os construtores definidos pela abordagem [2].

O modelo do exemplo contém três classes que representam os desenvolvedores que participam do projeto (classe "Developer"), os artefatos desenvolvidos (classe "Artifact") e as atividades que compõem o projeto (classe "Activity").

Além das classes, são definidos quatro relacionamentos no modelo. O relacionamento "Team", entre as classes "Developer" e "Activity" representa a ligação existente entre um desenvolvedor em uma atividade, isto é, determina a atividade que

[2] A notação completa do metamodelo pode ser encontrada em (Barros, 2001).

```
MODEL ProjectModel
{
        CLASS Developer
        {
           PROPERTY Experience 1;
           PROC Productivity Experience;
        };

        CLASS Artifact
        {
          PROPERTY Latent_errors 0;
          STOCK Errors Latent_errors;
        };

        CLASS Activity
        {
          PROPERTY Duration 0;
          STOCK TimeToConclude duration;
          RATE (TimeToConclude) Work if(DependOk, -Min (Prod*TimeToConclude / DT, Prod), 0);
          PROC DependOk GROUPSUM (Precedence, TimeToConclude)  < 0.001;
          STOCK ExecutingOrDone 0;
          RATE (ExecutingOrDone) RTExecuting if (AND(ExecutingOrDone < 0.001, DependOk), 1, 0);
          RATE (Outcome.Errors) ErrorsTransmit if(RTExecuting > 0.001, GROUPSUM (Income, Errors)/DT,0);
          PROC Prod GROUPMAX (Team, Productivity);
          PROC ErrorsPerDay 5;
          RATE (Outcome.Errors) ErrorsCommited -1 * ErrorsPerDay * Work / Prod;
        };

        MULTIRELATION Team Activity, Developer;
        MULTIRELATION Precedence Activity, Activity (NextActivities);
        MULTIRELATION Income Activity, Artifact;
        RELATION Outcome Activity, Artifact;
};
```

Figura 3.5: Modelo Simplificado para o domínio de projetos de software

um desenvolvedor desempenha. O relacionamento "Precedence", entre duas classes "Activity", determina uma ordem de precedência entre duas atividades. A utilização de um papel torna o relacionamento bidirecional. A classe destino pode utilizar o nome do papel para acessar informações das instâncias a ela associadas. Este relacionamento apresenta um papel "NextActivities".

Na figura 3.6, é apresentado um modelo específico de projeto criado a partir do modelo de domínio. Ele especifica as instâncias das classes definidas no modelo de domínio e os relacionamentos que existem entre estas instâncias no projeto.

Neste modelo, também podem ser feitas inicializações das propriedades das instâncias. Após sua construção, o modelo de projeto específico pode ser traduzido para os construtores tradicionais da Dinâmica de Sistemas, de forma que possa ser analisado em simuladores convencionais.

```
DEFINE MyProject ProjectModel
{
        Homer =  NEW Developer
                 SET Experience = 0.7;

        Chavez = NEW Developer
                 SET Experience = 0.8;

        AnalysisModel = NEW Artifact
                        SET latent_errors = 8;

        DesignModel = NEW Artifact
                      SET latent_errors = 3;

        SourceCode = NEW Artifact
                     SET latent_errors = 6;

        Designing = NEW Activity
                    SET duration = 10;
                    LINK Team Homer;
                    LINK Income AnalysisModel;
                    LINK Outcome DesignModel;

        Coding =  NEW Activity
                  SET duration = 5;
                  LINK Team Chavez;
                  LINK Precedence Designing;
                  LINK Income DesignModel;
                  LINK Outcome SourceCode;

};
```

Figura 3.6: Exemplo de modelo específico de um projeto de software a partir do modelo do domínio

A análise através de simulação consiste na observação, ao longo do tempo, de variáveis das instâncias das classes. Cada variável corresponde a uma equação definida na classe (propriedade, repositório, processo ou taxa). A figura 3.7 ilustra a análise de algumas variáveis do modelo instanciado no simulador.

3.3.2 Gerenciamento de Projetos Baseado em Cenários

O metamodelo da Dinâmica de Sistemas (Barros, 2001) foi desenvolvido no contexto de uma técnica denominada Gerenciamento de Projetos Baseado em Cenários, sendo aplicado no domínio de gerência de projetos de software. A representação proposta para os modelos, no entanto, pode ser utilizada em outros domínios (Barros *et al.*, 2001).

O objetivo da técnica é permitir que um gerente de projetos defina o compor-

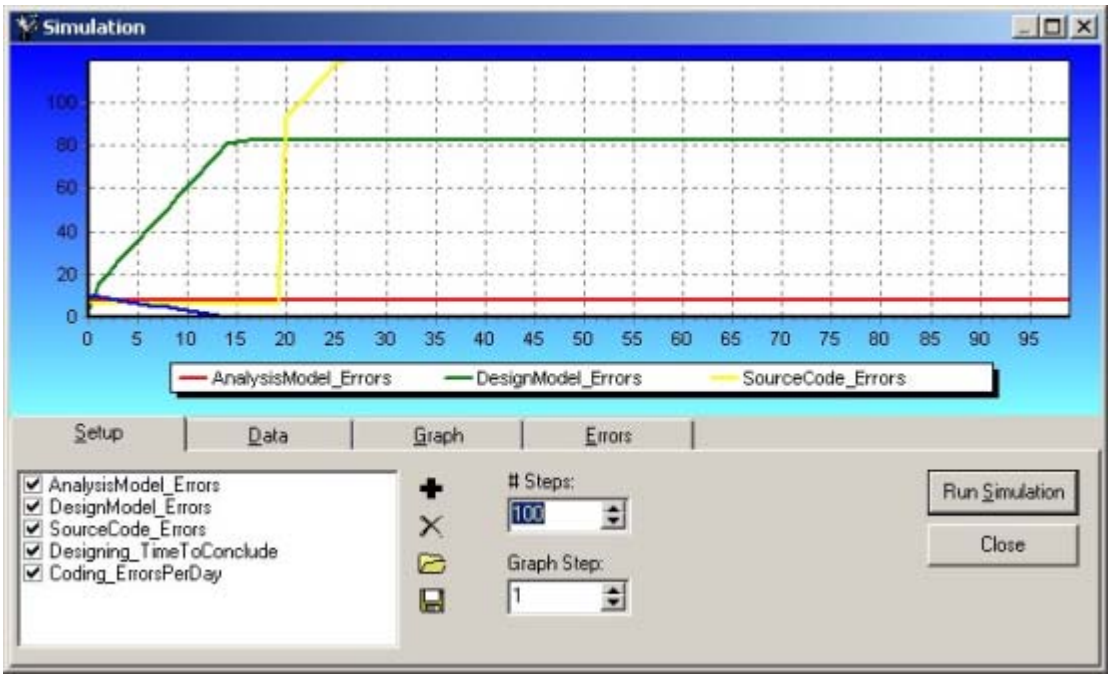

Figura 3.7: Modelo Simplificado para o domínio de gerenciamento de projetos de software

tamento esperado para um projeto e um conjunto de eventos que podem acontecer ao longo do seu desenvolvimento. Os eventos representam as incertezas do gerente sobre os rumos de seu projeto no futuro. Eles são representados através de cenários, que são utilizados para avaliar o impacto dos eventos sobre o comportamento do projeto (custo, cronograma, qualidade, entre outros). **Cenários** contêm conhecimento gerencial genérico e reutilizável. Eles mantêm representações de ações, teorias, políticas e procedimentos gerenciais que podem ser aplicados ou impostos a um projeto de desenvolvimento de software.

O paradigma do gerenciamento de projetos baseado em cenários é centralizado em dois artefatos: o modelo de projeto e os modelos de cenários. O modelo de projeto define o comportamento esperado para um projeto, enquanto os modelos de cenários descrevem rotas alternativas que o projeto pode seguir devido a ocorrência de eventos inesperados. Os cenários podem ser integrados ao modelo de projeto, modificando seu comportamento para representar as conseqüências de sua ocorrência sobre o projeto. O modelo de projeto descreve a interface de integração de cenários.

Os modelos de cenários são modelos abstratos, não podendo ser analisados iso-

ladamente. É necessário que seja feita uma integração com um modelo de projeto antes da simulação. Cenários são representados por uma sintaxe estendida do Metamodelo da Dinâmica de Sistemas. Um modelo de cenário está sempre associado a um modelo de domínio, contendo referências para as classes deste domínio.

Em Barros (2001), são apresentados, no domínio de gerência de projetos, alguns cenários que visam reproduzir eventos como:

- o aumento da produtividade, devido à experiência acumulada no projeto;
- a diminuição da taxa de geração de erros, causada pelo aumento da perícia;
- o aumento da taxa de geração de erros, provocado pelo aumento excessivo da carga de trabalho;
- a propagação de erros entre duas atividades seqüenciais, entre outros.

O modelo de cenários introduz dois conceitos sobre o metamodelo: as conexões e as ativações. Uma **conexão** representa um relacionamento entre o cenário e uma classe do modelo de domínio. Quando o cenário é integrado a um modelo específico construído a partir de um modelo de domínio, suas conexões devem ser especificadas entre as instâncias do modelo específico. As conexões de um cenário podem definir propriedades, processos, taxas, repositórios e ajustes. As propriedades representam novas características das instâncias associadas à conexão. Com os processos, taxas e repositórios, novas equações são incorporadas ao comportamento da instância sobre a qual a conexão foi ativada e podem referenciar outras equações e propriedades da conexão e relacionamentos da classe. Os ajustes de uma conexão permitem que o modelo de cenário redefina as equações de processos e taxas de uma classe do modelo de domínio.

As **ativações** correspondem a mecanismos de integração dos cenários aos modelos específicos (instanciados a partir do modelo de domínio). Uma ativação é uma indicação de que um cenário será associado a uma instância de classe.

O modelo de cenários pode indicar ainda a presença de **restrições**. As restrições

de um cenário permitem que este indique as conexões de outro ou do mesmo cenário que são esperadas nas instâncias afetadas por suas próprias conexões.

As definições dos modelos de cenários podem ser consultadas em maior detalhe em (Barros, 2001).

3.3.3 Modelos em Dinâmica de Sistemas com Estrutura Dinâmica

Na Dinâmica de Sistemas tradicional, a simulação é realizada após o ajuste de alguns parâmetros iniciais do modelo. A intervenção na simulação se dá basicamente na escolha do período de tempo a ser simulado e a seleção das variáveis que serão analisadas. Não há intervenção externa durante a simulação propriamente dita. O modelo evolui com uma estrutura fixa que determina o seu comportamento.

Para exemplificar de que maneira isso acontece, considere o modelo apresentado na seção anterior (figura 3.6). Após as configurações iniciais, ou seja, a definição das instâncias de classes e relacionamentos, o modelo é simulado e sua estrutura (relacionamentos entre suas instâncias) se mantém a mesma ao longo de toda a simulação. Assim, as instâncias de atividades, artefatos e desenvolvedores permanecem relacionadas da mesma forma. Apenas as variáveis que representam os níveis de repositórios e valores de processos se alteram. Não há, durante a simulação, alterações estruturais na organização dos elementos que compõem o sistema. A interação do usuário observador com a ferramenta se dá, portanto, apenas no início da simulação.

Os jogos de simulação, diferentemente das simulações simples, apresentam um maior grau de interação com o usuário. O efeito das ações do usuário durante o jogo provocam a alteração do modelo que representa o sistema observado. A alteração do modelo fornece base para reflexão do jogador sobre suas decisões. As interações com o jogo podem provocar alteração nos valores das variáveis existentes no modelo e a modificação dos relacionamentos entre os elementos que compõem o sistema. Desta maneira, a conjunção dos jogos com modelos de simulação torna necessário

ao simulador oferecer funcionalidades de manipulação estrutural do modelo.

Esse tipo de funcionalidade não está contemplada na Dinâmica de Sistemas tradicional. Uma extensão da abordagem do Metamodelo da Dinâmica de Sistemas apresenta uma perspectiva neste sentido (Dantas, 2003). A adaptação consiste em prover maneiras de se alterar estruturalmente o modelo simulado no decorrer da simulação. O simulador passa a se tornar uma ferramenta viável de ser utilizada por um jogo que se baseie em um modelo de simulação definido segundo a Dinâmica de Sistemas. Durante o jogo, o modelo pode ser alterado por meio de **operações** que atuam sobre suas instâncias.

Considerando o exemplo de projeto de software apresentado na seção 3.3.1, após determinar os desenvolvedores responsáveis por cada tarefa, um gerente no ambiente de simulação tradicional teria condições apenas de observar os desenvolvedores de sua equipe imaginária executando suas tarefas, sem maior poder de intervenção. Em um ambiente de simulação mais interativo, ele pode observar como o sistema evolui se ele trocar os desenvolvedores das atividades nas quais eles estão alocados. Poderia, ainda, considerar uma situação de inserção de uma atividade auxiliar, como de inspeção, e observar como se comporta a taxa de propagação de erros nos artefatos produzidos. O metamodelo, acrescido das operações que possibilitam alterações estruturais, fornece ao observador um ambiente de simulação mais interativo, permitindo maior grau de experimentação na simulação do problema ou fenômeno sob estudo. Conforme observado no capítulo 2, a interatividade é fundamental para o engajamento em situações de aprendizado (Ahdell e Andresen, 2001).

As operações são listadas abaixo, com exemplos correspondentes no modelo de projetos de software:

Alteração de valores de propriedades: provoca a alteração do valor de propriedades de instâncias de classes. Por conseguinte, altera os resultados dos cálculos dos comportamentos (taxas, repositórios e processos) que se baseiam nas propriedades. A execução desta operação necessita que sejam fornecidos como parâmetros o nome da instância que tem a propriedade, o nome da propriedade a ser alterada e seu novo valor. Um exemplo prático desta operação

é a alteração da duração de uma atividade do projeto.

Criação de uma instância de classe: implica a criação de uma nova instância de classe no modelo. Para a sua execução, é necessário informar o nome da instância a ser criada e a classe correspondente. Um exemplo desta operação é a contratação de um novo desenvolvedor, ou criação de um novo artefato ou atividade.

Remoção de uma instância de classe: implica a eliminação de uma instância de classe do modelo, removendo também os relacionamentos a ela conectadas. O único parâmetro necessário é o nome da instância a ser removida. Exemplos desta operação podem ser a demissão de um desenvolvedor ou a eliminação de um artefato ou atividade do projeto.

Alteração da instância destino de um relacionamento: atribui uma nova instância de classe como destino de um relacionamento. Os parâmetros necessários são o nome do relacionamento, a instância que está na origem e a nova instância que ficará no destino. Um exemplo da operação de mudança de destino no relacionamento é a troca do desenvolvedor alocado a uma determinada atividade.

Criação de um relacionamento: cria um relacionamento entre duas instâncias de classe. Os parâmetros para esta operação são o nome do relacionamento, a instância de origem e a instância de destino. Um exemplo é a alocação de um desenvolvedor a uma atividade, ou ainda, a criação de uma relação de dependência entre duas atividades no projeto.

Além das operações básicas descritas anteriormente, o simulador oferece operações para a ativação e a desativação de cenários ao longo da simulação.

3.4 O Projeto de Jogos de Treinamento

O projeto de jogos de treinamento pode ser analisado sob o ponto de vista de quem o utiliza, o jogador/aprendiz, e de quem o projeta e desenvolve. O primeiro

ponto de vista leva em conta as expectativas sobre o que o jogo oferece para o favorecimento do aprendizado. Ahdell e Andresen (2001) citam como alguns dos fatores mais importantes que influenciam o engajamento em um jogo de treinamento: a interatividade, o uso de efeitos dramáticos e a usabilidade.

A interação é produzida pelo acesso do usuário a dispositivos de entrada (como teclado, *mouse*, telas sensíveis a toque ou fala) que ativam uma determinada tecnologia para obtenção de uma resposta visual ou auditiva (texto, gráficos ou sons) (Ahdell e Andresen, 2001).

Os efeitos dramáticos cobrem os fatores que tornam o aprendizado mais divertido. São características como músicas, efeitos sonoros, humor, a contextualização do jogo em histórias, entre outros (Ahdell e Andresen, 2001).

A usabilidade é um componente fundamental. Em jogos, esta qualidade é chamada de "jogabilidade". Um consenso da literatura relacionado ao projeto de jogos é que um jogo é envolvente quando ele não é fácil, nem extremamente difícil a ponto de tornar remotas as chances do jogador chegar ao fim vitorioso. A jogabilidade deve ser um elemento de equilíbrio, tornando o jogo desafiante, porém possível de se vencer.

Do ponto de vista do projetista de jogos, é importante ter consciência a respeito das peculiaridades dos jogos em relação a outros tipos de projetos de software. Entre as particularidades, está o grande número de tecnologias de mídia que envolve recursos externos ao código, como imagens estáticas, seqüências de animações, *scripts*, sons, entre outros. O escopo e a complexidade do projeto de interfaces nos jogos também são superiores aos observados em software convencional, na medida em que devem ser considerados aspectos de usabilidade associados à emoção do jogador (Ye e Ye, 2002).

Outra diferença dos jogos está na quantidade de possíveis entradas, o que dificulta a previsibilidade do jogo. Um projetista de um jogo pode não imaginar a gama de estratégias possíveis para completá-lo (Rouse, 2001).

Além disso, os jogos apresentam fortemente aspectos de não-linearidade. De

acordo com Rouse (2001), a não linearidade provê o significado das interações no jogo e sem ela um jogo vira um filme. Em bons jogos há diferentes maneiras de se atingir um determinado resultado (Rouse, 2001). A não-linearidade pode se manifestar de diferentes formas: em termos da história do jogo, em termos de como o jogador supera os desafios do jogo, em termos da ordem em que os desafios são resolvidos e em termos da seleção dos desafios a serem resolvidos (Rouse, 2001). A não-linearidade em um jogo fornece a motivação para um jogador experimentar o jogo outras vezes, assumindo novas decisões e resolvendo os desafios de diferentes maneiras.

Como Ye e Ye (2002) observam, muitas empresas ainda projetam jogos de maneira caótica. O número de pessoas, atividades, artefatos e ferramentas envolvidas no projeto de jogos aumentaram exponencialmente nos últimos anos. Os autores observam que é cada vez mais difícil controlar e monitorar projetos que busquem atender os prazos e requisitos de qualidade. Desta forma, existe uma grande necessidade de processos que controlem a execução das atividades ao longo do desenvolvimento dos jogos.

Adicionalmente, os jogos para treinamento inserem outro componente de complexidade no projeto que é a conciliação entre os fatores que tornam o jogo envolvente e a valorização do aprendizado.

Em relação a abordagens que tratem da organização do desenvolvimento de jogos de simulação para treinamento, pouco é encontrado na literatura. Uma abordagem mais genérica para jogos educacionais é apresentada em (Greenblat, 1988), porém a autora não aborda a implementação de jogos em um ambiente computacional. Alguns trabalhos apresentam propostas de protótipos (Oh e van der Hoek, 2001) ou protótipos já implementados (Drappa e Ludewig, 2000). As poucas publicações sobre este assunto na área de jogos de treinamento podem ser atribuídas à recente penetração no meio acadêmico. Outro motivo está relacionado a questões de competitividade no lucrativo mercado de jogos, tanto para fins didáticos quanto para fins de entretenimento.

Prensky (2001) aborda a implantação de jogos de treinamento em empresas,

enfatizando aspectos de negócios e questões culturais. O autor, no entanto, não entra em detalhes sobre processos ou ferramentas que possibilitem o desenvolvimento dos jogos no ambiente computacional.

No projeto de jogos de entretenimento, pode ser encontrada uma vasta quantidade de ferramentas que auxiliam no desenvolvimento como, por exemplo, as chamadas máquinas de jogo (*engines*) e bibliotecas que fornecem rotinas reutilizáveis. Apesar de prover inúmeros recursos para manipulação de gráficos, animações e sons, não há suporte adequado ao desenvolvimento de jogos que se baseiem em um modelo de simulação que governe a evolução dos elementos do jogo. Este modelo, geralmente, faz parte do código fonte do jogo, não podendo ser isolado com facilidade. Isto diminui o potencial de reutilização dessas ferramentas nos jogos de simulação para treinamento, visto que elas precisariam ser integradas com o simulador do modelo em questão.

3.5 Conclusões

A abordagem do metamodelo da Dinâmica de Sistemas, analisada neste capítulo, fornece uma linguagem de alto nível para a descrição dos modelos de simulação e um ferramental de tradução entre os níveis de abstração e de execução (o simulador). O desenvolvimento dos modelos, no entanto, poderia ser feito com o uso de linguagens gráficas, que facilitassem a sua visualização. O simulador apresentado torna possível o aumento de interatividade da simulação por meio das operações de alteração dinâmica da estrutura do modelo ao longo da execução.

Na área de projeto de jogos de treinamento não foram encontradas publicações sobre processos ou ferramentas que auxiliem o desenvolvimento dos jogos para treinamento. Pode-se observar nos trabalhos aqui referenciados a demanda por técnicas que sistematizem as atividades de desenvolvimento. Além disso, a necessidade por tais técnicas surge na medida em que se deseja diminuir os esforços de construção de uma família de jogos de treinamento de um domínio para platéias de diferentes níveis de conhecimento neste domínio. Esta adequação pode demandar alterações

tanto nos modelos quanto na forma das interações que o jogo disponibiliza.

É oportuno dispor de meios para organizar o desenvolvimento do jogo, que estabeleçam as atividades e artefatos a serem trabalhados e a utilização das ferramentas adequadas (editores de modelos, tradutores, compiladores, simulador). Um requisito desejável de uma sistematização que se destine a tal propósito é o de enfatizar a reutilização dos modelos, promovendo uma economia de esforços quando da adequação dos jogos a platéias diferentes. A reutilização aqui defendida deve ser exercida tanto nos elementos que compõem os modelos de simulação, quanto nos elementos que compõem o jogo, como a sua apresentação (gráfica ou textual), contextualização (a história que envolve o jogo), entre outros. No próximo capítulo, é proposta uma sistematização de suporte ao desenvolvimento de jogos de simulação para treinamento que busca atender às necessidades aqui identificadas.

Capítulo 4

Uma Sistematização do Projeto de Jogos de Simulação Aplicados ao Treinamento

4.1 Introdução

Neste capítulo é proposta uma sistematização do projeto de jogos de treinamento baseados em modelos dinâmicos de simulação. A abordagem consiste na definição de um conjunto de atividades de modelagem e da especificação de um ferramental que busca auxiliar a construção, integração e execução dos modelos para a obtenção do jogo de treinamento.

As atividades e artefatos resultantes auxiliam na implementação do jogo. É enfatizada na abordagem, a reutilização dos diferentes aspectos do projeto que compõem os jogos de treinamento baseados em modelos dinâmicos.

Em cada atividade são definidos artefatos que representam os diferentes aspectos que constituem jogos de simulação para treinamento. Os artefatos são apresentados na forma de modelos que representam as dimensões do projeto do jogo. Os modelos, ao final do processo, são integrados para a obtenção do jogo.

Para a ilustração da abordagem, é utilizado como exemplo o projeto de um jogo para treinamento na área de gerência de projetos de software. Os exemplos dos conceitos aqui introduzidos são baseados neste jogo.

Na próxima seção, é apresentada uma visão ampla da abordagem proposta. Na seção 4.3 são enumeradas as características dos jogos considerados na abordagem. As atividades de modelagem são apresentadas na seção 4.4. As seções 4.5, 4.6, 4.7 e 4.8 abordam os artefatos produzidos nas atividades do modelagem. A seção 4.9 finaliza o capítulo com as conclusões.

4.2 Visão Ampla da Abordagem

A figura 4.1 fornece uma visão ampla da sistematização proposta, com as atividades, ferramentas e papéis envolvidos.

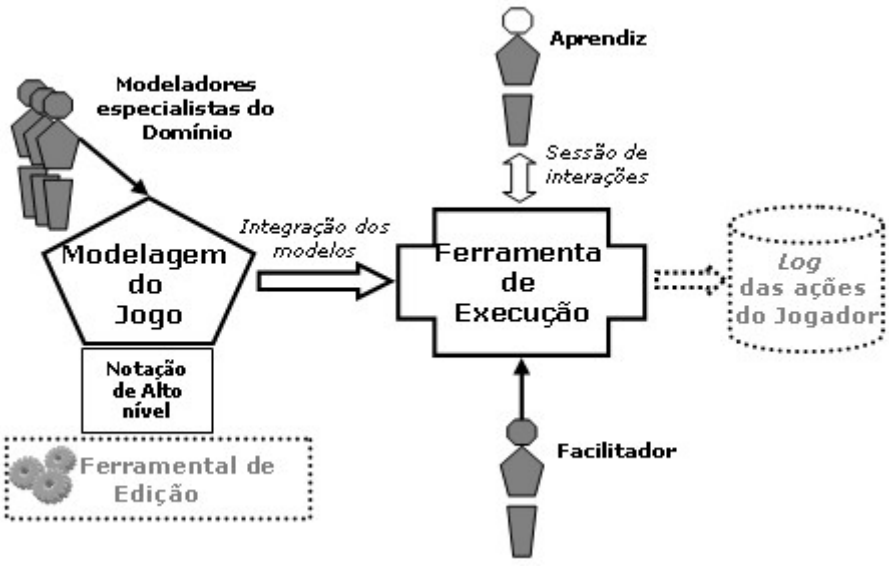

Figura 4.1: Estrutura geral da sistematização do projeto de jogos de simulação para treinamento

Projetistas especialistas no domínio do problema alvo do treinamento atuam na modelagem dos diferentes aspectos do jogo. Os modelos que constituem o jogo são então integrados e executados. Na fase de execução, são identificados outros dois papéis: o aprendiz, que é a pessoa submetida ao treinamento com o jogo, e

o facilitador, com as funções de esclarecer dúvidas e fornecer o suporte necessário à execução do jogo. No contexto do exemplo utilizado neste trabalho, supõe-se que o aprendiz seja um gerente de projetos novato e o facilitador uma pessoa com conhecimentos específicos sobre o jogo e na área de treinamento.

Durante a sessão de jogo, o jogador realiza as interações que constituem as suas decisões. As decisões podem ser armazenadas em um repositório para posterior análise. Ao término do jogo, são apresentados os resultados, sobre os quais os participantes do treinamento poderão refletir e realizar estudos.

Conforme observado na literatura, entende-se que a eficácia do treinamento é dependente não apenas da qualidade do jogo em questão, mas também da realização de sessões explanatórias antes e depois das sessões do jogo. A sessão explanatória inicial deve se destinar à exposição dos pontos que serão explorados e da base de conhecimento necessária para utilização do jogo. A sessão final deve apresentar pontos de discussão sobre a sessão do jogo, reforçando o conhecimento adquirido pelos aprendizes, as dificuldades encontradas e destacando as decisões que podem ter causado situações indesejáveis no jogo.

4.3 Gênero do Jogo

Por ser uma sistematização voltada para a fase de projeto e implementação do jogo, surgem algumas restrições tecnológicas que determinam uma categoria de jogos a serem desenvolvidos com a abordagem. A seguir, são apresentadas as características desta categoria:

- **Apresentação de uma história**: os jogos apresentam uma história na qual o jogador é inserido como um personagem central. No início da sessão de jogo, são apresentados textos introdutórios que auxiliam na contextualização do jogador. Por exemplo, na introdução do jogo de treinamento em gerência, pode ser exibido um texto que apresenta o projeto a ser realizado e os objetivos do jogo;

- **Gráficos em 2D**: a apresentação dos jogos considerados neste trabalho é feita por meio de gráficos planos de duas dimensões. Considera-se que este tipo de interface satisfatório, na medida em que possibilita a exibição dos elementos de fantasia dos jogos e ao mesmo tempo que se mostra mais simples de implementar do que interfaces mais sofisticadas, como interfaces isométricas de 2D - que tentam reproduzir visões em perspectiva - ou modelos gráficos em 3D;

- **Reprodução de ambientes de trabalho que se deseja treinar**: os jogos devem reproduzir uma atmosfera que represente o ambiente de trabalho do contexto de treinamento. Por exemplo, para o jogo de treinamento em gerência de projeto, é interessante reproduzir na interface gráfica o laboratório onde o projeto tem andamento, com os computadores, desenvolvedores, cadeiras e mesas;

- **Telas de configuração:** os jogos devem permitir a configuração da velocidade da simulação e também o volume do som;

- **Gráficos animados para representação dos elementos:** devem ser utilizados gráficos animados para representação de alguns elementos do jogo com os quais ocorrem as interações do usuário. Esses elementos podem ser personagens (como os desenvolvedores) ou outros objetos do jogo (como artefatos e atividades). A animação pode ser utilizada para produzir a percepção de mudança de estados de um elemento. Por exemplo, desenvolvedores cansados podem ser representados por seqüências de quadros de imagens e adereços gráficos que reproduzam esse estado nos bonecos que representam esses personagens. O jogador pode então interagir com o jogo no sentido de tentar alterar esse estado, decidindo, por exemplo, reduzir a carga horária de trabalho de alguns desenvolvedores;

- **Interação por cliques de *mouse*:** as ações que configuram as tomadas de decisão do jogador devem ser feitas por meio de cliques de *mouse* sobre os elementos do jogo. O clique sobre um elemento faz com que o jogo exiba uma janela de diálogo com informações de estado do elemento e ações válidas no

contexto corrente. Por exemplo, o clique sobre um desenvolvedor exibe uma janela com informações como seu nome, remuneração e estado de exaustão em que se encontra, além de um menu de opções com as ações que podem ser tomadas sobre o personagem, como demissão ou alteração da carga horária;

- **Exibição de relatórios:** relatórios podem ser exibidos durante o jogo, reportando ao jogador estados relevantes dos elementos. Por exemplo, podem ser exibidas mensagens informativas sobre o estado do cronograma do projeto em andamento;

- **Andamento do tempo em turnos:** os jogos devem ter passagem de tempo em turnos. A velocidade de passagem dos turnos pode ser configurada pelo jogador.

Jogos de simulação de gerência com as características abordadas acima se mostraram úteis em contextos de treinamento, conforme evidências extraídas por Dantas (2003) a partir de estudos de caso de foco qualitativo.

É importante estabelecer um escopo de projeto bem delimitado para tornar viável o desenvolvimento dos jogos no contexto da pesquisa. Considera-se, no entanto, que a sistematização proposta pode ser estendida a outros gêneros de jogos de simulação, havendo a necessidade de adaptação do ferramental aqui apresentado ou utilização de ferramentas auxiliares. A adaptação é necessária, especialmente, quando se considera a sofisticação da interface gráfica, como por exemplo a mudança para gráficos isométricos em 2D ou 3D.

4.4 Atividades de Modelagem

A figura 4.2 ilustra as atividades propostas para o desenvolvimento. A atividade inicial é a elaboração de uma descrição do jogo de simulação a ser construído. O objetivo da descrição é estabelecer uma definição clara, em linguagem natural, dos conceitos que nortearão a construção do jogo, como a história a ser vivida, os personagens, os ambientes e informações de contextualização do jogo.

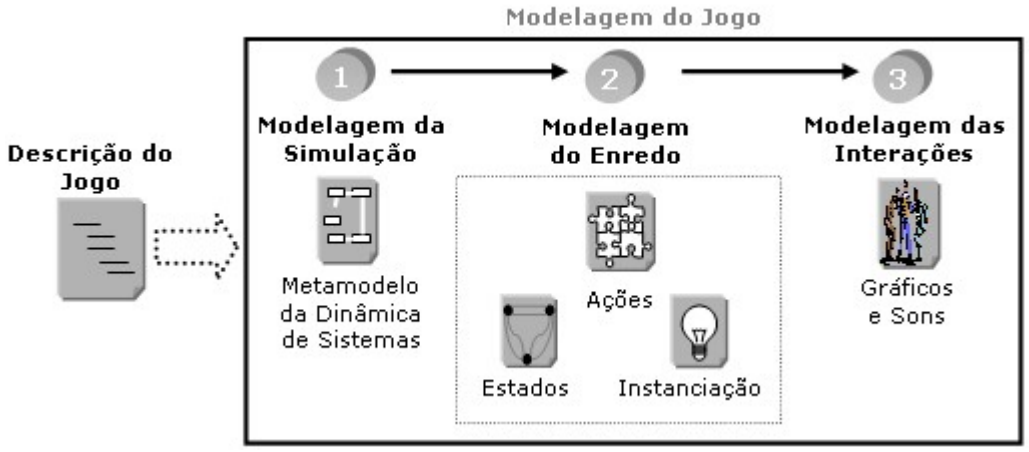

Figura 4.2: Atividades propostas para o desenvolvimento de jogos de treinamento baseados em simulação

As atividades seguintes correspondem à modelagem do jogo propriamente dito. São três as atividades de modelagem: **modelagem de simulação**, **modelagem de enredo** e **modelagem das interações**.

Na atividade de modelagem de simulação ocorre a criação de um modelo de simulação que descreve as propriedades, comportamentos e relacionamentos existentes entre os elementos do domínio alvo.

Na atividade de modelagem de enredo, são construídos modelos sobre os aspectos da lógica do jogo, tendo como base o modelo de simulação. A lógica do jogo consiste da definição de informações sobre elementos simulados que são relevantes no contexto do jogo. Na modelagem de enredo, é considerada, por exemplo, a descrição dos estados possíveis que os elementos do jogo podem assumir, bem como a descrição da reação desses elementos a transições de estados.

Na atividade de modelagem de interações, é construído um modelo que descreve os aspectos da apresentação do jogo ao usuário. Este modelo é dependente dos dois modelos anteriores.

A divisão do projeto do jogo de treinamento em três dimensões - simulação, enredo e interação - é uma forma de diminuir o esforço empregado na evolução e na reutilização de diferentes partes do jogo. Um jogo de treinamento baseado em

simulação pode evoluir nessas dimensões em ritmos diferenciados. Busca-se com este agrupamento em três dimensões aumentar a coesão entre aspectos comuns do projeto e diminuir o acoplamento entre aspectos distintos.

As seções a seguir abordam cada atividade em maior detalhe.

4.5 Descrição do Jogo

A descrição do jogo conta uma breve história sobre o contexto do treinamento. É importante que seja definido claramente o problema no domínio a ser estudado e as respectivas habilidades que se deseja aperfeiçoar no aprendiz no decorrer da sessão de treinamento.

A descrição deve identificar os elementos chave do domínio que estarão envolvidos no jogo, como eles estão relacionados e quais desses elementos constituirão personagens e outros objetos importantes do jogo. Cada elemento deve ter descrito um conjunto de estados que ele pode assumir no jogo. A descrição também deve conter a identificação dos ambientes onde serão apresentados os personagens e objetos.

A seguir, é apresentada a descrição de um jogo destinado ao treinamento em gerência de projetos, denominado "YAMM" (da sigla de "Yet Another ManageMent game").

> "O jogo consiste no controle de um projeto de software. Busca-se consolidar no aprendiz os conceitos relevantes da gerência de projetos, fazendo com que ele seja exposto a decisões com as quais se depararia em um ambiente real. O aprendiz é um gerente de projetos novato que deve controlar o projeto para que este seja concluído com sucesso. O gerente deverá tomar algumas decisões como o controle da carga de trabalho dos desenvolvedores, a aplicação de atividades de inspeção e a contratação de desenvolvedores.
>
> O objetivo do gerente é conseguir executar o projeto em um prazo es-

tipulado e com um orçamento definido no início do jogo. O ambiente principal de controle do jogo simula um laboratório, onde os desenvolvedores estarão trabalhando nas atividades que compõem o projeto. A interação do jogador ocorre por meio do acionamento de comandos com o clique de *mouse* sobre os desenvolvedores. O jogo tem um ambiente de recursos humanos no qual se encontram desenvolvedores disponíveis no mercado.

As ações que o gerente pode executar são as seguintes: alterar o número de horas diárias de trabalho do desenvolvedor, indicar os desenvolvedores responsáveis pelas atividades componentes do projeto, ativar ou desativar atividades de inspeção, demitir desenvolvedores e contratar novos desenvolvedores.

O andamento do jogo ocorre em unidades de dia, que se sucedem continuamente sem intervenção do jogador. O jogo termina com sucesso quando o projeto é concluído no prazo e com o orçamento planejado e termina com falha caso um destas duas condições não seja satisfeita."

4.6 Modelagem da Simulação

O modelo de simulação apresenta a definição da estrutura do sistema modelado, identificando os elementos e os relacionamentos que compõem o sistema. Em cada elemento, descreve-se matematicamente seu comportamento. O comportamento de cada elemento define o conjunto de regras que governam a sua evolução ao longo do tempo e determinam a maneira como ocorre sua interação com outros elementos. Estes elementos configurarão os personagens e outros objetos que exercem um papel relevante no enredo do jogo. O modelo de simulação utilizado como exemplo nesta seção descreve os conceitos envolvidos em projetos de software com base em informações da literatura e no conhecimento de gerentes de projetos experientes.

Este modelo contém a definição das classes do domínio, a especificação de suas propriedades e comportamentos e dos tipos de relacionamentos que podem aconte-

cer entre as classes. Ele é descrito de acordo com o metamodelo da Dinâmica de Sistemas, ilustrado na seção 3.3.1 do capítulo 3.

O metamodelo simplifica o entendimento dos modelos em Dinâmica de Sistemas, quando comparado com a forma tradicional de modelagem (baseada apenas em fluxos, taxas, repositórios, processos e conectores). No entanto, por utilizar uma notação textual, ainda pode ser complexo visualizar detalhes do comportamento definido para as classes e obter uma visão ampla da estrutura do sistema modelado quando este é composto por muitas classes e relacionamentos.

Os diagramas são um meio de compartilhamento de informações que permitem que elas sejam verificadas, discutidas e modificadas, facilitando a comunicação entre pessoas de diferentes disciplinas. Os diagramas reduzem a ambigüidade da linguagem natural, evitando diferentes interpretações de um mesmo assunto, e podem ser precisos o suficiente para expor inconsistências que seriam encobertas sem o seu uso. Notações que utilizam diagramas podem capturar e apresentar informações que seriam menos precisas, menos claras e menos sucintas se expressas em linguagem natural (Wills, 1997).

Por este motivo, é proposta neste trabalho uma notação gráfica para o metamodelo da Dinâmica de Sistemas. Esta notação tem como objetivo permitir uma visualização clara dos aspectos estruturais e dinâmicos do modelo, sem a necessidade de um estudo detalhado das equações, classes e relacionamentos que compõem a descrição textual do modelo de domínio.

A mesma analogia vale para as linguagens orientadas a objeto. O entendimento de programas extensos pode ser apoiado por diagramas da UML (*Unified Modeling Language*) (OMG, 2000). Para projetos orientados a objeto, habitualmente, faz-se o uso de diagramas de classes para a visualização da estrutura e de diagramas de seqüência e de colaboração, para visualização da dinâmica de colaboração entre os objetos. Os diagramas, além de proverem uma visão ampla do sistema em estudo, facilitam o intercâmbio de informações entre as pessoas envolvidas no desenvolvimento e evolução do sistema.

A UML foi investigada como uma possível notação gráfica para modelos descritos com o metamodelo da Dinâmica de Sistemas. O interesse na utilização da UML surgiu, principalmente, da adequação do diagrama de classes para representação estrutural das classes do metamodelo da Dinâmica de Sistemas. As classes do metamodelo apresentam correspondência direta com as classes da UML. As propriedades poderiam ser mapeadas, sem perda de semântica, para atributos dos objetos, já que estes representariam uma informação de estado das instâncias. Os tipos dos atributos seriam fixos como números reais. Repositórios, processos e taxas poderia ser mapeados como operações da classe, por estarem ligados ao comportamento. Estereótipos poderiam ser utilizados para diferenciá-los entre si.

No mapeamento de relacionamentos também não haveria perda de semântica, pois na UML existem identificadores para os papéis, navegabilidade e cardinalidade, necessários para a descrição de relacionamentos do metamodelo da Dinâmica de Sistemas.

No entanto, os construtores da UML não se mostraram satisfatórios para a exibição da dinâmica inerente às instâncias das classes simuladas. É importante, em uma visão diagramática de um modelo em Dinâmica de Sistemas, ter a clara distinção dos elementos de comportamento (fluxos, taxas e processos). Diagramas da UML que representam a dinâmica de colaboração entre objetos - diagramas de seqüência e de colaboração - não apresentam os elementos gráficos adequados. Decidiu-se utilizar uma adaptação dos diagramas de repositórios e fluxos para este fim, visto que estes diagramas comportam os construtores tradicionais da Dinâmica de Sistemas e sua utilização é consolidada na comunidade que utiliza esta técnica de simulação.

Outra dificuldade para utilização da UML para a modelagem de simulação está na representação de cenários e conexões. Estudou-se a possibilidade de utilização de superclasses abstratas ou interfaces para a representação das conexões dos cenários. Todavia, a relação de herança ou realização de interfaces são válidas, na UML, para todos os objetos da classe derivada, enquanto que as conexões de cenários definidas no metamodelo podem estar ativas em apenas algumas instâncias. Além do mais, os conceitos de herança ou realização de interface não se mostram apropriados ao

conceito de conexão de cenário, pois apresentam semânticas diferentes.

Avaliou-se, portanto,(i) uma adaptação da UML ao metamodelo nos aspectos de representação estrutural ao qual ela se adequa; (ii) a utilização da notação de diagramas de fluxos e repositórios da Dinâmica de Sistemas tradicional para representação do comportamento das classes e (iii) a criação de novos construtores para a representação de cenários.

A abordagem apresenta dois diagramas para a representação da estrutura: o **diagrama de classes de simulação** e o **diagrama de cenários**. A representação do comportamento definido para as classes e conexões de cenários é feita por meio do **diagrama de fluxos e repositórios** da Dinâmica de Sistemas tradicional.

Diagrama de Classes de Simulação

O diagrama de classes possibilita a visualização das propriedades de uma classe, do seu comportamento - composto por processos, repositórios e taxas - e dos relacionamentos entre as classes. A figura 4.3 apresenta um diagrama de classes.

Cada classe é representada por um retângulo e contém três seções, assim como na UML. A primeira seção apresenta o nome da classe; a seção central apresenta suas propriedades; a seção inferior apresenta seus comportamentos. Os símbolos à esquerda dos comportamentos são utilizados para indicação do seus tipos. O círculo simples é utilizado para representar os processos, o círculo com um detalhe na parte superior indica as taxas e o quadrado representa os repositórios. O símbolo quadriculado representa uma tabela. Tabelas, embora não sejam consideradas como parte da Dinâmica de Sistemas tradicional, são comuns em diversas implementações desta abordagem. Elas representam pontos discretos de uma curva e são utilizadas por equações de taxas, repositórios e processos.

Relacionamentos são representados por retas direcionadas. Acima de cada relacionamento deve ser indicado o seu nome. A cardinalidade em relacionamentos simples não recebe indicação e em relacionamentos múltiplos é indicada pelo símbolo de asterístico (“*”).

O diagrama da figura 4.3 apresenta duas classes de um modelo no domínio de

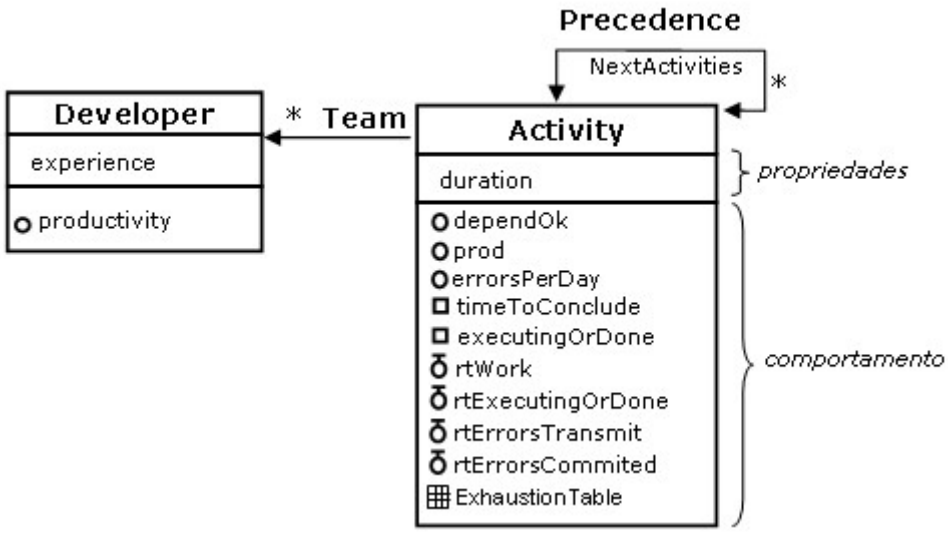

Figura 4.3: Diagrama de classes com os conceitos do domínio de Gerência de Projetos

gerência de projetos de software e dois relacionamentos. A classe "Developer" representa um desenvolvedor e a classe "Activity" representa uma atividade que compõe o projeto. Um desenvolvedor pode ser alocado a uma atividade. Uma atividade pode preceder várias outras atividades. Estas relações estão ilustradas, respectivamente, pelas associações "Team" e "Precedence".

Diagrama de Fluxos e Repositórios das Classes

Além da visão estática fornecida pelo diagrama de classes, é possível modelar a dinâmica de cada classe por meio do diagrama de repositórios e fluxos. Vale ressaltar que, na notação do metamodelo, os repositórios são alimentados sempre por taxas originadas de provedores infinitos. Na Dinâmica de Sistemas tradicional, por sua vez, as taxas podem transferir elementos entre repositórios. O efeito de transferência de elementos entre repositórios por meio de uma taxa pode ser facilmente reproduzido com a notação do metamodelo.

A figura 4.4 exibe a dinâmica da classe "Activity". Os repositórios externos à classe, mas que são afetados por taxas definidas no contexto da classe, são envolvidos por linhas tracejadas. Obrigatoriamente, a classe externa portadora do repositório deve estar relacionada à classe fonte.

Diagrama de Cenários

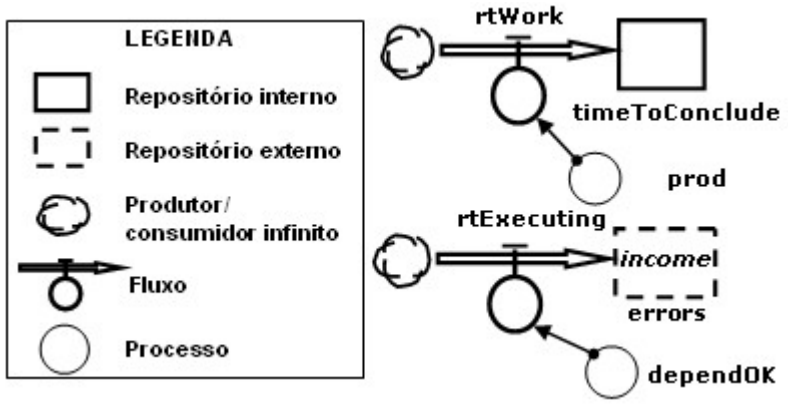

Figura 4.4: Diagrama de Fluxos e Repositórios que representa o comportamento da classe Atividade

Os cenários do modelo são apresentados por meio do diagrama de cenários. Cada cenário é apresentado em um diagrama próprio que exibe as suas conexões. Uma conexão pode definir propriedades, comportamentos e ajustes a serem realizados na classe subjacente. A conexão é representada de maneira similar a uma classe: um retângulo dividido em três seções. A primeira seção exibe o nome da conexão. A segunda indica as novas propriedades a serem incorporadas na classe. A terceira seção indica novos comportamentos e ajustes. Os comportamentos são representados pelos mesmos símbolos utilizados para diferenciá-los nas definições das classes. Os ajustes são representados em negrito e seu símbolo sinaliza se a redefinição ocorrerá sobre a equação de um processo ou de uma taxa.

As restrições dos cenários, que indicam uma condição para a conexão do cenário a uma instância de classe, não são explicitamente indicadas no diagrama. Elas podem ser definidas em informações detalhadas do cenário por meio de caixas de diálogo.

A figura 4.5 ilustra o cenário de exaustão, "Exhaustion". Este cenário apresenta uma conexão apenas. A conexão "TheDeveloper" atua sobre instâncias da classe "Developer" e reflete a falta de dedicação ao trabalho gerada pela exaustão do desenvolvedor. A ligação de uma conexão de cenário a uma classe é representada por uma linha com um círculo cheio na extremidade que intercepta a classe.

Em cada instância a qual o cenário se conecta, é definida uma série de novos comportamentos. O cenário apresentado não insere novas propriedades. As equa-

ções de comportamento desta conexão definem a proporção na qual o desenvolvedor diminui sua dedicação em função do seu grau de exaustão.

O processo indicado em negrito representa uma redefinição do processo original de instâncias da classe "Developer". Neste caso, "DWHAdjust" afetará a maneira como são calculadas as horas diárias de trabalho do desenvolvedor correspondente a instância referida (os cenários foram detalhados no capítulo 3).

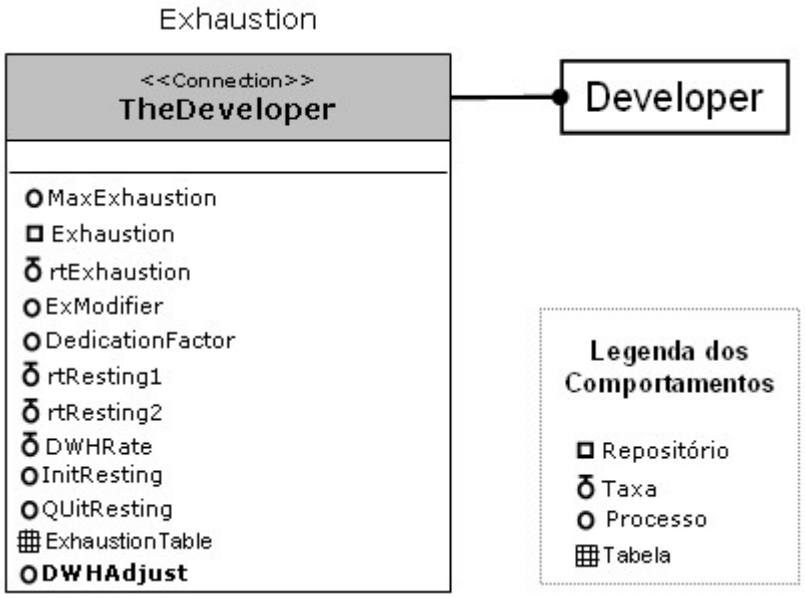

Figura 4.5: Diagrama de Cenários

É importante ressaltar que o modelo de simulação pode ser aplicado fora do contexto de construção de um jogo de simulação. Esta abordagem pode ser utilizada para elaboração de modelos a serem executados em simulações convencionais.

4.7 Modelagem do Enredo

A segunda etapa na criação de um jogo de treinamento compreende a modelagem do enredo do jogo. Os modelos construídos nesta etapa descrevem aspectos da lógica do jogo e fazem referência aos elementos definidos no modelo de simulação. Classes e relacionamentos modelados na fase anterior terão uma representação própria no contexto do jogo.

São três os modelos construídos nesta fase: o **modelo estrutural**, o **modelo de estados** e o **modelo de instanciação** dos elementos.

O modelo estrutural insere informações descritivas nas classes do modelo de simulação e determina quais cenários estarão ativos sobre as instâncias das classes. As classes com as novas informações são tratadas no modelo estrutural como **elementos do jogo** e configuram personagens ou outros objetos relevantes no enredo.

O modelo de estados apresenta as descrições de todos os estados relevantes que podem ser assumidos pelos elementos no jogo. Os estados são definidos em função de variáveis da simulação. Este modelo estabelece quais as reações de cada elemento a mudanças de estado.

O modelo de instanciação é utilizado para definir as instâncias de elementos que iniciarão o jogo, bem como os valores das suas propriedades.

As próximas três seções apresentam em detalhe cada modelo criado na etapa de modelagem de enredo.

4.7.1 Modelo Estrutural

Este modelo embute nas classes definidas no modelo de simulação informações relevantes apenas no contexto do jogo. As novas informações compreendem a inclusão de novos atributos, a especificação de ações que poderão ser executadas sobre as instâncias das classes durante a sessão do jogo e a indicação dos cenários que serão conectados às instâncias.

Os atributos são informações textuais dos elementos relevantes apenas no contexto do jogo, não possuindo significado para a simulação. Por exemplo, para o elemento que representa um desenvolvedor, podem ser definidos como atributos a idade, o nome completo e uma descrição do currículo. Esses atributos podem ser utilizados para dar maior contextualização ao enredo e enriquecer o jogo com elementos de fantasia.

Durante a sessão de jogo, os elementos podem sofrer **ações** decorrentes das

interações realizadas pelo jogador. A ação é um conceito do jogo ligado ao domínio modelado. As ações estão sempre associadas a instâncias dos elementos. A demissão de um desenvolvedor e a troca do desenvolvedor associado a uma atividade são exemplos de ações.

Cada ação pode disparar um conjunto de **operações**, que alteram o estado do modelo de simulação sendo executado. As operações apresentam uma ligação estreita com os serviços fornecidos pela camada de simulação do jogo. Basicamente, cada operação corresponde a manipulação de algum elemento da simulação. Estas operações correspondem às operações básicas providas pelo simulador e que permitem a manipulação do modelo em tempo de simulação. As operações foram definidas na seção 3.3.3 do capítulo 3.

A figura 4.6 ilustra o esquema de execução das ações. O jogador efetua uma interação sobre um determinado elemento do jogo. Esta interação dispara uma ação, definida no modelo estrutural, que, por sua vez, dispara um conjunto de operações sobre as instâncias das classes definidas no modelo de simulação. As operações alteram a estrutura do modelo executado na máquina de simulação.

Figura 4.6: Esquema da transformação das interações do jogador em ações que disparam operações sobre as instâncias do modelo de simulação

Ainda neste modelo, é possível determinar a ativação de cenários sobre os elementos. Para cada elemento definido no modelo estrutural, deve ser indicado quais cenários terão conexões ativas sobre as instâncias do elemento. Estas instâncias terão os valores de suas propriedades e equações afetados pelas conexões dos cenários indicados.

4.7.2 Modelo de Estados

O modelo de estados descreve os possíveis estados que podem ser assumidos pelas instâncias de elementos [1] durante o jogo. Nesta fase da modelagem, não são levados em conta aspectos visuais dos elementos, isto é, não é determinada a maneira como os estados são apresentados na tela. Por exemplo, se um personagem está andando ou sentado em frente ao computador.

O modelo de estados da fase de enredo descreve os estados lógicos dos elementos em função das suas variáveis simuladas. A alteração dos valores das variáveis ao longo da simulação pode provocar a modificação do estado em que a instância do elemento se encontra. A atuação do jogador é feita no sentido de manter os estados das instâncias de uma maneira que contribua para que ele conclua o jogo com sucesso. Um estado assumido por um elemento pode indicar a necessidade de que algumas ações sejam tomadas pelo jogador.

Por exemplo, uma atividade que assuma um estado de "Atrasada" torna necessário que uma atitude seja tomada pelo gerente para que o projeto retome seu cronograma planejado. O jogador no papel de gerente pode então decidir por uma realocação de desenvolvedores ou uma mudança de cronograma. Um desenvolvedor com estado de "insatisfeito" pode causar problemas no andamento do projeto, como aumento da geração de erros na atividade na qual ele está alocado ou até mesmo pode pedir demissão no meio do projeto, quando ele já adquiriu conhecimento razoável. Para melhorar a motivação do desenvolvedor, o jogador poderia, por exemplo, diminuir suas horas de trabalho diário.

As transições entre os estados são determinadas por condições sobre variáveis booleanas (especificamente processos) definidas na classe correspondente ao elemento. Para isto, pode ser necessário que essas variáveis booleanas sejam incluídas no modelo de simulação. Um exemplo prático é a inserção da variável de processo booleana

[1] As classes do modelo de simulação são referenciadas no enredo como "elementos" do jogo. A preferência pela utilização deste termo se deve ao fato de que nem todas as classes de simulação apresentam necessariamente um papel relevante no enredo. Basicamente, os elementos do jogo são as classes do modelo com atributos textuais e ações.

"OverTime" para indicar que o projeto extrapolou o tempo para ele estipulado. Esta variável inserida na classe "Project", no modelo de simulação, simplesmente compara o tempo atual consumido pelo projeto com o tempo limite do projeto (determinado por propriedade da mesma classe). Caso a primeira seja maior, o processo "OverTime" retorna um valor positivo. Esta abordagem simplifica o desenvolvimento do modelo de estados, deixando a realização de cálculos a cargo do modelo de simulação.

Suponha que o personagem "Desenvolvedor" do jogo de gerência de projetos apresente a máquina de estados exibida na figura 4.7.

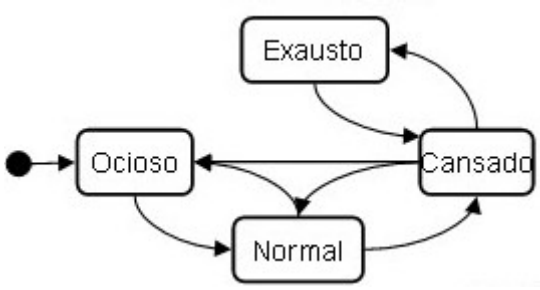

Figura 4.7: Máquina de estados do personagem "Desenvolvedor" do jogo de gerência de projetos

O desenvolvedor apresenta quatro estados. O estado "Ocioso" indica que nenhuma das atividades associadas ao desenvolvedor está em execução. O estado "Normal" indica que o desenvolvedor está submetido a uma carga horária suportável. O estado "Cansado" indica que o desenvolvedor está trabalhando em um regime de horas que causa um início de exaustão. O estado "Exausto" indica que o desenvolvedor se encontra esgotado pelo volume de trabalho. Neste estado, a taxa de produção de erros do desenvolvedor será certamente mais elevada do que é o esperado, caso ele estivesse em algum dos outros estados (Barros, 2001).

A alteração das variáveis que definem os estados pode ativar as transições. As transições são baseadas em condições sobre variáveis do modelo de simulação.

A seleção de quais estados um elemento pode assumir e quais variáveis determinam cada estado é uma decisão subjetiva e depende da experiência do projetista do jogo.

4.7.3 Modelo de Instanciação

Após a criação do modelo estrutural e do modelo de estados dos elementos do jogo, deve ser construído um modelo que define e inicializa as instâncias dos elementos do jogo e seus relacionamentos. Este modelo é chamado modelo de instanciação.

Este modelo determina a configuração inicial do jogo, instanciando os elementos específicos. Em cada instância de elemento, são inicializados os atributos de jogo (incorporados no modelo estrutural), as propriedades (definidas no modelo de simulação) e os relacionamentos que partem da instância.

Considere como exemplo no jogo de gerência de projetos uma configuração inicial que defina uma rede de três atividades: análise, projeto e codificação. Além disso, considere um desenvolvedor alocado às três atividades. O modelo de instanciação definiria as três instâncias do elemento atividade ("Activity"), uma instância do elemento desenvolvedor ("Developer") e a inicialização do relacionamento "Team", que liga cada instância de atividade ao desenvolvedor. A disposição das instâncias ao longo do jogo poderá mudar em decorrência da dinâmica das interações. Novas instâncias de desenvolvedores poderão surgir (representando ofertas do mercado) e relações entre desenvolvedores e atividades podem ser eliminadas (representando realocações ou demissões).

O modelo de instanciação exerce, no projeto do jogo, o mesmo papel do modelo específico definido pelo metamodelo da Dinâmica de Sistemas, apresentado na seção 3.3.1 do capítulo 3. A diferença básica é que o modelo de instanciação, além de definir valores para propriedades numéricas, permite a inicialização de atributos textuais do jogo.

Propõe-se um diagrama para este modelo contendo como elementos básicos as instâncias de classes e as ocorrências de seus relacionamentos. A figura 4.8 apresenta o diagrama proposto. Para cada instância, definem-se os valores de suas propriedades. As classes são indicadas nas instâncias por meio de estereótipos. Ocorrências de relacionamento são representadas por setas com a indicação do nome do relacionamento definido no modelo de simulação.

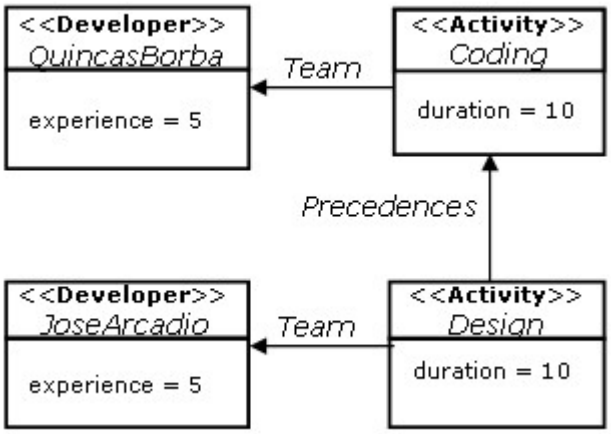

Figura 4.8: Artefatos e as suas relações de dependência

Na figura apresentada, o modelo de instanciação define os desenvolvedores "JoseArcadio" e "QuincasBorba". O primeiro é associado à atividade de projeto e o segundo à atividade de codificação, ambos pelo relacionamento "Team". A atividade de codificação é o destino da ocorrência do relacionamento "Precedences", que parte da instância que representa a atividade de projeto, indicando que a execução do projeto é precedida pela codificação.

A indicação da inicialização dos valores dos atributos textuais deve ser feita por meio de caixas de diálogo, visto que a sua extensão pode dificultar a adequação da instância na tela.

4.8 Modelagem das Interações

A atividade de modelagem de interações compreende a elaboração de um modelo que descreve os efeitos gráficos e eventos de interface do jogo. Este modelo é construído com base nos elementos do modelo de enredo. Os principais conceitos abordados são as animações e imagens de personagens e objetos decorativos, ambientes que reproduzem locais de treinamento, máquinas de estados que definem transições gráficas entre elementos e definição de apresentação de variáveis na tela. As seções do modelo de interação são descritas nas subseções a seguir.

4.8.1 Visões

No modelo de interações são definidas animações e imagens sem contexto e que podem ser referenciadas em outras seções. Uma outra seção as associa a personagens e elementos de adorno. As animações e imagens estáticas são chamadas aqui de visões[2]. A animação é construída pela indicação de uma seqüência de imagens em formato convencional (como gif, png ou jpg). Uma imagem estática pode representar, por exemplo, uma mesa no laboratório e uma animação a seqüência de imagens de um desenvolvedor andando com aspecto de cansado.

4.8.2 Ambientes

Um ambiente do modelo reproduz um local de treinamento e é a principal interface de interação do jogo. Um exemplo de ambiente é um laboratório, que pode conter desenvolvedores trabalhando com computadores, enfeites como quadros e uma ou mais ligações com outros ambientes, como a de recursos humanos. A ligação pode ser representada pelo desenho de uma porta que, quando acionada por meio de cliques de *mouse*, alteram o ambiente corrente do jogo. A representação do ambiente propriamente dito é feita por meio de uma imagem de fundo. Além da imagem, definem-se:

- **instâncias de elementos presentes no ambiente:** são indicadas as instâncias de elementos descritas no enredo (no modelo de instanciação) que aparecem no ambiente. Para cada instância, indica-se a sua posição inicial no ambiente. A posição poderá se alterar ao longo do jogo em função das interações e alterações no contexto. A representação gráfica de uma instância no ambiente é fornecida na seção que define os estados de interações, onde é realizada a associação das instâncias definidas no enredo a animações e seus estados lógicos;

- **adereços:** são elementos decorativos associados a imagens estáticas ou anima-

[2]Do inglês *view*. Este termo é utilizado por algumas *engines* e ferramentas de autoria de jogos construídas no início da década de 90 para denotar seqüências de imagens de personagens.

ções. Por exemplo: uma cafeteira ou uma janela animada. Para cada adereço também deve ser indicada a sua posição no ambiente, e

- ***links:*** são adereços associados a uma outro ambiente do jogo. Quando acionados conduzem ao ambiente destino. Um exemplo de *link* é a porta que conduz ao escritório de recursos humanos.

4.8.3 Máquina de estados de interação

Nesta seção do modelo, são definidos os estados e eventos gráficos de cada elemento do jogo. Cada elemento do modelo de enredo que se apresenta graficamente deve ter uma descrição da sua máquina de estados de interação. Esta máquina reflete os estados lógicos do jogo e gerencia a representação gráfica das instâncias para a qual ela foi definida. Por exemplo, as instâncias do elemento desenvolvedor podem conter um estado gráfico que reproduza um conjunto de animações, em diferentes momentos, indicando que ele está cansado, exibindo o boneco com uma postura corcunda e uma feição de desgaste.

Estados gráficos

A máquina de interação referencia os estados lógicos definidos para o elemento no modelo de enredo. Nem todos os estados lógicos do elemento necessitam de uma definição gráfica, mas todo estado gráfico deve referenciar um, e apenas um, estado lógico. Além disso, cada estado gráfico deve referenciar uma ou mais visões. As visões do elemento são alternadas durante o jogo, de acordo com uma distribuição fornecida pelo usuário. Supondo um desenvolvedor trabalhando em um estado lógico "Normal" pode-se representá-lo em 70% do tempo digitando, 20% andando e 10% coçando a cabeça, simulando a tentativa de solução de um problema encontrado. A alternância de visões fornece ao jogo uma possibilidade variar a apresentação dos elementos e, conseqüentemente, tornar o jogo menos monótono.

Eventos de interação

A transição entre estados lógicos do jogo pode disparar um conjunto de ações

gráficas. Para cada transição entre estados lógicos que implique alguma alteração nos gráficos deve ser descrita a seqüência de ocorrência das ações gráficas. As ações gráficas são listadas a seguir:

- movimentação da visão gráfica da instância para um ponto determinado;
- deslocamento relativo da visão da instância, pela indicação de um *offset* nas coordenadas X e Y;
- posicionamento da visão gráfica de uma instância em um outro ambiente;
- alteração da visão gráfica atual da instância, e
- alteração do ambiente corrente.

Um exemplo de evento de interação disparado por uma transição pode ser a troca do estado corrente de um desenvolvedor de "Ocioso" para "Normal". Considerando que em "Ocioso" o desenvolvedor está em pé, quando ocorrer a transição pode-se definir a seqüência:

1. troca da visão corrente para uma animação que reproduza o movimento de andar;
2. movimentação da instância até uma mesa;
3. troca da visão da instância para uma animação que reproduza o movimento de sentar, e
4. troca da visão da instância para uma animação de digitação.

A combinação das ações gráficas pode reproduzir uma série de comportamentos mais complexos.

4.8.4 Exibição de variáveis de monitoramento contínuo

Pode ser de interesse do projetista disponibilizar para visualização contínua algumas variáveis relevantes para o monitoramento do estado do jogo. Para o jogo de

treinamento em gerência, é importante que o jogador acompanhe as variáveis que indicam o tempo decorrido e o orçamento disponível. Desta forma, o projetista pode definir em uma seção do modelo de interação o conjunto de variáveis que estarão disponíveis para visualização contínua. Para cada variável deve ser indicado seu nome no modelo de simulação, a instância a qual a variável pertence e, opcionalmente, um *alias* (um nome amigável para ser exibido no lugar do nome original) e uma unidade (por exemplo, uma moeda).

4.9 Conclusões

A abordagem proposta neste capítulo apresenta uma tentativa de organizar o desenvolvimento de jogos de simulação para treinamento. Os jogos considerados na abordagem utilizam como base um modelo de simulação, descrito com o metamodelo da Dinâmica de Sistemas, que rege a evolução dos elementos do jogo ao longo do tempo.

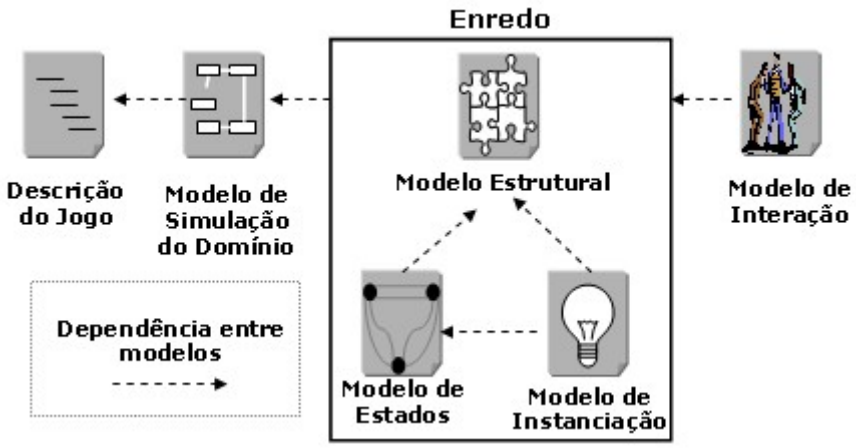

Figura 4.9: Artefatos e as suas relações de dependência

A sistematização é composta de um conjunto de atividades de modelagem, da definição dos artefatos produzidos em cada atividade e da especificação um ferramental de integração e execução, a ser descrito no próximo capítulo. Cada atividade de modelagem leva em conta um aspecto particular do jogo.

A figura 4.9 apresenta um resumo dos artefatos gerados nas atividades aqui

descritas e as suas relações de dependência.

A divisão do jogo nas dimensões de simulação, enredo e interações pode comportar a evolução do jogo sem o comprometimento de sua estrutura, possibilitando a reutilização em diferentes níveis e a adaptação do jogo a platéias distintas em um mesmo domínio.

Os modelos gerados em cada atividade foram apresentados em um alto nível de abstração. Maiores detalhes sobre a representação em baixo nível são discutidos no próximo capítulo.

Capítulo 5

Aplicação da Sistematização

5.1 Introdução

No capítulo anterior, a sistematização para o desenvolvimento de jogos de treinamento baseados em simulação foi apresentada em linhas gerais. As atividades nela definidas foram descritas e os modelos apresentados em um alto nível de abstração.

Este capítulo descreve a estrutura do ferramental de suporte, constituída por ferramentas de edição, integração e execução. Além disso, é apresentada uma aplicação completa da abordagem que contextualiza os conceitos em um jogo de treinamento em Gerência de Projetos. O jogo segue a descrição apresentada na seção 4.5. A representação dos modelos é abordada neste capítulo em maior nível de detalhe, da forma como é tratada pelo protótipo da ferramenta de integração e execução.

O capítulo está organizado da seguinte forma: a próxima seção descreve a estrutura do ferramental definido na abordagem. A seção 5.3 apresenta uma aplicação da abordagem, descrevendo os modelos desenvolvidos do jogo de treinamento em gerência de projetos. A última seção deste capítulo apresenta as suas conclusões.

5.2 Ferramental

A figura 5.1 exibe a estrutura geral do ferramental definido na proposta.

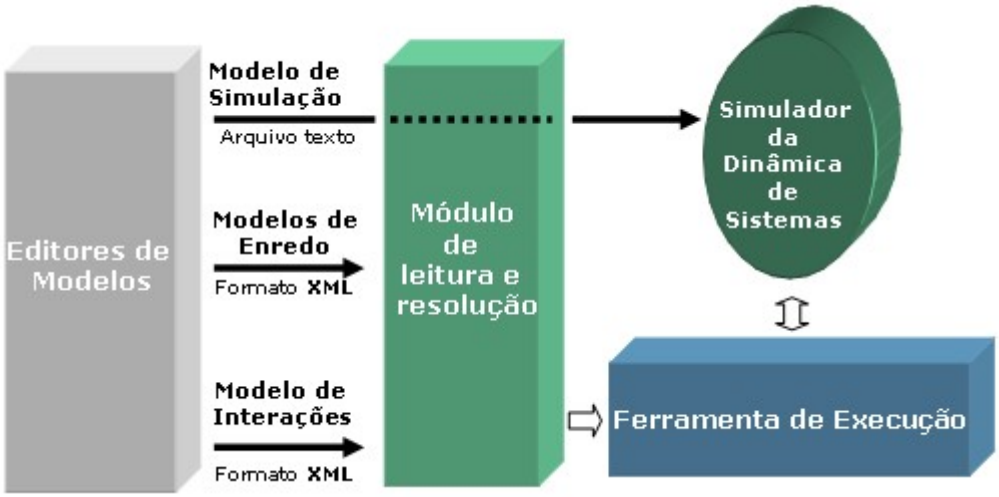

Figura 5.1: Estrutura geral das ferramentas da abordagem

Os editores de modelos devem prover as funcionalidades para a elaboração dos modelos utilizando as notações dos diagramas apresentadas no capítulo 4. Das ferramentas ilustradas, apenas os editores de modelos e o mecanismo de *log* não foram implementados, dado o tempo disponível para desenvolvimento desta pesquisa. A abordagem proposta, no entanto, abre outras frentes de trabalho que podem explorar os aspectos não tratados aqui e que podem ser desenvolvidos em projetos finais de curso ou em outras teses.

O diagrama do modelo de simulação deve ser traduzido para a notação textual do metamodelo da Dinâmica de Sistemas. Os modelos de enredo e de interações são traduzidos em arquivos XML.

Após o processo de tradução dos modelos em arquivo textual, o módulo de leitura faz a carga dos modelos e a resolução (tratamento das dependências) dos modelos em memória. Os modelos devem ser lidos na seguinte ordem:

1. Modelo de simulação
2. Modelos de enredo

   (a) Modelo Estrutural

   (b) Modelo de Estados

   (c) Modelo de Instanciação

3. Modelo de Interações

Esta ordem deve ser seguida em função da dependência existente entre os construtores dos modelos.

Como pode ser observado na figura 5.1, o modelo de simulação é tratado de maneira particular: o arquivo textual é repassado diretamente ao simulador, que mantém uma representação em memória do modelo de simulação. O simulador, desenvolvido antes desta pesquisa, fornece os serviços necessários para a compilação de modelos descritos de acordo com o metamodelo da Dinâmica de Sistemas. Os serviços oferecidos por este componente permitem carregar, compilar e gerar código (em construtores tradicionais da Dinâmica de Sistemas) de um modelo, executar passos de simulação, acessar os valores obtidos pela simulação e acessar mensagens de erro.

Após a leitura do arquivo com o modelo de simulação, os arquivos com os modelos de enredo e de interações são lidos e ocorre, então, a execução da sessão do jogo.

5.2.1 Ferramental de Execução

O ferramental de execução é dividido em três camadas, cada qual responsável por um aspecto do jogo. A figura 5.2 ilustra a estrutura da ferramenta.

A camada de simulação é representada por um simulador de modelos da Dinâmica de Sistemas (apresentado no capítulo 3). O módulo oferece métodos de acesso aos dados do modelo (cenários, classes, propriedades, relacionamentos e instâncias de classes) e métodos para o processamento de operações sobre instâncias de classes.

A camada de enredo mantém uma tabela com todas as instâncias de elementos existentes e os respectivos estados em que se encontram. A cada passo do jogo, os

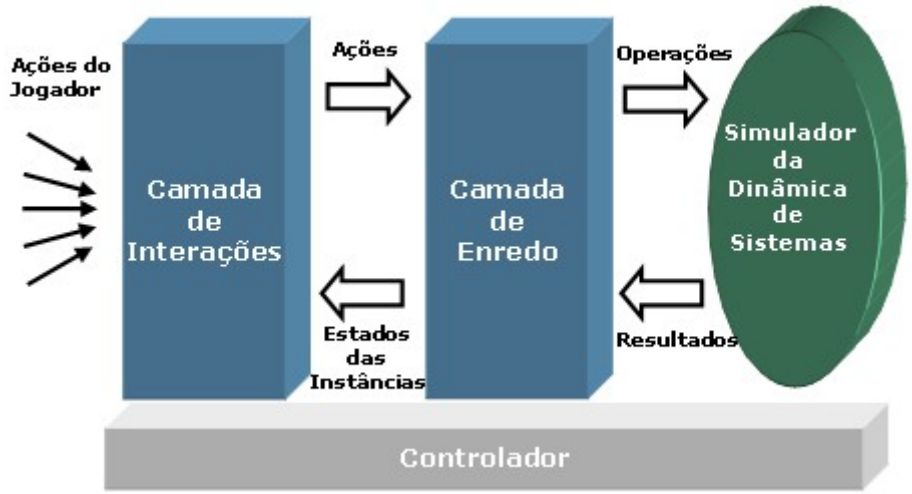

Figura 5.2: Organização da ferramenta de execução

estados das instâncias são atualizados de acordo com os resultados fornecidos pelo simulador. As alterações de estados são então notificadas a camada de interações. As ações recebidas da camada de interações são repassadas, como operações, à camada de simulação, alterando a estrutura do modelo de simulação durante o jogo.

A camada de interações é responsável pelo tratamento de eventos provindos do teclado e do *mouse*, exibição das imagens e animações. Esta camada realiza tarefas de renderização das animações e painéis de controle por meio de linhas de execução independentes (*threads*). O estado lógico dos elementos é refletido pela camada de interação com base nas definições de estados gráficos. As interações realizadas sobre elementos gráficos que representam instâncias de enredo são repassadas no formato de ações para a camada que gerencia o enredo.

Um módulo de controle faz o gerenciamento do fluxo de informações pelas camadas. O controle mantém uma *thread* que executa um passo de jogo na freqüência estipulada pelo projetista (a configuração inicial é de 2 segundos). Em cada passo do jogo, é feita (i) a passagem do turno de simulação, (ii) atualização dos estados das instâncias e (iii) a verificação do estado final do jogo (feita por uma consulta às definições de enredo).

Na etapa (i), o controlador envia uma mensagem de passo de simulação ao simu-

lador. A cada passo de simulação executado, o simulador calcula e guarda o valor de cada equação do modelo (referentes aos processos, taxas e repositórios). Todos os valores de cada instante de simulação podem ser consultados pelo nome da equação, que é único no modelo.

Na etapa seguinte (ii), o controlador envia ao gerenciador da camada de enredo uma mensagem de atualização dos estados das instâncias. A partir do estado corrente de cada instância, verifica-se qual o próximo estado. A transição dos estados é realizada de acordo com as condições sobre as variáveis do modelo de simulação.

A cada transição, o gerenciador da camada de enredo dispara um evento de notificação aos gerenciadores interessados. Para isto, é definida uma interface genérica que deve ser implementada pelos gerenciadores interessados. Estes gerenciadores, por sua vez, devem ser registrados na camada de enredo, durante a inicialização do jogo. O módulo de interações é o principal interessado nas transições, pois a ocorrência destes eventos pode disparar um conjunto de alterações na interface. Esta indireção, implementada por meio do padrão *observer*, garante a independência do módulo de enredo em relação ao módulo de interações.

Na última etapa (iii), ocorre a verificação do estado final do jogo. Esta verificação é feita por uma consulta do controlador ao gerenciador da camada de enredo. O gerenciador apenas verifica se a instância central do jogo encontra-se em um estado final. A instância informada é definida no modelo de instanciação.

As camadas de interações e enredo e o módulo de controle foram desenvolvidos em Java. A camada de interações utiliza o *framework Swing* (SUN, 2003) na apresentação dos componentes visuais. O gerenciamento de *Sprites* [1] é provido pela biblioteca *Gage* (Gage, 2003). O simulador foi desenvolvido em linguagem C e é encapsulado em uma biblioteca de linquedição dinâmica (dll). A comunicação dos módulos Java com o simulador é feita por meio de JNI (*Java Native Interface*).

[1] Quadros ou sequências de quadros de imagem que representam os elementos gráficos na tela do jogo

5.3 Aplicação da abordagem

Ao longo deste texto, exemplos apresentados na área de gerência de projetos nortearam a apresentação dos conceitos envolvidos na abordagem. Nesta seção, é apresentado o jogo "YAMM", destinado ao treinamento de gerentes de projetos de software.

5.3.1 Modelo de Simulação

O modelo de simulação utilizado como base deste jogo foi uma evolução do modelo utilizado em (Dantas, 2003). Parte do diagrama do modelo utilizado no jogo pode ser consultado na figura 4.3 do capítulo 4. O modelo na notação textual da Dinâmica de Sistemas é apresentado no apêndice A.

O jogo utiliza classes que representam desenvolvedores, atividades e o projeto como um todo. O desenvolvedor se constitui no personagem principal do jogo. Ele apresenta uma propriedade que representa o seu custo por hora e propriedades que quantificam sua experiência nos diversos tipos de atividade.

A atividade apresenta propriedades que indicam sua ordem de execução e informações de duração. Um grupo de propriedades booleanas (que assumem valores -1 ou 1) determina seu tipo, isto é, se a atvidade é de análise, projeto, inspeção, codificação ou de teste. Apenas uma das propriedades indicativas de tipo deve ser positiva.

O projeto apresenta propriedades que indicam os limites de tempo e custo para sua execução.

São dois os relacionamentos do modelo: *Precedences*, que identifica a ordem de precedência das atividades, e *Team*, que identifica a alocação de um desenvolvedor a uma atividade.

O cenário "StateInProject" foi desenvolvido para disponibilizar à camada de enredo variáveis relevantes aos estados do desenvolvedor (figura 5.3). O fragmento do

modelo na notação do metamodelo da Dinâmica de Sistemas é apresentado na figura 5.4.

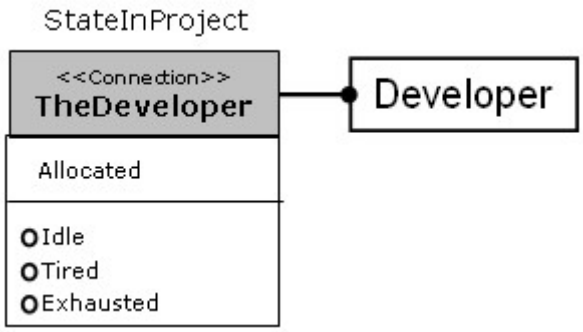

Figura 5.3: Diagrama do cenário "StateInProject" desenvolvido para prover funcionalidades para a camada de enredo

```
SCENARIO StateInProject ProjectModel
{
    CONNECTION TheDeveloper Developer
    {
        PROPERTY Allocated 0;   # Values 0 or 1

        PROC Idle (AssociatedTask >= 1000);
        PROC Tired (AND(Exhaustion > 10, Exhaustion < MaxExhaustion));
        PROC Exhausted (Exhaustion >= MaxExhaustion);
    };
    CONSTRAINT TheDeveloper, Exhaustion.TheDeveloper;
};
```

Figura 5.4: Código do cenário "StateInProject" na notação do metamodelo da Dinâmica de Sistemas

Este cenário insere no desenvolvedor uma propriedade ("Allocated") para indicar se a instância está alocada ou não a um projeto, assumindo os valores 0 ou 1. O processo "Idle" é utilizado para indicar que a tarefa a qual o desenvolvedor está alocado não se encontra em execução. Caso esteja, ela retorna o valor 0, indicando que o desenvolvedor não está ocioso. O processo "Tired" verifica se o grau de exaustão está em um intervalo de tolerância, determinado empiricamente. Caso esteja no intervalo, o processo retorna 1, indicando que o desenvolvedor está cansado. O processo "Exhausted" verifica se o grau de exaustão do desenvolvedor encontra-se acima do tolerável, retornando 1 caso a condição seja satisfeita. A restrição para a utilização deste cenário é que a conexão "TheDeveloper" do cenário "Exhaustion" esteja ativada na instância.

5.3.2 Modelo Estrutural

No modelo estrutural, os atributos e ações incorporados em cada classe são detalhados a seguir:

- **Desenvolvedor**

  Atributos: idade, currículo, nome completo e apelido

  Ações: contratação, demissão, alteração do número de horas de trabalho diárias

- **Atividade**

  Atributos: nome, descrição

  Ações: ligar/desligar atividade, alterar desenvolvedor alocado

- **Projeto**

  Atributos: nome e descrição

  Ações: nenhuma

O modelo estrutural é representado em um arquivo em formato XML. O trecho ilustrado na figura 5.5 apresenta as definições do elemento desenvolvedor.

A *tag* **Element** identifica um bloco de inserção de novas informações sobre uma classe do modelo de simulação, que é indicada pelo atributo **class**.

Os atributos de jogo incorporados aos elementos são identificados pela palavra reservada **Attribute** e as ações pela palavra **Action**. Opcionalmente, para atributos e ações, pode ser definido um nome amigável, indicado no atributo **alias**.

As ações são demarcadas em blocos com a *tag* **Action**. Cada ação pode conter uma ou mais operações e, opcionalmente, pode conter parâmetros que podem ser compartilhados pelas operações nela agrupadas. Os parâmetros devem ser informados no momento da chamada da ação.

```
<StructuralModel>
  <Element class="Developer">
    <Attribute name="fullName" alias="Full Name"/>
    <Attribute name="nickName" alias="Nick Name"/>
    <Attribute name="curriculum" alias="Curriculum"/>
    <Attribute name="age" alias="Age" />
    <Action name="changeWorkHours"
            alias="Change developer work hours"
            restrict="Allocated">
      <Parameter name="newWorkHours"
                 alias="Work Hours" type="double" min="0" max="20"/>
      <SetProperty property="WorkHours"
                   value="newWorkHours" object="self"/>
    </Action>
    <Action name="fireDeveloper" alias="Fire developer"
                                 restrict="Allocated" value="true">
      <SetProperty property="Allocated" value="0" object="self"/>
    </Action>
    <Action name="hireDeveloper" alias="Hire developer"
            restrict="Allocated" value="false">
      <SetProperty property="Allocated" value="1" object="self"/>
    </Action>
  <Scenarios>
    <Connect scenario="ProductivityLossDueCommunication"/>
    <Connect scenario="ProductivityDueExpertise"/>
    <Connect scenario="ErrorGenerationDueExpertise"/>
    <Connect scenario="ProductivityDueLearning"/>
    <Connect scenario="Overworking"/>
    <Connect scenario="Exhaustion"/>
    <Connect scenario="StateInProject"/>
   </Scenarios>
  </Element>
...
</StructuralModel>
```

Figura 5.5: Código XML do modelo estrutural do jogo

Considere a ação "changeWorkHours" do elemento "Developer", para exemplificação do funcionamento dos parâmetros. O objetivo desta ação é permitir a alteração do número de horas trabalhadas por dia por um desenvolvedor. A disponibilidade das ações pode ser restrita ao valor de uma variável booleana do modelo de simulação. Para isto, utiliza-se o atributo **restrict**, onde deve ser informada a variável condicionante (no exemplo, a propriedade "Allocated"). Caso o atributo de *tag* **value** seja informado, a ação estará disponível sempre que a variável retornar valor igual ao informado no atributo no passo de simulação corrente. Caso seja omitido, é considerado sempre como condição o valor verdadeiro.

A ação "changeWorkHours" apresenta um parâmetro e uma operação. As operações são detalhadas na tabela 5.1.

Operação	***Tag***	**Campos**	**Campos Parametrizáveis**
Alteração de valor de propriedade	SetProperty	nome da instância, nome da propriedade e valor	valor
Criação de Instância	CreateObject	nome da classe e nome da instância	nome da instância
Remoção de Instância	RemoveObject	nome da instância	nome da instância
Criação de Relacionamento	CreateLink	nomes das instâncias fonte e destino, nome do relacionamento	nome da instância destino
Alteração de destino de relacionamento	SetLink	nome da instância fonte, nome do relacionamento, nome da instância destino	nome da instância destino

Tabela 5.1: Operações sobre o modelo de simulação

A primeira coluna indica as operações. A segunda exibe a *tag* utilizada no documento XML que representa o modelo estrutural. A terceira coluna indica quais são os campos necessários na execução das operações. A quarta indica quais desses campos podem ser preenchidos por parâmetros.

Cada operação apresenta um ou mais campos parametrizáveis. Os valores desses campos são utilizados para a execução da operação. O preenchimento do campo pode ser feito diretamente, informando-se o valor na tag da operação, ou indiretamente, quando este é passado via parâmetros. Todo parâmetro tem um nome, um tipo e opcionalmente um alias. O nome é utilizado pela operação. O tipo pode ser numérico (*double*) ou um elemento declarado no modelo estrutural. Como o parâmetro do exemplo é destinado a alteração de uma propriedade, o seu tipo é *double*. Quando o conteúdo de um campo indica o nome de um parâmetro definido para ação, utiliza-se o valor do parâmetro na execução da operação. Caso o valor do campo não corresponda a um parâmetro, ele é utilizado diretamente na operação.

A operação do exemplo (**SetProperty**) altera o valor da propriedade "WorkHours". O novo valor da propriedade a ser ajustado pela operação é dado pelo parâmetro indicado no atributo **value**. A instância sobre a qual a operação é executada é indicada no atributo **object**. Neste caso, a palavra reservada *self* indica que a própria

instância será afetada pela operação. Na declaração de um parâmetro numérico, é possível estabelecer limites de valores para o seu conteúdo através dos atributos **min** e **max**. Desta forma, a carga horária de um desenvolvedor poderá variar de 0 a 20 horas de trabalho.

Além dos atributos e ações, definem-se os cenários que serão conectados às instâncias da classe do elemento. As conexões são informadas no interior do bloco indicado pela *tag* **Scenarios**. A tag **Connect** identifica o cenário que será conectado as instâncias.

5.3.3 Modelo de Estados

A figura 5.6 apresenta o modelo de estados dos três elementos utilizados no jogo.

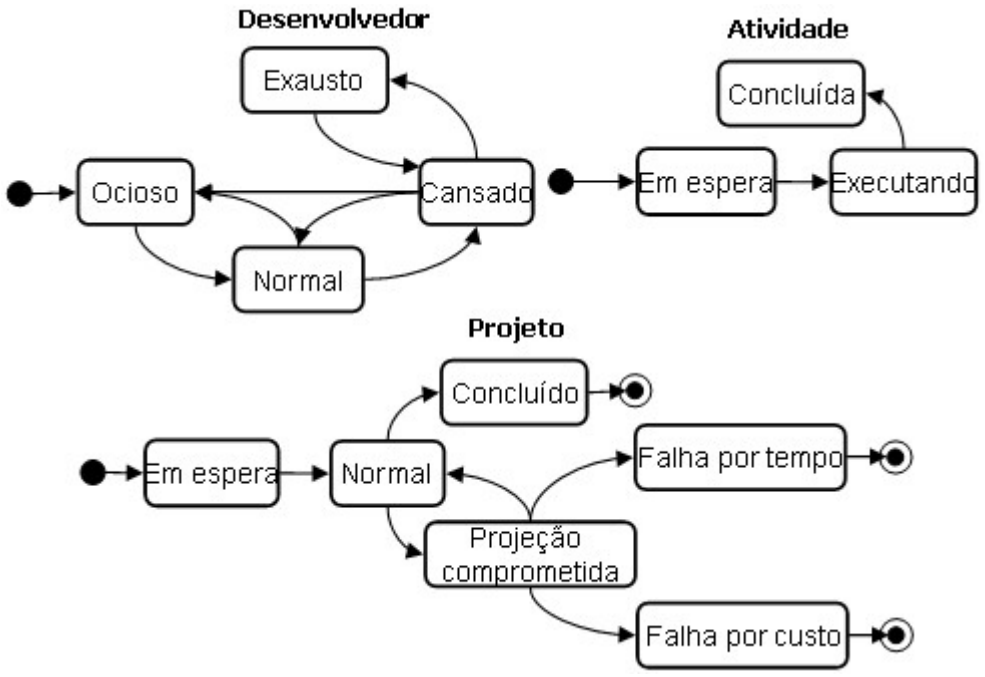

Figura 5.6: Diagrama de estados dos elementos do jogo

Considere a máquina que determina os estados possíveis das instâncias do elemento desenvolvedor. O estado "Ocioso" indica que o desenvolvedor está alocado no projeto, mas não está trabalhando, pois não está alocado a alguma atividade ou a atividade ao qual ele está alocado não está sendo executada. Este é o estado inicial de todos os desenvolvedores. Ao começar o projeto, os desenvolvedores alocados às

atividades iniciais passam ao estado "Normal".

O estado "Normal" indica que o desenvolvedor está trabalhando, normalmente, na atividade e se baseia, também, na variável que determina que seu grau de exaustão (inserida pelo cenário "Exhaustion" no modelo de simulação) está em níveis baixos. Deste estado, o desenvolvedor poderá passar novamente para o estado de ócio ou para o estado de cansaço, caso esteja trabalhando em um regime de horas desgastante.

O desenvolvedor permanece no estado "Cansado" se o seu grau de exaustão permanece maior do que o normal, mas ainda em um nível tolerável.

No modelo de simulação, o processo booleano "Tired" retorna positivo se a exaustão está entre os valores 15 e 50, determinados empiricamente. Do estado de cansaço o desenvolvedor pode passar aos estados de ócio, normal ou de exaustão.

O estado de exaustão é indicado pelo processo booleano "Exhausted" que retorna positivo, caso o grau de exaustão seja maior que 50. Neste estado, o desenvolvedor tende a trabalhar menos, pelo que é descrito matematicamente no cenário "Exhaustion".

Como não há indicação de estado final para este elemento específico, todos os estados desta máquina são considerados finais. Isto não faz diferença para a execução do jogo, pois o estado final do jogo é determinado pela máquina de estados do Projeto (esta informação é fornecida no modelo de instanciação, detalhado na próxima seção).

Um resumo do modelo a ser gerado correspondente à máquina de estados do desenvolvedor é ilustrado na figura 5.7.

Cada elemento deve ter uma descrição de uma máquina de estados. A descrição da máquina é feita com o bloco **EventSet**. Cada máquina de estados é composta pela descrição dos seus estados, feita em blocos demarcados pela *tag* **State**. A máquina pode ter apenas um estado inicial e este deve ser identificado pelo atributo **initial**.

No interior da definição de cada estado, é feita a descrição das transições para

```
<StateModel>
  <EventSet element="Developer">
    <State name="idle" alias="Idle" initial="true">
      <Transition target="normal">
        <Condition variable="Idle" value="false"/>
        <ShowReport message="developerIdleToNormal"/>
      </Transition>
      <Transition target="unemployed">
        <Condition variable="Allocated" value="false"/>
        <ShowReport message="developerIdleToUnemployed"/>
      </Transition>
    </State>
    <State name="normal" alias="Normal">
      ...
    </State>
    <State name="tired" alias="Tired">
      ...
    </State>
    <State name="exhausted" alias="Exhausted">
      ...
    </State>
    <State name="unemployed" alias="Unemployed">
      ...
    </State>
  </EventSet>
  ...
</StateModel>
```

Figura 5.7: Fragmento do código XML do modelo de estados do desenvolvedor

os estados alvo. Cada transição é um conjunto de condições sobre as variáveis que definem o estado. A transição é executada quando a conjunção das condições é obedecida. Isto faz com que o jogo atribua ao elemento o estado destino da transição.

As transições são identificadas pela *tag* **Transition**. O atributo **target** indica o estado destino da transição.

Cada condição sobre uma variável do estado é identificada pela *tag* **Condition**. As variáveis condicionadas são sempre processos booleanos. Para cada condição, devem ser informados a variável condicionada (atributo **variable**) e o valor alvo da comparação (atributo **value**). Caso seja omitido, o valor considerado é sempre verdadeiro.

A estratégia de utilização de processos booleanos para determinar variáveis de transição foi adotada com o intuito de simplificar o modelo de estados e deixar a cargo do modelo de simulação todo o tipo de cálculo matemático. Desta forma,

a construção de modelos de estados pode implicar a construção de novos cenários que introduzam no modelo de simulação, os processos booleanos necessários para a descrição completa dos estados. O desenvolvimento do jogo passa, portanto, a ser iterativo para a aplicação dos ajustes necessários em modelos construídos em fases anteriores.

Além da indicação das condições, pode-se especificar uma mensagem de notificação na transição. A mensagem é notificada ao jogador quando ocorre a transição nas instâncias regidas pela máquina de estados em questão. As mensagens de notificação são identificadas em *tags* **ShowReport**. O nome da mensagem é informado no atributo message. Este nome é uma referência para uma *tag* de mensagem específica em um aquivo XML auxiliar que armazena as mensagens do jogo. Desta maneira, toda vez que o desenvolvedor passa do estado "Ocioso" para o estado "Normal" ocorre uma notificação ao jogador com uma mensagem textual (além dos possíveis reflexos na representação gráfica do desenvolvedor, como será visto na seção que aborda o modelo de interações).

5.3.4 Modelo de Instanciação

O diagrama da figura 5.8 apresenta uma parte do diagrama do modelo de instanciação utilizado no jogo.

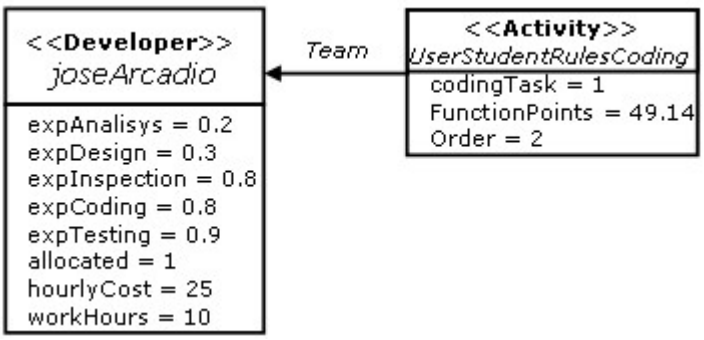

Figura 5.8: Parte do diagrama do modelo de instanciação do jogo de gerência de projetos

O trecho da figura 5.9 ilustra o trecho XML do modelo correspondente ao dia-

grama apresentado.

```
<SetupModel>
  <Instance name="joseArcadio" class="Developer">
    <Attribute name="fullName" value="Jose Arcadio Buendia"/>
    <Attribute name="nickName" value="Arcadio"/>
    <Attribute name="curriculum"
      value="Recent undergraduated. 2 years in OO programming."/>
    <Attribute name="age" value="23"/>
    <Property name="ExpAnalysis" value="0.2"/>
    <Property name="ExpDesign" value="0.3"/>
    <Property name="ExpInspection" value="0.8"/>
    <Property name="ExpCoding" value="0.9"/>
    <Property name="ExpTesting" value="0.8"/>
    <Property name="HourlyCost" value="25"/>
    <Property name="WorkHours" value="10"/>
    <Property name="Allocated" value="1"/>
  </Instance>
  <Instance name="UserStudentRulesCoding" class="Activity">
    <Attribute name="Description" value="Coding of functionalities
                                         provided for students"/>
    <Property name="CodingTask" value="1"/>
    <Property name="FunctionPoints" value="49.14"/>
    <Property name="Order" value="2"/>
  </Instance>
    <Link relation="team" source="UserStudentRulesCoding" target="joseArcadio"/>
  ...
  <MasterInstance instance="CtrlPESC"/>
</SetupModel>
```

Figura 5.9: Trecho do modelo de instanciação do jogo de gerência de projetos

O modelo é declarado com a *tag* **SetupModel**. As instâncias dos elementos são inicializadas em blocos com a *tag* **Instance**. O atributo **name** da *tag* indica o nome da instância e o atributo **class** indica a classe da instância.

Cada atributo de jogo e propriedade é identificado pelo par nome-valor, onde o nome indica o nome do atributo textual definido na classe correspondente no modelo estrutural ou o nome da propriedade definida na classe correspondente no modelo de domínio. O valor indica o valor de inicialização que corresponde a um número real para propriedades ou um texto para atributos. A *tag* **Attribute** demarca a inicialização de um atributo de jogo e a *tag* **Property** a de uma propriedade.

A *tag* **Link** identifica uma instância de um relacionamento definido no modelo de domínio de simulação. Nesta *tag*, o atributo **relation** refere-se ao nome da relação, o atributo **source** refere-se a instância origem do relacionamento e o atributo **target** refere-se à instância destino. Seguindo o exemplo na atividade "Coding", isto quer

dizer que na instância desta atividade há uma instância da relação de precedência cujo destino é a instância "Design".

A *tag* **MasterInstance** indica qual a instância que determina o estado final do jogo. O estado final do jogo será informado pela máquina de estados do elemento correspondente. No caso deste jogo, a instância do projeto ("CtrlPESC") determina de que maneira o jogo termina.

5.3.5 Modelo de Interações

No modelo de interações do jogo, é feita a descrição de dois ambientes com os quais um gerente se depara no dia a dia: o laboratório de desenvolvimento e um escritório de recursos humanos.

No laboratório, estão posicionados os desenhos dos desenvolvedores alocados ao projeto, das mesas com computadores, das cadeiras, de uma mesa de servidor e de uma porta que liga ao escritório de recursos humanos.

No ambiente de recursos humanos, estão posicionados desenvolvedores que não estão alocados ao projeto. Os desenvolvedores encontram-se sentados em cadeiras. Quando um desenvolvedor é despedido, ele passa para o ambiente de recursos humanos e quando ele é contratado ele é posicionado no laboratório.

Visões

A figura 5.10 exibe algumas imagens estáticas utilizadas no jogo. A figura 5.11 exibe a animação de um desenvolvedor caminhando.

O trecho da figura 5.12 ilustra um trecho de código XML com declarações de *visões* estáticas e dinâmicas.

Todas as visões têm um nome associado para ser referenciado em outras seções do modelo. Visões estáticas são identificadas pela *tag* **StaticImage**. Visões de animações são identificadas pela *tag* **AnimatedImage**. Para imagens estáticas, além do nome deve ser indicado o arquivo da imagem (atributo **path**). Para animações,

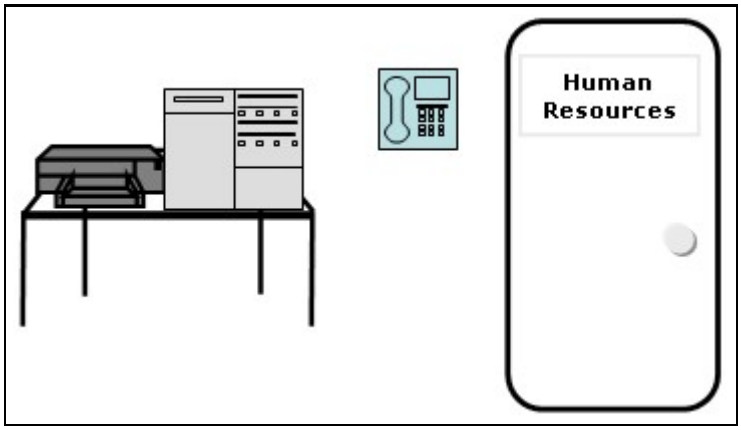

Figura 5.10: Exemplos de imagens estáticas utilizadas no jogo

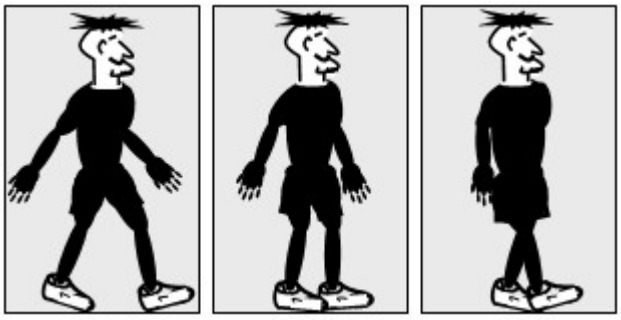

Figura 5.11: Exemplo de animação utilizada no jogo: desenvolvedor andando.

```
<InteractionModel>

  <Views>
    <!-- Sequence for developer walking to the left side-->
    <AnimatedImage name="WalkingLeft" delay="2" loop="true">
      <Frames>
        <Frame path="/images/sprites/walking_left_1.png"/>
        <Frame path="/images/sprites/walking_left_2.png"/>
        <Frame path="/images/sprites/walking_left_3.png"/>
      </Frames>
    </AnimatedImage>
    ...
    <StaticImage name="door" path="/images/sprites/door.png"/>
    <StaticImage name="server" path="/images/sprites/serverTable.png"/>
    <StaticImage name="computer" path="/images/sprites/computer.png"/>
    <StaticImage name="phone" path="/images/sprites/phone.png"/>
    <StaticImage name="blackboard" path="/images/sprites/blackboard.png"/>
  </Views>
  ...
</InteractionModel>
```

Figura 5.12: Trecho XML com declarações de visões

pode ser informado o atraso na sucessão de quadros em milessegundos e se ocorrerá ciclo na animação. A sucessão de quadros é indicada no bloco **Frames**, no qual cada *tag Frame* informa a figura que representa um quadro componente da animação.

Como pode ser observado no trecho do modelo, a definição de uma visão estática deve conter apenas o nome e o caminho do arquivo da imagem no sistema operacional.

Estados gráficos

Neste projeto de jogo, atividades e desenvolvedores são apresentados graficamente. Para cada estado lógico, pode ser definido um estado gráfico correspondente. O estado pode conter várias visões.

Os estados gráficos do desenvolvedor apresentam apenas uma visão por estado. A figura 5.13 exibe o trecho XML dos estados e eventos gráficos do desenvolvedor.

```
<InteractionStates element="Developer">
  <State name="idle">
    <View name="sitIdle"/>
  </State>
  <State name="normal">
    <View name="SitNormal"/>
  </State>
  <State name="tired" >
      <View name="SitTired"/>
  </State>
  <State name="exhausted">
    <View name="SitExhausted"/>
  </State>

  <InteractionEvents>
    <OnAction action="fireDeveloper">
      <SetView view="WalkingLeft"/>
      <MoveTo x="-100" y="300"/>
    </OnAction>
    ...
  </InteractionEvents>
</InteractionStates>
```

Figura 5.13: Trecho XML com declarações dos estados e eventos gráficos do desenvolvedor

Os estados gráficos são declarados na *tag* **State** e referenciam o estado lógico correspondente pelo atributo **name**. No interior deste bloco, podem ser feitas refe-

rências às visões declaradas no modelo.

O trecho correspondente a declaração dos eventos gráficos é demarcado pela *tag* **InteractionEvents**. Como pode ser observado na figura 5.13, o disparo da ação "fireDeveloper" pelo jogador desencadeia as ações gráficas de troca de visão da instância de desenvolvedor e a posterior movimentação do *sprite* correspondente.

Ambientes

A figura 5.14 ilustra o ambiente do laboratório. Neste exemplo, dois desenvolvedores estão alocados ao projeto. O trecho da figura 5.15 corresponde a definição em XML do ambiente de laboratório. As descrições de ambientes são feitas dentro de blocos com a *tag* **Rooms**. Cada bloco com a *tag* **Room** identifica um ambiente, com o seu nome e a imagem de fundo correspondente. Os elementos gráficos do ambiente são demarcados pelas *tags* **Item** para decorações, **Link** para *links*, ou **GraphicInstance** para elementos que representam instâncias definidas na camada de enredo. Todos os elementos gráficos do ambiente devem conter um nome e uma coordenada que indica a sua posição inicial no ambiente.

5.4 Exemplo de sessão

Nesta seção é apresentado um exemplo específico de utilização do jogo.

Após uma introdução do jogo, na qual é apresentado o projeto a ser conduzido pelo gerente, o jogador se depara com uma configuração inicial do projeto (figura 5.16), com desenvolvedores alocados ao projeto e uma rede de atividades montada. A tela inicial apresenta o ambiente de laboratório. Como pode ser observado, a tela do jogo apresenta na parte inferior um painel de controle. Este painel é dividido em três áreas: uma área com botões de controle, a área central para exibição de mensagens informativas e, na parte esquerda, uma região com variáveis de monitoramento. O primeiro botão permite a pausa ou continuação do jogo. O segundo exibe a opção de configuração de velocidade de andamento da simulação, e o terceiro exibe uma

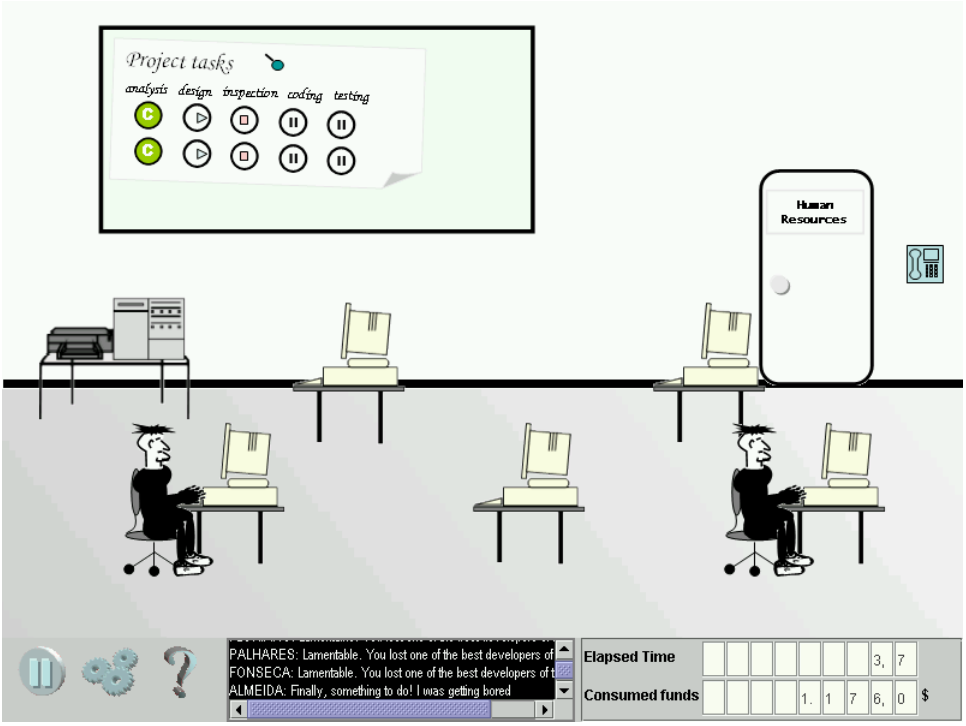

Figura 5.14: Laboratório com dois desenvolvedores

```
<Rooms>
  <Room name="lab" background="/images/rooms/workplace.gif">
    <Decorations>
      <Item name="Server" view="server" x="30" y="220"/>
      <Item name="Computer1" view="computer" x="308" y="249"/>
      <Item name="Computer2" view="computer" x="508" y="249"/>
      <Item name="Blackboard" view="blackboard" x="80" y="20"/>
      <Item name="Phone" view="Phone" x="750" y="200"/>
    </Decorations>
    <GraphicInstances>
      <GraphicInstance instance="quincasBorba" x="250" y="250"/>
      <GraphicInstance instance="joseArcadio" x="450" y="250"/>
      <GraphicInstance instance="UserStudentRulesAnalysis" x="110" y="83"/>
      <GraphicInstance instance="UserStudentRulesDesign" x="150" y="85"/>
      <GraphicInstance instance="UserStudentRulesInspection" x="190" y="87"/>
      <GraphicInstance instance="UserStudentRulesCoding" x="230" y="89"/>
      <GraphicInstance instance="UserStudentRulesTesting" x="270" y="91"/>
      <GraphicInstance instance="AreaTeacherDisciplineAnalysis" x="110" y="113"/>
      <GraphicInstance instance="AreaTeacherDisciplineDesign" x="150" y="115"/>
      <GraphicInstance instance="AreaTeacherDisciplineInspection" x="190" y="117"/>
      <GraphicInstance instance="AreaTeacherDisciplineCoding" x="230" y="119"/>
      <GraphicInstance instance="AreaTeacherDisciplineTesting" x="270" y="121"/>
    </GraphicInstances>
    <Links>
      <Link name="doorLink" view="door" x="630" y="137" targetRoom="office"/>
    </Links>
  </Room>
  ...
</Rooms>
```

Figura 5.15: Trecho XML correpondente à descrição do laboratório

mensagem de ajuda. Os botões e a área de mensagens são fixas para todos os jogos contruídos com a sistematização. A área das variáveis é configurável no modelo das interações.

A parte superior apresenta o painel onde ocorrem as interações do jogador com o ambiente de treinamamento, no caso, o laboratório e o escritório de recursos humanos.

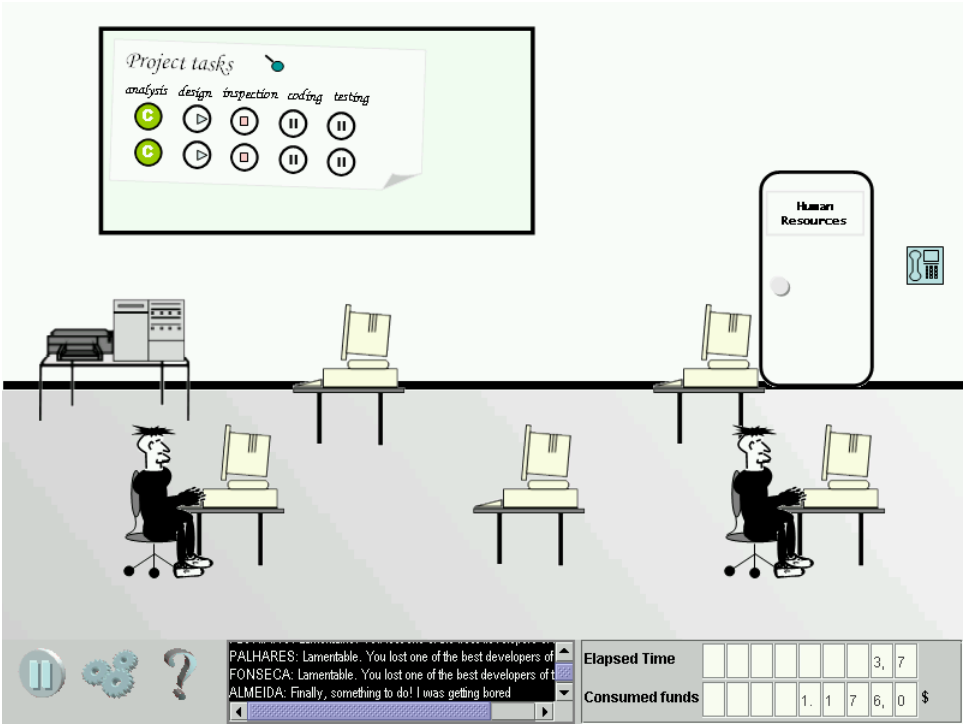

Figura 5.16: Ambiente do laboratório.

A configuração inicial do projeto contém um total de dois desenvolvedores e dez atividades, divididas em dois casos de uso do projeto chamado CtrlPESC, apresentado na introdução do jogo. As atividades estão indicadas no canto superior da tela e são divididas pelo seu tipo (análise, projeto, inspeção, codificação e teste). O jogador pode executar ações e visualizar propriedades de atividades e desenvolvedores. O cursor do *mouse* se altera a cada vez que se sobrepõe a região de uma instância, indicando ao jogador a possibilidade de uma interação. As propriedades são visualizadas com um clique duplo sobre a instância de desenvolvedor ou atividade.

A figura 5.17 ilustra a exibição de detalhes do desenvolvedor. Esta tela é acessada por meio de um duplo clique sobre o desenvolvedor. A tela apresenta o estado do desenvolvedor, os seus atributos textuais e os valores das propriedades de simulação.

O acesso às ações disponíveis em um determinado contexto de interação é feito por meio do clique no botão direito do mouse. Um menu é montado dinamicamente

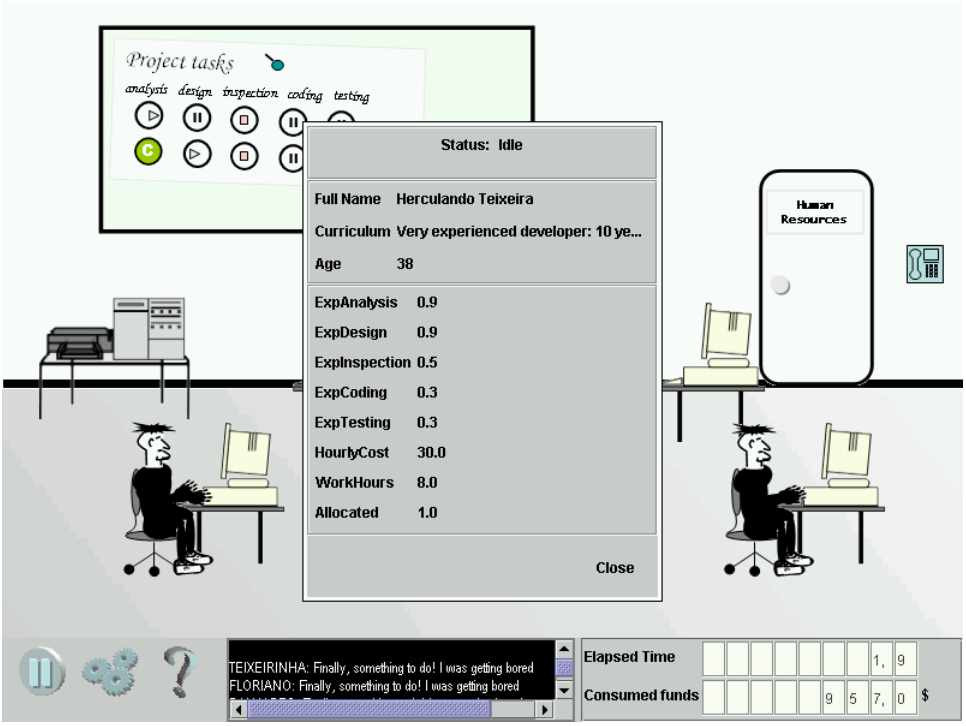

Figura 5.17: Detalhes de um desenvolvedor.

com as ações disponíveis dado o estado da instância. A figura 5.18 mostra as ações que podem ser executadas sobre um desenvolvedor alocado no projeto.

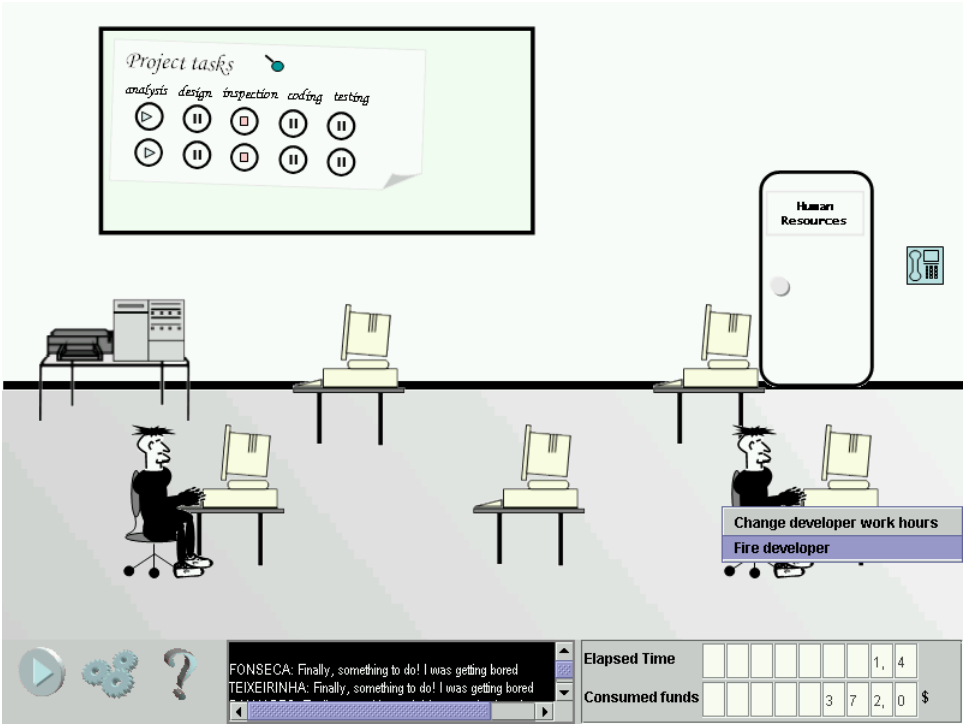

Figura 5.18: Ações disponíveis para o desenvolvedor alocado no projeto.

Caso a ação selecionada apresente parâmetros, é exibida uma caixa de diálogo com os campos necessários para o preenchimento dos valores (figura 5.19). Neste contexto específico de uso, determinou-se um regime diário de trabalho de 14 horas para o desenvolvedor "Almeida".

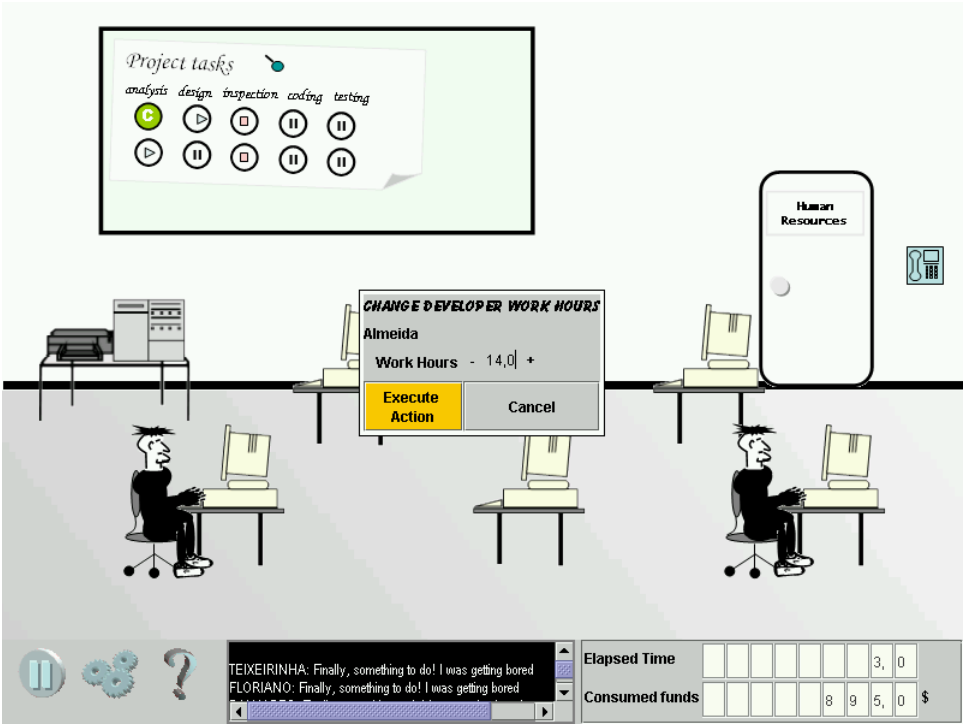

Figura 5.19: Execução uma ação com parâmetro.

Como é previsto no modelo, este regime torna-se desgastante com o passar do tempo, resultando no cansaço do desenvolvedor, conforme refletido na tela da figura 5.20.

Um clique na porta conduz ao ambiente de recursos humanos. Neste novo ambiente, pode-se encontrar novos desenvolvedores para o projeto. A figura 5.21 ilustra uma situação de contratação de um desenvolvedor.

Após a contratação, o desenvolvedor aparece no ambiente do laboratório e pode ser alocado a uma tarefa (figura 5.22).

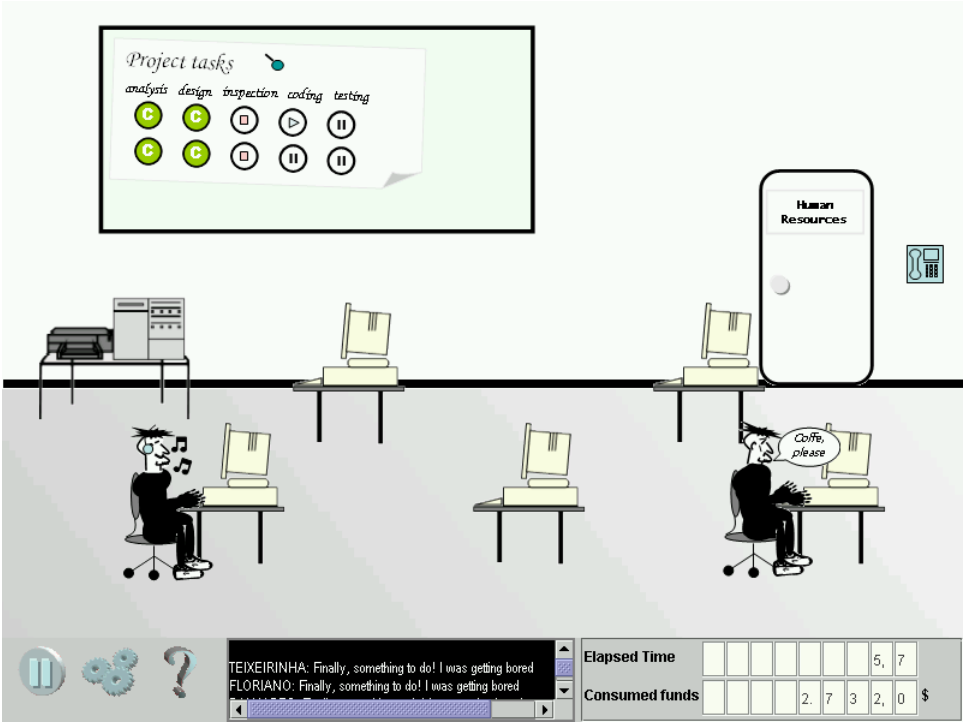

Figura 5.20: Apresentação do desenvolvedor cansado.

5.5 Conclusões

Neste capítulo, foi apresentada a estrutura do ferramental de suporte e a aplicação da sistematização, proposta no capítulo 4, na construção de um jogo de simulação para treinamento em gerência de projetos. O jogo em si não apresenta inovações na área de jogos de simulação. Porém, é importante ressaltar a maneira como o mesmo foi desenvolvido: é colocada ênfase na divisão de responsabilidades de cada dimensão do jogo.

A abordagem aqui apresentada permite que o projetista enfoque nos aspectos do domínio do treinamento, poupando tempo de desenvolvimento que seria dedicado a questões comuns em jogos computacionais como gerenciamento de *threads*, rotinas de renderização, atualização de estados de personagens, tratamentos de eventos, entre outros. Esta estrutura comum é de responsabilidade da máquina de execução e é reutilizada nos diferentes jogos de treinamento. A separação de interesses no projeto de jogos de simulação possibilita a reutilização dos aspectos modelados em diferentes jogos em um mesmo domínio.

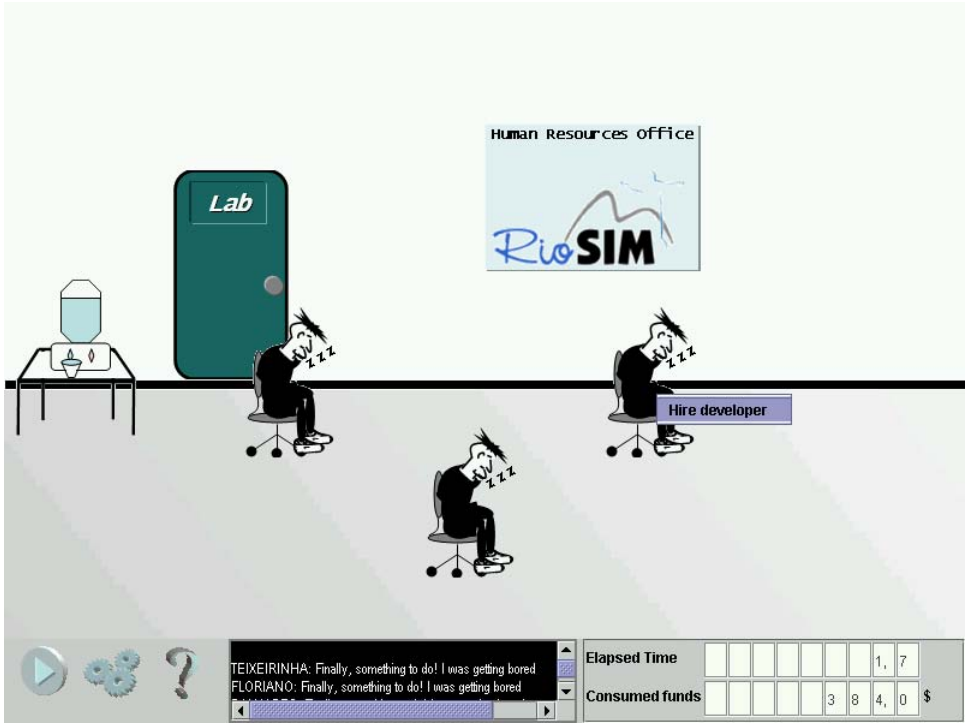

Figura 5.21: Contratação de um desenvolvedor no ambiente de recursos humanos.

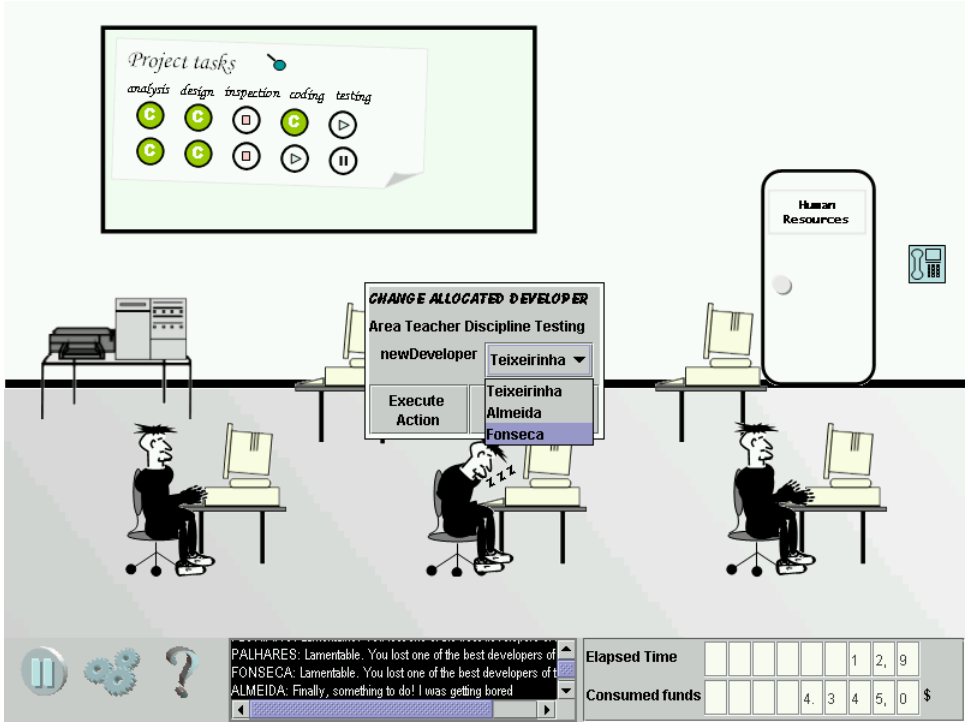

Figura 5.22: Alocação de um desenvolvedor a uma atividade.

Capítulo 6

Considerações Finais

6.1 Contribuições

Este trabalho apresentou uma proposta de sistematização do desenvolvimento de jogos baseados em modelos dinâmicos de simulação voltados para treinamento. Destacam-se como principais contribuições deste trabalho:

- a identificação dos aspectos que envolvem o projeto e implementação dos jogos de treinamento baseados em modelos de simulação, que possibilitou a definição de atividades de modelagem e artefatos a serem produzidos como resultados destas atividades. Os artefatos representam as diferentes dimensões que constituem este gênero de jogo: a simulação, o enredo (incluindo os modelos estrutural, de estados e instanciação) e as interações. As atividades ressaltam a separação de interesses existentes no projeto de jogos e a reutilização dos modelos em um domínio de aplicação;

- a criação de notações gráficas que apóiam a descrição dos modelos. Foram definidas notações gráficas para os modelos de simulação, modelo de estados e modelo de instanciação. Em Dantas (2003), foi identificada a demanda por notações gráficas mais intuitivas para o desenvolvimento dos modelos de simulação. Neste trabalho, esta demanda é suprida com a notação gráfica do

metamodelo da Dinâmica de Sistemas, que pode ser utilizada para o desenvolvimento de modelos não necessariamente destinados à construção de jogos de treinamento;

- a implementação de um propótipo da ferramenta de integração e execução dos modelos que constituem o jogo. A máquina de execução implementa as funcionalidades que são comuns em diversos jogos como gerenciamento de linhas de execução (*threads*), tratamento de eventos de interface e controle de estados. Os projetistas podem, assim, concentrar seus esforços nos desenvolvimento dos aspectos do domínio do problema;

- a modelagem de um jogo de simulação para treinamento em gerência de projetos de software, que visou avaliar se os artefatos e atividades propostos, em conjunto com o ferramental de execução, seriam capazes de apoiar o projeto de um jogo para treinamento.

6.2 Limitações

Entre as principais limitações do trabalho realizado, destacam-se:

- a abordagem foi concebida para apoiar o desenvolvimento de jogos com base em modelos da Dinâmica de Sistemas. A sua utilização em outros tipos de jogos pode ser bastante limitada ou impraticável;

- o simulador utilizado não apresenta mecanismos para o tratamento de expressões de caminho. Esta limitação dificulta o desenvolvimento de certas funcionalidades nos jogos. Esta situação pode ser ilustrada utilizando-se como o exemplo o modelo de simulação do jogo **Yamm**. A partir da classe que representa um desenvovedor, não se pode acessar diretamente dados da instância da atividade a qual o desenvolvedor está alocado. A alocação é realizada pelo relacionamento "Team", que tem como fonte a instância de atividade e como destino a instância do desenvolvedor. O efeito de consulta aos dados da atividade relacionada só pode ser conseguido, neste caso, por meio da criação de

variáveis auxiliares. Estas acabam, de certa forma, poluindo o modelo por não apresentarem um resultado direto;

- uma limitação do modelo de interações está relacionada ao grau de sofisticação dos gráficos. A estrutura dos elementos gráficos utilizada na abordagem é bastante simplificada e poderia ser estendida para possibilitar a obtenção de gráficos mais sofisticados;

- o trabalho necessita de uma validação que possibilite verificar a viabilidade da construção de jogos em diferentes domínios e também a relação entre facilidade/benefício do desenvolvimento dos jogos em relação as maneiras convencionais (baseados em linguagens de programação).

Uma percepção evidente neste trabalho é a de que a arte em jogos gráficos é um quesito fundamental para o sucesso. A riqueza de recursos de simulação e enredo pode não ser percebida pelos usuários, caso o trabalho de arte não esteja satisfatório. Isto pode ser obtido por meio de boas animações, expressividade nos gráficos dos personagens, variedade de elementos gráficos, utilização de uma combinação de cores agradável ao contexto do jogo, entre outros recursos.

6.3 Trabalhos Futuros

Além dos trabalhos que tenham como motivação a resolução das limitações apresentadas na seção anterior, são identificadas neste trabalho novas possibilidades de extensão a serem exploradas em projetos finais e teses, como:

- implementação do ferramental de edição dos modelos, que inclui a construção de diagramadores e assistentes;

- implementação dos mecanismos de registro (*logging*) das decisões do jogador. Para isto, deve ser especificada uma representação das decisões e definida a maneira como os dados de registro devem ser analisados. A análise de dados resultantes das interações do usuário pode auxiliar na detecção de padrões de

decisão dos aprendizes e pode ressaltar pontos que devem ser explorados em aulas e discussões;

- elaboração de mecanismos que permitam que o aprendizado seja realizado de maneira incremental. O aprendizado incremental poderia ser conseguido por meio da apresentação de novos desafios à medida que o jogador adquirisse experiência durante o jogo. Estes novos desafios poderiam ser implementados como novas fases do jogo;

- a extensão do modelo de interações para permitir a definição de efeitos sonoros nos jogos. Músicas e outros efeitos agregam aos jogos maior fantasia, tornado-os mais interativos. Cada ambiente do jogo poderia ter associada uma música ambiente. Além disso, transições decorrentes de alterações de estados no enredo, poderiam disparar alterações da música ambiente. Situações de tensão no jogo seriam reforçadas por sons que indicassem tensão e as situações menos tensas seriam reforçadas por sons mais alegres;

- observou-se, ao final da implementação do ferramental de execução, que o conceito de ambientes seria mais adequado ao enredo do que propriamente ao modelo de interações. Isto por que, embora um ambiente tenha uma representação gráfica, o semântica de espaço físico de trabalho não é meramente uma informação de representação. Neste sentido, poderia ser definido no enredo um modelo de ambientes, que informaria os ambientes existentes, suas conexões e a disposição das instâncias pelos ambientes. O modelo de interação, por sua vez, conteria informações de como ambientes e instâncias seriam representadas graficamente;

- o modelo de interações da forma como está estruturado, permite a associação de seqüências de imagens a classes. Por exemplo, no jogo YAMM, todos os desenvolvedores são representados graficamente da mesma forma. A abordagem poderia ser estendida para permitir a associação de gráficos específicos a instâncias, possibilitando assim uma maior variedade nos gráficos exibidos. Uma conseqüência disso no jogo exemplo seria a possibilidade de representar desenvolvedores com bonecos que fossem do sexo masculino ou feminino, altos

ou baixos, gordos e/ou magros, e assim por diante. Uma segunda conseqüência, esta negativa, seria o aumento da complexidade da modelagem da parte de interações;

- o modelo de interações poderia ser aperfeiçoado para contemplar gráficos isométricos (em duas dimensões). Este tipo de gráfico produz resultados bastante satisfatórios quando bem empregado e é utilizado por uma imensa gama de jogos de entretenimento disponíveis no mercado;

- mecanismos de retrocesso nas decisões poderiam ser implementados na máquina do jogo. Isto permitiria aos jogadores retornar o jogo a um estado anterior a uma determinada decisão, fornecendo a eles maiores possibilidades de experimentação no processo de tomada de decisões;

- desenvolvimento na máquina do jogo de uma funcionalidade de persistência do estado geral do jogo, possibilitanto recomeçar o jogo a partir de um ponto atigindo em alguma sessão anterior;

- o jogo de gerência de projetos proposto poderia ser aperfeiçado para contemplar um maior número de estados gráficos e maior número de ambientes, ampliando o universo de interações a serem exploradas pelos aprendizes de gerentes.

Referências Bibliográficas

Ahdell, R. e Andresen, G. (2001). Games and simulation in workplace e-learning. Dissertação de Mestrado, Norwegian University of Science and Technology, Noruega.

Barros, M. O. (2001). *Gerenciamento de Projetos Baseado em Cenários: Uma Abordagem Baseada em Modelagem Quantitativa e Simulação*. Tese de Doutorado, COPPE - Universidade Federal do Rio de Janeiro.

Barros, M. O., Werner, C. M. L., e Travassos, G. H. (2001). From models to metamodels: Organizing and reusing domain knowledge in system dynamics model development. In *Proceedings of The 19th International Conference of the System Dynamics Society*, Atlanta, USA.

Basnet, C. (1996). Simulation games in production management education - a review. Technical report, Department of Management Systems. University of Waikato.

Cavaleri, S., Raphael, M., e Filleti, V. (2002). Evaluating the performance efficacy of systems thinking tools. In *Proceedings of The 20th International Conference of the System Dynamics Society*, Palermo, Italy.

Dantas, A. R. (2003). Jogos de simulação no treinamento de gerentes de projetos de software. Dissertação de Mestrado, COPPE - Universidade Federal do Rio de Janeiro.

Dempsey, J. V., Rasmussen, K., e Lucassen, B. (1996). The instructional gaming literature: Implications and 99 sources. Technical Report 96-1, College of Education. University of South Alabama.

Drappa, A. e Ludewig, J. (2000). Simulation in software engineering training. In *Proceedings of The 22nd International Conference on Software Engineering - ICSE*, pages 199–208, Limerick, Ireland.

EA, S. (2004a). Sim city. Disponível na web. URL: http://thesims.ea.com/. Acessado em 25/03/2004.

EA, S. (2004b). The sims. Disponível na web. URL: http://thesims.ea.com/. Acessado em 25/03/2004.

Feinstein, A. H. e Cannon, H. M. (2001). Fidelity, verifiability, and validity of simulation: Constructs for evaluation. Working Paper. Wayne State University. Marketing Department.

Forrester, J. W. (1961). *Industrial Dynamics*. The MIT Press, Massachussets, USA.

Forrester, J. W. (1991). System dynamics and the lessons of 35 years. Technical Report D-4224-4, Sloan School of Management,MIT, Cambridge, Massachussets, USA. A chapter for The Systemic Basis of Policy Making in the 1990s.

Gage (2003). Biblioteca de jogos para java. Publicado na web. URL: http://www.dnsalias.com. Acessado em 12/12/2003.

Greenblat, C. S. (1988). *Designing Games and Simulations*. Sage Publications, Inc., USA, 2 edition.

Größler, A. (2000). Methodological issues of using business simulators in teaching and research. In *Proceedings of The 18th International Conference of The System Dynamics Society*, 18, Bergen, Norway.

Hsu, E. (1989). Role-event gaming simulation in management education. a conceptual framework and review. *Simulation & Games*, 20(4):409–438.

Kearsley, G. (2001). Explorations in learning & instruction. Publicado na web. URL: http://tip.psychology.org. Acessado em 11/03/2002.

Klabbers, J. H. (2002). Enhancing corporate change: the case of strategic human resource management. In *Proceedings of Manufacturing Complexity Network Conference.'Tackling Industrial Complexity: the ideas that make a difference'*, Downing College, Cambridge, UK.

Klabbers, J. H. G. (2001). The emerging field of simulation & gaming: Meanings of a retrospect. *Simulation & Gaming*, 32(4):471–480.

Kriz, W. C. e Rizzi, P. (1999). Environmental education and training of systems-competence with gaming/simulation. In *Proceedings of Euroconference, Quality of Life, Susteinability, Environmental Changes*, Rust, Österreich.

Madachy, R. J. (2001). Software process modeling with system dynamics. Publicado na web. URL: http://www-rcf.usc.edu/ madachy/sd/sd.html. Acessado em 18/09/2003.

Maier, F. H. e Größler, A. (1998). A taxonomy for computer simulations to support learning about socio-economic systems. In *Proceedings of 16th International Conference of The System Dynamics Society*, 16, Quebec, Canada.

Maier, F. H. e Stohhecker, J. (1996). Do management flight simulators really enhance decision effectiveness? In *Proceedings of 14th International System Dynamics Conference*, number 14 in 14, Cambridge, Massachusetts, USA.

Martin, A. (2000). A simulation engine for custom project management education. *Instructional Journal of Project Management*, (18):201–213.

Martin, L. A. (1997). The first step. Technical Report D-4694, MIT, Cambridge, Massachussets, USA. Prepared for the MIT System Dynamics in Education Project under supervision of Dr. Jay W. Forrester.

Murphy, E. (1997). Construtivism, from philosophy to practice. Publicado na web. URL: http://www.stemnet.nf.ca/ elmurphy/emurphy/cle.html. Acessado em 08/03/2002.

Oh, E., Baker, A., e van der Hoek, A. (2004). Teaching software engineering using simulation games. In *In Proceedings of the 2004 International Conference on Simulation in Education,*, San Diego, California.

Oh, E. e van der Hoek, A. (2001). Adapting game technology to support individual and organizational learning. In *Proceedings of The 13th International Conference on Software Engineering and Knowledge Engineering*, 13, pages 347–354, Buenos Aires, Argentina.

OMG (2000). Unified modeling language specification. Publicado na web. URL: http://www.omg.org/uml. Acessado em 11/01/2003.

P.E.D., L., Holt, G., Shen, L., H.Li, e Irani, Z. (2002). Using system dynamics to better understand change and rework in contruction project management systems. *International Journal of Project Management*, 20:425–436.

Pfahl, D. e Ruhe, G. (2001). System dynamics as an enabling technology for learning in software organisations. In *Proceedings of The 13th International Conference on Software Engineering and Knowledge Engineering*, number 13 in 13, pages 355–362, Buenos Aires, Argentina.

Prensky, M. (2001). *Digital Game-Based Learning*. McGraw-Hill.

Rieber, L. P. (1996). Seriously considering play: Designing interactive learning environments based on the blending of microworlds,simulations, and games. In *Educational Technology Research & Development*, pages 43–58.

Romme, A. G. L. (2002). Microworlds for management education and learning. Working Paper. Tilburg University. Faculty of Economics & Bussiness Administration.

Rouse, R. (2001). Game design: Theory and practice, chapter seven. the elements of gameplay. Publicado na web. URL: http://www.gamasutra.com/features/20010627/rouse_pfv.htm. Acessado em 11/02/2003.

Saunders, J. H. (2002). The management flight simulator. Publicado na web. URL: http://www.johnsaunders.com/papers/mfs.htm. Acessado em 30/08/2002.

Schmucker, K. (1999). Learning technology review: A taxonomy of simulation software. Disponível na web. URL: http://www.apple.com/education/LTReview/spring99/simulation/. Acessado em 25/05/2003.

Spector, J. M. (1998). Tools and principals for the design of collaborative learning environments for complex domains. Based on a presentation at AERA 98, San Diego, CA.

Spector, J. M. (2000). System dynamics and interactive learning environments: Lessons learned and implications for the future. *Simulation & Gaming*, 31(4):509–516.

Spector, J. M. e Davidsen, P. I. (1998). Constructing learning environments using system dynamics. *Journal of Courseware Engineering*, 1:5–11.

Sterman, J. D. (1988). A skeptic's guide to computer models. Technical Report D-1401-1, Sloan School of Management, MIT, Cambridge, Massachussets, USA.

SUN, M. (2003). Java 2 standard edition 1.4.2. Disponível na web. URL: http://java.sun.com/. Acessado em 10/09/2003.

Wills, A. C. (1997). Models and code: the connection. Publicado na web. URL: http://www.trireme.u-net.com/catalysis/refinement2.pdf. Acessado em 20/12/2003.

Ye, Z. e Ye, D. (2002). A process model for game design and development. In *Proceedings of Game Technology Conference*, China.

Apêndice A

Modelos do Jogo de Gerência de Projetos

A.1 Modelo de Simulação

```
#
# Metamodel that represents a software development project
#
MODEL ProjectModel
{
# ------------------------------------------------------------------------
# Class that represents a developer participating in a software project
# ------------------------------------------------------------------------
CLASS Developer
{
PROPERTY ExpAnalysis 0; # range [0, 1]
PROPERTY ExpDesign 0; # range [0, 1]
PROPERTY ExpCoding 0; # range [0, 1]
PROPERTY ExpTesting 0; # range [0, 1]
PROPERTY ExpInspection 0; # range [0, 1]
PROPERTY HourlyCost 0; # in $

# Activity to which the developer is associated
PROC AssociatedTask Groupmin(Bound([Activity], Team), DeveloperNeed);

# Developer's Productivity for each activity
PROC Cost HourlyCost * 8;
```

```
# Developer's Productivity for each activity
PROC  Productivity 1;
PROC  ProdAnalysis Productivity;
PROC  ProdDesign Productivity;
PROC  ProdCoding Productivity;
PROC  ProdTesting Productivity;
PROC  ProdInspection Productivity;

# Developer's error generation rate for each activity
PROC  ErrorGenerationRate 1;
PROC  ErrorRateAnalysis ErrorGenerationRate;
PROC  ErrorRateDesign ErrorGenerationRate;
PROC  ErrorRateCoding ErrorGenerationRate;
PROC  ErrorRateTesting ErrorGenerationRate;
};

# ------------------------------------------------------------------------
# Class that represents an activity in a software project
# ------------------------------------------------------------------------
CLASS Activity
{
PROPERTY AnalysisTask 0; # 0 or 1
PROPERTY InitialDesignTask 0; # 0 or 1
PROPERTY DetailedDesignTask 0; # 0 or 1
PROPERTY CodingTask 0; # 0 or 1
PROPERTY TestingTask 0; # 0 or 1
PROPERTY InspectionTask 0; # 0 or 1
PROPERTY MinimumDuration 0; # in days
PROPERTY ExpectedDuration 0; # in days
PROPERTY MaximumDuration 0; # in days
PROPERTY Order   0; # 0 or +

# Set the activity execution time
PROC ExpectedTime ExpectedDuration;
PROC MaximumTime MaximumDuration;
PROC MinimumTime MinimumDuration;

STOCK ExecutionTime IF(OR(MaximumDuration <=0, MinimumDuration <=0),
ExpectedTime, BETAPERT(MinimumTime, ExpectedTime, MaximumTime));
PROC ETAjuste (ExpectedTime - ExecutionTime) / DT;
RATE (ExecutionTime) RTExecutionTime ETAjuste;

PROC PrecedenceTime GROUPMAX(Precedences, ExpectedTime);
PROC AccumulatedTime PrecedenceTime + ExpectedTime;

# Determine if precedent activities are concluded
PROC PrecConcluded AND (GroupMax (Precedences, PrecConcluded) >= 0,
```

```
GroupMax (Precedences, RemainingTime) < 0.001);

# Determine if the activity is concluded
PROC Concluded RemainingTime < 0.001;

# Determine if the activity is ready to run
PROC Ready AND (PrecConcluded, NOT(Concluded));

# Determine if there are resources available for the activity
PROC DeveloperNeed IF (Ready, Order, 1000);
PROC Executing AND (Ready, Team.AssociatedTask = Order);

# Determine developer producticity
PROC Productivity ((AnalysisTask * Team.ProdAnalysis)  +
(InitialDesignTask * Team.ProdDesign) + (DetailedDesignTask * Team.ProdDesign) +
(CodingTask * Team.ProdCoding) + (TestingTask * Team.ProdTesting) +
(InspectionTask * Team.ProdInspection)) * DT;

# Determine activity executed time based on developers productivity
STOCK ExecutedTime 0;
RATE (ExecutedTime) RTExecTime IF (Executing, if
(OR(InspectionTask>0, TestingTask>0), MIN(RemainingTime, DT),
MIN(RemainingTime, Productivity)), 0) / DT;
PROC RemainingTime ExecutionTime - ExecutedTime;

# Calculates conclusion time for an activity
STOCK ConclusionTime 0;
RATE (ConclusionTime) RTConclusionTime
if(AND(ConclusionTime < 0.01, RemainingTime-RTExecTime*DT < 0.01), TIME/DT+1, 0);

# Accumulates activity cost
STOCK Cost 0;
RATE (Cost) RTCost IF (Executing, Team.Cost, 0);

# Accumulates days executing
STOCK DaysExecuting 0;
RATE (DaysExecuting) RTDaysExecuting IF (Executing, 1, 0);

# Errors latent in the activity
STOCK Errors 0;
RATE (Errors) RTErrors 0;
};

# ------------------------------------------------------------------------
# Class that represents a software project
# ------------------------------------------------------------------------
CLASS Project
{
```

```
PROC Concluded GROUPMIN ([Activity], Concluded);

STOCK ProjectTime 0;
RATE (ProjectTime) RTProjectTime IF (Concluded, 0, 1);

PROC ProjectCost GROUPSUM([Activity], Cost);



PROC ExpectedTime GROUPMAX([Activity], AccumulatedTime);

PROC AvgDevWorkHours 8;
PROC AvgDevCost 30.00;

PROC ExpectedTotalTime GROUPSUM([Activity], ExpectedTime);
PROC ExpectedCost ExpectedTotalTime * AvgDevCost * AvgDevWorkHours;

###### Properties and equations add due to the game

#Maximum permitted project cost
PROPERTY MaximumCost 0;

#Maximum time permited
PROPERTY MaximumTime 0;

#When the project cost extrapolates
PROC OverBudget (ProjectCost > MaximumCost);

#When the project time extrapolates
PROC OverTime (ProjectTime > MaximumTime);

#Bad projections in time
PROC BadTimeEstimation (ExpectedTime > MaximumTime);

#If the project is being executed
PROC Executing GROUPMAX ([Activity], Executing);

};

# ------------------------------------------------------------------------------
# Relationships among classes
# ------------------------------------------------------------------------------

MULTIRELATION Precedences Activity, Activity (Successors);

RELATION Team Activity, Developer;
};
```

```
#
# Scenario that reflects the variation in productivity due to communication overhead
#
SCENARIO ProductivityLossDueCommunication ProjectModel
{
CONNECTION TheDeveloper Developer
{
PROC COHFactor Count ([Developer]);
PROC  COHModifier LOOKUP (COHTable, COHFactor, 0, 30);
TABLE COHTable 0, 0.015, 0.06, 0.135, 0.24, 0.375, 0.54;

AFFECT Productivity Productivity * (1 - COHModifier);
};
};



#
# Scenario that reflects the variation in productivity due to experience
#
SCENARIO ProductivityDueExpertise ProjectModel
{
CONNECTION TheDeveloper Developer
{
AFFECT ProdAnalysis (0.667 + ExpAnalysis * 0.666) * ProdAnalysis;
AFFECT ProdDesign (0.667 + ExpDesign * 0.666) * ProdDesign;
AFFECT ProdCoding (0.667 + ExpCoding * 0.666) * ProdCoding;
AFFECT ProdTesting (0.667 + ExpTesting * 0.666) * ProdTesting;
AFFECT ProdInspection (0.667 + ExpInspection * 0.666) * ProdInspection;
};
};



#
# Scenario that reflects the variation in error generation rates due to experience
#
SCENARIO ErrorGenerationDueExpertise ProjectModel
{
CONNECTION TheDeveloper Developer
{
AFFECT ErrorRateAnalysis (0.667 + (1 - ExpAnalysis) * 0.666) * ErrorRateAnalysis;
AFFECT ErrorRateDesign (0.667 + (1 - ExpDesign) * 0.666) * ErrorRateDesign;
AFFECT ErrorRateCoding (0.667 + (1 - ExpCoding) * 0.666) * ErrorRateCoding;
AFFECT ErrorRateTesting (0.667 + (1 - ExpTesting) * 0.666) * ErrorRateTesting;
};
};



#
```

```
# Scenario that reflects the gain in productivity due to learning
#
SCENARIO ProductivityDueLearning ProjectModel
{
CONNECTION TheDeveloper Developer
{
STOCK DaysInProject 0;
RATE  (DaysInProject) DaysInProjectCounter 1;

PROC  DIPFactor MIN (DaysInProject / 20, 1.0);
PROC  DIPModifier LOOKUP (LearningTable, DIPFactor, 0, 1);
TABLE LearningTable 1.0, 1.0125, 1.0325, 1.055, 1.09, 1.15, 1.2, 1.22, 1.245, 1.25, 1.25;

AFFECT Productivity Productivity * DIPModifier;
};
};



#
# Scenario that reflects the variation in productivity/error generation due to overworking
#
SCENARIO Overworking ProjectModel
{
CONNECTION TheDeveloper Developer
{
PROPERTY WorkHours 8; # 8 to 12 hours

STOCK DailyWorkHours WorkHours;
PROC DWHAjuste (WorkHours - DailyWorkHours) / DT;
RATE (DailyWorkHours) DWHRate1 DWHAjuste;
PROC  WHModifier 1 + (DailyWorkHours - 8) / (12 - 8);

PROC  SEModifier LOOKUP (SchErrorsTable, WHModifier-1, 0, 1);
TABLE SchErrorsTable 0.9, 0.94, 1, 1.05, 1.14, 1.24, 1.36, 1.5;

AFFECT Cost Cost * DailyWorkHours / 8;
AFFECT Productivity Productivity * WHModifier;
AFFECT ErrorGenerationRate ErrorGenerationRate * SEModifier;
};
};



#
# Scenario that reflects the willingness to overwork due to exhaustion
#
SCENARIO Exhaustion ProjectModel
{
CONNECTION TheDeveloper Developer
```

```
{
STOCK Exhaustion 0;
PROC  MaxExhaustion 50;
RATE  (Exhaustion) ExhaustionRate if(OR(Resting = 1, AssociatedTask >= 1000),
 Max(-Exhaustion/DT, -MaxExhaustion / 20.0), EXModifier);

PROC  EXModifier LOOKUP (ExaustionTable, DedicationFactor, 0, 1.5);
PROC  DedicationFactor 1 - (1 - Dedication) / 0.4;
PROC  Dedication 0.6 + (WHModifier - 1) * (1.2 - 0.6);
TABLE ExaustionTable 0.0, 0.0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9, 1.15, 1.3, 1.6, 1.9, 2.2, 2.5;

STOCK Resting 0;
RATE  (Resting) RestingRate1 IF (InitResting, 1 / DT, 0);
RATE  (Resting) RestingRate2 IF (QuitResting, -1 / DT, 0);
RATE  (DailyWorkHours) DWHRate2 IF (Resting = 1, (8 - DailyWorkHours) / DT, 0);

PROC  InitResting AND(Resting = 0, Exhaustion > MaxExhaustion);
PROC  QuitResting AND(Resting = 1, Exhaustion < 0.1);

AFFECT DWHAjuste IF (Resting = 1, 0, DWHAjuste);


};

CONSTRAINT TheDeveloper, Overworking.TheDeveloper;
};

#
# Scenario that introduces query variables about the developer's state in the project
#
SCENARIO StateInProject ProjectModel
{
CONNECTION TheDeveloper Developer
{
#Determines if the developes is allocated to the project
PROPERTY Allocated 1;  # Values -1 or 1

# Verify is developer has no task to work on
PROC Idle NOT(GROUPMAX(Bound([Activity],Team),Executing));
#PROC Idle (AssociatedTask = 1000);

PROC Tired AND(Exhaustion > 10, Exhaustion < MaxExhaustion);

PROC Exhausted (Exhaustion >= MaxExhaustion);

};
```

```
CONNECTION TheActivity Activity
{
#Determines if the activity must be executed on the project.
It is valid only for inspection tasks
PROPERTY On 1;  # Values -1 or 1

#Verify if the activity can be turned off
PROC CanTurnOff AND(On,NOT(Concluded));

#If is inspection taskl inspection then considers on, else doesn't care
PROC InspectionPlugged IF(InspectionTask,On,1);

# Redefinition of process to consider developer's allocation in the project
and if the activity must be executed
AFFECT Executing AND (AND(Executing,Team.Allocated),On);

};

CONSTRAINT TheDeveloper, Exhaustion.TheDeveloper;
};



#
# Scenario that allows measuring an activity in function points
#
SCENARIO FunctionPointMeasure ProjectModel
{
CONNECTION TheActivity Activity
{
PROPERTY FunctionPoints 0; # in FP
PROC  TaskFunctionPoints 0; # FunctionPoints * TaskContribution
};
};



#
# Scenario that estimates activity duration using function points
#
SCENARIO EstimationFunctionPoints ProjectModel
{
CONNECTION TheActivity Activity
{
# Developer's productivity
PROC AvgFPMonth 27.80;
PROC MinFPMonth 15.41;
PROC MaxFPMonth 40.19;

# Evaluate the contribution of the activity type to the project
```

```
PROC  ContrAnalysis AnalysisTask * 4.11144;
PROC  ContrInitialDesign InitialDesignTask * 4.55393;
PROC  ContrDetailedDesign DetailedDesignTask * 6.07652;
PROC  ContrCoding CodingTask * 25.31653;
PROC  ContrTesting TestingTask * 59.94159;
PROC  Contribution ContrAnalysis + ContrInitialDesign + ContrDetailedDesign
+ ContrCoding + ContrTesting;

# Evaluate the expected activity execution time
AFFECT ExpectedTime (ExpectedTime * 0.0) + FunctionPoints * (30.0 / AvgFPMonth) * (Contribution / 100);
AFFECT MaximumTime (MaximumTime * 0.0) + FunctionPoints * (30.0 / MinFPMonth) * (Contribution / 100);
AFFECT MinimumTime (MinimumTime * 0.0) + FunctionPoints * (30.0 / MaxFPMonth) * (Contribution / 100);

AFFECT TaskFunctionPoints (TaskFunctionPoints * 0.0) + FunctionPoints * (Contribution / 100);
};

CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents error generation in an activity
#
SCENARIO ErrorGeneration ProjectModel
{
CONNECTION TheActivity Activity
{
PROC NewErrors (AnalysisTask + InitialDesignTask + DetailedDesignTask + CodingTask)
* ErrorGeneration;
PROC ErrorGen ((AnalysisTask * Team.ErrorRateAnalysis)  +
(InitialDesignTask * Team.ErrorRateDesign) + (DetailedDesignTask * Team.ErrorRateDesign)
+ (CodingTask * Team.ErrorRateCoding));
PROC ErrorGeneration if (ExecutionTime > 0, ErrorsFP *
FunctionPoints * ErrorGen * RTExecTime / ExecutionTime, 0);
PROC ErrorsFP (AnalysisTask * 0.6)  + (InitialDesignTask * 0.18) +
(DetailedDesignTask * 0.57) + (CodingTask * 1.05);
AFFECT RTErrors RTErrors + NewErrors;
};

CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that represents error propagation between two sequential activities
#
SCENARIO ErrorPropagation ProjectModel
{
```

```
CONNECTION TheActivity Activity
{
PROC SuccessorCount COUNT(Successors);
PROC ErrorsToGo if (SuccessorCount > 0, (Errors + RTErrors*DT) /  SuccessorCount, 0);

PROC Initializing AND (ExecutionTime > 0.0, ExecutedTime < 0.001, Executing);
PROC ReadyToInherit OR (AND(ExecutionTime < 0.001, PrecConcluded, Errors < 0.001), Initializing);
PROC InheritedErrors if (ReadyToInherit, GROUPSUM (Precedences, ErrorsToGo), 0);

AFFECT RTErrors RTErrors + InheritedErrors / DT;
};

CONSTRAINT TheActivity.Precedences, ErrorPropagation.TheActivity;
};



#
# Scenario that represents error regeneration on sequential activities
#
SCENARIO ErrorRegeneration ProjectModel
{
CONNECTION TheActivity Activity
{
PROC InheritedDensity InheritedErrors / FunctionPoints;
PROC RegenErrors (AnalysisTask + InitialDesignTask + DetailedDesignTask + CodingTask)
* InheritedErrors * 0.24 * RegenFactor;
PROC RegenFactor Max (1, LOOKUP (ActiveErrosDens, InheritedDensity , 0, 10));
TABLE ActiveErrosDens 1, 1.1, 1.2, 1.325, 1.45, 1.6, 2.0, 2.5, 3.25, 4.35, 6.0;
AFFECT RTErrors RTErrors + RegenErrors / DT;
};

CONSTRAINT TheActivity, ErrorPropagation.TheActivity;

CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};



#
# Scenario that represents error correction in an activity
#
SCENARIO ErrorCorrection ProjectModel
{
CONNECTION TheActivity Activity
{
PROPERTY Target 95.0; # in %

# Average errors per function point
PROC AvgErrorFP 2.4;
```

```
# Errors corrected acummulator for the activity
STOCK ErrorsCorrected 0;
RATE (ErrorsCorrected) RTCorrection CorrErrors;

# Error correction in the activity
RATE (Errors) RTCorrErrors -CorrErrors;
PROC CorrErrors (InspectionTask + TestingTask) * RTExecTime * Productivity / (DetectionCost * DT);

# Cost to correct an error
PROC DetectionCost 0.28;

# Adjustment of time for testng activities
RATE (ExecutionTime) RTTesting if (AND(Executing, TestingTask > 0),
-ExecutionTime+ExecutedTime+TestingEffort, 0) / DT;
PROC TestingEffort TestingDifference * DetectionCost;
PROC TestingDifference Max(Errors + InheritedErrors - CorrErrors - TestingTarget, 0);
PROC TestingTarget FunctionPoints * AvgErrorFP * (1 - Target / 100.0);
};

CONSTRAINT TheActivity, FunctionPointMeasure.TheActivity;
};


#
# Scenario that introduces new erros from error corretion
#
SCENARIO BadFixes ProjectModel
{
CONNECTION TheActivity Activity
{
PROC DefaultBadFixes 7.5;
PROC BadFixes CorrErrors * (DefaultBadFixes / 100.0) * Team.ErrorRateTesting;

RATE (Errors) RTBadFixes BadFixes;
};

CONSTRAINT TheActivity, ErrorCorrection.TheActivity;
};


#
# Scenario that represents the density effect over error correction
#
SCENARIO ErrorCorrectionDensity ProjectModel
{
CONNECTION TheActivity Activity
{
```

```
PROC ErrorDensityMultiplier Max (1, LOOKUP (TableErrorDensityMultiplier, ErrorDensity, 0, 1));
TABLE TableErrorDensityMultiplier 8, 6.75, 5.25, 4, 3, 2, 1.8, 1.6, 1.2, 1.1, 1;
PROC ErrorDensity (Errors + RTErrors*DT) / FunctionPoints;
AFFECT DetectionCost DetectionCost * ErrorDensityMultiplier;
};

CONSTRAINT TheActivity, ErrorCorrection.TheActivity;
};
```

A.2 Modelo Estrutural

```
<StructuralModel>
<Element class="Developer">
<Attribute name="fullName" alias="Full Name"/>
<Attribute name="nickName" alias="Nick Name"/>
<Attribute name="curriculum" alias="Curriculum"/>
<Attribute name="age" alias="Age" />
<Action name="changeWorkHours" alias="Change developer work hours"
restrict="Allocated" vaue="true">
<Parameter name="newWorkHours" alias="Work Hours" type="double" min="0" max="20"/>
  <SetProperty property="WorkHours" value="newWorkHours" object="self"/>
</Action>
<Action name="fireDeveloper" alias="Fire developer" restrict="Allocated" value="true">
  <SetProperty property="Allocated" value="-1" object="self"/>
    </Action>
<Action name="hireDeveloper" alias="Hire developer" restrict="Allocated" value="false">
  <SetProperty property="Allocated" value="1" object="self"/>
</Action>
  <Scenarios>
    <Connect scenario="ProductivityLossDueCommunication"/>
    <Connect scenario="ProductivityDueExpertise"/>
    <Connect scenario="ErrorGenerationDueExpertise"/>
    <Connect scenario="ProductivityDueLearning"/>
    <Connect scenario="Overworking"/>
    <Connect scenario="Exhaustion"/>
    <Connect scenario="StateInProject"/>
   </Scenarios>
  </Element>



<Element class="Activity">
<Attribute name="name" alias="Activity"/>
<Attribute name="description" alias="Description"/>
```

```
<Action name="changeDeveloper" alias="Change allocated developer"
restrict="On" restrictionValue="true">
      <!-- Parameter rectricted to allocated developers -->
      <Parameter name="newDeveloper" alias="New Developer" type="Developer"
      restrict="Allocated" value="true"/>
<SetLinkTarget relation="team" source="self" newTarget="newDeveloper"/>
    </Action>
<Action name="turnOn" alias="Turn On" restrict="On" value="false">
<SetProperty property="On" value="1" object="self"/>
    </Action>
<Action name="turnOff" alias="Turn Off" restrict="CanTurnOff" value="true">
<SetProperty property="On" value="-1" object="self"/>
    </Action>

    <Scenarios>
      <Connect scenario="FunctionPointMeasure"/>
      <Connect scenario="EstimationFunctionPoints"/>
      <Connect scenario="ErrorGeneration"/>
      <Connect scenario="ErrorPropagation"/>
      <Connect scenario="ErrorRegeneration"/>
      <Connect scenario="ErrorCorrection"/>
      <Connect scenario="ErrorCorrectionDensity"/>
      <Connect scenario="BadFixes"/>
      <Connect scenario="StateInProject"/>

    </Scenarios>
</Element>

<Element class="Project">
<Attribute name="name" alias="Project"/>
<Attribute name="description" alias="Project Description"/>
</Element>


</StructuralModel>
```

A.3 Modelo de Estados

```
<StateModel>
<EventSet element="Developer">
<State name="idle" alias="Idle" initial="true">
<Transition target="normal">
<Condition variable="Idle" value="false"/>
```

```
<ShowReport message="developerIdleToNormal"/>
</Transition>
<Transition target="unemployed">
<Condition variable="Allocated" value="false"/>
<ShowReport message="developerIdleToUnemployed"/>
</Transition>
</State>
<State name="normal" alias="Normal">
<Transition target="tired">
<Condition variable="Tired" value="true"/>
<ShowReport message="developerNormalToTired"/>
</Transition>
<Transition target="idle">
<Condition variable="Idle" value="true"/>
<ShowReport message="developerNormalToIdle"/>
</Transition>
<Transition target="unemployed">
<Condition variable="Allocated" value="false"/>
<ShowReport message="developerNormalToUnemployed"/>
</Transition>
</State>
<State name="tired" alias="Tired">
<Transition target="normal">
<Condition variable="Tired" value="false"/>
<Condition variable="Exhausted" value="false"/>
<ShowReport message="developerTiredToNormal"/>
</Transition>
<Transition target="exhausted">
<Condition variable="Exhausted"/>
<ShowReport message="developerTiredToExhausted"/>
</Transition>
<Transition target="idle">
<Condition variable="Idle"/>
<ShowReport message="developerTiredToIdle"/>
</Transition>
<Transition target="unemployed">
<Condition variable="Allocated" value="false"/>
<ShowReport message="developerTiredToUnemployed"/>
</Transition>
</State>
<State name="exhausted" alias="Exhausted">
<Transition target="tired">
<Condition variable="Tired" value="true"/>
<Condition variable="Exhausted" value="false"/>
<ShowReport message="developerExhaustedToTired"/>
</Transition>
<Transition target="idle">
<Condition variable="Idle" value="true"/>
```

```
<ShowReport message="developerExhaustedToIdle"/>
</Transition>
<Transition target="unemployed">
<Condition variable="Allocated" value="false"/>
<ShowReport message="developerExhaustedToUnemployed"/>
</Transition>
</State>

<State name="unemployed" alias="Unemployed">
<Transition target="Idle">
<Condition variable="Allocated" value="true"/>
<ShowReport message="developerUnemployedToIdle"/>
</Transition>
</State>

</EventSet>

<EventSet element="Activity">
<State name="waiting" alias="Waiting" initial="true">
<Transition target="executing">
<Condition variable="Executing" value="true"/>
</Transition>
<Transition target="off">
<Condition variable="On" value="false"/>
</Transition>
</State>

<State name="executing" alias="Executing">
<Transition target="concluded">
<Condition variable="Concluded" value="true"/>
</Transition>
<Transition target="waiting">
<Condition variable="Executing" value="false"/>
<Condition variable="On" value="true"/>
</Transition>
<Transition target="off">
<Condition variable="On" value="false"/>
</Transition>
</State>

<State name="off" alias="Turned off">
<Transition target="waiting">
<Condition variable="On" value="true"/>
<Condition variable="Executing" value="false"/>
</Transition>
<Transition target="executing">
<Condition variable="On" value="true"/>
<Condition variable="Executing" value="true"/>
```

```
</Transition>
</State>

<State name="concluded" alias="Concluded" final="true">
</State>
</EventSet>

<EventSet element="Project">
<State name="waiting" alias="Waiting" initial="true">
<Transition target="normal">
<Condition variable="Executing" value="true"/>
</Transition>
</State>

<State name="normal" alias="Executing on schedule">
<Transition target="concluded">
<Condition variable="Concluded" value="true"/>
</Transition>
<Transition target="BadEstimations">
<Condition variable="BadTimeEstimation" value="true"/>
</Transition>
</State>

<!-- Bad projections of time  -->
<State name="BadEstimations" alias="Executing and bad estimated">
<Transition target="normal">
<Condition variable="BadTimeEstimation" value="false"/>
</Transition>
<Transition target="failedLowFunds">
<Condition variable="OverBudget" value="true"/>
</Transition>
<Transition target="failedLowTime">
<Condition variable="OverTime" value="true"/>
</Transition>
</State>

<!-- Final states -->
<State name="failedLowFunds" final="true">
</State>

<State name="failedLowTime" final="true">
</State>

<State name="concluded" final="true">
</State>

</EventSet>
```

```
</StateModel>
```

A.4 Modelo de Instanciação

```
<SetupModel>
<Instance name="almeida" alias="Almeida" class="Developer">
<Attribute name="fullName" value="Onofre Almeida"/>
<Attribute name="curriculum" value="Recent undergraduated.
2 years in OO programming."/>
<Attribute name="age" value="23"/>
<Property name="ExpAnalysis" value="0.2"/>
<Property name="ExpDesign" value="0.3"/>
<Property name="ExpInspection" value="0.8"/>
<Property name="ExpCoding" value="0.9"/>
<Property name="ExpTesting" value="0.8"/>
<Property name="HourlyCost" value="25"/>
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="1"/>
</Instance>

<Instance name="teixeira" alias="Teixeirinha" class="Developer">
<Attribute name="fullName" value="Herculando Teixeira"/>
<Attribute name="curriculum" value="Very experienced developer: 10
years in structured programming\n MSc in math, 12 years as System analyst"/>
<Attribute name="age" value="38"/>
<Property name="ExpAnalysis" value="0.9"/>
<Property name="ExpDesign" value="0.9"/>
<Property name="ExpInspection" value="0.5"/>
<Property name="ExpCoding" value="0.3"/>
<Property name="ExpTesting" value="0.3"/>
<Property name="HourlyCost" value="30"/>
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="1"/>
</Instance>


<Instance name="palhares" alias="Palhares" class="Developer">
<Attribute name="fullName" value="Juvenal Palhares"/>
<Attribute name="curriculum" value="Middle developer"/>
<Attribute name="age" value="38"/>
<Property name="ExpAnalysis" value="0.4"/>
<Property name="ExpDesign" value="0.5"/>
<Property name="ExpInspection" value="0.5"/>
```

```
<Property name="ExpCoding" value="0.4"/>
<Property name="ExpTesting" value="0.6"/>
<Property name="HourlyCost" value="13"/>
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="-1"/>
</Instance>

<Instance name="fonseca" alias="Fonseca" class="Developer">
<Attribute name="fullName" value="Antero Fonseca"/>
<Attribute name="curriculum" value="Good system analyst. He doesn't like programming."/>
<Attribute name="age" value="32"/>
<Property name="ExpAnalysis" value="0.9"/>
<Property name="ExpDesign" value="0.2"/>
<Property name="ExpInspection" value="0.3"/>
<Property name="ExpCoding" value="0.2"/>
<Property name="ExpTesting" value="0.3"/>
<Property name="HourlyCost" value="17"/>
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="-1"/>
</Instance>

<Instance name="tavares" alias="Floriano" class="Developer">
<Attribute name="fullName" value="Floriano Tavares"/>
<Attribute name="curriculum" value="Old programmer.
Experienced on assembly, cobol, fortran and clipper.
Very good in procedural programming and experienced in developing client/server applications."/>
<Attribute name="age" value="52"/>
<Property name="ExpAnalysis" value="0.1"/>
<Property name="ExpDesign" value="0.3"/>
<Property name="ExpInspection" value="0.3"/>
<Property name="ExpCoding" value="0.95"/>
<Property name="ExpTesting" value="0.84"/>
<Property name="HourlyCost" value="30"/>
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="-1"/>
</Instance>

<!--
<Instance name="praxedes" alias="Praxedes" class="Developer">
<Attribute name="fullName" value="Galdencio Praxedes"/>
<Attribute name="curriculum" value="Praxedes has skills in all activities."/>
<Attribute name="age" value="27"/>
<Property name="ExpAnalysis" value="0.6"/>
<Property name="ExpDesign" value="0.65"/>
<Property name="ExpInspection" value="0.45"/>
<Property name="ExpCoding" value="0.55"/>
<Property name="ExpTesting" value="0.6"/>
<Property name="HourlyCost" value="23"/>
```

```
<Property name="WorkHours" value="8"/>
<Property name="Allocated" value="-1"/>
</Instance>
-->

<Instance name="UserStudentRulesAnalysis" alias="User Student Rules Analysis" class="Activity">
<Attribute name="Description" value="Analysis of the rules related to the students users"/>
<Property name="AnalysisTask" value="1"/>
<Property name="FunctionPoints" value="49.14"/>
<Property name="Order" value="0"/>
</Instance>


<Instance name="UserStudentRulesDesign"
alias="User Student Rules Design" class="Activity">
<Attribute name="Description" value="Design of the rules related to the students users"/>
  <Property name="DetailedDesignTask" value="1"/>
<Property name="FunctionPoints" value="49.14"/>
<Property name="Order" value="2"/>
</Instance>

<Instance name="UserStudentRulesInspection"
alias="User Student Rules Inspection" class="Activity">
<Attribute name="Description" value="Inspection activity for the user student rules"/>
  <Property name="InspectionTask" value="1"/>
<Property name="FunctionPoints" value="49.14"/>
<Property name="Order" value="4"/>
<Property name="On" value="-1"/>
  </Instance>

<Instance name="UserStudentRulesCoding"
alias="User Student Rules Coding" class="Activity">
<Attribute name="Description" value="Coding of the rules related to the students users"/>
  <Property name="CodingTask" value="1"/>
<Property name="FunctionPoints" value="49.14"/>
<Property name="Order" value="6"/>
</Instance>



<Instance name="UserStudentRulesTesting"
alias="User Student Rules Testing" class="Activity">
<Attribute name="Description"
value="Testing activity for the user student rules"/>
  <Property name="TestingTask" value="1"/>
<Property name="FunctionPoints" value="49.14"/>
<Property name="Order" value="8"/>
</Instance>
```

```
<Instance name="AreaTeacherDisciplineAnalysis"
alias="Area Teacher Discipline Analysis" class="Activity">
<Attribute name="Description"
value="Analysis of the rules related to the disciplines CRUD"/>
<Property name="AnalysisTask" value="1"/>
<Property name="FunctionPoints" value="50.70"/>
<Property name="Order" value="1"/>
</Instance>


<Instance name="AreaTeacherDisciplineDesign"
alias="Area Teacher Discipline Design" class="Activity">
<Attribute name="Description"
value="Design of the rules related to the disciplines CRUD"/>
  <Property name="DetailedDesignTask" value="1"/>
<Property name="FunctionPoints" value="50.70"/>
<Property name="Order" value="3"/>
</Instance>

<Instance name="AreaTeacherDisciplineInspection"
alias="Area Teacher Discipline Inspection" class="Activity">
<Attribute name="Description" value="Inspection activity for the disciplines CRUD"/>
  <Property name="InspectionTask" value="1"/>
<Property name="FunctionPoints" value="50.70"/>
<Property name="On" value="-1"/>
<Property name="Order" value="5"/>
  </Instance>

<Instance name="AreaTeacherDisciplineCoding"
alias="Area Teacher Discipline Coding" class="Activity">
<Attribute name="Description"
value="Coding of the rules related to the disciplines CRUD"/>
  <Property name="CodingTask" value="1"/>
<Property name="FunctionPoints" value="50.70"/>
<Property name="Order" value="7"/>
</Instance>


<Instance name="AreaTeacherDisciplineTesting"
alias="Area Teacher Discipline Testing" class="Activity">
<Attribute name="Description" value="Testing activity for the disciplines CRUD"/>
  <Property name="TestingTask" value="1"/>
<Property name="FunctionPoints" value="50.70"/>
<Property name="Order" value="9"/>
</Instance>
```

```
<Instance name="CtrlPESC" class="Project">
<Attribute name="Name" value="CtrlPESC Project"/>
<Attribute name="Description" value="A system for academic management"/>
  <Property name="MaximumCost" value="15000"/>
  <Property name="MaximumTime" value="40"/>
</Instance>

<Links>
<!-- Links for precendences relations among activities-->

<Link relation="Precedences" source="UserStudentRulesDesign"
target="UserStudentRulesAnalysis"/>
<Link relation="Precedences" source="UserStudentRulesInspection"
t arget="UserStudentRulesDesign"/>
<Link relation="Precedences" source="UserStudentRulesCoding"
target="UserStudentRulesInspection"/>
<Link relation="Precedences" source="UserStudentRulesTesting"
target="UserStudentRulesCoding"/>

<Link relation="Precedences" source="AreaTeacherDisciplineDesign"
target="AreaTeacherDisciplineAnalysis"/>
<Link relation="Precedences" source="AreaTeacherDisciplineInspection"
target="AreaTeacherDisciplineDesign"/>
<Link relation="Precedences" source="AreaTeacherDisciplineCoding"
target="AreaTeacherDisciplineInspection"/>
<Link relation="Precedences" source="AreaTeacherDisciplineTesting"
target="AreaTeacherDisciplineCoding"/>

<!-- Links for team relation -->

<Link relation="team" source="UserStudentRulesAnalysis" target="teixeira"/>
<Link relation="team" source="UserStudentRulesDesign" target="almeida"/>
<Link relation="team" source="UserStudentRulesInspection" target="almeida"/>
<Link relation="team" source="UserStudentRulesCoding" target="almeida"/>
<Link relation="team" source="UserStudentRulesTesting" target="teixeira"/>

<Link relation="team" source="AreaTeacherDisciplineAnalysis" target="teixeira"/>
<Link relation="team" source="AreaTeacherDisciplineDesign" target="almeida"/>
<Link relation="team" source="AreaTeacherDisciplineInspection" target="almeida"/>
<Link relation="team" source="AreaTeacherDisciplineCoding" target="almeida"/>
<Link relation="team" source="AreaTeacherDisciplineTesting" target="teixeira"/>

</Links>

<MasterInstance instance="CtrlPESC">
<Success state="concluded" message=""/>
```

```
<Failed state="failedLowFunds" message=""/>
<Failed state="failedLowTime" message=""/>
</MasterInstance>

</SetupModel>
```

A.5 Modelo de Interações

```
<InteractionModel>

<!-- Images sequences Definitions -->
<Views>

<!-- Sequence for developer walking to the left side-->
<AnimatedImage name="WalkingLeft" delay="2" loop="true">
<Frames>
<Frame path="/images/sprites/walking_left_1.png"/>
<Frame path="/images/sprites/walking_left_2.png"/>
<Frame path="/images/sprites/walking_left_3.png"/>
<Frame path="/images/sprites/walking_left_4.png"/>
</Frames>
</AnimatedImage>

<!-- Sequence for developer walking to the right side-->
<AnimatedImage name="WalkingRight" delay="2" loop="true">
<Frames>
<Frame path="/images/sprites/walking_right_1.png"/>
<Frame path="/images/sprites/walking_right_2.png"/>
<Frame path="/images/sprites/walking_right_3.png"/>
<Frame path="/images/sprites/walking_right_4.png"/>
</Frames>
</AnimatedImage>


<!-- Sequence for developer walking tired to the right side-->
<AnimatedImage name="WalkingRightTired" delay="4" loop="true">
<Frames>
</Frames>
</AnimatedImage>


<!-- Sequence for developer walking tired to the left side-->
<AnimatedImage name="WalkingLeftTired" delay="4" loop="true">
<Frames>
</Frames>
```

```
</AnimatedImage>

<AnimatedImage name="SitNormal" delay="3" loop="true">
<Frames>
<Frame path="/images/sprites/sit_normal_1.png"/>
<Frame path="/images/sprites/sit_normal_2.png"/>
</Frames>
</AnimatedImage>

<AnimatedImage name="SitNormal2" delay="3" loop="true">
<Frames>
<Frame path="/images/sprites/sit_normal_problem_1.png"/>
<Frame path="/images/sprites/sit_normal_problem_2.png"/>
</Frames>
</AnimatedImage>

<!-- Sequence for developer working tired -->
<AnimatedImage name="SitTired" delay="5" loop="true">
<Frames>
<Frame path="/images/sprites/sit_tired_1.png"/>
<Frame path="/images/sprites/sit_tired_2.png"/>
</Frames>
</AnimatedImage>

<!-- Sequence for developer working Exhausted -->
<StaticImage name="SitExhausted" path="/images/sprites/
sit_exhausted_1.png"/>

<!-- Sequence for developer idle -->
<StaticImage name="SitIdle" path="/images/sprites/sit_idle_1.png"/>


<!-- Sequence for developer idle2 -->
<AnimatedImage name="SitIdle2" delay="2" loop="true">
<Frames>
<Frame path="/images/sprites/sit_idle2_1.png"/>
<Frame path="/images/sprites/sit_idle2_2.png"/>
</Frames>
</AnimatedImage>

<!-- Sequence for developer sitting on his workplace
<AnimatedImage name="Sitting" delay="2" loop="true">
<Frames>
</Frames>
</AnimatedImage>-->

<!-- Sequence for developer standing up
<AnimatedImage name="Standing" delay="2" loop="true">
```

```
<Frames>
</Frames>
</AnimatedImage>-->


<StaticImage name="doorOffice" path="/images/sprites/decoration/doorOffice.png"/>
<StaticImage name="doorLab" path="/images/sprites/decoration/doorLab.png"/>
<StaticImage name="server" path="/images/sprites/decoration/serverTable.png"/>
<StaticImage name="computer" path="/images/sprites/decoration/computer.png"/>
<StaticImage name="phone" path="/images/sprites/decoration/phone.png"/>
<StaticImage name="water" path="/images/sprites/decoration/water.png"/>
<StaticImage name="chair" path="/images/sprites/decoration/chair.png"/>
<StaticImage name="blackboard" path="/images/sprites/decoration/blackboard.png"/>

<!-- Views for activities -->
<StaticImage name="activityWaiting" path="/images/sprites/ball_pause.png"/>
<StaticImage name="activityConcluded" path="/images/sprites/ball_concluded.png"/>
<StaticImage name="activityDisabled" path="/images/sprites/ball_stop.png"/>

<AnimatedImage name="activityExecuting" delay="10" loop="true">
<Frames>
<Frame path="/images/sprites/ball_play_1.png"/>
<Frame path="/images/sprites/ball_play_2.png"/>
<Frame path="/images/sprites/ball_play_3.png"/>
</Frames>
</AnimatedImage>

</Views>


<!-- Rooms -->
<Rooms>
<Room name="lab" background="/images/rooms/workplace.gif">
<Decorations>
<Item name="Server" view="server" x="30" y="240"/>
<Item name="Computer1" view="computer" x="140" y="349"/>
<Item name="Computer2" view="computer" x="240" y="249"/>
<Item name="Computer3" view="computer" x="390" y="349"/>
<Item name="Computer4" view="computer" x="540" y="249"/>
<Item name="Computer5" view="computer" x="640" y="349"/>
<Item name="Blackboard" view="blackboard" x="80" y="20"/>
<Item name="Phone" view="Phone" x="750" y="200"/>
</Decorations>

<GraphicInstances>
<GraphicInstance instance="teixeira" x="100" y="350"/>
<GraphicInstance instance="almeida" x="600" y="350"/>
```

```
<GraphicInstance instance="UserStudentRulesAnalysis" x="110" y="83"/>
<GraphicInstance instance="UserStudentRulesDesign" x="150" y="85"/>
<GraphicInstance instance="UserStudentRulesInspection" x="190" y="87"/>
<GraphicInstance instance="UserStudentRulesCoding" x="230" y="89"/>
<GraphicInstance instance="UserStudentRulesTesting" x="270" y="91"/>

<GraphicInstance instance="AreaTeacherDisciplineAnalysis" x="110" y="113"/>
<GraphicInstance instance="AreaTeacherDisciplineDesign" x="150" y="115"/>
<GraphicInstance instance="AreaTeacherDisciplineInspection" x="190" y="117"/>
<GraphicInstance instance="AreaTeacherDisciplineCoding" x="230" y="119"/>
<GraphicInstance instance="AreaTeacherDisciplineTesting" x="270" y="121"/>
</GraphicInstances>

<Links>
<Link name="doorLink" view="doorOffice" x="630" y="140" targetRoom="office"/>
</Links>

</Room>

<Room name="office" background="/images/rooms/workplace.gif">
<Decorations>
<Item name="Water" view="water" x="10" y="230"/>
<!--
<Item name="Chair1" view="chair" x="690" y="270"/>
<Item name="Chair2" view="chair" x="680" y="310"/>
<Item name="Chair3" view="chair" x="670" y="350"/>
<Item name="Chair4" view="chair" x="660" y="390"/>
-->
</Decorations>

<GraphicInstances>
<GraphicInstance instance="tavares" x="200" y="250"/>
<GraphicInstance instance="fonseca" x="350" y="350"/>
<GraphicInstance instance="palhares" x="500" y="250"/>
</GraphicInstances>


<Links>
<Link name="doorLink" view="doorLab" x="140" y="140" targetRoom="lab"/>
</Links>

</Room>
</Rooms>

<!-- Graphic States definition -->
<InteractionStates element="Developer">
<State name="idle">
<View name="sitIdle" prob="0.5"/>
```

```
<View name="sitIdle2" prob="0.5"/>
</State>
<State name="normal">
<View name="SitNormal" prob="0.95"/>
<View name="SitNormal2" prob="0.05"/>
</State>
<State name="tired" >
<View name="SitTired"/>
</State>
<State name="exhausted">
  <View name="SitExhausted"/>
</State>
<State name="unemployed">
<View name="sitIdle" prob="1"/>
</State>

<!-- Interaction events caused by transitions between logic states-->
<InteractionEvents>

<!-- Go out of the lab and put him in the human resources -->
<OnAction action="fireDeveloper">
<SetView view="WalkingRight"/>
<MoveTo x="650" y="160"/>
<PutOnRoom room="office" x="100" y="300"/>
</OnAction>

<OnTransition from="normal" to="idle">
</OnTransition>
<OnTransition from="idle" to="normal">
</OnTransition>

<!-- Put developer in the lab -->
<OnAction action="hireDeveloper">
<SetView view="WalkingLeft"/>
<MoveTo x="150" y="160"/>
<PutOnRoom room="lab" x="100" y="300"/>
</OnAction>
</InteractionEvents>
</InteractionStates>

<!-- Graphic States definition for activities-->
<InteractionStates element="Activity">
<State name="waiting">
<View name="activityWaiting"/>
</State>

<State name="executing">
<View name="activityExecuting"/>
```

```
</State>

<State name="concluded">
<View name="activityConcluded"/>
</State>

<State name="off">
<View name="activityDisabled"/>
</State>

<!-- Interaction events caused by transitions among logic states-->
<InteractionEvents>
</InteractionEvents>

</InteractionStates>

<!-- Graphic initialization -->
<Initialization>
<InitialRoom name="lab"/>
<IntroductionMessage message="introduction"/>
</Initialization>


<!-- Variables visualization -->
<ControlPanel>
<Variable name="ProjectTime" instance="CtrlPESC" alias="Elapsed Time"/>
<Variable name="ProjectCost" instance="CtrlPESC" alias="Consumed funds" unit="$"/>
</ControlPanel>


</InteractionModel>
```

A.6 Arquivo de Mensagens

```
<Messages>

<!-- Introduction -->
<Message code="introduction">You've been hired to be our new manager.
We must develop an academic control system for a famous university.
The system has about 160 adjusted function points and 2 main threads of tasks.
The first thread relates to users, students records and special search rules on the records.
The second thread relates to research areas, teachers and disciplines records.
Stakeholders demand the product in about 49 days and will spent about $ 60000.
The project has an initial configuration, in which to developers are allocated to the project activities.
```

```
You can (and should) change it during the game by hiring and firing developers, swithing
on and off inspections and changing developers' work hours.
Quality is important. The tests will only finish with 95% of the errors corrected.
Complete the project within the estimates. Do it fast and do it with few errors as possible.
</Message>

<!-- Developers' messages -->
<Message code="developerNormalToIdle">Nothing to do ? Ok, I'll download some MP3s</Message>
<Message code="developerNormalToTired">I am tired...</Message>
<Message code="developerNormalToUnemployed">Lamentable. You lost one of the best developers
of the market.</Message>

<Message code="developerIdleToUnemployed">Ok...I didn't like this job, anyway.
I am going to look for something better.</Message>
<Message code="developerIdleToNormal">Finally, something to do! I was
getting bored</Message>


<Message code="developerTiredToNormal">Now i feel fine</Message>
<Message code="developerTiredToIdle">I am going to rest a little</Message>
<Message code="developerTiredToExhausted">Dear Boss, can't you see i exhausted ?
Is it slavery?</Message>
<Message code="developerTiredToUnemployed">It's not fair! And now ? I have kids
to take care. </Message>

<Message code="developerExhaustedToTired">I am still tired... I need holidays</Message>
<Message code="developerExhaustedToIdle">zzzzzzz</Message>
<Message code="developerExhaustedToUnemployed">Thank you! Finally, it was all i wanted! </Message>

<Message code="developerUnemployedToNormal">Be sure i'll do a god job!</Message>
<Message code="developerUnemployedToIdle">It is job of my dreams!</Message>


<!-- Activities' messages -->
<Message code="activityConcluded">Activity concluded!</Message>
<Message code="activityWaitingExecuting">Started execution</Message>
<Message code="activityOff">Activity turned off</Message>
<Message code="activityOffWaiting">Activity turned on. Now waiting to be executed</Message>
<Message code="activityOffExecuting">Activity turned on. Now executing</Message>

<!-- Project's messages -->
<Message code="projectWaitingNormal">The project has just began !</Message>
<Message code="projectNormalBadEstimations">The project seems to be bad estimated</Message>
<Message code="projectFailedLowFunds">You failed. You haven't all money off the world to spend.
Try to find another job</Message>
<Message code="projectFailedLowTime">You Failed. Your team is very lazy, and I think you
haven't all necessary competencies to be a project manager. Looser!</Message>
<Message code="projectConcluded">Congratulations! The project has finished
```

```
with success.</Message>
</Messages>
```