Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

UM MODELO INTEGRADO PARA CONSTRUÇÃO DE JOGOS DE COMPUTADOR APLICADO À CAPACITAÇÃO EM GESTÃO DE PROJETOS

Marcelo Santiago Guedes

DISSERTAÇÃO DE MESTRADO

Recife
13 de setembro de 2006

Universidade Federal de Pernambuco
Centro de Informática

Marcelo Santiago Guedes

UM MODELO INTEGRADO PARA CONSTRUÇÃO DE JOGOS DE COMPUTADOR APLICADO À CAPACITAÇÃO EM GESTÃO DE PROJETOS

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Hermano Perrelli de Moura*

Recife
13 de setembro de 2006

AGRADECIMENTOS

Agradeço primeiramente a Deus pela inspiração e pela vontade de vencer e pelos obstáculos colocados a frente, sem os quais de certo que não seria quem hoje sou, mas sobretudo pelo conforto vindo da certeza do sucesso nos momentos de fraqueza.

Ao Prof. Hermano Perrelli pela orientação, apoio, trocas de conhecimento e amizade dedicada sem a qual este trabalho não seria possível de ser realizado.

À Profa. Patrícia Tedesco, pelas incansáveis ajudas e incentivos ao longo de todo o mestrado. Também pela coordenação do projeto SmartSim.

Ao Prof. Geber Ramalho pelas grandes contribuições ao projeto SmartSim.

À André Felipe, João, Nancy, Thierry, Victor, Ivanoska, George, Allan, Daniel, Eduardo, Pablo e Danielly por sua dedicação ao desenvolvimento do *Virtual Team*.

À minha mãe, meus irmãos e meu filho pela compreensão de minha ausência que eu espero um dia ser possível recuperar com muitos momentos felizes em vossa companhia.

Aos meus amigos do CIn e da Qualiti pela amizade, trocas de experiências, incentivo e oportunidades.

Aos amigos que ficaram em Brasília, mas que sempre estiveram próximos.

Aos amigos que ao longo dos últimos dois anos fizeram parte de minha vida e que de alguma forma contribuíram para a concretização deste trabalho.

A todos que por mim torceram.

RESUMO

Projetos de software são caracterizados como não triviais, visto que são fortemente baseados em conhecimento e na experiência de seus atores, em particular do gerente de projetos. Este contexto faz com que seja necessário não apenas conhecimento mas, principalmente, habilidades e atitudes. No entanto, o ganho de experiência tende a ser moroso e de alto custo, dado que se baseia na vivência de projetos reais, com recursos e custos reais.

O uso de ambientes de simulação e jogos permite uma experimentação do processo de gerenciamento de projetos a um menor custo. A dinâmica de sistemas vem sendo utilizada para criar modelos de simulação de gerenciamento de projetos de software, pois permite a modelagem de sistemas complexos e dinâmicos, sendo inclusive utilizada para a criação de jogos de simulação.

Para o processo de capacitação é fundamental que os alunos tenham uma sensação maior de proximidade com a realidade, quando do uso de ambientes simulados, pois a isto está relacionada a qualidade de suas decisões. Os jogos baseados em simulação fornecem uma aproximação nesse sentido, pois há interatividade, objetivo e desafio. No entanto, ao analisar jogos de entretenimento observa-se uma clara discrepância no sentido de imersão e comprometimento que o jogador possa ter com as decisões tomadas e seu efeito. Criar características nestes ambientes de simulação para que estes sejam capazes de apresentar maior sensação de realidade é, portanto, fundamental para o sucesso no uso destes jogos no processo de capacitação. Neste sentido, a utilização de atores sintéticos pode trazer um pouco desta aproximação. Este trabalho define uma arquitetura híbrida para o jogo Virtual Team que permita a utilização do ambiente de simulação da dinâmica de sistemas e de atores sintéticos, bem como a troca de informações entre estes ambientes. Ainda com o objetivo de aumentar a imersibilidade, este trabalho apresenta um modelo de gerência de projetos e de engenharia de software mais próximo da realidade dos projetos de desenvolvimento de software, e capaz de ser modelado e simulado no ambiente de simulação do Virtual Team.

Palavras-chave: Gerenciamento de Projetos, Engenharia de Software, Simuladores,

Dinâmica de Sistemas, Jogos Educacionais.

ABSTRACT

Software Projects are characterized as non-trivial, based on the fact that it is strongly based on the knowledge and experience of its actors, in particular the Project Manager. This context makes necessary not only the knowledge but mainly that these might be transformed into skills and attitude. Unfortunately, the growth of experience is a slow and high cost process, because it is based on the real project experience.

The use of simulation environments and games allow a highly experimentation of the project management process in a low price. System dynamics is been used to create simulation models into software project management field, based on its capacity of modeling complex system.

For the learning process it is important that the students have a sensation of proximity with the reality, because this is related to quality of its decisions. The simulation games offer an approach on this way, because it involves interactivity, objectives and challenges. However, analyzing entertainment games it is possible to observe a clear gap in the sense of immersion and engagement that the student could have with its decisions and its effects. Add more capabilities to these simulation environments making it able to transmit more reality is important for the success of these games as a training tool. On this way, the use o synthetic actors can bring this approximation. This work defines an hybrid architecture for the game VIRTUAL TEAM that allow the use o system dynamics simulation environment and the synthetic agents, and also allow the interchange of information between them. This work presents a project management model and a software engineering more near of the software development projects reality, capable of been modeling and simulating in the VIRTUAL TEAM.

Keywords: Project management, Software Engineering, Simulation, System Dynamics, Serious Games

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

O gerenciamento de projetos vem sendo estudado ao longo das ultimas décadas [Min00]. As premissas das teorias utilizadas em gerenciamento de projetos atuais são fortemente baseadas na "escola do planejamento" [Min00], sob a qual é possível imaginar cenários futuros e traçar um caminho sob esses cenários e sobre os possíveis eventos que possam provocar mudanças de curso, chamados de riscos. No entanto, muito se tem evoluído e acrescido às técnicas gerenciais, tais como gerência de comunicação, pessoas, conflitos, etc.

O PMBOK™ – *Project Management Body of Knowledge* – organiza o corpo de conhecimento em gerenciamento de projetos em nove áreas do conhecimento [Ins04], tentando desta forma sistematizar e explicitar o conhecimento para que o mesmo seja objeto de melhor aprendizado por parte de novos gerentes. No entanto, a simples explicitação não garante que a curva de aprendizado seja menor, porém é um avanço em termos de uma padronização sobre o conjunto de conhecimento mínimo necessário ao gerente de projetos.

As estratégias em educação gerencial vêem ao longo do tempo fazendo uso de ferramentas didáticas, que extrapolam o uso de livros ou aulas expositivas. A atividade gerencial é fortemente baseada em conhecimento e experiência [P+03], sendo natural que gerentes mais experientes tenham melhores resultados que gerentes menos experientes. Desta forma, o uso de técnicas didáticas que privilegiem a experimentação é fator chave para o sucesso de uma boa estratégia de capacitação.

A troca de experiências e discussões sobre estudos de caso, de sucesso ou fracasso, é largamente utilizado nas escolas de educação gerencial, porém estas, apesar de permitir reflexões sobre os contextos, decisões e conseqüências da experiência de terceiros, não submete o interlocutor a situação real sob a qual aquelas decisões foram tomadas. Ou seja, o nível de experimentação é limitado, dado que o aluno está numa situação confortável e paciente diante do problema.

A melhor situação de experimentação possível seria a submissão do gerente a situações reais, em projetos e responsabilidades reais. No entanto esta possibilidade oferece grande risco organizacional, uma vez que lida com recursos e custos reais. Normalmente o que se adota então é submeter um gerente junior a projetos piloto ou de menor risco, a fim

de que esse possa melhorar o desenvolvimento de suas habilidades. De qualquer forma, o processo é moroso e custoso para as organizações.

A criação de ambientes ou contextos simulados por computador para que os gerentes junior possam melhorar a sua capacitação tende a minimizar estes custos, na medida em que, a sua utilização não implica no comprometimento de recursos reais.

1.1 CONTEXTO

Os ambientes de simulação vêm ao longo do tempo respondendo a esta demanda para as mais diversas áreas do conhecimento. Na educação gerencial vemos ao longo do tempo o surgimento dos chamados jogos de negócio [Gab06], que implementam dentro de si processos organizacionais. Para Gabardo [Gab06] Jogos de Negócio, ou jogos de empresas, é um conjunto de ferramentas e técnicas que simulam situações de gestão de empresas. Ainda segundo Gabardo, a simulação de um ambiente empresarial permite gerar e vivenciar fenômenos grupais onde os estados emocionais são fortemente presentes. A vivência desses fenômenos grupais ligados à atividade empresarial se constitui numa poderosa alavanca para a aprendizagem gerencial [Gab06].

O uso destes software no processo de capacitação normalmente é parte de um processo maior, no qual há a presença de um professor ou facilitador, que faz com que os alunos tenham acesso aos modelos que se desejam ensinar, e depois fazem uso dos jogos de negócio como mais um recurso pedagógico. Normalmente o uso destes jogos são feitos em sala de aula ou laboratório e em grupo. As sessões de uso dos jogo são intercaladas com sessões de compartilhamento de informações com toda a turma.

Por motivos como este, as sessões de jogo são menos demoradas e exigem uma menor imersão, provocando requisitos menores de interface com relação a jogabilidade e ao desafio, pois seu objetivo é menor com relação ao entretenimento e maior com relação a simulação.

No entanto, o uso de jogos de entretenimento baseados em modelos de simulação mais reais podem assumir papel complementar na capacitação de gerentes de projeto. Ao se pensar mais em entretenimento e menos em simulação, questões como maior riqueza de interface gráfica e interatividade são importantes. O jogos de entretenimento procuram fazer com que o ambiente do jogo se pareça com o ambiente real, e não apenas o acompanhamento de variáveis através de gráficos.

A criação de jogos de negócio para o ensino de gerenciamento de projetos vem sendo pesquisado a algum tempo, com trabalhos como [AHM61], [Mer96], [Bar01], entre outros. Já o uso de ferramentas de simulação como um jogo foi utilizado em alguns poucos

trabalhos como [Lud89], [vdH04], [DBW04]. No entanto, em todos esses trabalhos a característica de simulação está muito forte, não só por uma limitação de interface gráfica, mas também porque os objetos, que normalmente compõem o projeto e que são representados na interface, são parte de um modelo sistêmico, se comportando da forma descrita pelo modelo. Modelo este que se deseja comunicar aos alunos.

1.2 MOTIVAÇÃO

Tendo como finalidade o desenvolvimento das capacidades gerenciais dos jogadores, a comunicação do modelo de gerencia de projetos é apenas parte do processo. Outra parte sensível desde processo de capacitação está relacionada ao reconhecimento de situações de ação, a decisão pela ação e o acompanhamento dos efeitos destas ações. A qualidade das decisões está bastante relacionada a imersibilidade do jogo.

Neste sentido, o uso de recursos tecnológicos que aumentem o grau de realidade do jogo é benéfico ao processo de capacitação. O uso de atores sintéticos para transformar os membros da equipe de desenvolvimento de software em personagens visa criar essa aproximação entre a realidade observada no jogo e a realidade. Os atores sintéticos, personagens, correspondem a um tipo especial de agente inteligente [Rus95], que possuem propriedades extras, como emoções e personalidade, e têm como objetivo principal transmitir "ilusão de vida"a usuários de aplicações educacionais e de histórias interativas [Cen99]. No dia-a-dia, o gerente de projetos é obrigado a lidar com pessoas reais que estão alocadas a execução de tarefas e que exercem um papel, como desenvolvedores, analistas, arquitetos, etc. As pessoas possuem uma identidade, experiências, personalidade, etc. que ajudam a definir suas ações. No ambiente simulado, os atores sintéticos buscam representar estes membros de uma equipe e permitem a exploração de uma série de aspectos comportamentais típicos a situações de software, como por exemplo, a perda de produtividade devido a problemas de relacionamento com outros membros da equipe.

No entanto, a transformação dos membros da equipe em personagens faz com que estes passem a ter comportamentos nem sempre determinísticos, dado que este dependerá agora do modelo que cada agente tem do mundo, que é totalmente independente dos demais atores e do modelo sistêmico do projeto. Esta forma de comportamento torna o jogo mais próximo da realidade.

1.3 OBJETIVOS

O objetivo deste trabalho é construir mecanismos arquiteturais que permitam aos jogos de negócio voltados a capacitação em gerenciamento de projetos estar mais próximos da realidade do gerenciamento de projetos, em especial o gerenciamento de projetos de software.

No entanto, para que isto seja possível é necessário que hajam modelos mais próximos dos atualmente utilizados pelas organizações de software, tanto relacionados a gerenciamento de projetos, como os relacionados a engenharia de software. Desta forma um dos mecanismos a serem utilizados é permitir aos jogos utilizarem processos de software e de gerenciamento que seja de fato utilizados nos projetos de desenvolvimento de software.

Aproveitando a capacidade de modelagem e simulação da dinâmica de sistemas para a criação de jogos, há a necessidade de um modelo de extensão arquitetural que permita a troca de informações entre os atores sintéticos e o ambiente de simulação da dinâmica de sistemas. Assim, é apresentado um modelo arquitetural integrado que permita a utilização dos modelos de dinâmica de sistemas e de atores sintéticos.

1.4 VISÃO GERAL DO TRABALHO

Esta dissertação esta organizada em seis capítulos. Este é o primeiro capítulo, que apresenta o contexto do trabalho, suas motivações e seus objetivos.

O Capítulo 2 apresenta uma revisão sobre gerenciamento de projetos de software, apresentando as metodologias atualmente utilizadas e seus benefícios.

O Capítulo 3 apresenta o contexto da modelagem de sistemas e as técnicas de simulação mais utilizadas em ambientes de simulação. Apresenta também o uso destas técnicas para a criação de jogos que vem sendo utilizados na capacitação de engenheiros de software e gerentes de projetos de software.

O Capítulo 4 apresenta a proposta de uma metodologia de desenvolvimento de software e de gerenciamento de projetos a ser utilizada em um jogo de negócio para capacitação de gerentes de projeto. Esta proposta segue os processos de customização de processos utilizados em projetos de software reais. Apresenta também a representação do processo proposto em uma estrutura estendida de um ambiente de *PSSE – Process Centered Engineering Enviroment*– no qual a simulação de processos pode ser utilizada com objetivo de otimização, mas também pode ser aproveitada para ser implementada em um jogo de negócio e com isso capacitar a equipe que usará o processo otimizado. Também apresenta o modelo de domínio baseado na dinâmica de sistemas a ser utilizado

como base para os processos de simulação.

O Capítulo 5 apresenta a estrutura do jogo Virtual Team, a arquitetura utilizada, as tecnologias integradas no mesmo e algumas telas do jogo. Dado que o Virtual Team foi criado em um projeto de pesquisa, no qual outros projetos de pesquisa também estavam envolvidos, será apresentada também a delimitação da contribuição deste trabalho à criação do jogo e do *framework*.

Por fim, no Capítulo 6, as contribuições, limitações e possibilidades de trabalhos futuros são apresentados.

CAPÍTULO 2

GERENCIAMENTO DE PROJETOS DE SOFTWARE

Os empreendimentos humanos desde a antiguidade impressionam por sua grandiosidade e complexidade. A observar monumentos como as pirâmides do Egito, muralha da china, entre outros; é difícil imaginar que tenham sido construídos sem o apoio do ferramental científico que temos hoje disponíveis.

No entanto a aplicação sistemática de seus conceitos só passou a ocorrer na segunda metade do século XIX [vdH04] com as mudanças nos modos de produção e a ocorrência da Revolução Industrial, que obrigou às empresas a produzirem para mercados em escala mundial. A Revolução Industrial alterou profundamente a estrutura econômica do mundo ocidental e teve como principais conseqüências o desenvolvimento do capitalismo industrial. As relações de produção foram drasticamente modificadas, e iniciou-se assim, uma cadeia de transformações, que tornou cada vez mais exigente a tarefa de gerir as novas organizações econômicas [Sis98]. Destaque-se nessas transformações àquelas relacionadas a forma de comunicação e ao transporte, que viabilizaram o comércio em escala mundial. A visão sistêmica do processo de produção industrial, e sua conseqüente preocupação com o aumento de produtividade, foi aplicada por Taylor a atividades ligadas a industria de aço [Sis98]. O aumento significativo no tamanho das organizações e da competição devido em muitos casos a presenças transnacionais, fez aumentar também a busca por maior eficiência em suas operações. Este fato é notório com a difusão na década de 70 das teorias de qualidade japonesas como a TQM –*Total Quality Management*, onde o controle do processo produtivo era elemento fundamental para o sucesso do negócio. Este contexto de maior competitividade fez com que as empresas também buscassem não só produtos e serviços que atendessem as solicitações dos clientes, mas que também atendesses a necessidades implícitas desses clientes, procurando assim superar suas expectativas. Segundo Mintzberg as escolas estratégicas passaram historicamente a observar mais o cliente. Desta forma, o desenvolvimento de produtos com características técnicas aderentes aos requisitos levantados juntos aos clientes, mas sobretudo, que estes estejam disponíveis em tempo hábil a sua utilização e dentro dos custos previstos, passou a ser obrigatório para as organizações. Neste ambiente competitivo o processo de tomada de decisão é requerido ser mais ágil e eficaz, ou seja, que as decisões sejam tomadas mais

rapidamente e com uma qualidade maior. Desta forma, na final do século XX o mundo foi submetido a chamada revolução da informação, onde ter informação de qualidade e no tempo necessário é de fundamental importância para a sobrevivência das organizações.

Este ambiente fez com que surgissem técnicas como o Gráfico de Gantt para o planejamento e acompanhamento de projetos de forma gráfica. Henry Gantt, desenvolveu a época da I Guerra Mundial, diagramas com barras de tarefas e marcos, que permitem a visualização da sequência das atividades e a sua duração [Sis98]. Os diagramas de gantt provaram ser uma ferramenta poderosa para os gerentes de projetos e permaneceram inalterados por quase 1 século, quando foram acrescentados as linhas de dependencias entre as tarefas já nos anos 1990.

A seguir descreveremos, na Seção 2.1, um conjunto de definições importantes para o Gerenciamento de Projetos, na Seção 2.2, um breve panorama de como a disciplina é tratada nas organizações, na Seção 2.3, abordaremos as peculiaridades do gerenciamento de projetos de desenvolvimento de software e na Seção 2.4, falaremos sobre o processo de capacitação dos gerentes de projetos de software.

2.1 DEFINIÇÕES

Entende-se como projeto, o emprego de um esforço temporário para criar um produto, serviço ou resultado único [Ins04].

Ele deve ser temporário, no sentido de que deve ter um início e um final bem determinados, no qual o final é atingido quando os objetivos do projeto são totalmente atingidos, ou fica claro que eles nao serão ou nao podem ser atingidos, ou que haja a necessidade do projeto não existir mais, fazendo com que ele seja terminado. Temporário não necessariamente entende-se por curto, mas de duração finita [Ins04]. Não deve ser confundido o temporário do projeto, com a temporalidade de seus resultados, que podem ser duradouros. Dois aspectos concorrem pra sua temporalidade, a saber [Ins04]:

- A oportunidade é usualmente temporal.
- O time do projeto, como unidade de trabalho, raramente sobrevivem ao projeto, pois normalmente os times são criados dentro do contexto de um projeto, e desfeitos após a sua conclusão.

Os projetos devem produzir resultados, serviços ou produtos únicos. Os projetos podem criar [Ins04]:

- Um produto ou artefato produzido, quantificável, que podem ser o item final em si ou um componente de outro.

- A capacidade de prestar um serviço, tais como funções de negócio de suporte a produção ou distribuição.
- Resultados, como resultados finais ou documentos. Por exemplo, o conhecimento produzido por um projeto de pesquisa.

Os projetos são produzidos de forma progressiva, organizado em etapas e produzidos de forma incremental. Desta forma nas fases mais iniciais do projeto, seu escopo está nebuloso, e a medida que o tempo vai passando, e o trabalho sendo realizado, ele vai sendo explicitado, dado que a equipe vai tendo mais entendimento sobre o escopo do trabalho a ser feito.

O gerenciamento de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender aos seus requisitos. O gerenciamento de projetos é realizado através da aplicação e da integração dos grupos de processos: iniciação, planejamento, monitoramento e controle e fechamento [Ins04]. Cabe ao gerente de projetos:

- Identificar as necessidades do projeto
- Estabelecer objetivos claros e atingíveis
- Balanceamento das demandas conflitantes de qualidade, escopo, tempo e custo.
- Adaptação das expectativas. dos planos e da abordagem às diferentes preocupações e expectativas das diversas partes.

Freqüentemente os projetos são afetados por variáveis de interesses conflitantes, como por exemplo: qualidade, escopo, tempo e custos. O gerente de projetos deve então conduzir o andamento do projeto de forma que essas restrições sejam atendidas (Figura 2.1).

As atividades a serem executadas pelo gerente de projetos são organizadas em um processo, que orienta sobre quais tarefas serão executadas, em que momento, qual o seu responsável e que produtos elas irão gerar. Na Seção 2.2.2 procurar-se-á detalhar um pouco como o PMBOK™ *Guide* define uma estrutura para a criação do processo de gerenciamento de projetos.

As atividades a serem executadas, são chamadas de processos no PMBOK™ *Guide*. Este processos representam um conjunto de ações e atividades relacionadas que são executadas de forma a atingir um conjunto pré-especificado de resultados, produtos ou serviços. Estes processos são executados pela equipe do projetos e geralmente são divididos em 2 categorias [Ins04]:

Figura 2.1 Variáveis de interesse conflitantes [Ins04]

- Os processos de gerenciamento de projetos. São comuns a maioria dos projetos na maior parte do tempo. Seus propósitos estão relacionados ao iniciar, planejar, monitorar e controlar, e encerrar um projeto. As formas de interação entre esses processos são normalmente complexas, de forma que não podem ser completamente descritos em um documento ou de gráficos. No entanto, na Seção 2.2.2, falaremos mais sobre como estes processos estão organizados, segundo a estrutura sugerida pelo PMBOK™ *Guide*.

- Os processos orientados ao produto que especificam e criam o produto do projeto. Estes processos são definidos pelos ciclo de vida do projeto e variam sensivelmente dependendo da área de aplicação. Por exemplo, o processo produtivo para a criação de um software não é o mesmo do que um processo para a construção de uma casa, ou de um hardware.

Os processos de gerenciamento de projetos e de produtos se sobrepõem e interagem entre si durante todo o projeto. O gerenciamento de projetos é um empreendimento integrador, que exige que os processos de gerenciamento de projetos e de produtos sejam adequadamente associados e conectados a outros processos pra facilitar sua coordenação.

Os processos de gerenciamentos de projetos possuem naturezas diferenciadas e podem ser organizadas em grupos de processos [Ins04]:

- Grupo de processos de iniciação

- Grupo de processos de planejamento

- Grupo de processos de execução

- Grupo de processos de monitoramento e controle
- Grupo de processos de fechamento

Esta organização permite uma visão de aplicação tendo como base o tempo, no entanto eles também podem ser organizados segundo as áreas do conhecimento que atuam [Ins04], a saber:

- Gerenciamento da integração do projeto
- Gerenciamento do escopo do projeto
- Gerenciamento de custos do projeto
- Gerenciamento do tempo do projeto
- Gerenciamento da qualidade do projeto
- Gerenciamento dos recursos humanos do projeto
- Gerenciamento das comunicações do projeto
- Gerenciamento dos riscos do projeto
- Gerenciamento das aquisições e sub-contratações do projeto

Na Seção 2.2.2 falaremos um pouco mais sobre os objetivos dos grupos de processo e das áreas do conhecimento.

2.2 PANORAMA GERAL DO GERENCIAMENTO DE PROJETOS

No início dos anos 1960, o gerenciamento de projetos surge como ciência e ao longo do tempo vem sendo uma das disciplinas mais estudadas e utilizadas nas organizações. Não que a evolução destas práticas e técnicas adotadas pelos gerentes de projetos tenha se dado de forma mais recente, mas principalmente pela organização do conhecimento na área e pela divulgação dada a este conhecimento, feita em grande parte pelo *PMI - Project Management Institute*.

Em 1969, foi fundado e atualmente é a maior instituição internacional dedicada à disseminação do conhecimento e ao aprimoramento das atividades de gestão profissional de projetos [Tor05]. No entanto, o gerenciamento de projetos como é visto atualmente é fruto da evolução das décadas seguintes, fazendo uso de marketing, psicologia e relações

humanas [Sis98]. Vários modelos de negócio se desenvolveram neste período, todos eles compartilhando a mesma estrutura de suporte: Um time sendo liderado por um gerente de projetos assegurando a comunicação e a integração do trabalho [Sis98].

Os gastos com projetos dentro das organizações vem aumentando. O PMI estima que cerca de 25% do PIB mundial seja gasto em projetos e cerca de 16,5 milhões de profissionais estejam envolvidos com gerenciamento de projetos em todo o mundo [Tor05].

Segundo relatório CHAOS do *Standish Group* [cha99] as taxas de falhas e os custos tem caído, assim como as taxas de sucesso nos projetos tem aumentado, justificando assim a adoção do gerenciamento de projetos pelas organizações de tecnologia da informação. As taxas de sucesso em grandes projetos aumentaram de 16% em 1994 para 26% em 1998. De uma forma geral o sucesso tem aumentado nas companhias, no entanto esse aumento na taxa de sucesso tem sido maior nas grandes organizações. No entanto as taxas de sucesso em empresas menores, ou em projetos menores, são maiores do que em grandes empresas ou em grandes projetos.

este aumento no investimento pode ser observado pelo aumento na quantidade de profissionais filiados ao PMI na última década Figura 2.2.

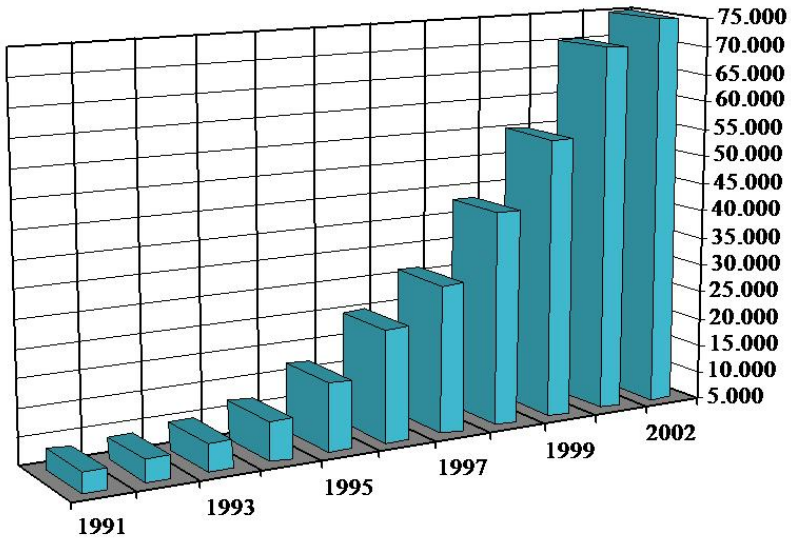

Figura 2.2 Número de profissionais filiados ao PMI

2.2.1 PMI

O PMI é uma associação sem fins lucrativos, cujo principal objetivo é difundir a gestão de projeto, de forma a promover a ética e a profissionalização desta atividade, visando

promover e ampliar o conhecimento existente sobre gerenciamento de projetos, assim como melhorar o desempenho dos profissionais e organizações nesta área [Tor05].

O PMI elaborou um código de ética a ser seguido por todos os profissionais e também um modelo de certificação na qual os profissionais avaliados recebem um selo de PMP™– *Project Management Professional*. No entanto pelo modelo adotado pelo PMI, não é condição suficiente ter o conhecimento avaliado em uma prova, mas também experiência e evolução contínua.

A fim de garantir uma rede de ensino que pudesse garantir o ensino das práticas de gerenciamento de projetos, o PMI mantém o registro de provedores de treinamento e produtos em gerenciamento de projetos, os chamados *REPs - Registered Education Providers*. Estes provedores são empresas de mercado com o objetivo de prover a este uma educação continuada. Do lado acadêmico, mantém convênios com programas de graduação e de pos-graduação (*strictu e lato sensu*) [Tor05].

O PMI é formado atualmente por cerca de 150.000 associados distribuídos em 150 países, que tem a sua disposição publicações periódicas como *PM Network, Project Management Journal, PMI Today*. O PMI é líder mundial em publicações sobre gerenciamento de projetos, no entanto sua maior publicação é o PMBOK™ *Guide – Project Management Body of Knowledge*– que organiza o corpo de conhecimento básico necessário a um gerente de projetos [Tor05]. Os associados do PMI dispõem de 3 componentes básicos para se filiar: Os *Chapters*, Grupos de Interesse Específicos e *Colleges*.

Os *Chapters* totalizam atualmente cerca de 235 em todo o mundo. São seções regionais/locais que promovem localmente a difusão do conhecimento [Tor05], através da organização de seminários e reuniões periódicas onde os associados podem além de trocar informações sobre novas técnicas e metodologias, promover o *networking*, facilitando assim o desenvolvimento de seus negócios.

2.2.2 O PMBOK™ *Guide*

O PMBOK™ *Guide* foi inicialmente publicado em 2000, tendo uma nova edição feita em 2004. Ele reúne o corpo de conhecimento básico que o PMI julga necessário a um gerente de projetos. A confecção do guia é parte do esforço do PMI em transformar a atividade de Gerente de Projetos em uma profissão única, com um currículo próprio. Apesar de criar este corpo mínimo de conhecimento, o guia procura mostrar que mais habilidades são necessárias ao gerente de projetos. Na Figura 2.3 pode-se observar a abrangência dos conhecimentos contidos no PMBOK™ e os necessários a plena atividade da gerência. O PMBOK™ se coloca em uma área de fronteira, e procura trazer práticas de várias áreas

para que de forma integrada consigam promover um melhor andamento do projeto.

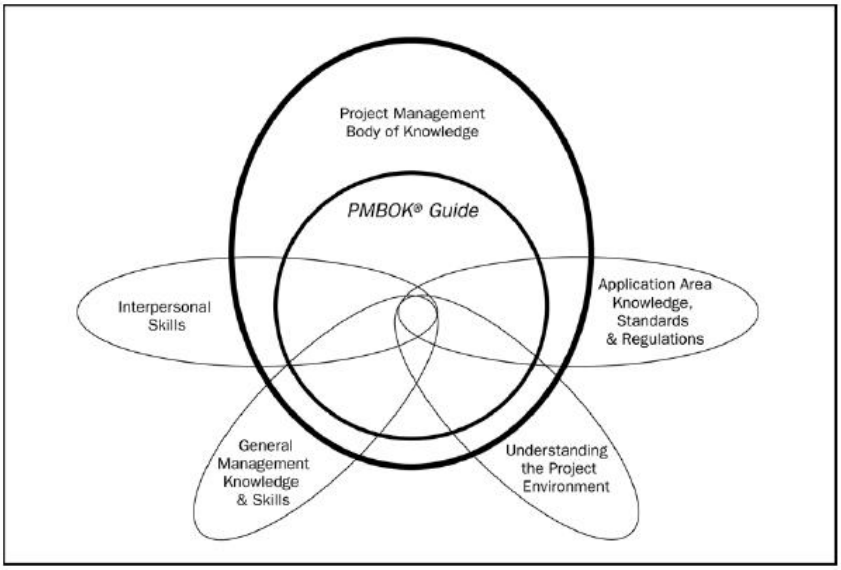

Figura 2.3 Áreas de conhecimento necessário à equipe de projeto [Ins04]

Estruturalmente o PMBOK™ *Guide* é divido em 3 grandes partes: a primeira fala sobre conceitos básicos e sobre a estrutura do guia, a segunda sobre o ciclo de vida e a organização do projeto e a terceira sobre as áreas dos conhecimento e os processos envolvidos. A seguir falaremos sobre os grupos de processos e em seguida sobre as áreas do conhecimento. É importante ressaltar que tratam-se de visões diferentes, porém integráveis, da organização de um conjunto de processos.

2.2.2.1 Grupos de Processo Um projeto possui um tempo finito para sua execução e os processos a serem executadas ao longo deste tempo são organizadas no tempo pelo modelo de ciclo de vida. O modelo de processo do definido no PMBOK™ *Guide* pode ser observado na Figura 2.4. Ao longo do tempo um projeto pode assumir vários estágios de maturidade, fazendo com que seu desenvolvimento possa ser dividido em fases. Baseado no modelo PDCA - *Plan, Do, Check, Act*, o ciclo com estas ações podem ser repetidos nestas várias fases. Observemos, entretanto, que os processos de gerenciamento de projetos não se confundem com os processo produtivo, ou seja, do processo específico dos produtos a serem desenvolvidos pelo projeto. Os processos de gerenciamento de projetos são processos de suporte, que objetivam fazer com que o projeto obtenha sucesso, ou seja, atenda aos requisitos definidos, dentro do prazo, no custo e na qualidade prevista.

O grupo de processo de Iniciação consiste de processos para facilitar autorização formal para iniciar um novo projeto ou fase. Um projeto inicia-se frequentemente fora do escopo

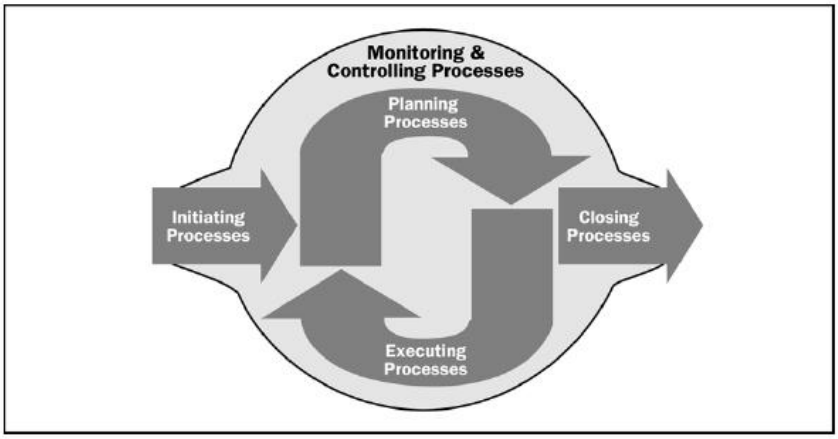

Figura 2.4 Grupos de Processos mapeados no PDCA [Ins04]

de controle do mesmo, baseado no processo organizacional, que irão fornecer as entradas necessárias ao processos de iniciação do projeto. São desenvolvidas descrições claras dos objetivos do projeto, incluindo as razões pelas quais um determinado projeto é a melhor alternativa para os requisitos apresentados. O principal artefato, ao qual entenderemos como um produto de uma ou mais atividades, a ser produzido nesta fase é o*Project Charter* ou Termo de Abertura do Projeto. Nele devem conter uma descrição básica do escopo do projeto, produtos a serem entregues, duração do projeto, previsão de uso de recursos para uma análise de investimento por parte da organização. A relação entre o projeto e os objetivos estratégicos da organização também podem ser descritos. Em projetos que tenham múltiplas fases, a iniciação pode ser representada pela reavaliação das hipóteses levantadas inicialmente [Ins04].

O grupo de processo de planejamento é formado por processos cujo objetivo é planejar e gerenciar um projeto de sucesso para a organização. Os processos ajudam a obter informações de diversas fontes, com cada uma tendo seus níveis de completude e confidencialidade. O principal artefato a ser produzido é o plano de projeto. Os processos deste grupo ajudam a definir, identificar ou maturar o escopo, custo, cronograma de atividades que ocorrem no projeto. Devido ao processo de refinamento das informações obtidas no *Project Charter*, novas informações são descobertas fazendo com que novas dependencias, riscos, oportunidades, hipóteses e restrições sejam identificadas e resolvidas [Ins04].

Devido a natureza dos negócios das empresas, os projetos devem estar devidamente preparados para mudanças. Os processo de gerenciamento de projetos é iterativo, fazendo com que as possíveis mudanças no contexto externo possam ser devidamente incorporadas pelo projeto. Estas mudanças devem ser então avaliadas e o plano de projeto alterado. Desta forma, o plano de projeto pode ser alterado com freqüência ao longo do projeto,

dado que os ciclos de execução e avaliação (PDCA) são periódicos.

O grupo de processos de execução e controle consiste dos processos que objetivam a observação do projeto em execução, de forma que potenciais problemas sejam identificados em tempo hábil e as devidas ações corretivas sejam tomadas caso necessário [Ins04]. O principal benefício deste grupo de processo é que a performance é observada e medida regularmente para identificar variações em relação ao plano de projeto. Os processos de monitoramento e controle também envolvem controlar as mudanças e recomendar ações para antecipar a problemas. Os processos de monitoramento e controle monitoram o trabalho, o esforço em todo o projeto. Em projetos divididos em fases, fornece informações para que o plano de projeto possa ser alterado, caso discrepâncias entre o planejado e o realizado tenham sido encontradas [Ins04].

O grupo de processo de fechamento objetiva o fechamento formal do projeto, ou de uma fase, de todas as suas atividades, suas dependências, distribuir todos os seus produtos. Incluindo os fechamentos de contratos formais, devoluções de recursos aos seus centros de custo.

No entanto, os grupos de processos acima mencionado não constituem em fases, ou seja, não estão necessariamente separados no tempo. A Figura 2.5 apresenta um diagrama mostrando as possíveis sobreposições destes processos e esforço envolvido no mesmo ao longo do tempo.

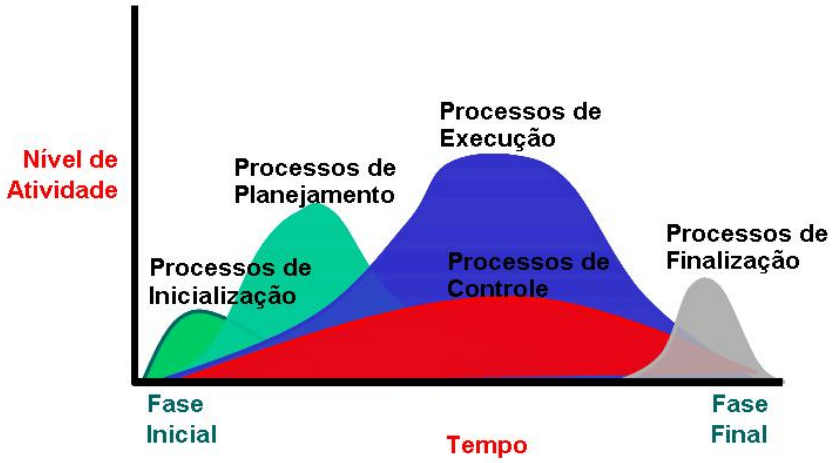

Figura 2.5 Interação entre os grupos de processo em um projeto [Ins04]

2.2.2.2 Áreas do Conhecimento O PMBOK™ [Ins04] também organiza os processos em nove áreas do conhecimento. Esta visão fornece uma espécie de estrutura estática dos processos, enquanto a visão por grupos de processos apresenta uma organização mais

dinâmica. As áreas do conhecimento são:

- Integração: esta área do conhecimento inclui os processos para identificar, definir, combinar, unificar e coordenar os diversos processos e atividades de gerenciamento de projetos dentro dos grupos de processo de gerenciamento de projetos. Ela inclui características de unificação, consolidação, articulação e ações integradoras que são essenciais para o término do projeto, para atender com sucesso às necessidades do cliente e das outras partes interessadas e para gerenciar as expectativas.

- Escopo: esta área inclui os processos necessários para garantir que o projeto inclua todo o trabalho necessário, e somente ele, para terminar o projeto com sucesso. O gerenciamento de escopo trata principalmente da definição e controle do que está e do que não está incluído no projeto.

- Tempo: esta área inclui os processos que visam garantir que o projeto seja realizado no prazo. Incluindo processos como estimativas de prazo e sequenciamento de atividades, definição e controle de cronograma, gerenciamento do caminho críticos, entre outros.

- Custo: esta área inclui os processos envolvidos em planejamento, estimativa, orçamentação e controle de custos, de forma que seja possível ao projeto terminar dentro do orçamento aprovado.

- Qualidade: esta área inclui todas as atividades das organização executora que determinam as responsabilidades, os objetivos e as políticas de qualidade, de modo que o projeto atenda às necessidades que motivaram sua realização. As atividades aqui realizadas implementam o sistema de gerenciamento da qualidade da organização.

- Recursos Humanos: esta área inclui os processos relativos a organização e gerenciamento da equipe do projeto. Dado que o projeto será executado por pessoas que executam papeis dentro do projeto, é necessário que estas pessoas estejam mobilizadas, disponíveis, devidamente capacitadas e principalmente motivadas a fim de que a equipe consiga atingir os objetivos do projeto.

- Comunicações: esta área inclui os processos para garantir a geração, coleta, distribuição, armazenamento, recuperação e destinação final das informações sobre o projeto de forma oportuna e adequada.

- Riscos: esta área inclui todos os processos relativos a identificação, análise, planejamento, respostas, monitoramento e controle dos riscos de um projeto. O objetivo destes processos é aumentar a probabilidade e o impacto dos eventos positivos e diminuir dos eventos negativos.

2.3 GERENCIAMENTO DE PROJETOS DE DESENVOLVIMENTO DE SOFTWARE

O gerenciamento de projetos consiste de técnicas de aplicação genérica, ou seja, a qualquer tipo de projeto. No entanto, cada sub-área do conhecimento tem peculiaridades próprias. Em especial a área de software, dado que este tipo de produto e o contexto no qual ele esta inserido possuem características únicas.

Ao falarmos em software, não estamos falando apenas na construção de um programa de computador, estamos falando em tecnologia, estamos falando de um segmento de negócio que faz largo uso de tecnologia e com a qual está acostumado. Desta forma o perfil de comportamento dos usuários de tecnologia é diferenciado. A palavra tecnologia está sobretudo relacionada a inovação, que traz consigo todo o contexto de promoção a mudanças. Ou seja, mais do que esta acostumado a mudanças as organizações desta área, são os promotores das mudanças. Isto faz com que sejam muito sensíveis aos contextos de mercado e perfil de consumo. A dinâmica de mudanças no mercado é muito grande, fazendo com que as tecnologias e produtos mais utilizados no mercado sejam, ou possam ser, rapidamente mudados. Estas mudanças trazem naturalmente um re-arranjo das posições de mercado. Às empresas competidores nestes mercados restam querem manter suas posições, caso estas sejam confortáveis, ou que queiram ampliar suas posições. No entanto, o grande fator que promove mudanças significativas na estrutura de mercado é a inserção de inovações.

As inovações não estão apenas relacionadas a produtos, mas também a técnicas, metodologias que permitam às organizações estarem mais próximas de seus objetivos empresariais.

Segundo SWEBOK™ [A+04] existem aspectos específicos aos produtos de software e ao ciclo de vida de um software que complicam o gerenciamento do projeto, eis alguns:

- A percepção do cliente acerca do produto não consegue perceber sua complexidade, particularmente no que diz respeito a solicitação de mudanças.

- Software é construído de forma iterativa

- Processo de construção de software envolve aspectos de criatividade e disciplina, e manter um equilíbrio adequado entre estes dois aspectos é difícil.
- O grau de inovação e complexidade é sempre muito alto
- A alta taxa de mudança nas tecnologias utilizadas.

Por características como estas é que o produto de software tenham preços de aquisição nem sempre acessíveis a seu público.

Aliado a este fato, outro ponto importante a ser observado é o tempo de consumo. Cabe lembrar entretanto que o software é um produto que se materializa para o usuário no momento em que este solicita um determinado serviço, sendo este então o momento de percepção do seu valor agregado. Mas esta materialização só é possível graças ao conhecimento que nele foi depositado. Conhecimento este captado de especialistas e devidamente implementado em alguma linguagem de programação. No entanto, este conhecimento tem aplicação finita no tempo, pois a rotina das organizações é dinâmica. Portanto, por mais que o software não sofra a ação do tempo, o conhecimento nele embutido torna-se obsoleto. A conseqüência disto é que a visão do software como um ativo da organização seja questionado, a ponto muitos modelos de negócios atuais fazerem uso de provedores de serviço e de solução. A depreciação de um software é muito grande.

O SWEBOK™ [A+04] divide a engenharia de software em várias áreas do conhecimento, entre elas o gerenciamento de projetos. Ela define que a área de gerenciamento de projetos de software inclua as 5 primeiras áreas de conhecimento definidas pelo PMBOK™ (integração, escopo, tempo, custos e qualidade) acrescida da área de métricas. Entretanto, estas áreas devem encontram-se intimamente relacionadas a outras áreas de conhecimento em software: Requisitos, Gerência de Configuração, Engenharia de Processo e Qualidade. Mais especificamente o SWEBOK™define que as seguintes sub-áreas sejam consideradas como parte do gerenciamento de projetos de software:

- Iniciação e definição de escopo
- Planejamento
- Execução
- Revisão e avaliação
- Fechamento
- Métricas

As atividades de gerenciamento a serem utilizadas e o processo mais detalhado são dependentes do processo de software a ser utilizado. Por exemplo, um projeto que utilize um modelo de ciclo de vida baseado no processo unificado [JB+98] será diferente de um que utilize metodologias ágeis.

A Figura 2.6 apresenta de forma mais detalhada a engenharia de processos. Com ela é possível visualizar os aspectos considerados no momento de se definir um processo para construção de um software.

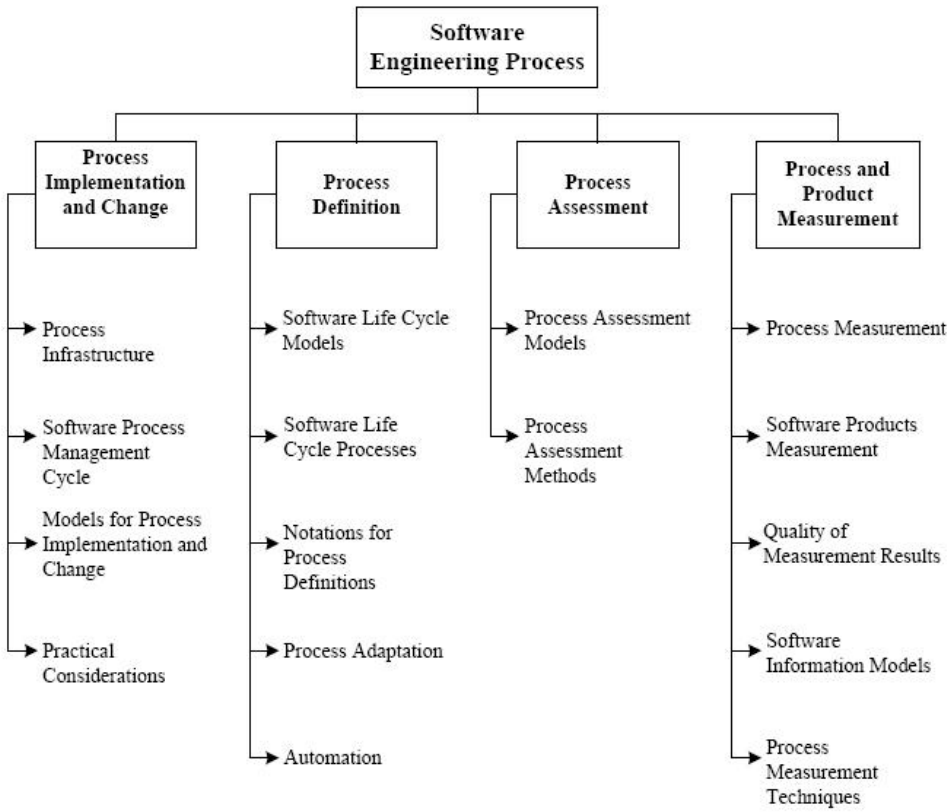

Figura 2.6 Engenharia de processos [A+04]

2.4 CAPACITAÇÃO EM GERENCIAMENTO DE PROJETOS E SUAS FERRAMENTAS

A capacitação em gerenciamento de projetos vem ganhando cada vez mais importância, na medida em que as organizações valorizam cada vez mais os benefícios de profissionais capacitados e certificados em seus projetos, ou seja, resultados positivos estão sendo cada vez mais obtidos.

Conforme observado na Figura 2.2 a quantidade de profissionais tem aumentado significativamente. Também tem aumentado a quantidade de profissionais a busca do conhecimento de gerenciamento de projetos, haja vista a grande quantidade de *REPs* no mercado e o aumento dos cursos de pós-graduação específicos em gerenciamento de projetos.

A capacitação em gerenciamento de projetos tem tido dois grandes objetivos:

- Formação: Neste caso o aluno procura adquirir os conhecimentos sobre gerenciamento de projetos de uma forma geral.

- Certificação: Neste caso o aluno busca a consolidação de um conhecimento já adquirido e também discutir aspectos da prova de certificação PMP™.

Atualmente a certificação PMP™é muito valorizada pelo mercado e representa para seu detentor uma maior empregabilidade e maiores níveis de remuneração.

No entanto, os mecanismos de ensino utilizados são os tradicionais, ou seja, através de aulas expositivas. Essas aulas permitem que seja comunicado um modelo, no caso o de gerenciamento de projeto. A fim de aumentar sua captação e entendimento por parte dos alunos, os professores e instrutores fazem uso de exemplificações que mostram a aplicação do modelo em uma situação hipotética.

Outro expediente muito utilizado na educação gerencial é a comunicação da experiência real obtida em projetos através de *cases*, ou seja, busca-se ai o aprendizado com a experiência de aplicação de terceiros. Entretanto, por questão de mercado, em sua maioria os *cases* são experiências de sucesso de aplicação de um determinado modelo. No entanto, tão importante quanto saber como fazer é saber como não fazer, ou seja, é importante conhecer casos de insucesso na aplicação de uma determinada teoria ou modelo. Estes casos já são menos comunicados pois acabam por expor o nome de profissionais ou de organizações.

A experiência entretanto é individual, pois por mais que um gerente possa analisar um contexto e entender o porque de uma determinada decisão naquele contexto, ele não terá vivenciado a experiência dele mesmo ter tomado àquela decisão. O período de aprendizagem então continua após a aquisição do conhecimento, acontecendo normalmente através da vivencia do papel de gerente em um projeto piloto em sua organização. No entanto, esta possibilidade faz com que a organização envolva recursos reais, custos reais, riscos reais a um projeto no qual ainda está capacitando um gerente.

O uso de jogos de negócio tem sido uma saída interessante para estas organizações, pois evitam o uso de recursos reais e permitem a seus gerentes a experimentação de contextos problema, da tomada de decisão e de seu resultado no contexto. Os jogos de

negócio, sejam eles computacionais ou não, fornecem então um ambiente simulado que modela o sistema real.

CAPÍTULO 3

MODELAGEM DE SISTEMAS E SIMULAÇÃO

A indústria de software vem buscando ao longo do tempo a redução dos seus custos de produção e o aumento da qualidade de seus produtos. Novos modelos de desenvolvimento surgem com uma forma de resposta a essa busca. Para as organizações a simples adoção de novos modelos ou paradigmas não é suficiente, pois para atestar a eficiência de um novo modelo ou processo, é necessário tempo para o amadurecimento do processo na organização, para que as pessoas estejam devidamente treinadas, para que o processo reflita a realidade da organização e de seu ambiente. Os programas de qualidade de uma forma geral, pregam a prática da melhoria contínua, onde os processos produtivos passam continuamente por otimizações.

No entanto, este processo gera altos custos, seja na capacitação de pessoal, seja nas iterações de otimização. Neste ponto os simuladores representam grandes ferramentas aliadas, permitem a aplicação e teste de um processo em um ambiente virtual, onde os recursos envolvidos não são reais. O tempo de simulação geralmente é abreviado, fazendo com que o ciclo de amadurecimento do processo seja abreviado.

Kellner et al [KMR99] apresentam vários propósitos para os quais a simulação pode ser utilizada:

- gerenciamento estratégico
- planejamento
- gerenciamento e controle operacional
- melhoria de processo e inovação tecnológica
- entendimento de um processo
- treinamento e aprendizado

O uso de simuladores apresenta grandes vantagens como ferramenta de auxilio ao aprendizado, na medida que o tempo de execução do processo em ambiente simulado seja da ordem de horas, ao invés de meses ou anos, facilitando assim o processo de reflexão

sobre as decisões tomadas, dado o pequeno tempo entre a tomada de decisão e suas conseqüências. Outra vantagem é o não comprometimento de recursos reais e o isolamento do ambiente real no que tange às conseqüências de decisões ditas "incorretas"para o negócio da empresa. Quando utilizado em treinamentos, permite focar em eventos e cenários relevantes ao objetivo do treinamento, pois em um cenário real os projetos estão submetidos a várias situações que não são relevantes ao um objetivo especifico, mas que consomem tempo, como por exemplo a execução de tarefas administrativas e burocráticas.

Apesar de benéficos o uso de simuladores apresentam alguns problemas ao processo de aprendizagem relacionados ao engajamento e da sensação de realidade passada aos alunos, que fazem com que muitas decisões sejam tomadas sem a devida preocupação com o resultado. O uso de jogos no treinamento permite que o jogador tenha essa maior imersão no ambiente e a interface e interatividade permitem maior relação com o ambiente real, aumentando assim o comprometimento com o aprendizado.

Neste capítulo são apresentados os principais conceitos sobre modelagem de sistemas e os tipos de modelos: mentais, explícitos e formais. Por sua capacidade de simulação, os modelos formais são melhor estudados e suas principais técnicas de simulação: discreta, contínua e dinâmica de sistemas. Em seguida as aplicações destas técnicas de simulação ao processo de desenvolvimento de software são apresentadas. Por fim veremos o uso de jogos de simulação.

3.1 MODELAGEM DE SISTEMAS

Um sistema representa uma parte da realidade, sendo formado por componentes que interagem entre si para realizar tarefas que não podem ser realizadas isoladamente [Bar01, MB99]. Os relacionamentos entre estes componentes formariam a estrutura do sistema [Bar01, Mar97]. Sistemas podem ser classificados como abertos ou fechados, o segundo se diferenciaria do primeiro pela presença de ciclos de re-alimentação ou *feedback*, possuindo assim seqüências cíclicas de atividades [Bar01, R$^+$83].

Todo modelo é uma representação de um sistema. A utilidade dos modelos está no fato deles simplificarem a realidade. Este sistema pode ser um sistema ecológico, social, ou por exemplo, um projeto de software [Ver03]. Os modelos podem ser classificados como mentais ou explícitos.

Os modelos mentais representam a percepção que o indivíduo forma a respeito das interações entre os componentes de um sistema e do comportamento resultante destas interações. Os modelos mentais são flexíveis [Sta88], podendo se adaptar a novas situações, seja através de uma simplificação ou de uma expansão, geralmente fruto de um processo

de generalização e comparação do sistema ou ambiente observado com as informações contidas na mente. Estas novas informações constituem-se em novos fatos que podem contradizer o modelo, ou parte dele, o que leva a um re-arranjo da estrutura do modelo mental. Estes novos fatos também podem reforçar relações de conceitos e acrescentar novas informações, provocando assim a sua expansão. No entanto, os modelos mentais estão restritos pela incapacidade da mente humana em lidar com um grande número de fatores distintos [Bar01]. Por conseguinte, os modelos mentais são normalmente simples, considerando apenas um número reduzido de componentes e de relações entre eles. Essa capacidade reduzida faz com que características importantes de sistemas passem desapercebidas, possibilitando a geração de interpretações equivocadas a respeito do sistema ou ambiente observado, o que leva frequentemente a decisões incorretas. Aliado a isso, tem-se o fato de que os modelos mentais residem no conhecimento tácito de cada indivíduo, sendo difíceis de compartilhar e validar [Bar01].

Os modelos explícitos são representações dos modelos reais em uma linguagem capaz de ser compartilhada com outros indivíduos. Segundo Sterman e Forrester, os modelos explícitos são limitados a influenciar os modelos mentais, visto que nenhuma decisão é tomada a partir deles, mas a partir dos modelos mentais [Bar01].

Os modelos explícitos possuem normalmente um conjunto de símbolos e semânticas, e são úteis para a representação, validação e difusão do conhecimento. A representação de um modelo permite a explicitação de suas premissas e a transposição da barreira da complexidade imposta aos modelos mentais, ajudando ainda a organizar a estrutura das informações do sistema, facilitando assim o seu entendimento. A validação do modelo é importante na medida em que os resultados reais obtidos da simulação de sua execução podem diferir dos resultados esperados por suas premissas, especialmente em sistemas complexos essa divergência pode acontecer. A difusão do conhecimento é possível pois o conhecimento esta representado em locais acessíveis ao grupo, numa semântica inteligível. Os modelos mentais por encontrarem-se armazenados no indivíduo, compõem a sua experiência individual, que não pode ser comunicada. Outra característica interessante de modelos explícitos, é que por poder capturar conhecimento de sistemas reais, eles podem aproveitar o mesmo para aplicá-los a situações hipotéticas, avaliando assim quais seriam os resultados reais naquele novo cenário.

Os modelos explícitos podem ser classificados em informais e formais. Os modelos formais possuem uma semântica não ambígua, e neste caso, são passíveis de verificação e de simulação, dado que podem ser transformados para postulações matemáticas e lógicas.

3.2 MODELOS INFORMAIS

Os modelos informais são muito importantes para a comunicação de um determinado modelo ou processo, porém sua semântica não é suficiente para garantir que ele seja simulado computacionalmente. No entanto, são passíveis de serem executados por pessoas dentro de uma organização, por exemplo. Este mecanismo tem sido utilizado para formalizar processos produtivos, em especial a Engenharia de Software. As técnicas de modelagem de processos de software possuem normalmente uma linguagem gráfica com conceitos comuns a este tipo de atividade. Vários tipos de diagrama foram utilizados com este fim ao longo do tempo, um exemplo é o DFD – Diagrama de Fluxo de Dados– que tem a utilidade de descrever um fluxo de informações.

A UML – *Unified Modeling Language* – é uma linguagem que ajudou a unificar os conceitos e graficismos comuns a modelagem de sistemas de software, fazendo uso de diversos diagramas de forma a apresentar varias visões de como um sistema de software pode ser observado [RJB99]. No entanto, esta mesma linguagem possui mecanismos de extensão, chamadas *profiles*, que permitiram o seu uso para modelagem de processos [Joh04]. A modelagem de processos passou a ser utilizada dentro da engenharia de software como uma disciplina com a criação do UP – *Unified Process* [JB+98]. Com isso a UML passou a poder representar não apenas características de sistemas, mas também do processo organizacional no qual o produto de software seria inserido, permitindo assim que os engenheiros de software pudessem ter uma transição mais tênue entre os processos organizacionais e os processos de software.

Esta utilização da UML abriu caminho para que a mesma pudesse ser utilizada também para a explicitação de processos de software. Neste contexto foi criado o SPEM – *Software Process Engineering MetaModel* [Gro05] – como um meta-modelo para a criação e representação de processos de software, de forma que este possa ser comunicado e executado por membros de uma equipe de desenvolvimento de software. Outra importante contribuição do SPEM é a sua utilização em ambientes de desenvolvimento de software (ADS) centrados no processo, também conhecidos como PSEE – *Process-Centered Software Engineering Enviroment*. Estes ambientes suportam não só a função de desenvolvimento de software, mas também funções relacionadas a gerência e garantia da qualidade [Ara95, Gro05]. No Capítulo 4 discorreremos mais sobre modelagem de processos de software. Para o momento, é de fundamental importância o entendimento da relevância da representação de modelos explícitos, mesmo que informal.

3.3 MODELOS FORMAIS

Os modelos formais podem ser vistos como modelos que são capazes de serem executados computacionalmente. De acordo com o seu estado os modelos formais podem ser classificados em estáticos ou dinâmicos, onde os estáticos são aqueles que cujas variáveis não sofrem alterações ao longo do tempo; já os dinâmicos apresentam alteração nos seus estados à medida que a simulação ocorre. Modelos dinâmicos podem ser classificados em discretos ou contínuos, de acordo com a forma como ocorrem as mudanças de estado, onde o primeiro é baseado na ocorrência de eventos, cujo intervalo de tempo entre os vários eventos não é previamente definido. Já em um modelo contínuo as variáveis mudam em um intervalo de tempo constante [Bar01]. Os modelos formais são passíveis de serem simulados. A simulação é um processo que reproduz o comportamento de um sistema através de operações numéricas realizadas por computador [Mar97]. A simulação pode ser utilizada para verificação do modelo, especialmente quando o efeito das interações entre os componentes e o sistema como um todo, apresentam-se separadas no tempo e no espaço [Bar01]. As técnicas de simulação podem ser classificadas em Simulação Discreta e Simulação Contínua.

3.3.1 Simulação Discreta

Em uma simulação discreta o simulador possui uma fila de eventos ordenada pelo tempo para a ocorrência do evento. Em cada iteração do processo de simulação, o simulador trata o primeiro evento da fila, adiantando um relógio interno até o próximo instante de ocorrência de cada evento e atualizando as variáveis do modelo de acordo com o evento. O tratamento de um evento também pode gerar novos eventos, que serão inseridos na fila de eventos do simulador. O intervalo de tempo entre cada par de eventos não pode ser previamente determinado, fazendo com que a simulação ocorra sem um período constante. A simulação é encerrada em um tempo previamente determinado, ou quando não existirem eventos a serem tratados [Bar01].

Um exemplo de interpretação destes eventos poderia ser uma lista de atividades para execução de um determinado projeto. As atividades possuem dependências, as quais só permitem o inicio de uma atividade se suas predecessores já tiverem concluído. Cada evento seria uma atividade a ser executada. A conclusão de uma atividade, através da espera do seu tempo previsto, fará com que novas atividades, notadamente aquelas que possuam relação de dependência, sejam colocadas na fila de execução.

Os sistemas baseados em regras são um particular caso de simulação discreta, no qual

os estados do sistema são representados por fatos, e as ações a serem executadas são representadas como predicados [Bar01].

Tomando o mesmo exemplo anterior, pode-se representar uma lista de atividades através de um conjunto de regras, apresentando assim sua estrutura hierárquica e suas dependências. A execução de uma atividade é sinalizada através da inserção de um fato na memória de trabalho. A conclusão do de uma atividade também é sinalizada com a inserção de outro fato na memoria de trabalho. A inserção destes fatos provoca o disparo de outras regras, que fornecerão novas atividades a serem executadas. A simulação continua ocorrendo através deste ciclo, até que não sejam mais disparadas novas regras.

3.3.2 Simulação Contínua

A simulação de modelos contínuos ocorre em intervalos de tempo infinitesimais, constantes e previamente determinados. As variáveis são alteradas a cada iteração de simulação. A simulação termina após o fim do número de iterações previstos para a simulação [Bar01]. A simulação contínua é suportada pela execução de equações que trazem dentro de si as informações do modelo a ser simulado [WP99]. São utilizados dentro de pesquisa operacional para simular modelos estocásticos e com isso validá-los. As variáveis pertencentes ao modelo possuem um universo de domínio que são selecionados randomicamente [Per03].

3.4 DINÂMICA DE SISTEMAS

A dinâmica de sistemas é uma disciplina de modelagem criada na década de 1960 por Forrester [For61], no MIT. Ela apresenta um conjunto de ferramentas que nos permite entender a estrutura e a dinâmica de um sistema de dinâmica complexa [Ste00]. Onde um sistema de dinâmica complexa é um sistema com componentes ou atividades inter-relacionadas, cujas relações não são lineares, e com relação causa-efeito distantes no tempo. Outra característica comum a sistemas de dinâmica complexa é a existência de retro-alimentações [Dug04].

As técnicas da dinâmica de sistemas podem ser aplicadas para entender e influenciar o comportamento de um sistema ao longo do tempo. A dinâmica de sistemas surgiu das apreciações de Forester sobre as dificuldades enfrentadas pelos gerentes, segundo as quais, estas seriam advindas, não do lado da engenharia mas do lado do gerenciamento, dado que os sistemas sociais são muito mais árduos de se entender do que os sistemas físicos. Sua aplicação para a *General Eletric*, em meados dos anos 1950, permitiu concluir que o ciclo de permanência dos empregados em uma de suas plantas industriais advinha da

estrutura interna, diferente do que imaginavam os dirigentes. Os trabalhos posteriores, ainda nos anos 1950 e início dos anos 1960, permitiram a evolução da simulação manual para a simulação baseada em computador através da linguagem, SIMPLE – *Simulation of Industrial Management Problems with Lots of Equations* – e, posteriormente, DYNAMO – *DYNAmic MOdels* [Rad].

Posteriormente, Forrester criou um modelo para simular a dinâmica urbana (URBAN DYNAMICS), segundo o qual foi possível ilustrar que muitas políticas de urbanização conhecidas são ineficientes, ou tornam os problemas urbanos ainda pior. Foi também aplicada no campo sócio-econômico, onde foi criado um modelo para simular o comportamento e os efeitos do crescimento exponencial da população mundial (WORLD DYNAMICS) [Rad].

3.4.1 Os Modelos da Dinâmica de Sistemas

A Dinâmica de Sistemas disponibiliza dois tipos de diagrama: Diagramas de Causa e Efeito e Diagramas de Repositório e Fluxo.

3.4.1.1 Diagramas de Causa e Efeito O diagrama de causa e efeito é uma técnica simples e flexível que permite aos tomadores de decisão identificar as variáveis de interesse e explorar as relações entre essas variáveis.

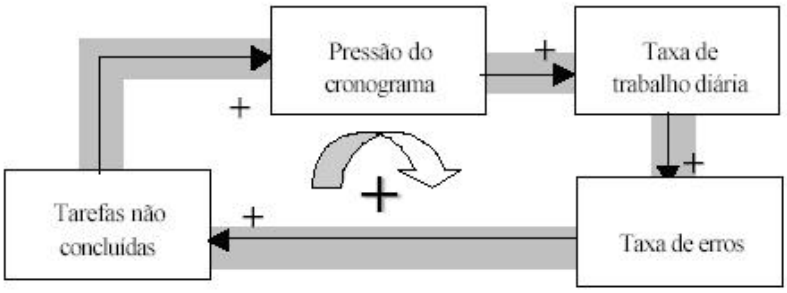

Figura 3.1 Diagrama de causa e efeito da dinâmica de sistemas [Bar01]

A Figura 3.1 apresenta um exemplo de um diagrama de causa e efeito num contexto típico de gerenciamento de projetos. Neste caso pode-se observar a relação positiva entre Taxa de trabalho diária e a Taxa de Erros, indicando que estas são diretamente proporcionais, ou seja, se a pressão no cronograma aumentar então a taxa de trabalho diária também irá aumentar. A mesma coisa acontece com as demais variáveis. O sinal no meio do ciclo, indica a polaridade do ciclo.

O ciclo, também chamado de retro-alimentações, possui uma polaridade que indica o comportamento esperado pelo mesmo. Caso a polaridade seja positica, há a relação de reforço e a tendência do comportamento é o crescimento ou o descrescimento dos valores das variáveis neles contidas. Caso seja negativa há a tendência a um equilíbrio. [Rad].

O diagrama de causa e efeito é construído através de setas que ligam a causa ao efeito. Uma ligação entre duas variáveis pode ser positiva ou negativa, indicando se o efeito será diretamente ou inversamente proporcional, respectivamente. Sterman define que para uma dada ligação $X \rightarrow Y$ ela será [Ste00]:

- Uma ligação positiva, caso o crescimento (decréscimo) no valor de X implique no crescimento (decréscimo) do valor de Y. Tomando como premissa que todas as outras variáveis permaneçam com o mesmo valor.

- Uma ligação negativa, caso o crescimento (decréscimo) no valor de X implique no decréscimo (crescimento) no valor de Y. Tomando como premissa que todas as outras variáveis permaneçam com o mesmo valor.

Um ciclo de retro-alimentação é caracterizado pela possibilidade de se traçar um caminho, através das relações de causa e efeito, que retorne à variável original. Aliado a esse fato, dado que todas as ligações foram caracterizadas como positivas ou negativas, é possível determinar a polaridade do ciclo, permitindo assim prever o comportamento geral do ciclo. Segundo Sterman [Ste00] a polaridade de um ciclo é:

- Positiva, se o número de ligações negativas em um ciclo for par.

- Negativa, se o número de ligações negativas em um ciclo for ímpar.

Duggan [Dug04] apresenta, na Figura 3.2, um exemplo de um diagrama de causa e efeito para o processo de desenvolvimento de software.

As principais variáveis do sistema modelado são representadas, juntamente com suas relações causais. É possível identificar três ciclos de retro-alimentação. As relações entre as variáveis são apresentadas abaixo:

- *Completed Design Specification* $\rightarrow$ *Backlog*: A medida que as especificações de projeto vão sendo produzidas o backlog de especificações para serem implementadas pelos desenvolvedores vai aumentando. Neste modelo a unidade de medida para o backlog é o método (menor unidade de trabalho em sistemas orientados a objeto)

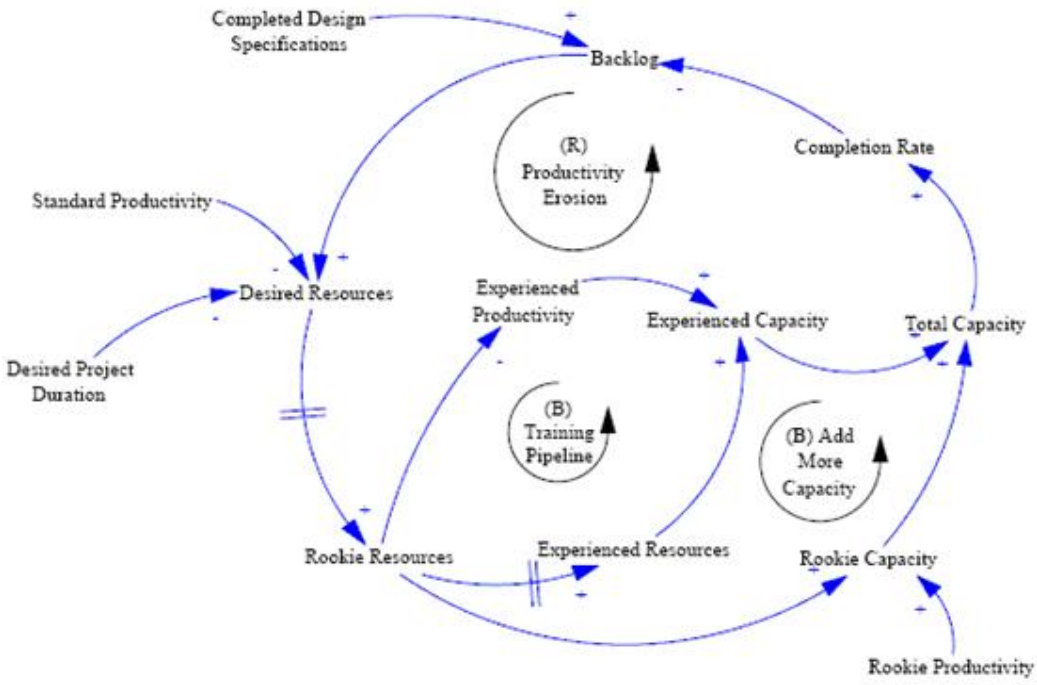

Figura 3.2 Diagrama de causa e efeito para o desenvolvimento de software [Dug04]

- *Backlog* → *Desired Resources*: A medida que o backlog cresce, a quantidade de recursos desejados (desenvolvedores) para sua implementação também aumenta. A quantidade de recursos desejados também é função da duração desejada (dias) para o projeto e da produtividade padrão (medida em métodos/pessoa/dia).

- *Standard Productivity* → *Desired Resources*: O aumento na produtividade implica na redução de recursos necessários.

- *Desired Resources* → *Rookie Resources*: Esta relação mapeia o processo de contratação de pessoal, que ao ser contratado ira demandar interação com desenvolvedores mais experientes, o que irá afetar a produtividade geral. As duas linhas que cruzam a seta representam um retardo, indicando que o efeito estará distante no tempo em relação à causa.

- *Rookie Resources* → *Experienced Productivity*: A medida que o número de desenvolvedores sem experiência é adicionado a equipe, a sua demanda por interação com os desenvolvedores mais experientes aumenta, o que provoca a redução nos níveis de produtividades.

- *Rookie Resources* → *Experienced Resources*: Ao longo do tempo, seja através de treinamento ou do contato com o trabalho, os desenvolvedores iniciante se tornarão desenvolvedores experientes.

- *Rookie Resources* → *Experienced Capacity*: A medida que a quantidade de desenvolvedores iniciante aumenta, também aumenta a capacidade deste grupo de desenvolvedores.

- *Rookie Productivity* → *Rookie Capacity*: O aumento na produtividade dos desenvolvedores iniciantes provoca o aumento na capacidade deste grupo de desenvolvedores.

- *Rookie Capacity* → *Total Capacity*: O aumento na capacidade dos desenvolvedores iniciantes provoca o aumento na capacidade de desenvolvimento total.

- *Experienced Productivity* → *Experienced Capacity*: O aumento na produtividade (métodos/dia) dos desenvolvedores experientes provoca o aumento na capacidade deste grupo de desenvolvedores.

- *Experienced Resources* → *Experienced Capacity*: O aumento na quantidade de desenvolvedores experientes provoca o aumento na capacidade dos desenvolvedores experientes.

- *Total Capacity* → *Completion Rate*: A medida que a capacidade total aumenta, a taxa de completude do projeto também aumenta.

- *Completion Rate* → *Backlog*: A medida que a taxa de completude do projeto aumenta o backlog de métodos a serem implementados vai diminuindo.

No modelo apresentado na Figura 3.2, três ciclos de retro-alimentação podem ser observados, sendo dois chamados Balance (polaridade negativa) e um de Reforço (polaridade positiva):

- *Add More Capacity (BackLoop* → *Desired Resources* → *Rookie Resources*→ *Rookie Capacity* → *Total Capacity* → *Completion Rate* → *Backlog*): Ciclo de balanceamento (polaridade negativa) mostra que quando um desenvolvedor sem experiência é contratado para a equipe, imediatamente ele adiciona capacidade a equipe, o que aumentar a taxa de completude do projeto, e conseqüentemente diminuirá a necessidade de contratar novos desenvolvedores.

- *Training Pipeline (Backlog → Desired Resources → Rookie Resources → Experienced Resources → Experienced Capacity → Total Capacity → Completion Rates → Backlog*: Ciclo de balanceamento (polaridade negativa) que faz com que a medida que os desenvolvedores iniciante forem treinados, pela execução de suas atividades, sua produtividade ira aumentar, e conseqüentemente a capacidade total do geral também aumentara.

- *Productivity Erosion (Backlog → Desired Resources → Rookie Resources → Experienced Productivity → Experienced Capacity → Total Capacity → Completion Rate → Backlog*): Ciclo de reforço (polaridade positiva) que mostra que a contratação de novos desenvolvedores demandada pelo aumento no backlog, irá provocar a diminuição das pessoas experientes, e conseqüentemente a capacidade total e a taxa de completude, o que ira aumentar o backlog.

Os diagramas de causa e efeito são mais simples e devem ser refinados para diagramas de repositório e fluxo. O diagrama de causa e efeito apresenta vários componentes do sistema e o efeito da acumulação ou redução de volume em um componente provoca sobre os demais. Devido a sua simplicidade o diagrama de causa e efeito é utilizado para explicar conhecimentos retirados do modelo, porém não é adequado para análise de regras e simulações.

3.4.1.2 Diagramas de Repositório e Fluxo Os diagramas de repositório e fluxo apresentam nível de detalhamento maior, simbologia diferente. A Na Figura 3.3 apresenta a simbologia gráfica dos elementos do diagrama.

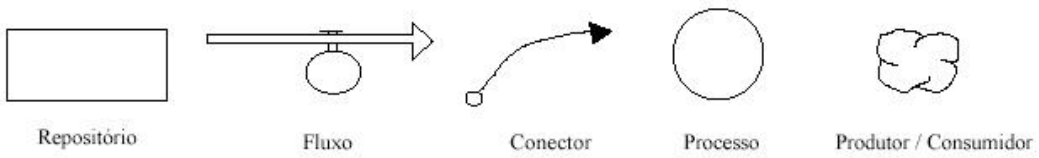

Figura 3.3 Símbolos gráficos do diagrama repositório e fluxo da Dinâmica de Sistemas

Onde:

- Repositório: Representa um elemento que possa ser acumulado ou consumido ao longo do tempo. Por exemplo a equipe envolvida no projeto ou o conjunto de artefatos a serem entregues podem representar repositórios.

- Fluxo: Representam uma taxa de variação do repositório em relação a um instante de tempo, e estão normalmente ligados a um ou dois repositórios. A taxa de contratação ou demissão de membros da equipe poderia ser vista como um fluxo.
- Processo: Utilizado para calcular informações a partir de um conjunto de parâmetros (taxa de variação indicada por um fluxo, nível de um repositório ou produto de um outro processo). A data de conclusão de um projeto pode ser calculada por um processo.
- Conector: representam uma via de transmissão de informações no modelo, assim como o fluxo transfere elementos entre os repositórios os conectores permitem a troca de informações entre os processos.
- Produtor/Consumidor: Representar produtores e consumidores infinitos, ou seja, fonte infinita de informações a um fluxo, ou um destino capaz de consumir tudo o que um determinado fluxo gera.

Em [Rad] Radzicki afirma que a modelagem dinâmica é baseada no principio da acumulação, que afirma que todo comportamento dinâmico no mundo ocorre quando um fluxo acumula em um estoque.

A Figura 3.4 apresenta um diagrama de estoque e fluxo simplificado.

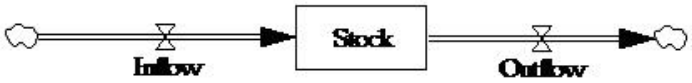

Figura 3.4 Diagrama de estoque e fluxo simplificado

É importante observar que o nível acumulado pelo estoque Stock é controlado pelos fluxos de entrada (Inflow) e de saída (Outflow). O aumento no nível do estoque acontecerá se a taxa do fluxo de entrada for maior que a taxa de saída. O nível diminuirá se a taxa de entrada for menor que a taxa de saída.

Na Figura 3.5, temos um exemplo bem simples, que pode ser observado no estudo de Forrester (*World Dynamics*) [For61], onde ele mostra que não basta haver a redução na emissão de CFC para que a camada de ozônio deixe de ser degradada, pois uma vez liberados na atmosfera, esses gases não encontram um fluxo de saída da mesma. Portanto, apenas a política da redução da emissão desses gases não é suficiente, sendo então necessário também que haja a criação de algum mecanismo para o consumo deste gás da atmosfera.

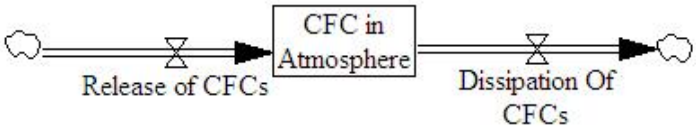

Figura 3.5 Estrutura da dinâmica do sistema de acumulação de CFC na atmosfera [Rad]

Na Figura 3.6, Duggan [Dug04] apresenta o diagrama de repositório e fluxo resultado do detalhamento do diagrama de causa e efeito da Figura 3.2. Nele é possível observar que os principais estoques do problema modelado são o *backlog*, o número de desenvolvedores inexperientes e o número de desenvolvedores experientes.

Algumas novas informações também foram adicionadas dado que o sistema real possui alguns limitantes que não foram representados no diagrama de causa e efeito, como a quantidade máxima de recursos disponíveis e a quantidade máxima disponível para contratação. Aliado a isso também são representados no modelo a entrada de informações externas, que se tornam constantes no modelo, como por exemplo: tamanho do projeto, produtividade base ou inicial, data de finalização do projeto.

Os diagramas de repositório e fluxo possuem uma interpretação matemática, em que seus elementos são transformados em equações, que são passíveis de execução em uma máquina de simulação. A criação destes diagramas possibilita a compilação para equações que são executadas durante a simulação.

3.5 MODELAGEM DE PROCESSOS DE SOFTWARE

A aplicação das técnicas de modelagem formal e sua conseqüente simulação atende a dois principais objetivos: a otimização do processo modelado e a capacitação de pessoas no conhecimento do processo.

A otimização do processo é alcançada através do confronto entre os resultados esperados pela simulação do modelo e os resultados reais, onde cada simulação serve para a geração de resultados que são comparados aos esperados. As diferenças encontradas fomentam a mudança no processo, que mais uma vez será simulado. A execução cíclica deste processo faz com que a melhoria implementada nas modificações aproxime os resultados reais dos esperados. Esta técnica é utilizada no processo de melhoria contínua de processos.

A simulação aplicada à melhoria contínua tem particular importância no contexto

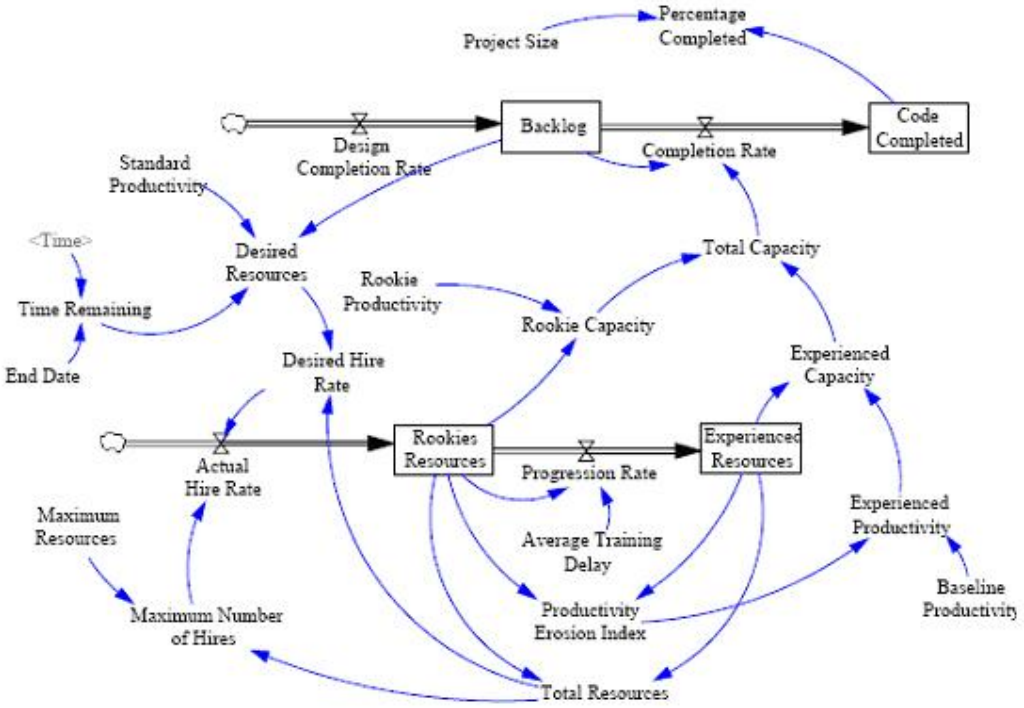

Figura 3.6 Diagrama de repositório e fluxo do desenvolvimento de software

atual de software das organizações, na medida em que os principais modelos de qualidade de software são alicerçados neste conceito. Observando modelos como o CMMI™ [BKS03], MPS.BR [Sof06], entre outros, observa-se que todos procuram reduzir a complexidade do processo de aumento da qualidade em níveis, mas sobretudo no conceito de melhoria continua, onde em cada nível um determinado aspecto é dominado e, no próximo, além das capacidades do nível anterior, novos requisitos são incorporados ao novo modelo, obrigando assim que os modelos de desenvolvimento de software sejam evolutivos e cujo processo de evolução seja incremental. No entanto, um dos grandes aspectos na obtenção do nível de maturidade, é que para a consolidação das capacidades em um determinado nível, há a necessidade de aplicação em vários projetos, a fim de verificar se o modelo aplicado de fato garante os requisitos do nível. Deste modo a simulação de modelos tem grande importância neste contexto.

No dia a dia das organizações boa parte do processo produtivo de software é executado por pessoas. Portanto, há a necessidade que essas pessoas estejam devidamente capacitadas no processo. O uso da simulação neste contexto também tende a agregar muito valor, na medida em que permite que as pessoas tenham contato com o processo sem que o mesmo esteja acontecendo de fato, evitando assim custos e riscos, inerente a

execução real de um projeto.

3.5.1 Aplicação das Técnicas de Modelagem

Barros apresenta uma relação de aplicações das técnicas de modelagem mencionadas à modelagem de processos de software, as quais serão detalhadas abaixo [Bar01].

3.5.1.1 Eventos Discretos A modelagem por eventos discretos foi utilizada por Rus e Collefelo para criar um modelo com o objetivo de avaliar a eficiência de políticas de aprimoramento de processos e seus efeitos na confiabilidade dos produtos de software construídos [RCL98]. Entre as políticas de aprimoramento analisadas, encontram-se a especificação formal de requisitos, inspeções, revisões, reutilização de software, entre outras. A viabilidade de cada política depende de um conjunto de fatores relacionados ao processo e ao produto, como o tamanho do código fonte, a disponibilidade da equipe, tempo de testes, entre outros. As simulações permitem a analise das políticas que resultam em maiores benefícios [Bar01].

O modelo criado por Hansen [Han96] demonstra como o volume de re-trabalho e os momentos em que as atividades de re-trabalho são inseridas no processo de desenvolvimento afetam os resultados do projeto. O modelo utiliza um processo de desenvolvimento em cascata, analisando o impacto de diversas estratégias de alocação de trabalho [Bar01].

Martinez-Garcia e Warboys [MGW98] apresentam uma técnica capaz de traduzir modelos de projetos de software para modelos de eventos discretos, usando a metodologia *RAD - Role Activity Diagram*. Ela captura as atividades que compõem o projeto e os papéis que os agentes devem assumir para realizarem suas atividades. A modelagem de eventos discretos complementa os modelos descritivos com informações acerca das atividades, assim como outros atributos durante a simulação. O modelo de eventos discretos permite a simulação, oferecendo ao engenheiro de processos dados sobre os gargalos, tempos de interação dos agentes, entre outras informações dinâmicas dos agentes [Bar01].

Silva [Ste00] apresenta um modelo de simulação de processos de software baseado em eventos para ser inserido em um ambiente de engenharia de processos de software, com o objetivo de otimizar o processo a ser simulado, procurando inconsistências que possam gerar problemas durante a sua execução, como o aumento de custos ou a queda na qualidade do produto. No entanto, a fim de representar melhor a dinâmica do sistema e o conhecimento, utiliza regras para representar os eventos.

3.5.1.2 Simulação Baseada em Regras A simulação baseada em regras é um subcaso da simulação por eventos discretos, pois mantém os mesmos conceitos, mudando apenas a sua forma de representação. Schacci [Sca99] apresenta um modelo de projetos de software descrito através de regras e apoiados por simulação discreta. Os estados do modelo são representados por fatos e as postulações por predicados. Baseado nos fatos as regras são disparadas, que geram novos fatos e que habilitam a execução de novas regras. Este tipo de simulação permite que o sistema armazene um histórico dos fatos acontecidos [Bar01]. Em particular, este tipo de técnica apresenta grande importância, pois é uma técnica muito comum dentro da inteligência artificial para a representação do conhecimento e do modelo do mundo de agentes inteligentes [Rus95]. Com sistemas baseados em regras é possível trabalhar com processos de inferência e fazer com que o simulador, ou partes dele, possa assumir comportamento não determinístico e inteligente.

Os modelos baseados em eventos discretos ou por regras apresentam uma particularidade muito importante, que é a capacidade de representação em níveis de detalhe maior, e a observação do comportamento individualizado de cada variável. A visão necessária a construção do modelo é uma visão mais próxima do dia a dia do engenheiro de processos e dos gerentes de projeto. No entanto, a visão sistêmica fica prejudicada, ou seja, a visão das correlações entre as variáveis não fica tão clara. Os modelos de Martinez-Garcia, Warboys e Schacci trazem como principal característica a possibilidade de descrição dos modelos em uma linguagem mais próxima ao domínio de gerência de projetos. Entretanto, os modelos referenciados não contemplam aspectos do processo de desenvolvimento de software que não se caracterizam em suas relações de alto nível, sendo necessário alterar sua representação de baixo nível, caso alguma outra característica seja necessária. [Bar01].

3.5.1.3 Modelos Baseados em Diagramas de Estados O diagrama de estados é uma ferramenta de modelagem que apresenta diversos estados que uma determinada entidade pode assumir ao longo de sua existência e as ações, chamadas transição, que provocam a mudança de um estado para outro. A ocorrência ou não de uma transição é controlada por uma condição de guarda. Os diagramas de estados podem ser decompostos hierarquicamente, na medida em que um sistema pode ser dividido em componentes, e esses componentes podem ter seus diagramas de estados independentes. O conjunto dos diagramas forma o diagrama de estados do sistema. Humphrey e Kellner [HK89] apresentam uma técnica para modelagem de processos de desenvolvimento de software baseada em diagrama de estados.

3.5.1.4 Modelos Baseados na Dinâmica de Sistemas A técnica de dinâmica de sistemas foi inicialmente aplicada nos campos de administração e engenharia, e posteriormente em análise de sistemas sociais, econômicos, agrícolas, físicos, químicos, biológicos e ecológicos [Mar97]. A aplicação engenharia de software ocorreu em meados dos anos 1980 com Abdel-Hamid e Madnick [AHM61]. Este modelo foi posteriormente estendido, existindo atualmente várias aplicações da dinâmica de sistemas ao ambiente de desenvolvimento de software [L+99, RW97]. Tendo inclusive sua aplicação sendo estendida à capacitação em gerenciamento de projetos [Mer96].

Barros apontou uma série de limitações da dinâmica de sistemas para a aplicação a gerência de projetos e criou uma extensão que permite a construção dos modelos, representação, detalhamento do conhecimento e de incertezas [DBW04]. Este conhecimento detalhado e a representação de incertezas são feitos através da criação de modelos complementares (cenários) que podem ser integrados ao modelo original. As incertezas também permitem maior flexibilidade e aplicação ao desenvolvimento de software, dado a natureza estocástica e humana de um processo de desenvolvimento de software.

As incertezas são representadas no modelo Barros, através da criação cenários que podem ser disparados ou não ao longo da simulação do modelo. Estes cenários são estruturas próprias, que irão alterar o as equações atuais do modelo, caso sua condição de ativação seja executada. Desta forma pode-se influenciar parte do comportamento do modelo baseado em condições específicas e deixar outra parte do modelo intacta. O comportamento das variáveis de um modelo pode ser representado por equações que mapeiam funções, ou quando não se tem exatamente as funções que se deseja implementar, pode-se criar uma tabela com os valores discretos e por interpolação chegar-se a função que a originou.

3.6 O USO DE JOGOS DE SIMULAÇÃO

Para Greenblat [Gre88] nos jogos de simulação, o ambiente e as atividades dos participantes têm características de jogos: jogadores possuem papeis a desempenhar, metas a serem cumpridas, atividades a realizar, restrições e recompensas (positivas e negativas) como resultado de suas ações e das ações de outros elementos no sistema . Greenblat [Gre88] conclui que jogos de simulação envolvem a realização de atividades de jogos em contextos simulados. Järvinen [Jär06] diz que nem toda simulação é um jogo, pois o jogo com suas regras, formam um particular forma de se estruturar uma simulação, sendo no entanto justificável discutir determinados tipos de simulação como jogos.

Jogos possuem características como interação, flexibilidade, competição, feedback vi-

sual, efeitos dramáticos, usabilidades, grau de fidelidade a realidades, níveis de dificuldade, etc. fazem com que sua utilização atue diretamente nos problemas dos ambientes de simulação, a saber: motivação e engajamento [Bar01].

Alguns estudos [vdH04], [Bar01] apresentam o desenvolvimento de jogos com esta finalidade, no entanto, observa-se que do ponto de vista de interação e do mercado de games para diversão há um grande diferença, haja vista jogos de simulação como: Microsoft Flight Simulator™, Sim City™ e The Simms™.

No entanto, apesar dos indícios de que o uso de jogos e ferramentas de simulação seja promissora no auxilio a capacitação de pessoas, pouco se tem de evidencia empírica de sua eficácia. Entre as possíveis causas para este fato, destacam-se [Ver03]:

- A eficácia da simulação depende da fidelidade do modelo ao mundo real
- Dificuldade de validação
- Dificuldade de se avaliar os resultados da simulação
- Inexistência de métodos e medições que avaliem alterações na curva de aprendizado

Na área de engenharia de software e de gerenciamento de projetos existem algumas iniciativas para a criação de jogos para treinamento, nesta seção apresentaremos algumas destas iniciativas.

3.6.1 SESAM

O projeto SESAM – *Software Engineering for Software by Animated Models* – foi inicialmente descrito em 1989 por Ludewig [Lud89] e posteriormente evoluído por Drappa [DDL95]. Os modelos do SESAM são estruturados em duas partes: Uma descrição estática que apresenta a definição e os tipos dos objetos envolvidos no processo de software e seus possíveis relacionamentos. Uma perspectiva dinâmica onde o comportamento é representado através de regras que especificam ações e efeitos que provocam mudanças no estado do projeto simulado.

- SESAM *Schema*: Representa a descrição estática. É utilizada uma notação de entidade-relacionamento estendida, na qual os objetos do mundo real são representados, bem como suas características relevantes e seus relacionamentos com outros objetos.

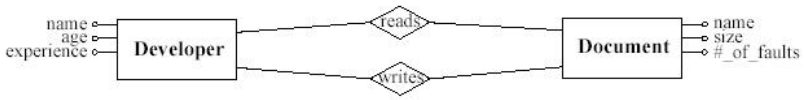

Figura 3.7 Exemplo de diagrama com a definição de *Schema* de SESAM [DDL95]

A Figura 3.7 apresenta um diagrama simples, onde são definidos dois objetos: *Developer* e *Document*. Também são apresentados seus respectivos atributos e seus relacionamentos.

- SESAM *Rules*: As regras representam o comportamento a ser seguido pela simulação. Essas regras podem ser disparadas em um determinado tempo da simulação, ou pela ocorrência de algum evento ou condição.

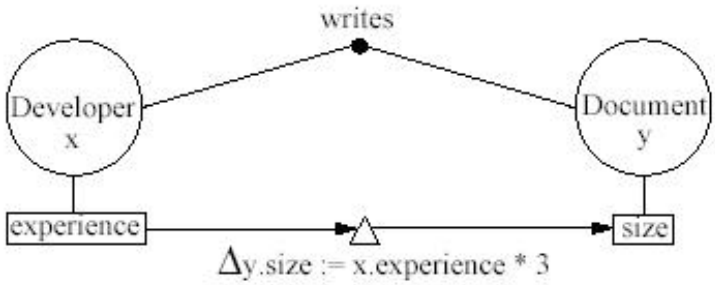

Figura 3.8 Exemplo de diagrama com a definição de regras do SESAM *Schema* [DDL95]

Na Figura 3.8 é possível observar a descrição de uma regra na qual se tem que a medida que a experiência do desenvolvedor aumenta a quantidade de documento que ele é capaz de escrever em um dado intervalo de tempo é multiplicada por 3. Desta forma tem-se uma regra de produtividade. A notação gráfica das regras do SESAM é baseada na combinação de dinâmica de sistemas e gramática de grafos (*graphs grammars*).

Para o inicio da simulação entretanto ainda é necessário prover a máquina de simulação com um conjunto inicial de estados, capaz de disparar as regras da simulação. Este conjunto de estados representa as instâncias dos projetos a serem simulados. A interação acontece através de uma interface bem simples e baseada em texto, conforme é possível ver na Figura 3.9.

Ao fim da simulação o SESAM permite a análise do comportamento das variáveis ao longo do tempo de forma a permitirão aluno refletir sobre o resultado de suas decisões.

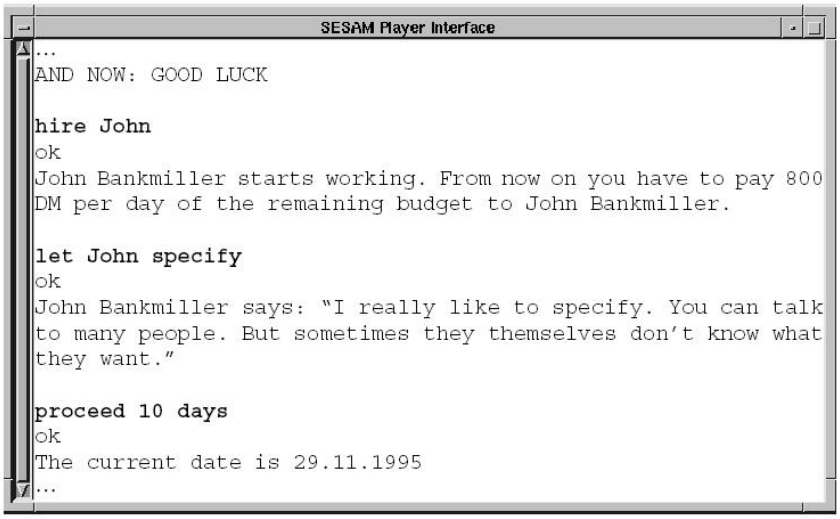

Figura 3.9 Exemplo de GUI do SESAM *Schema* [DDL95]

3.6.2 SIMSE

Emily OH [vdH04] descreve a ferramenta SimSE, que é um jogo criado para o ensino do processo de engenharia de software. É um jogo monousuário no qual o jogador assume o papel de um gerente de projetos. Entre as tarefas que poderá exercer incluem demitir e contratar desenvolvedores, delegar tarefas, monitorar o progresso e comprar ferramentas de trabalho para a equipe. A interação se dá através de interface gráfica, na qual o jogador visualiza o escritório de trabalho, com salas de reunião, mesas, computadores, etc. A Figura 3.10 apresenta a arquitetura do SimSE, na qual o *Model Builder* permite a criação do modelo que será simulado. Neste modelo encontrar-se-ão as definições dos tipos, atividades, regras, representação gráfica de cada elemento da simulação e um conjunto inicial de estados para o inicio da simulação. O *Model Builder* é uma aplicação gráfica e de fácil uso, que facilita a criação do modelo.

Com base nessas informações o *Generator* fará a interpretação do modelo e a geração de código dos componentes de gerenciamento de estados e de execução de regras, que estão inseridos na máquina de simulação genérica (*Simulation Environment*). A simulação irá ocorrer através da geração de *ticks* em intervalos de tempo constante. A cada *tick* do componente *Clock*, o componente de execução de regras (*Rule Execution*) verifica quais ações estão em execução no momento, através da invocação do componente de gerenciamento de estados (*State Mgmt*). As regras associadas a estas ações são executadas, e os efeitos desta execução são propagadas às entidades e ações que estão no *State Mgmt*. Terminada a alteração, o *Clock* sinaliza o componente de interface gráfica (*User Interface*) para que o

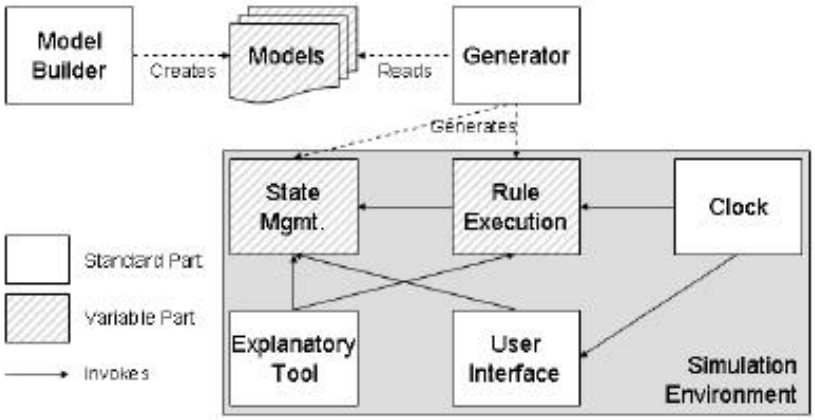

Figura 3.10 Arquitetura do SIMSE [vdH04]

mesmo se atualize e reflita o novo estado. Aliado a este fato a ferramenta SimSE tem um componente (*Explanatory Tool*), que permite ao jogador verificar a qualquer momento um registro de suas ações no tempo e com isso verificar a relação entre suas decisões e os efeitos das mesmas ao longo do tempo. O modelo a ser criado, com o auxilio do *Model Builder*, se baseia em alguns meta-objetos e em e em ações e regras. Os meta-objetos são definidos em Empregados, Artefatos, Ferramentas, Projeto e Cliente. Já as ações e regras trarão a implementação dos comportamentos do modelo. Na construção dos modelos específicos correspondente a metodologia a ser simulada, deve-se criar entidades que sejam sub-tipos dos meta-objetos citados.

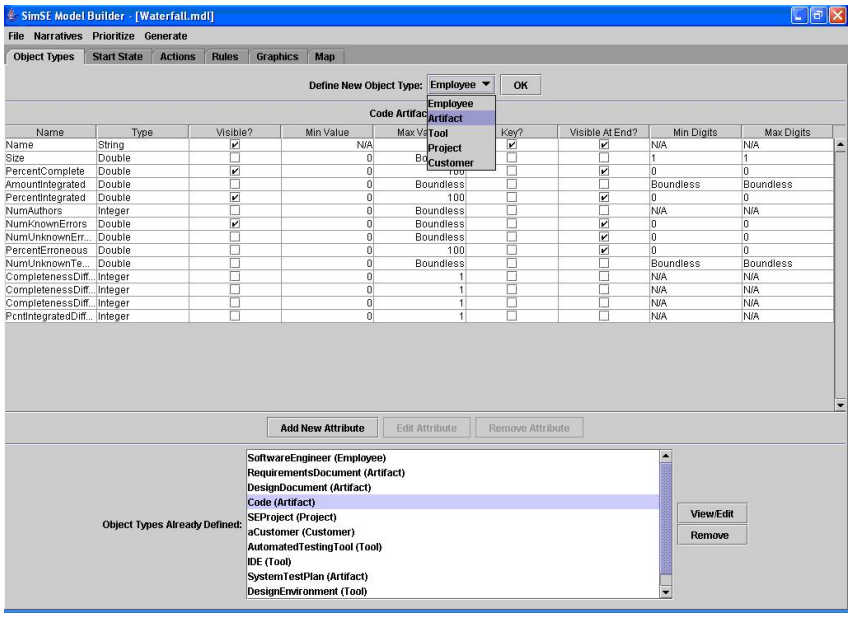

Figura 3.11 GUI do *Model Builder* [vdH04]

Na Figura 3.11 é possível ver a interface de criação dos tipos.

No cenário apresentado acima têm se a criação de um artefato de nome *Code* e atributos como: *Name, Size, PercentComplete, AmountIntegrated*, etc. Por esta ferramenta é possível também definir o conjunto de estados inicial, ou seja, os valores dos atributos dos objetos criados. É possível também criar as regras, ações e aparência gráfica de cada objeto criado. O SimSE fornece desta forma uma simplificação significativa quanto a complexidade da linguagem e forma para a criação de modelos, pois sua linguagem possui uma semântica de nível mais alto e próxima da realidade dos engenheiros de software. Trata-se de uma ferramenta OSS - Open Source Software - extensões em desenvolvimento, como por exemplo a implementação de um modelo baseado no RUP™.

3.6.3 The Incredible Manager

Dantas [DBW04] apresenta o jogo The Incredible Manager, fruto de seus trabalhos de pesquisa na COPPE-UFRJ e extensão dos trabalhos de Barros [Bar01]. A estrutura do jogo apresentada por Dantas pode ser vista na Figura 3.12.

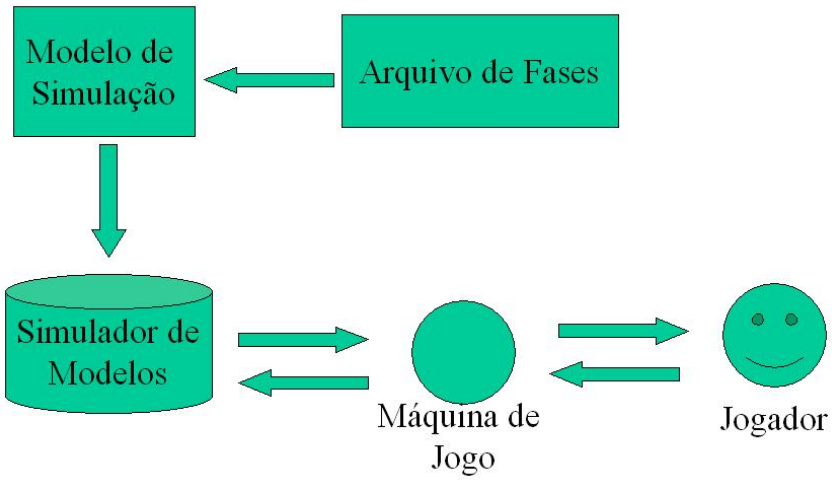

Figura 3.12 Estrutura do jogo The Incredible Manager [DBW04]

A criação do modelo de simulação obedece a um meta-modelo criado para a dinâmica de sistemas [Bar01], onde se tem a definição dos tipos (artefatos, papeis, atividades) e relações. Na Figura 3.13 tem-se um exemplo da criação de um modelo de domínio.

Entre os modelos para a simulação tem-se ainda o modelo de projeto, no qual são criadas as instâncias dos tipos definidos no modelo de domínio, e o modelo de cenário, no qual é possível criar situações especificas para a execução do projeto. O simulador de modelos é o elemento responsável por controlar os passos de simulação. Já a máquina de

```
MODEL ProjectModel
{
        CLASS Developer
        {
                PROPERTY experience 1;
        };
        CLASS Artifact
        {
                PROPERTY latent_errors 0;
        };
        CLASS Activity
        {
                PROPERTY duration 0;
        };
        MULTIRELATION Precedence Activity, Activity (NextActivities);
        MULTIRELATION Team Activity, Developer;
        MULTIRELATION Income Activity, Artifact;
        RELATION Outcome Activity, Artifact;
};
```

Figura 3.13 Exemplo de modelo de domínio do jogo The Incredible Manager [DBW04]

jogo é o elemento com que o jogador interage e recebe feedback visual dos resultados da simulação.

3.7 CONCLUSÃO

As técnicas de modelagem de sistemas permitem uma explicitação e formalização da dinâmica interna e do conhecimento desse sistema. As várias técnicas apresentam facilidades e limitações, dependendo do objetivo da simulação. A simulação por dinâmica de sistemas permite a representação e a visualização explícita das relações entre os conceitos em alto nível, permitindo assim uma melhor compreensão sobre o comportamento geral do sistema. No entanto apresenta dificuldades em representar informações mais detalhadas, na medida em que a criação de um modelo com muitas variáveis, ou muito detalhado, provoca a criação de muitas equações, o que pode tornar o modelo muito complexo e de difícil manutenção. Neste sentido os modelos baseados em regras apresentam certa vantagem, sem no entanto apresentar uma visão concreta e geral do comportamento sistêmico.

No contexto atual da engenharia de software os modelos de processo utilizados por equipes de desenvolvimento não utilizam uma linguagem formal, mas uma linguagem de modelagem explicita e gráfica que permite a comunicação dos diversos conceitos por membros da equipe e também a orientação sobre as atividades a serem executadas e em que momento. O SPEM, por exemplo, para ser simulado, deve passar por transformações e mapeamentos para as técnicas de simulação de modelos formais tradicionais.

No entanto, quando se fala em jogo, as técnicas de modelagem de processo e de simulação são apenas parte, dado que este deve criar um ambiente de imersão, interação e desafio para o aluno. Os jogos de estratégia contam em sua maioria com um cenário que permite ao jogador imergir na história. No caso dos jogos analisados é apresentado um pequeno contexto sobre o projeto a ser conduzido, a sua complexidade, as funcionalidades a serem desenvolvidas, os recursos disponíveis. Os cenários são criados dentro dos jogos de forma a permitir que habilidades sejam testadas e desenvolvidas pelo jogador, ou seja, caso o jogador não intervenha da forma como o modelo interno do jogo espera, o final do jogo pode não levar a uma situação de sucesso. Nos jogos de treinamento em gerenciamento de projetos normalmente o jogador é colocado no papel de gerente de projetos, que deve realizar atividades de gerenciamento como: alocação de recursos às atividades, seleção de equipe, alocação de horas-extras, acompanhamento de custos, etc.

.

CAPÍTULO 4

MODELOS DE GESTÃO E DE DESENVOLVIMENTO DE SOFTWARE

No Capítulo 2 apresentamos algumas características peculiares do processo de criação de software. Um software é um produto, que como qualquer outro, necessita de um processo produtivo. No entanto, seu processo produtivo é diferenciado, e mesmo se levarmos em consideração apenas a área de software, este pode sofrer muitas variações.

O processo de software é muito complexo e centrado nas decisões e julgamentos humanos. Não há processo ideal e muitas organizações desenvolveram suas próprias abordagens. Processos tem sido desenvolvidos para explorar a capacidades das pessoas de uma organização e das características específicas do sistema a ser desenvolvido. Por exemplo, para sistemas de missão crítica um processo bem estruturado é necessário, já para sistemas voltados para negócios, cujo contexto de negócio vária bastante, um processo ágil, flexível deverá ser mais adequado [Som04]. Sommerville também coloca que apesar disso algumas atividades são fundamentais aos processos de software:

- Especificação: As funcionalidades e restrições devem ser definidas.
- Projeto e implementação: O software para atender as especificações deve ser construído.
- Validação: O software deve ser validado para garantir que ele atende ao que o cliente quer.
- Evolução: O software precisa evoluir para atender as mudanças nas necessidades do cliente.

De uma forma geral alguns modelos de ciclo de vida são utilizados como estrutura para a criação de processos de software [Som04].

- Modelo cascata: Este modelo organiza os processos que compõem as atividades fundamentais e as organiza em fases como especificação de requisitos, projeto, implementação, testes e assim por diante.

- Desenvolvimento evolucionário: Esta abordagem intercala atividades de especificação, desenvolvimento e validação, de forma que um sistema inicial seja rapidamente desenvolvido apartir de especificações abstratas, que vai sendo refinada ao longo do tempo com base nas informações colhidas dos clientes.

- Engenharia de software baseada em componentes: Esta abordagem baseia-se na existência de um número significativo de componentes reutilizáveis, de forma que o processo de desenvolvimento foca na integração destes componentes.

Estes três modelos de processos, ou paradigmas, são largamente utilizados pelas organizações desenvolvedores de software. O seu uso não é mutuamente exclusivo, ao contrário, com freqüência são utilizados de forma conjunta, especialmente para sistemas de grande porte [Som04].

Neste capítulo apresentamos, na Seção 4.1, os paradigmas utilizados em engenharia de software para a construção de processos de software de uma forma mais detalhada. Na Seção 4.2, propomos uma instância de um processo de software. Na Seção 4.3 acrescentamos ao processo sugerido, os processos de gerenciamento de projetos. Na Seção 4.4 será apresentada uma estrutura do processo numa linguagem XML, a fim de que a mesma possa ser posteriormente simulada. Por fim, na Seção 4.5, é apresentado o modelo que será efetivamente simulado pelo jogo.

4.1 MODELOS DE PROCESSO DE SOFTWARE

Os processos de software são normalmente construídos baseados em modelos de processo, que também podem ser chamados de *frameworks*. Estes modelos trazem dentro de si paradigmas e modelos de ciclo de vida que ajudam a organizar o processo ao longo do tempo, definindo atividades, objetivos, responsáveis e uma sequência lógica de execução e agregação de valor.

4.1.1 Modelo Cascata

Este modelo de processo foi primeiramente introduzido em 1970 por Winston W. Royce [Roy87]. A Figura 4.1 ilustra o modelo [Som04].

Os principais estágios que modelo mapeia são:

- *Requirements analysis and definition*: Os serviços do sistema, restrições e objetivos são estabelecidos através de levantamentos juntos aos usuários. Eles são definidos em detalhes e servem de especificação para o sistema.

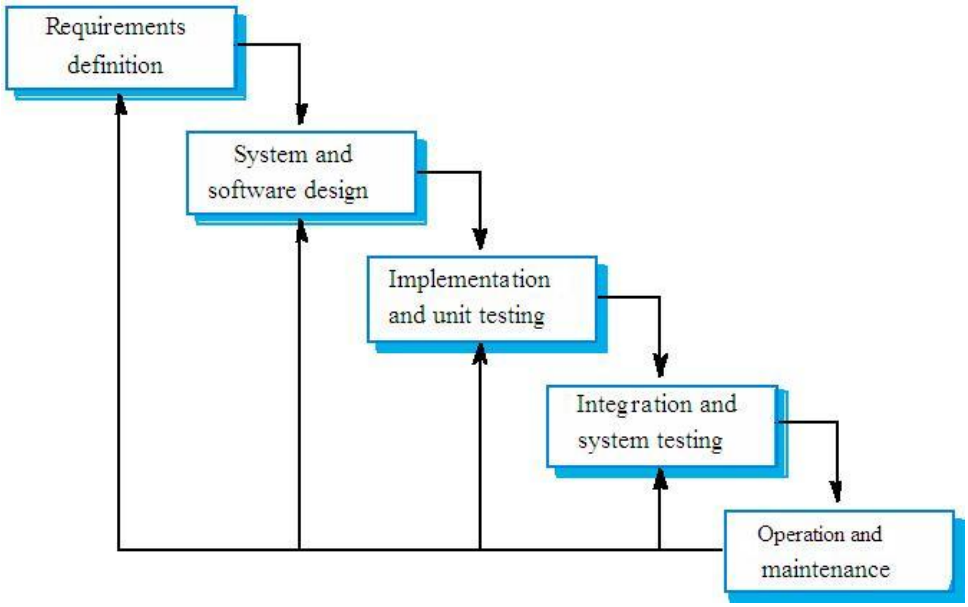

Figura 4.1 Modelo cascata [Som04]

- *System and software design*: O objetivo e estabelecer a arquitetura geral do sistema. O projeto envolve identificar e descrever as abstrações principais do sistema e suas relações.

- *Implementation and test unit*: Nesta etapa o projeto do sistema é concretizado em um conjunto de programas ou unidades de programa. Os testes de unidade envolvem a verificação de que cada parte atende a sua especificação.

- *Integration and system testing*: Os programas ou unidades de programa são integradas e testadas de forma conjunta, a fim de verificar a funcionalidade do sistema como um todo, e se estas conjuntamente atendes aos requisitos estabelecidos. Após os testes o sistema é entregue ao cliente.

- *Operation and maintenance*: Esta etapa normalmente é a mais prolongada do ciclo de vida, quando o sistema é colocado em uso. A manutenção envolve a correção de erros que porventura não tenham sido descobertos nas fases anteriores, a melhoria da implementação das unidades de programa e a incorporação de novos requisitos que eventualmente tenham sido descobertos.

Apesar do nome cascata sugerir uma separação total das fases, organizadas linearmente, na prática existe uma certa sobreposição, ou seja, atividades de uma fase subseqüente pode ser iniciada mesmo antes do fim da fase anterior. Outra importante questão

é a possibilidade de *feedback*, que é o retorno a atividades de fases anteriores. Por exemplo, em uma fase de implementação e testes pode-se concluir que há problemas de projeto, fazendo assim que atividades de projeto sejam retomadas. Fazendo com que hajam alguns ciclos de iteração. No entanto, as iterações são onerosas e envolvem bastante re-trabalho. Desta forma, após um pequeno número de iterações, é normal o congelamento de partes do produtos gerados até então, como por exemplo as especificações. Os problemas são deixados para posterior solução. Este congelamento prematuro nos requisitos pode significar que o sistema não irá fazer o que os usuários realmente necessitam, ou podem fazer com que o sistema possua problemas com a projeto da aplicação [Som04].

As vantagens do modelo cascata encontram-se na documentação que ele gera a cada fase e na capacidade dele se integrar a outros modelos de processo. Suas principais desvantagens estão na incapacidade de ser particionado em estágios diferentes e na dificuldade de responder a mudanças nos requisitos [Som04].

4.1.2 Desenvolvimento Evolucionário

O desenvolvimento evolucionário é baseado na idéia do desenvolvimento inicial de uma implementação, que é submetida aos comentários do usuário, e então submetida a refinamentos sucessivos até que um sistema adequado tenha sido desenvolvido. As atividades des especificação, desenvolvimento e validação são intercaladas ao invés de separadas, permitindo assim uma rápida integração entre as atividades [Som04]. A Figura 4.2 apresenta o desenvolvimento evolucionário.

A abordagem evolucionária é frequentemente mais efetiva do que o modelo em cascata quando trata-se do desenvolvimento de sistemas que endereçam necessidades imediatas de um cliente. A sua vantagem encontra-se no fato as especificações podem ser desenvolvidas incrementalmente e as versões vão sendo entregues ao usuário, que vão tendo um melhor entendimento do seu problema e assim refletir esse melhor entendimento na estrutura do sistema [Som04].

No entanto, do ponto de vista de engenharia e de gerenciamento esta abordagem apresenta dois grandes problemas:

- Processo não é visível: Os gerentes necessitam de entregas regulares para poderem medir o andamento do projeto. Se um sistema é desenvolvido rapidamente é contraprodutivo financeiramente a produção de documentação para cada versão do sistema
- Estrutura pobre: A mudanças contínuas tendem a corromper a estrutura do sistema. A inserção de mudanças vai se tornando mais difícil e cara.

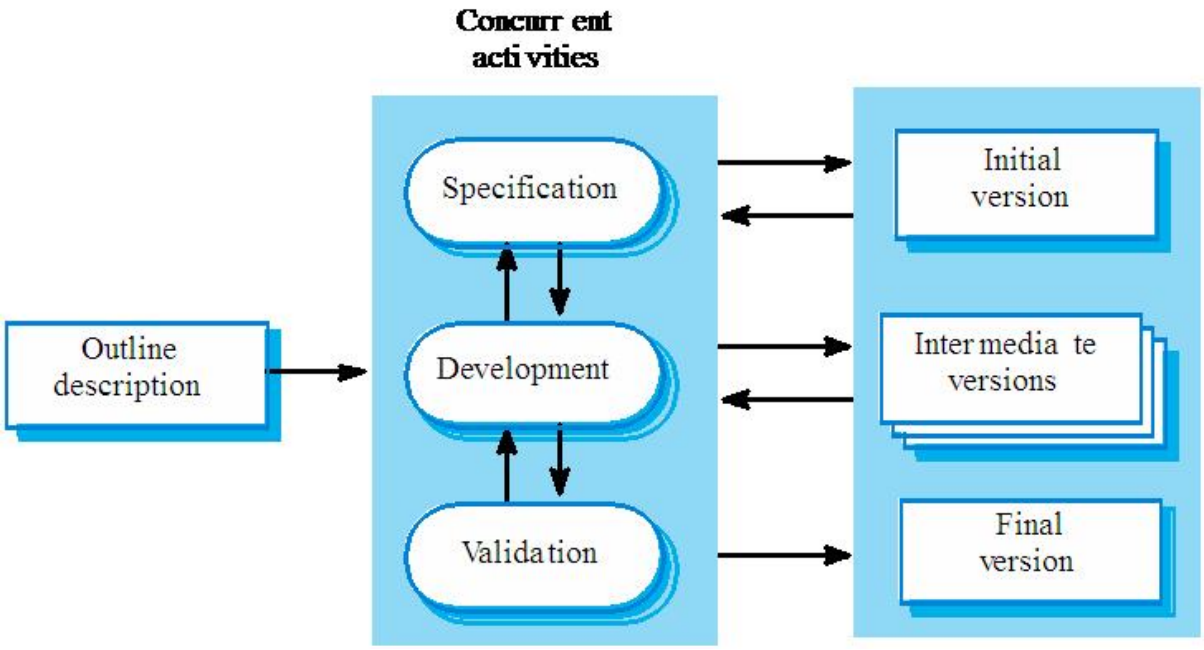

Figura 4.2 Desenvolvimento evolucionário [Som04]

Sommerville indica que para sistemas de pequeno e médio porte o modelo evolucionário apresenta melhor abordagem, pois os problemas deste modelo quando aplicados a grandes sistemas tornam-se agudos. Sistemas grandes farão que hajam várias equipes desenvolvendo funcionalidades diferentes, criando assim dificuldade para a integração das partes e na estabilização de uma arquitetura [Som04]. Para sistemas de grande porte, Sommerville sugere que se faca uma mistura dos modelos cascata e evolucionário, aproveitando assim o melhor de suas características.

4.1.3 Engenharia de Software Baseada em Componentes

Nos projetos de software existe algum tipo de reuso, seja formal através do aproveitamento de componentes previamente desenvolvidos e testados, ou informalmente, através do aproveitamento da experiência dos membros da equipe ou de projetos semelhantes.

Nos últimos anos uma nova abordagem em processo de software tem emergido, baseado justamente no conceito de reuso (CBSE – *Component-Based Software Engineering*). O alicerce desta abordagem está no uso de uma grande base de softwares reutilizáveis e de *frameworks* de integração [Som04].

A Figura 4.3 apresenta um processo genérico para o CBSE.

Os estágios do CBSE são:

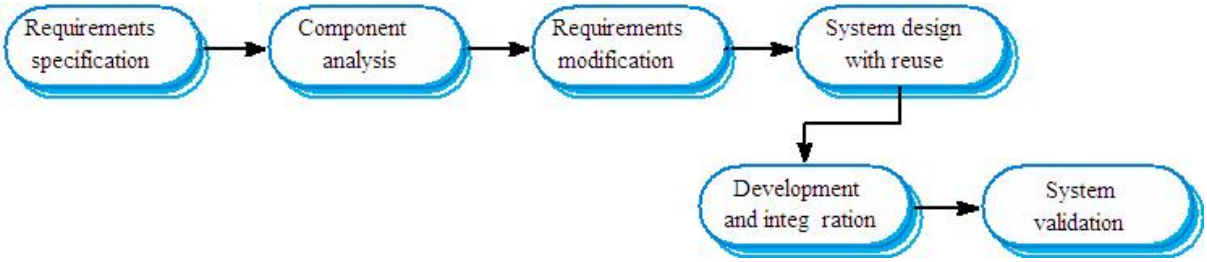

Figura 4.3 Engenharia de software baseada em componentes [Som04]

- *Component analysis*: Com base na especificação de requisitos uma busca pelos componentes que a implementam é efetuada. Usualmente os componentes só implementam partes da funcionalidade.

- *Requirements modification*: Durante este estágio, os requisitos são analisados com bases nas informações dos componentes encontrados. Os requisitos são então alterados para refletir as funcionalidades capazes de serem servidas pelos componentes. Quando a modificação não é possível, uma nova busca é efetuada.

- *System design with reuse*: A estrutura do sistema é desenvolvida ou reutilizada. Os projetistas levam em consideração os componentes que serão reutilizados. Para partes que não foi possível encontrar componentes elas deverão ser projetadas.

- *Development and integration*: As partes que não puderam ser atendidas por algum componente serão desenvolvidas e integrados juntamente com os demais componentes para compor o novo sistema. A integração dentro deste modelo é parte do desenvolvimento, ao invés de ser visto como uma atividade em separado.

O CBSE tem a vantagem de promover o reuso, fazendo com isso que o custo e o tempo de desenvolvimento sejam reduzidos, porém pode levar ao risco do sistema final não atender exatamente as necessidades reais dos usuários [Som04]. Una-se a isso que os padrões de integração dos componentes podem não ser compatíveis, fazendo assim com que haja muito esforço na integração dos componentes.

4.1.4 Processos Iterativos

O contexto do desenvolvimento de software é altamente regido por necessidades de mudanças, portanto ter um processo que saiba incorporar essas características é fundamental nos tempos atuais. Particularmente em sistemas de grande portes isto é praticamente

inevitável [Som04]. Apesar das possibilidades de combinação dos modelos cascata e evolucionário estes não tratam diretamente a questão do tratamento dessas mudanças. Dois modelos em especial foram explicitamente criados para suportar a execução de iterações:

- Entregas incrementais: Este modelo faz com que as especificações, projeto e implementação sejam quebradas em em partes que serão desenvolvidas a cada iteração.
- Desenvolvimento Espiral: Este modelo também procura dividir as especificações e partes e ir fazendo o seu desenvolvimento, mas o ciclo de vida segue o formato de uma espiral, no qual cada volta na espiral representa um ciclo de iteração.

A essência do desenvolvimento iterativo é que a especificação seja desenvolvida em conjunto com o sistema, fazendo assim com que o conjunto completo de especificações seja parte do processo de desenvolvimento e só seja conhecido de fato ao final do projeto.

O modelo de entregas incrementais possui algumas variantes atuais como, por exemplo, o XP – eXtreme Programming – desenvolvido por Beck em 2000 e que se baseia no desenvolvimento de pequenos incrementos de funcionalidade a cada versão, envolvimento constante do usuário no desenvolvimento, melhoria constante do código e programação em pares [Som04].

O modelo espiral pode ser visto na Figura 4.4. Ele foi originalmente proposto por Boehm em 1988 [Boe86]. Ao invés de apresentar o processo de software como uma sequência de atividades com alguma possibilidade de retrocesso, o processo é representado por uma espiral.

Cada volta na espiral representa uma fase do processo de desenvolvimento. Cada volta tem um objetivo, como por exemplo a volta mais interna preocupa-se com a viabilidade, já a proxima volta com a definição de requisitos, a próxima com o projeto e assim sucessivamente [Som04]. Cada volta na espiral é dividida em quatro quadrantes:

- *Objective setting*: Os objetivos específicos para a fase são definidos. Restrições ao processo e ao produto são identificadas e um plano de gerenciamento é detalhado. Riscos são identificados e estratégias alternativas para os riscos são montadas.
- *Risk Assessment and reduction*: Para cada risco identificado uma analise detalhada é feita e ações para reduzir os riscos são tomadas.
- *Development and validation*: Após a avaliação dos riscos, um modelo de desenvolvimento é escolhido. Por exemplo, se os riscos relacionados a interface são dominantes, então um modelo evolucionário com entrega de sucessivos protótipos pode ser mais apropriado.

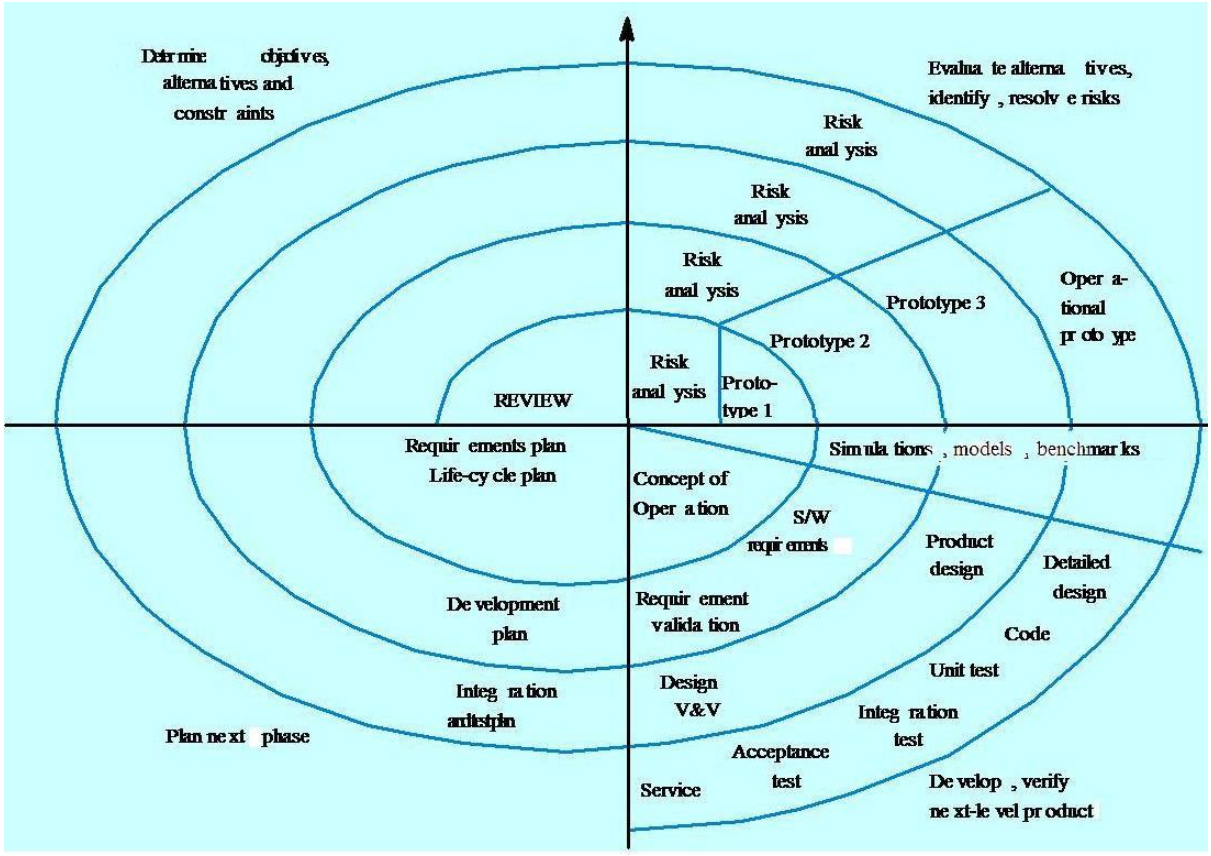

Figura 4.4 Modelo Espiral [Som04]

- *Planning*: O projeto é revisto a toma-se a decisão sobre a continuidade de mais uma volta na espiral, indicando uma nova fase. Caso seja essa a opção então os planos para a nova fase são feitos.

4.1.5 O Processo Unificado

O Processo Unificado é um modelo de processo que surgiu como um trabalho derivado da criação da UML (*Unified Modelling Language*), que procurou unificar as linguagens de modelagens de sistemas até então utilizadas. No entanto as principais linguagens utilizadas traziam consigo práticas processuais e uma visão de aplicação dentro de um ciclo de vida. Desta forma a criação do processo unificado, promoveu a unificação de metodologias até então utilizadas.

Os criadores do processo unificado criaram, a época da criação do processo, uma empresa chamada Rational™, com o objetivo de transformar o processo unificado em um produto mais acabado. A Rational™criou uma séria de ferramentas para dar suporte

as atividades do processo. Desta forma, foi criado o RUP™ – *Rational Unified Process* [Rat06] – que além de conter a metodologia do processo unificado, possui uma série de tutoriais de como implementar as atividades sugeridas utilizando o ferramental da Rational™. No contexto deste trabalho entretanto o as duas expressões são equivalentes, sendo o seu uso indistinto.

Ela trás elementos de todos os modelos de processos mais genéricos, suporta iterações e utiliza boas práticas de especificação e projeto [Som04].

O processo unificado reconhece que os modelos de processos tradicionais apresentam apenas uma visão do processo. Normalmente o RUP™ é descrito em três perspectivas:

- Uma perspectiva dinâmica que apresenta as fases do modelo.
- Uma perspectiva estática na qual apresenta as atividades do processo que deve ser executadas.
- Uma perspectiva prática na qual sugere boas práticas a serem utilizadas no processo.

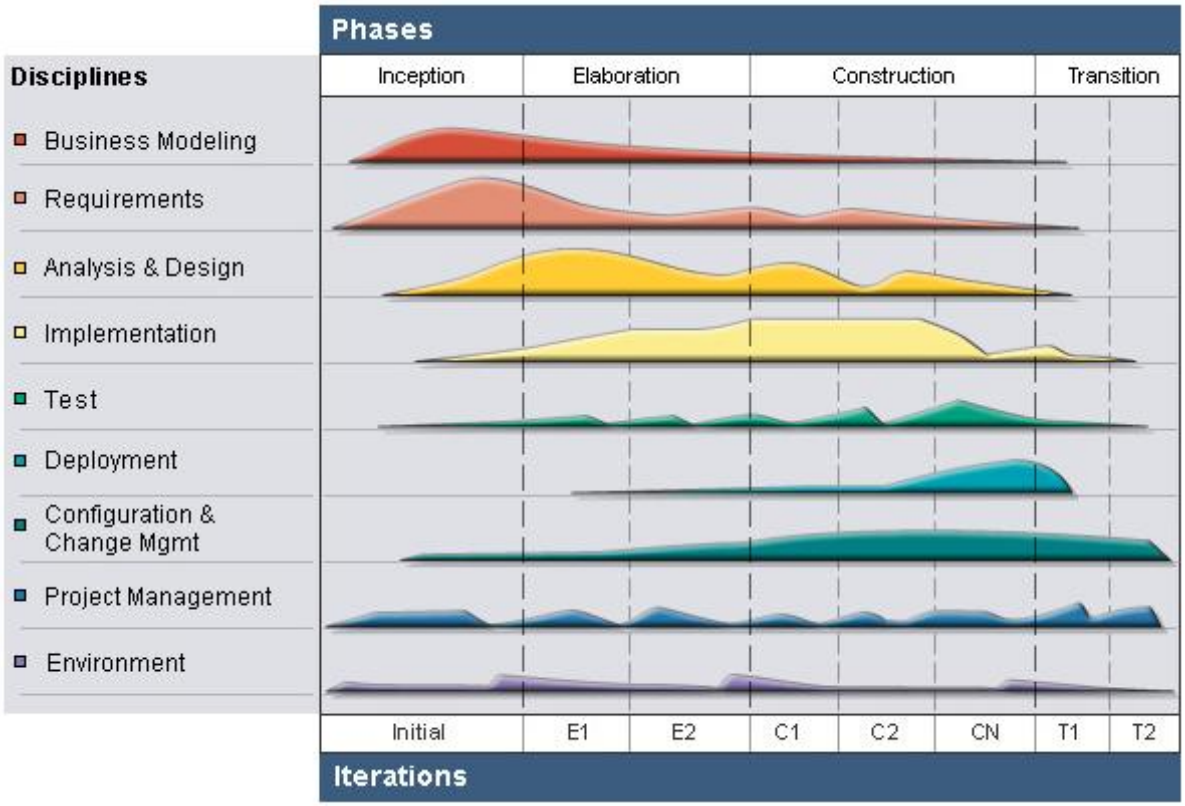

Figura 4.5 Processo Unificado [Rat06]

Podemos observar na Figura 4.5 a combinação das visões estática e dinâmica. Na visão dinâmica ela divide o ciclo de vida em quatro fases:

- *Inception*: Constitui na fase de iniciação cujo objetivo é ter uma visão clara do sistema que será desenvolvido, do ambiente de negócio no qual o sistema irá operar e nas pessoas que serão afetadas por sua operação.

- *Elaboration*: Os objetivos desta fase estão relacionados ao desenvolvimento do entendimento do domínio do problema, no estabelecimento de uma arquitetura estável para o sistema, no desenvolvimento do plano de projeto e na identificação de riscos, principalmente os técnicos.

- *Construction*: Esta fase está principalmente relacionada ao projeto, implementação e testes. Partes do sistema são desenvolvidas em paralelo e integradas durante esta fase. Ao fim desta fase deve haver uma versão executável e funcional do sistema e a documentação a ele necessária para implantar o sistema.

- *Transition*: A fase final está relacionada a transpor o sistema do ambiente de desenvolvimento para o ambiente do usuário, deixando disponível no ambiente real.

As iterações acontecem normalmente dentro de cada fase, mas também pode acontecer de se repetir todo o ciclo de vida, ou seja, havendo uma transição da *transition* para a *inception* [Som04].

A visão estática do RUP™ foca nas atividades a serem executadas durante o processo de desenvolvimento. Estas atividades encontram-se agrupadas em fluxos de atividades ou disciplinas, a saber:

- *Business Modelling*: Agrupa as atividades relacionadas a elicitação dos casos de uso de negócio, ou seja, os processos de negócio que cercam, interagem ou que serão implementados por funcionalidades do sistema.

- *Requirements*: Agrupa as atividades relacionadas a elicitação, análise, validação e rastreamento dos requisitos do sistema.

- *Analysis and Design*: Agrupa as atividades relacionadas a transformação dos requisitos em abstrações representadas nos modelos de analise, arquitetural, projeto, dados, objetos, componentes. Para representar esses modelos é feito uso da linguagem UML.

- *Implementation*: Engloba as atividades relacionadas a implementação dos componentes do sistema e a sua estruturação em subsistemas.

- *Testing*: Relaciona as atividades relacionadas aos testes do sistema que podem envolver a criação, implementação e execução dos casos de teste, bem como a avaliação dos resultados dos mesmos.

- *Deployment*: Envolve as atividades relacionadas a criação e empacotamento do produto final, a sua distribuição e disponibilização aos seus usuários em seus locais de trabalho.

- *Configuration and change management*: Trata-se de fluxo de suporte que permite o gerenciamento das mudanças do sistema e a rastreabilidade de quais versões estão implementadas quais funcionalidades ou solicitações.

- *Project management*: Fluxo de suporte que reúne as atividades que permitem o gerenciamento do desenvolvimento do sistema.

- *Enviroment*: Fluxo de suporte que se preocupa com a montagem do ambiente adequado de desenvolvimento necessário a equipe de desenvolvimento.

A Figura 4.5 é popularmente chamada de gráfico das baleias, pois apresenta em cada disciplina uma curva em formato que se assemelha ao dorso de uma baleia. Cada curva procura representar o esforço empreendido pelas atividades daquela determinada disciplina ao logo das fases.

A visão prática do RUP™ prega seis boas práticas fundamentais:

- Desenvolvimento iterativo

- Gerenciamento de requisitos

- Uso de arquitetura baseada em componentes

- Modelagem visual

- Verificação da qualidade de software

- Controle de mudanças

Apesar de aparentar um processo aplicável diretamente ao desenvolvimento de um projeto, o RUP™, se propõe a ser um *framework* para criação de processos. Desta forma há a necessidade que cada organização ao adotá-lo, tenha que criar uma metodologia específica própria. A este processo chamamos de instanciação do processo. Será possível assim a organização definir seu processo e colocar dentro dele peculiaridades próprias do seu negócio e de sua organização.

Quanto os projetos forem de fato acontecer, o gerente de projetos pode adotar o processo definido pela organização ou dependendo do contexto pode promover mudanças, ou seja, definir um processo especifico para o projeto. É importante lembrar que o projeto é algo dinâmico e que pode ter seu processo alterado mais a frente pelo gerente de projetos, afim de atender alguma necessidade especifica do projeto.

4.2 MODELOS DE DESENVOLVIMENTO DE SOFTWARE - UMA INSTÂNCIA DO RUP™ A SER SIMULADA

O processo unificado é um *framework* largamente utilizado na indústria de software atual para a criação de processos de desenvolvimento de software. Seu uso envolve boa parte das boas práticas de engenharia de software ensinadas nas universidades. Por este motivo a criação de um processo apartir do processo unificado com o objetivo de simulação tende a manter esta próxima da realidade das organizações. Por obedecer a premissas especificas este processo é chamado de instância. Este processo será materializado posteriormente através da criação de um projeto de desenvolvimento de software que o utilizará como processo produtivo. Este projeto é apresentado e detalhado no Capítulo 5.

O processo de customização ou instanciação de processos normalmente é feito com base em um *framework* que fornece um conjunto de atividades, relações de precedência entre atividades, responsáveis, produtos, etc. Normalmente é criado um subconjunto com atividades, responsáveis, produtos, entre outros, com base nos requisitos utilizados na customização.

Os princípios utilizados na criação desta instância de processo foram:

- Processo de software simples e de fácil assimilação por parte do jogador.
- Estrutura do processo também simples afim de que a simulação possa focar os elementos mais importantes.
- Processo que siga uma metodologia de desenvolvimento atual e amplamente utilizado.

O processo proposto, tal como o processo unificado, é dividido em quatro fases: Concepção, Elaboração, Construção e Transição.

As fases possuem atividades de natureza diferenciada e por este motivo possuem coloração diferente nos diagramas. Essas naturezas diferenciadas constituem as disciplinas ou fluxo de atividades do processo unificado. As atividades constantes dos diagramas foram obtidas com base nas atividades do processo unificado [Rat06].

Podemos ver na Figura 4.6 as atividades que compõem a fase de concepção, sendo basicamente composta por atividades da disciplina de requisitos. O objetivo aqui foi que os requisitos fossem bem trabalhados. Em projetos de pequeno e médio porte essa configuração é frequentemente utilizada.

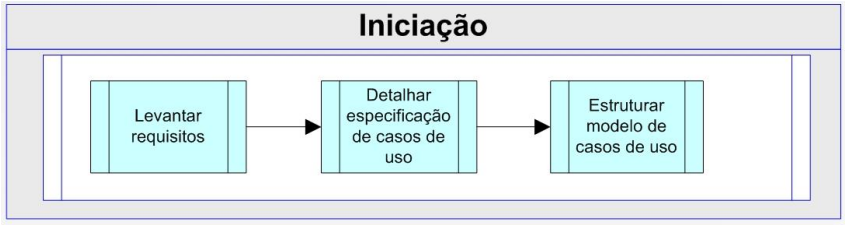

Figura 4.6 Concepção

A fase de elaboração pode ser vista na Figura 4.7, sendo composta por atividades do fluxo de analise e projeto e de implementação. A inclusão de atividades do fluxo de implementação em uma fase cujo objetivo primário não é a geração de código, mas a estabilização de uma arquitetura, se dá pelo fato de poder adiantar como os componentes serão estruturados e integrados, permitindo assim que na próxima fase haja uma fluidez maior na produção de código.

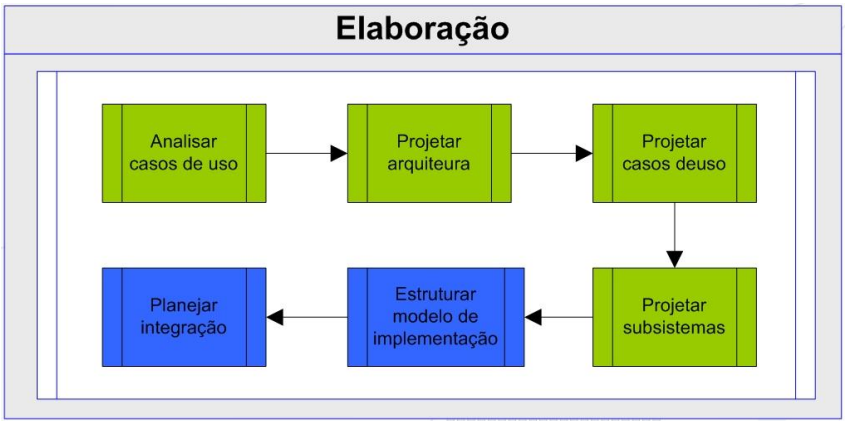

Figura 4.7 Elaboração

A fase de construção pode ser vista na Figura 4.8, sendo composta por atividades do

fluxo de implementação e do fluxo de testes. O objetivo aqui é que os componentes implementados já serem testados não só individualmente, atribuição das atividades de teste unitário, mas também sistemicamente de acordo com os parâmetros de testes definidos no plano de testes.

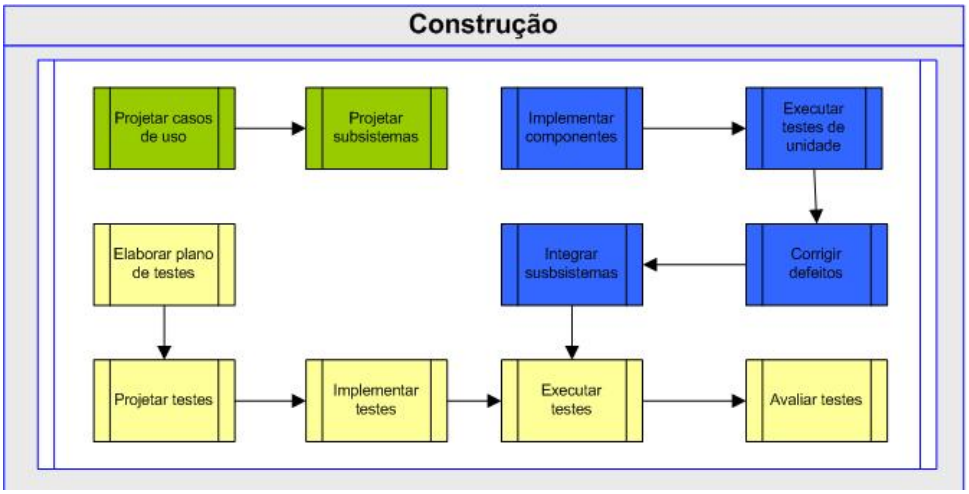

Figura 4.8 Construção

A fase de transição pode ser vista na Figura 4.9, sendo composta por atividades do fluxo de testes e de implantação.

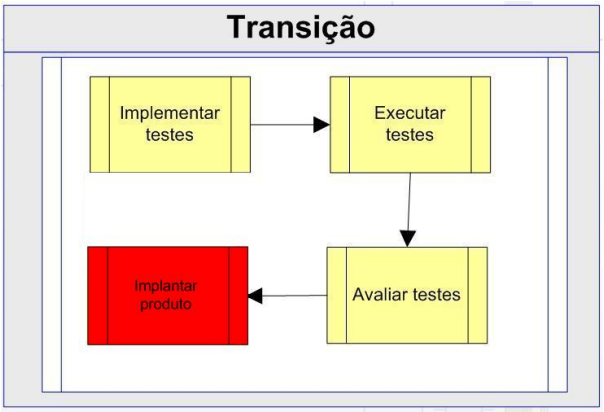

Figura 4.9 Transição

É de fundamental importância aqui compreender que este processo de software não é genérico a ponto de ser aplicável em qualquer contexto, mas a pequenos projetos, podendo ser necessárias algumas modificações, principalmente no que tange ao ciclo de re-alimentação entre a correção de erros e a implementação e a possível existência de implementação na fase de transição.

Na figura Figura 4.10 podemos ver uma legenda de associação das atividades aos fluxos de atividades.

Figura 4.10 Legenda de cores para os fluxos de atividades

4.3 A INTEGRAÇÃO DO MODELO AOS PROCESSOS DO PMBOK™

O fluxo de atividades de gerenciamento de projetos é um fluxo de suporte, ou seja, suas atividades geram artefatos que são voltados a condução de um processo de desenvolvimento de software. Suas atividades requerem esforços constantes ao longo de todo o ciclo de vida. O processo unificado define neste fluxo um conjunto de atividades, seu sequenciamento e suas práticas de aplicação.

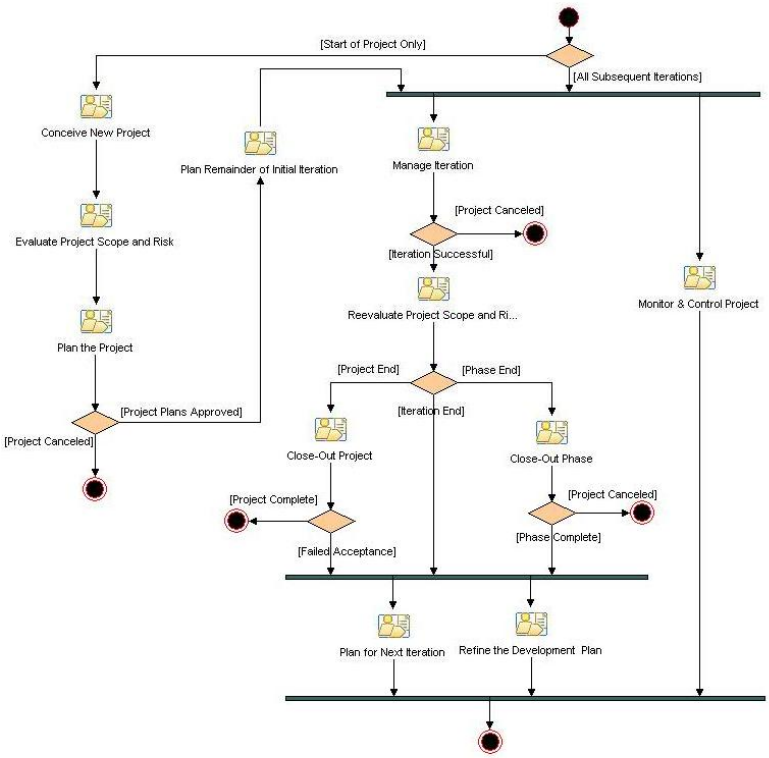

Figura 4.11 Fluxo de atividades de gerenciamento de projetos do RUP™ [Rat06]

A Figura 4.11 apresenta o fluxo de atividades de gerenciamento de projetos do RUP™

[Rat06].

Estas atividades focam nos aspectos mais importantes para o desenvolvimento iterativo, como gerenciamento de riscos, planejamento de um projeto iterativo e o monitoramento e a mensuração de um projeto iterativo. No entanto, aspectos como gerenciamento de pessoas, gerenciamento de custos e gerenciamento de sub-contratação não são cobertos pelo RUP™ [Rat06].

Desta forma faz-se necessário a um projeto de software, que utilize uma metodologia baseada no processo unificado, não só a observação dos aspectos definidos no fluxo de atividades modelo, mas também a inserção de atividades que estejam definidas no *framework* do PMBOK™ *Guide* [Ins04].

Ne entanto este mapeamento necessita ser feito com cuidado, pois não há uma correspondência direta entre as fases do processo unificado e os grupos de processo do PMBOK™. Para qualquer mapeamento que se deseja fazer é antes necessário fazer um resgate dos objetivos apresentador na Seção 4.1.5. As fases do processo unificado já fazem parte do processo produtivo de software, não havendo portanto fases preliminares no ciclo de vida de produto do software. A Concepção objetiva ter uma visão clara do produto a ser desenvolvido; A Elaboração objetiva o desenvolvimento de uma arquitetura estável; A Construção objetiva a criação de componentes executáveis e a Transição a disponibilização do produto ao cliente. No entanto para que o processo de software ocorra, o conceito do projeto já existe, tendo inclusive recursos materiais e humanos alocados. Portanto, quando do início da fase de concepção o plano de projeto, artefato que, segundo o PMBOK™, contém o planejamento de todo o projeto, já existe. Portanto a etapa de Iniciação (vide Figura 2.4) ocorrem antes do ciclo produtivo de software. As etapas de Planejamento, Execução e Controle ocorrem durante todas as fases do ciclo produtivo, uma vez que estas formam um ciclo de execução no formato PDCA. De acordo com as fases do processo unificado o início de cada fase é o início de um ciclo PDCA. As fases do processo unificado podem ser divididas em iterações. Estas iterações também representam uma volta no ciclo PDCA. Ou seja, há vários ciclos de planejamento, execução e controle ocorrendo de forma paralela, um mais amplo, tendo como escopo todo o projeto; Outro menos amplo, tendo como escopo as fases; e outro mais interno tendo como escopo as iterações. A etapa de fechamento do projeto, conforme visto no PMBOK™ *Guide*, tem por objetivo o fechamento formal do projeto, havendo então como premissa a finalização das fases do processo de software.

Para que o processo seja materializado e que os personagens do jogo tenham um projeto para ser desenvolvido, foi criado um projeto, que será melhor explorado no Capítulo

5. Trata-se de um projeto simples, de curta duração e para ser desenvolvido por uma equipe pequena. Algumas premissas retiradas deste contexto foram utilizadas afim de selecionar as atividades que comporão o processo de gerenciamento de projetos do processo proposto:

- Projeto de duração de 4 a 6 meses;
- Ênfase nos processos e eventos de gestão de pessoas; e
- Equipe de desenvolvimento pequena

Procurou-se focar nos aspectos mais relacionados a fase de construção, onde as atividades relacionadas a gestão de pessoas são de fato executadas, de forma que as atividades relacionadas a iniciação e fechamento, como a criação do *Project Charter*, não fazem parte do processo.

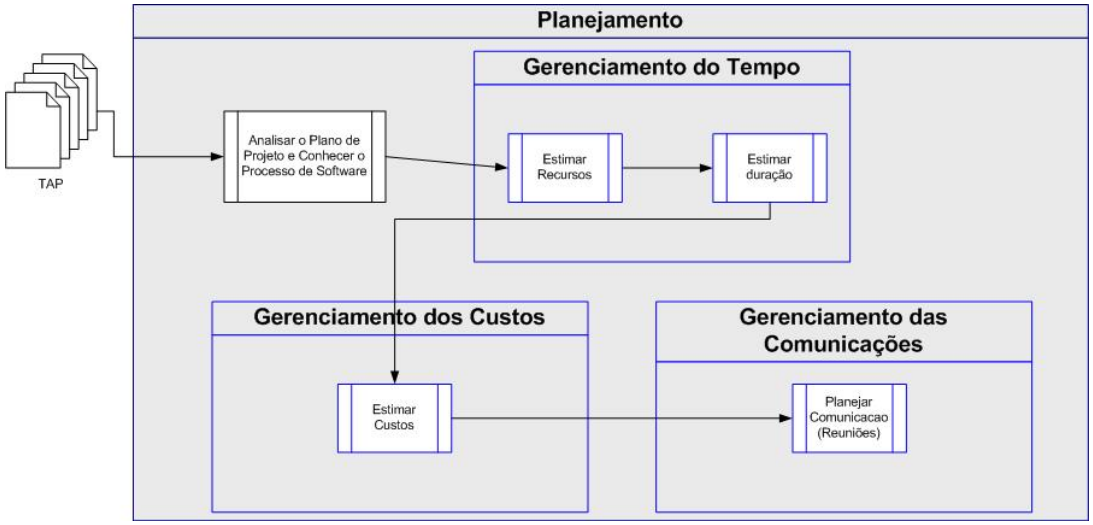

Figura 4.12 Grupo de processo de Planejamento

A Figura 4.12 apresenta as atividades a serem executadas pelo jogador e que fazem parte da fase de planejamento. Estas atividades não serão simuladas ou executadas por agentes, mas serão de responsabilidade do jogador no papel de gerente de projetos. Basicamente as atividades que serão executadas com relação ao planejamento serão: Estimar recursos, Estimar duração, Estimar custos e Planejar comunicações.

Ao final dessas atividades ter-se-á chegado ao fim da fase de planejamento e a execução do mesmo poderá ser iniciada, ou seja, estas atividades não consomem recursos do projeto.

A fase seguinte é a de execução, que pode ser observada na Figura 4.13. Em função das premissas adotadas e do foco a ser dado no jogo, as atividades e ações que compõem

o processo estão relacionadas a gestão de pessoas (recursos-humanos), comunicação e integração.

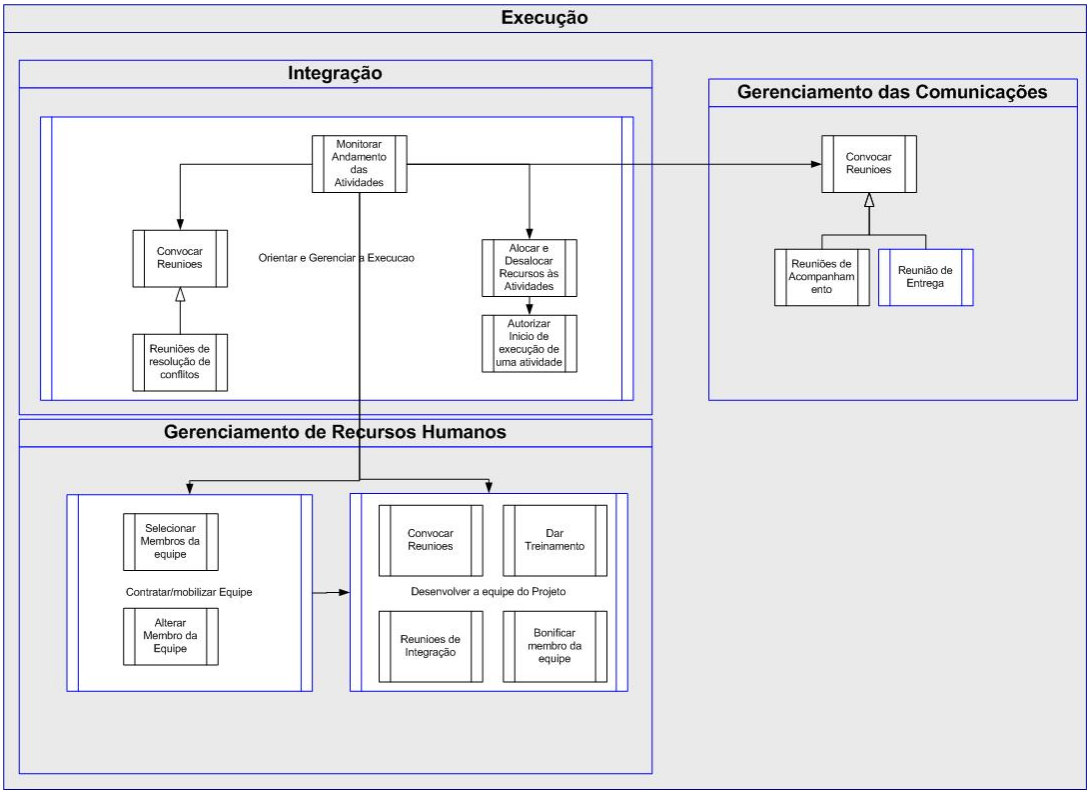

Figura 4.13 Grupo de processo de Execução

As atividades de monitoramento e controle estão colocadas no processo, porém na maioria das atividades a sua execução não representará feedback para o jogo ou para a equipe, será apenas a observação das informações fornecidas pelo jogo, através de seus mecanismos de *feedback*. Essas atividades podem ser vistas na Figura 4.14.

Nas figuras apresentadas é possível observar que alguns processos constantes das figuras possuem dentro de si atividades. Isto acontece, pois neste caso existem várias atividades a serem executadas no jogo, que segundo o PMBOK™ *Guide* pertencem a apenas um processo. Como, por exemplo, na Figura 4.14 a atividade "Monitorar Nível de Satisfação do Cliente", que no jogo é representada pela observação de um indicador de satisfação, no PMBOK *Guide* é uma atribuição de um processo chamado "Gerenciar as Partes Interessadas"e que por sua vez faz parte da área do conhecimento de "Gerenciamento das Comunicações".

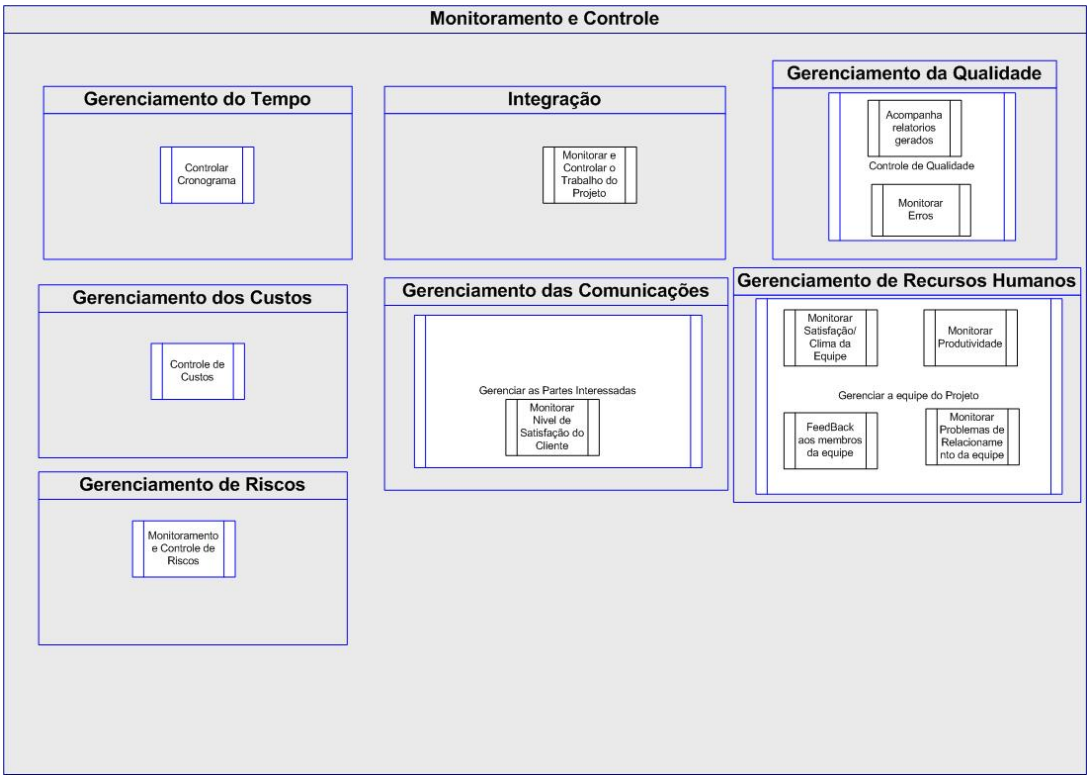

Figura 4.14 Grupo de processo de monitoramento e controle

4.4 REPRESENTAÇÃO DO MODELO INSTANCIADO

A fim de descrever o processo a ser simulado, pesquisamos modelos de descrição de processos. Apesar de existirem modelos gráficos para representar processos de software, como o utilizado pelo RUP™ para definição de processo, ou o SPEM [Gro05] que utiliza a UML para a representação gráfica do processo, há a necessidade de um mecanismos de descrição e armazenamento do processo, a fim de que o mesmo possa ser recuperado, validado, simulado, otimizado e publicado.

Falbo [Fal98] apresenta uma ontologia para descrição de processos de software. Esta ontologia apresenta relações de precedência entre as atividades, de decomposição de atividades e de classificação (execução, gerenciamento e controle da qualidade). A ontologia de Falbo representa os artefatos criados por estas atividades e os recursos, sejam materiais ou humanos. O elemento central da ontologia é a atividade.

A ontologia apresentada por Falbo permite então a estruturação dos conceitos pertinentes a um processo de software.

O ImpPros [OV06] define uma estrutura de representação em arquivo XML (*eXtensible Markup Language*) no qual ele utiliza a ontologia definida por Falbo para ajudar a

estruturar um arquivo que fosse capaz de armazenar um processo de software. ImpPros tem o objetivo de criar um ambiente para apoio a implementação de um processo de software em uma organização, da seguinte forma:

- Especificar um meta-modelo de processo de software a fim de definir uma terminologia única entre os vários modelos de qualidade de processo de software existentes, para uso do ambiente em seus serviços providos;
- Apoiar a definição de um processo de software para organização;
- Permitir a modelagem e instanciação deste processo;
- Permitir a simulação do processo a partir das características instanciadas para um projeto específico;
- Dar apoio à execução do processo de software tomando como base uma máquina de inferência;
- Possibilitar a avaliação do processo de software;
- Apoiar a melhoria contínua do processo de software e o reuso através da re-alimentação e coleta das experiências aprendidas.

A estrutura definida por Oliveira só permite definir o processo, e não o projeto. No entanto, é no projeto que o processo irá materializar-se, que as atividades especificadas terão de fato contexto, objetivo, responsáveis, estimativas, custo, tempo, ou seja, de uma certa forma podemos dizer que estas de fato ocuparão "lugar no espaço". Assim sendo uma série de atributos ficam faltando a um projeto. Desta forma, é necessário criar extensões a estrutura definida pelo ImpPros, para que a mesma possa representar um projeto e que possa ser utilizada dentro do jogo. Oliveira define então uma estrutura de um arquivo em formato XML, através de um DTD, no qual um processo de software pode ser especificado usando as *tags* por ele definidas, que pode ser observado no B.

A definição estrutural de arquivos XML utilizando o formato DTD (*Document Type Definition*) é considerada obsoleta, devido suas limitações no que tange a capacidade de descrição do conteúdo e conseqüentemente das possibilidades de validação da estrutura de um documento que siga o formato especificado. Atualmente o formato utilizado para descrição de estruturas de arquivo XML é o XML *Schema*. Por este motivo, ao invés de fazer a extensão na própria estrutura do DTD, decidimos por criar um XML *Schema*

para então acrescentar as extensões necessárias aos objetivos desta pesquisa, que pode ser observada no Apêndice C.

Neste *Schema* temos a estrutura para a definição do processo de software que será simulado no *Virtual Team*. Podemos observar que a estrutura sugerida pela ontologia do Falbo e mantida no ImpPros, no entanto, algumas informações a mais são requeridas.

Temos então a definição das *tags* "processoinstanciado", "processo", "modelociclovida", "fase", "atividade", todas seguindo a mesma estrutura definida no DTD do ImpPros.

```
<xs:element name="processo">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="artefato" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="tech-skill" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="role" minOccurs="0" maxOccurs="unbounded"/>
            <!-- adicionados os elementos acma para relacionamento com as atividades -->
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="atividade"/>
        </xs:sequence>
        <xs:attribute name="nome" use="required"/>
        <xs:attribute name="descricao" use="required"/>
        <xs:attribute name="origem" use="required"/>
        <xs:attribute name="infoimpl" use="required"/>
        <xs:attribute name="restricao" use="required"/>
        <xs:attribute name="situacao" use="required"/>
    </xs:complexType>
</xs:element>
```

Figura 4.15 *Tag* Processo

A Figura 4.15 apresenta a estrutura da *tag* "processo". Nesta estrutura além de replicar todos os ítens definidos por Oliveira [OV06], acrescentamos ao processo a existência de outros três atributos: Artefato, Tech-Skill e Role.

A Figura 4.16 apresenta a *tag* de definição de um artefato que possui os atributos: Nome, que trás a denominação do artefato; Size, que trás o tamanho do artefato; e Complexity, que trás uma classificação de complexidade do artefato.

As atividades definidas possuem artefatos de entrada e saída, no entanto, na estrutura proposta por este trabalho ela é definida como um tipo a parte. A conseqüência prática disto é que os artefatos deverão ser definidos em *tags* em separado fora da estrutura hierárquica da atividade e apenas referenciada dentro desta. Isto caracteriza o formalmente o artefato como parte do processo, além dar uma melhor visibilidade ao caso em que várias atividades ajudam a produzir o mesmo artefato, de forma que não haverá

```
<!-- adicionado elemento artefato para relacionamento com as atividades-->
<xs:element name="artefato">
    <xs:annotation>
        <xs:documentation xml:lang="pt-br"> Elemento de domínio representando um artefato. Artefatos
            podem ser usados como entrada ou saída para atividades. </xs:documentation>
        <xs:documentation xml:lang="pt-br"> Apresenta dois atributos numéricos para tamanho e
            complexidade. </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attributeGroup ref="id_name"/>
        <!-- TODO remover required e definir valores default -->
        <xs:attribute name="size" type="xs:byte" use="required"/>
        <xs:attribute name="complexity" type="xs:byte" use="required"/>
    </xs:complexType>
</xs:element>
```

Figura 4.16 *Tag* Artefato

necessidade de se redefinir a cada atividade aquele mesmo artefato, caso que aconteceria caso esta estrutura não seja definida como um tipo a parte.

```
<xs:element name="tech-skill">
    <xs:annotation>
        <xs:documentation xml:lang="pt-br">Elemento de domínio representando uma capacitação
            técnica.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attributeGroup ref="id_name"/>
    </xs:complexType>
</xs:element>
```

Figura 4.17 *Tag* Tech-Skill

A *tag Tech-Skill*, apresentada na Figura 4.17, tem o objetivo definir quais são as habilidades requeridas pelo processo. Elas são definidas como tipos a parte para melhor caracterizar estas características e para auxiliar a definição dos diversos papéis que comporão o processo. No caso o seu único atributo é o nome do skill técnico.

A *tag Role*, apresentada na Figura 4.18, tem o objetivo de definir os papéis que farão parte do processo. Seus atributos são além do nome do papel uma lista de *tech-skills*, que compõem o conjunto de habilidades requeridos para o papel.

Outra extensão importante diz respeito a definição do recurso humano responsável pela execução de uma determinada atividade. Ela pode ser observada na Figura 4.19 em destaque no retângulo vermelho. Para cada atividade temos um responsável e associamos a este um perfil "ideal"em termos de habilidades necessárias à plena execução desta.

```
<xs:element name="role">
    <xs:annotation>
        <xs:documentation xml:lang="pt-br">Elemento de domínio representando um papel. Apresenta uma
            sequência de tech-skills aplicáveis.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="tech-skill-ref" type="xs:IDREF" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="id_name"/>
    </xs:complexType>
</xs:element>
```

Figura 4.18 *Tag* Role

Quando em execução, o VIRTUAL TEAM verifica este perfil "ideal"e o compara com as habilidades do recurso de fato alocado a execução da atividade, promovendo mudanças em variáveis como produtividade e geração de erros. A *tag* recurso-humano pode ser vista na Figura 4.20.

O IMPPROS atua nas várias fazes de definição e implantação de processo de software em uma organização, fazendo inclusive a simulação do ponto de vista de otimização de processos. No entanto, por mais que o processo esteja otimizado do ponto de vista estrutural, ao ser executado na realidade, o mesmo será conduzido por pessoas reais, e várias situações podem surgir, fazendo com que o comportamento do processo não seja conforme o esperado. Uma das questões diz respeito ao conhecimento e experiência da equipe com o processo, ou seja, a equipe necessita ser capacitada no processo, para que uma maior produtividade seja atingida.

Utilizar o mesmo meta-modelo para definição de processo, apenas estendendo e acrescentando algumas informações pertinentes ao contexto do jogo, permitirá que o jogo de simulação seja uma ferramenta a ser utilizada na capacitação dos membros das equipes. No entanto, apesar de facilitar, esta não é a única dependência para que isto aconteça, pois o jogo objeto deste trabalho, objetiva a capacitação do gerente de projetos, portanto o jogador está neste papel. Seria então necessário que vários cenários novos fossem implementados no jogo e o modelo de domínio fosse estendido para que um novo papel pudesse estar na primeira pessoa do jogo.

A representação completa do processo apresentado na Seção 4.2 pode ser vista no Apêndice D.

No Capítulo 5 teremos o detalhamento da estrutura do jogo e a explicação de um arquivo complementar a este, que é o arquivo de definição de fases.

```
<xs:element name="atividade">
    <xs:complexType>
        <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" ref="superatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="subatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="preatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="posatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="artefatoentrada"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="artefatosaida"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="procedimento"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursohardware"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursosoftware"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursohumano"/>
        </xs:sequence>
        <xs:attributeGroup ref="id_name"/>
        <xs:attribute name="descricao"/>
        <xs:attribute name="tipo"/>
        <xs:attribute name="origem"/>
        <xs:attribute name="restricao"/>
    </xs:complexType>
</xs:element>
```

Figura 4.19 *Tag* Atividade

4.5 O MODELO DE SIMULAÇÃO DO JOGO

Para construir o modelo de simulação será utilizado o meta-modelo definido por Barros [Bar01], e utilizado por Dantas [DBW04] e Veronese [Ver03] em seus estudos e na criação do *The Incredible Manager* [DBW04].

O meta-modelo definido por Barros apresenta a vantagem de aproximar a linguagem de definição do modelo da área de domínio, no caso o gerenciamento de projetos. Apesar de já haver modelos nesta área de domínio descrito nesta meta-linguagem, os exemplos disponíveis não mapeavam diretamente as variáveis desejadas no projeto do jogo *Virtual Team*. Desta forma foi criada um novo modelo.

O jogo foi planejado para dar um enfoque maior na gestão de pessoas. No entanto as características relacionadas ao comportamento dos membros da equipe e suas correlações não foram mapeadas no modelo de simulação que será simulado pela Dinâmica de Sistemas. Neste jogo cada membro da equipe foi mapeado como um ator sintético, que traria dentro de si características humanas como personalidade, motivação, estresse, entre outros e que naturalmente podem ter impacto em variáveis mais diretas da gerência de projetos como a produtividade individual e a taxa de geração de erros.

No entanto, não é objeto deste trabalho explorar os aspectos dos atores sintéticos

```
<xs:element name="recursohumano">
    <xs:annotation>
        <xs:documentation xml:lang="pt-br"> Elemento de domínio representando um recurso humano.
            Extendido para inclusão de skills. </xs:documentation>
        <xs:documentation xml:lang="pt-br"> Apresenta uma sequência de tech-skill-levels, cada um
            contendo uma referência ao skill em questão e um valor numérico para a habilidade.
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="tech-skill-level" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="tech-skill-ref" type="xs:IDREF" use="required"/>
                    <xs:attribute name="level" type="xs:byte" default="3"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="role-ref" type="xs:IDREF"/>
    </xs:complexType>
</xs:element>
```

Figura 4.20 *Tag* Recurso-Humano

e sua influência em variáveis como produtividade. É objeto sim de contribuição a ser explorada no Capitulo 5, a proposta de uma arquitetura para o jogo que permita o uso do ambiente de atores sintéticos integrados com o ambiente da dinâmica de sistemas.

O modelo de domínio proposto completo encontra-se disponível no Apêndice A.

São definidas três estruturas principais no modelo: *Project, Activity e HumanResource.*

```
CLASS Project {
    STOCK ActualTime 0;
    PROC getActualTime ActualTime;
    RATE (ActualTime) RTActualTime DT;
    STOCK PlannedValue 0;
    STOCK EarnedValue 0;
    STOCK ActualCost 0;
};
```

O projeto possui os estoques acumuladores de custo e tempo do projeto como um todo. Neste projeto é implementada a análise de valor agregado, desta forma os atributos de Valor Planejado (*Planned Value*), Valor Agregado (*Earned Value*) e Custo Real (*Actual Cost*), são representados na forma de estoque na estrutura do projeto. Os projetos encontram-se relacionados as atividades, porém a direcionalidade da ação é das atividades

para o projeto, ou seja, as atividades possuem uma forma de referenciar o projeto a que pertencem. Desta forma as acumulações nos estoques do projeto serão feitas pelas atividades.

```
CLASS Activity {

    PROPERTY Size 100;
    PROPERTY PlannedStartTime 0;

    PROC PlannedRunningTime ActProjRel.getActualTime - PlannedStartTime;

    STOCK Errors 0;
    PROC EnergyFactor 0.667 * (2 - PerformedBy.getEnergy);
    PROC SkillFactor 0.667 * (2 - PerformedBy.getSkill);
    PROC ErrGenFactor 0.1 * (EnergyFactor*SkillFactor);
    RATE (Errors) RTErrors
        IF(IsReady,
        #then
            ErrGenFactor,
        #else
            0);

    STOCK Completeness 0;
    PROC IsComplete Completeness >= Size;
    RATE (Completeness) RTCompleteness
        IF(IsReady,
        #then
            PerformedBy.getCapProdIndex,
        #else
            0);


    RATE (ActProjRel.PlannedValue) RTPlannedValue
        IF(AND(PlannedRunningTime >= 0, PlannedRunningTime <= Size),
        #then
            DT * PerformedBy.getCostPerHour,
        #else
            0);

    # Calcula o Valor Agregado da atividade até o momento
    RATE (ActProjRel.EarnedValue) RTEarnedValue
        IF(IsReady,
        #then
```

```
            PerformedBy.getCapProdIndex * PerformedBy.getCostPerHour,
        #else
            0);

    #PROC getEarnedValue Completeness * PerformedBy.getCostPerHour;

    RATE (ActProjRel.ActualCost) RTActualCost
        IF(IsReady,
        #then
            PerformedBy.getCostPerHour,
        #else
            0);

    # Determine if the activity is concluded
    PROC IsConcluded Completeness >= 100;

    # Determine if precedent activities are concluded
    PROC IsAllDepsConcluded GroupMax (Predecessors, IsConcluded) >= 0;

    # Determine if the activity is ready to run
    PROC IsReady AND (IsAllDepsConcluded, NOT(IsConcluded));
};
```

A atividade é principal estrutura do modelo, pois ela é que refletirá o andamento, os erros e o custo. Toda atividade possui um tamanho e uma complexidade, que neste modelo está sendo representada pela propriedade "Size". A unidade de complexidade é medida em pessoa/hora. Na Seção 5.2.1 é apresentado um projeto que será objeto de simulação, onde também é possível observar o processo de estimativa para se chegar aos valores da variável "Size"para todas as atividades do projeto.

A medida que a atividade for sendo executada seu percentual de completude vai sendo acumulado no estoque "Completeness". Cada passo da simulação representa uma hora de trabalho, de forma que o que será acumulado no estoque de completude da atividade será o que o seu recurso responsável conseguiu produzir, ou a sua capacidade produtiva, durante aquele passo de simulação.

As atividades podem ter vínculos de precedência, que criam restrições para que dada atividade possa ser iniciada, ou seja, para que dada atividade esteja pronta para ser iniciada, todas as suas dependências têm que estar terminadas. Isso é verificado pelo processo "IsReady".

A fim de que cada atividade possa acumular suas parcelas nos estoques de custo do projeto, foram criadas taxas ("RTPlanedValue", "RTEarnedValue", "RTActualCost")

para acumularem os valores referentes as três variáveis do cálculo do valor agregado. O cálculo do valor planejado é feito com base na multiplicação entre a quantidade de horas trabalhadas na atividade e o custo por hora do recurso alocado. O valor agregado é a medida de contribuição de valor de determinado trabalho ou atividade para o projeto, que no caso é representado pelas horas que o recurso conseguiu de fato produzir. Por exemplo, em 1 hora de atividade o recurso pode ter produzido de valor apenas 0.6 horas, ou seja, apesar de ter custado 1 hora de trabalho, a atividade só avançou 0.6. Este valor representa o valor agregado daquela hora de trabalho. Portanto o valor agregado é feito com base na capacidade produtiva do recurso naquela hora multiplicada pelo seu custo por hora. O custo real representa o efetivamente gasto na execução da atividade, uma vez que o recurso está alocado à atividade e esta esteja pronta para iniciar, o valor hora do recurso é acrescido ao custo real.

Observando a ontologia de Falbo [Fal98], é possível ver que as atividades encontram-se ligadas a artefatos, no sentido de que as atividades geram como produtos artefatos. No entanto, do ponto de vista do jogo, a característica fundamental do artefato a ser contabilizada seria a sua qualidade, ou seja, a quantidade de erros gerados em sua confecção. No entanto, se fixarmos o critério de qualidade, a existência de erros irá, na verdade, indicar mais tempo em atividades de revisão para que os erros sejam corrigidos e o artefato entregue com a qualidade mínima exigida. Desta forma a característica da quantidade de erros gerados em uma atividade é o atributo a ser observado, não havendo então a necessidade da criação de uma estrutura diferenciada só para sua alocação. Por este motivo os erros estão colocados como estoques na estrutura da atividade.

Barros [Bar01] apresenta algumas variáveis que afetam a quantidades de erros geradas por uma atividade. Em nosso modelo tomamos duas destas variáveis: Energia("Energy") e Habilidades("Skill"). Nestas duas variáveis Barros apresenta o mesmo raciocínio: A quantidade de erros geradas por um recurso com sua energia máxima é a metade da gerada por um recurso com energia mínima. Tomando como unidade a contribuição média, o fator de contribuição de um recurso com energia no valor mínimo é de 0,667 e no valor máximo de 1,334. O intervalo de variação da energia e do skill é do 0 (mínimo) a 1 (máximo). Seguindo a premissa de que o recurso com maior energia tem a mínima contribuição na geração de erros (0,667) então o restando da equação tem q ser igual a 1, chega-se a formação do fator $(2 - PerformedBy.getEnergy)$.

A taxa de geração de erros combina os dois fatores através de uma multiplicação. No entanto, os valores apresentados por esta combinação direta foram considerados muito altos. Lembrando que cada passo de execução correnponde a uma unidade de tempo, a

possibilidade do fator de geração de erros (”ErrGenFactor”) assumir valores entre 0.44, correspondente a $2/3 * 2/3$, e 1,77, correspondente a $4/3 * 4/3$. Ou seja, 44% e 177% de taxa de geração de erros, que no modelo apresentado significa que uma atividade de 10 horas de duração ela poderá gerar entre 4,4 e 17,7 horas a serem gastas em uma atividade de correção de erros. Por este motivo, foi aplicado um redutor de 0,1 no cálculo do ”ErrGenFactor”.

```
CLASS HumanResource {
    PROPERTY WorkingHoursPerDay 8;    # Jornada diaria de trabalho
    PROPERTY OverWorkTime 0;          # Horas Extras
    PROPERTY CostPerHour 10.00;       # Custo por hora normal de trabalho
    PROPERTY CapProdIndex 0.8;        # Capacidade Produtiva
    PROPERTY Skill 0.8;               # Habilidade

    PROC getWorkingHoursPerDay WorkingHoursPerDay;
    PROC getOverWorkTime OverWorkTime;
    PROC getCapProdIndex CapProdIndex;
    PROC getCostPerHour CostPerHour;
    }
```

As atividades devem ser executadas por algum membro de uma equipe, neste caso ele é representado pela estrutura *HumanResource*. Basicamente ele mapeia as propriedades que serão obtidas dos atores sintéticos e que influenciarão no andamento da atividade, como por exemplo: *Skill, Indice de Capacidade produtiva*(CapProdIndex).

Este índice de capacidade produtiva é função de várias variáveis importantes em gerenciamento de projetos como por exemplo: habilidades, motivação, personalidade, entre outras. No entanto, esta modelagem apesar de fazer parte do jogo, não faz parte do escopo deste trabalho, pois estes aspectos estão modelados nos modelos dos atores sintéticos. Estes aspectos são tratados em um trabalho de pesquisa a parte. A arquitetura de integração entre o ambiente dos atores sintéticos e o ambiente de simulação de dinâmica de sistemas é apresentada no Capítulo 5

```
SCENARIO ProductivityLossDueAdminTask ProjectModel {
   CONNECTION AffectedActivity Activity {

       PROC ProductivityLossFactor 0.2;
```

```
        AFFECT RTCompleteness RTCompleteness * (1 - ProductivityLossFactor);
        AFFECT RTEarnedValue RTEarnedValue * (1 - ProductivityLossFactor);

    };
```

O modelo de Barros [Bar01] define uma estrutura chamada cenários na qual podemos definir situações de exceção ou eventos que atuam parcialmente sobre o modelo. Em nosso caso definimos que além dos elementos de perda de produtividade que atingem cada membro da equipe de forma individual e particular, haveria uma perda de produtividade genérica associada a execução de atividades administrativas que não agregam valor ao projeto. Ou seja, por mais que um determinada pessoa trabalhe durante 8 horas diárias, as horas de fato dedicadas ao projeto, serão menores que estas 8 horas, pois uma parte será gasta atendendo telefones, conversando com os amigos, indo ao café, tomando água, respondendo emails, navegando na internet, etc. Por mais que o valor colocado no jogo não represente alguma teoria comportamental, ou seja é colocado de forma empírica, o modelo prevê esse tipo de perda e seu valor pode ser ajustado rapidamente.

Diferentemente do que é apresentado nos trabalhos de Barros [Bar01], Dantas [DBW04], e Veronese [Ver03], onde a definição do processo contém não apenas o modelo de domínio, mas também o modelo de projeto, este modelo contém apenas o modelo de domínio. O motivo para esta decisão é de que o modelo de projeto será criado dinamicamente no Hector [Equ06] apartir do código do jogo, com base nas informações carregadas do XML do processo e dos arquivos XML dos níveis (a serem apresentados no Capitulo 5).

O Hector [Equ06] é o ambiente de simulação de Dinâmica de Sistemas criado por Barros [Bar01] para dar suporte a sua meta-linguagem e a estrutura de cenários. Este ambiente é utilizado através de uma interface de programação na qual após a carga do modelo é feita a tradução para os construtores normais da Dinâmica de Sistemas. Esta mesma interface permite a interação durante a simulação e criação de atividades, desenvolvedores em tempo real, ou seja, durante a execução do jogo.

CAPÍTULO 5

O VIRTUAL TEAM

Os jogos baseados em simulação estudados na Seção 3.6 não apresentam um nível de interatividade e usabilidade característico de jogos de entretenimento. São ambientes mais voltados a apresentar ou validar modelos arquiteturais que, fazendo uso da dinâmica de sistemas, podem ser utilizados na criação de jogos educativos. No entanto, ao observarmos alguns jogos de estratégia disponíveis no mercado como Microsoft Flight Simulator™, Sim City™ ou The Simms™, é possível verificar uma clara distância em seus níveis de interatividade e qualidade de interface. Apesar da alta carga educativa destes jogos de entretenimento, eles não tem como objetivo primordial a capacitação, ou seja, foram planejados para serem utilizados somente pelo jogador e com período de experimentação mais longo, tendo portanto objetivos diversos.

Já nos jogos de simulação utilizados com objetivo educacional, a experimentação por parte do jogador está dentro de um contexto de um modelo que provavelmente já foi ou será discutido em sala. Normalmente a experimentação do jogo está vinculada à presença de um facilitador, que alterna sessões de interação com o jogo com sessões de discussão e compartilhamento das experiências.

Neste capítulo é apresentado, na Seção 5.1, o projeto SmartSim, que objetiva a criação de um framework de simulação com uso de atores sintéticos para a criação de jogos sérios. Na Seção 5.2 é apresentado o jogo Virtual Team, que é um protótipo de um jogo de negócio voltado a gestão de pessoas em um ambiente de desenvolvimento de software. Nesta mesma seção também são apresentados o projeto de software que será objeto de simulação por parte do jogo além de algumas telas do jogo. Na Seção 5.3, é apresentada arquitetura de integração entre o ambiente de simulação da dinâmica de sistemas e os atores sintéticos. Por fim é apresentado uma validação do modelo de domínio em gerenciamento de projetos utilizado no Virtual Team.

5.1 O PROJETO SMARTSIM

O projeto SmartSim surgiu da união desta pesquisa com pesquisas sobre atores sintéticos. Trata-se de um projeto multidisciplinar com o objetivo de desenvolver um *framework* de

código aberto, sob licença LGPL [FSF99], para o desenvolvimento de jogos sérios baseados em simulação e que utilizam atores sintéticos para simular as personagens envolvidas no processos [Sma06].

As pesquisas envolveram áreas multidisciplinares como por exemplo: Inteligência Artificial [Rus95], Jogos Sérios baseados em simulação [Bar01], psicologia organizacional [Rob05] e Gerenciamento de Projetos [Ins04].

A fim de entender melhor as interações entre os atores sintéticos e o ambiente de simulação definiu-se como o foco do jogo uma aplicação na área de gerenciamento de projetos, mais especificamente na sub-área de gestão de pessoas. Desta forma, poder-se-ia não apenas verificar e interagir com os personagens em suas atribuições no projeto, mas também os aspectos relacionados ao seu trabalho em equipe. Para que esta equipe, formada pelo atores sintéticos, pudesse trabalhar junta em um projeto de desenvolvimento de software, é necessário, não apenas, a modelagem de uma parte do mundo do gerenciamento de projetos para guiar a simulação, dentro do que Barros chama de modelo de domínio [Bar01], mas também representar o processo de software, que contém as atividades a serem de fato simuladas pelos atores, dentro do que Barros chama modelo de projeto [Bar01].

Este trabalho de pesquisa contribuiu com o projeto justamente no fornecimento do modelo de processo de software, na composição do modelo de domínio em gerenciamento de projetos e na definição da arquitetura de integração entre os atores sintéticos e a dinâmica de sistemas.

O projeto SmartSim também foi objeto de contribuição de outro estudo, que o permitiu ter uma interface gráfica diferenciada e mais lúdica do que os jogos de simulação aqui estudados, além de um processo para construção de personagens [Gur06]. Esta interface pode ser melhor observada na Seção 5.2.2.

5.2 O VIRTUAL TEAM

O Virtual Team foi concebido como o protótipo de um jogo para auxiliar na capacitação de gerentes de projeto com ênfase em gestão de pessoas. Ele procura simular cenários de um ambiente de desenvolvimento de software em que os atores sintéticos, serão utilizados para modelar os participantes da equipe de desenvolvimento, que desempenharão os diversos papéis, tais como: engenheiros de software, arquitetos, implementadores, testadores, entre outros. O jogador exercerá o papel do gerente de projetos.

O objetivo de se utilizar os atores sintéticos para modelar os membros da equipe objetiva levar um maior realismo nos cenários simulados, permitindo assim ao jogador/gerente

uma experimentação mais rica dos processos organizacionais, metodológicos, pessoais e culturais inerentes a uma equipe de desenvolvimento de software.

5.2.1 O Cenário do Jogo: Projeto SES

O sistema a ser desenvolvido no projeto é um sistema de recomendação de compras via web, denominado SES – *SmartSim Economic Shopping*. Neste sistema o usuário cadastra sua cesta de compras e pode acompanhar em quais estabelecimentos comerciais reais aquela cesta encontra-se mais em conta.

O sistema constitui-se de seis casos de uso, que podem ser observados na Figura 5.1.

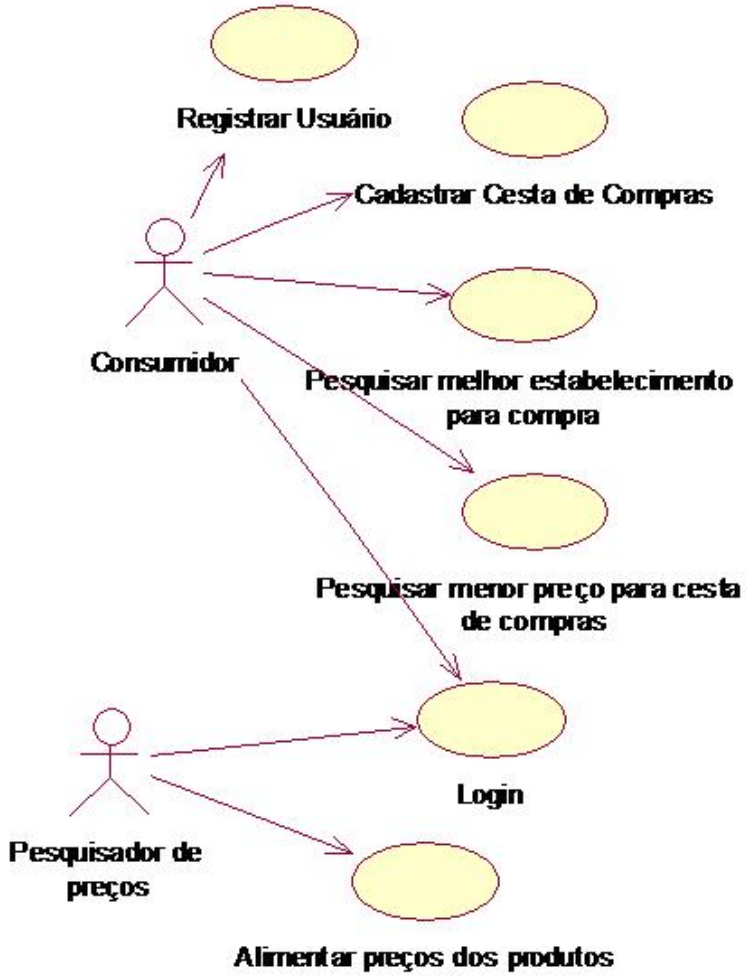

Figura 5.1 Diagrama de Casos de uso do SmartsimEconomicShopping

Os casos de uso tem a seguinte finalidade:

- Registrar Usuário: Este caso de uso permite aos usuários não cadastrados no serviço registrarem-se para poder fazer uso dos serviços do site. Ele deve fazer registro de informações que permitam definir o seu perfil de consumo.

- Efetuar Login: Este caso de uso permite ao usuário cadastrado abrir a sua sessão de uso e ter acesso aos serviços do site.

- Cadastrar Cesta de Compras: Este caso de uso permite ao usuário cadastrar no seu perfil a sua cesta de compras. Esta coleção de produtos será objeto de pesquisa por parte do site, e de alimentação de preços por parte dos pesquisadores de preços.

- Pesquisar Melhor Estabelecimento para Cesta de Compras: Este caso de uso faz um comparativo do valor de toda a cesta nos vários estabelecimentos cadastrados e apresenta o de menor valor. Para fins de comparação pode-se colocar o segundo estabelecimento com melhor preço.

- Pesquisar Menor Preço para Cesta de Compras: Este caso de uso fornecerá produto a produto o melhor estabelecimento para a compra.

- Alimentar Preços dos Produtos: Este caso de uso permite aos pesquisadores de preços do site alimentarem a base com os preços dos produtos. Os pesquisadores trabalham remotamente e acessam o sistema através de aplicativos que rodam em PDAs. Esta atualização de preços também pode ser efetuada diretamente pelos varejistas, através de convênios e parcerias estabelecidas.

Para a simulação metrificar o sistema é importante, na medida em que as atividades no processo de software tem naturezas diferentes e conseqüentemente possuem estimativas de duração diferenciadas. Desta forma, faz-se necessário metrificar o projeto a ser simulado, no caso o SES, para saber a previsão da duração de cada atividade do projeto.

A técnica utilizada para medir o tamanho do sistema foi a Pontos por Caso de Uso (PCU) [Kar93]. A técnica de PCU é derivada da técnica de Pontos de Função, no entanto apresenta-se mais adequado a medir sistemas industriais que utilizam um processo de software baseado no *Objectory*. O *Objectory* é um dos processos que deu origem ao RUP™, assim sendo o RUP™preserva suas características, fazendo com que a técnica de PCU seja uma das mais utilizadas atualmente para estimativas de sistemas que utilizam orientação a objetos, UML e RUP™.

Segundo o modelo de Karner [Kar93] o sistema possui 98,00 pontos por casos de uso não ajustados (PCUNA), que medem as funcionalidades do sistemas apartir do modelo de casos de uso. Basicamente esse fator considera a quantidade de casos de uso e de atores do sistema e as suas complexidades.

Outros dois fatores ajudam a ajustar esse valor de complexidade são os fatores técnicos e os fatores ambientais. Os fatores técnicos relacionados as funcionalidades são avaliados de forma semelhante aos pontos de função, analisando questões como: distribuição, performance, segurança, eficiência, reuso, facilidade de instalação, manutenibilidade, portabilidade, concorrência, entre outros. O fator de complexidade técnica (FCT) é de 1,23.

Os fatores ambientais ajudam a estimar qual eficiente o projeto é avaliando questões como: familiaridade da equipe com o RUP™, experiência da equipe, experiência da equipe com orientação a objetos, capacidade dos analistas da equipe, motivação, estabilidade dos requisitos, domínio da tecnologia e configuração do ambiente. Neste caso os fatores ambientais somam 1,40.

Desta forma temos:

$$PCUNA = 98,00$$
$$FCT = 1,235$$
$$FA = 1,40$$

Segundo Karner os pontos de casos de uso então são calculados através de:

$$PCU = PCUNA * FCT * FA$$
$$PCU = 169,44$$

Karner sugere que sejam utilizados 20 pessoa-hora/PCU. Desta forma o projeto gastaria 3.388,84 pessoa-hora para ser concluído. Esta é uma medida de esforço, que permite ao gerente de projetos saber a complexidade de todo o projeto.

No contexto Virtual Team a equipe de desenvolvimento será formada por 5 membros. Tendo desta forma um total de 677,77 horas. A considerar a semana de trabalho como tendo 40 horas de trabalho, isto totalizaria 4,24 meses de trabalho.

Há a necessidade ainda de estimar a complexidade de cada atividade que compõe o projeto. A Tabela 5.1 apresenta uma tabela com dados sobre o percentual de tempo gasto em cada atividade do processo na coluna "% Esforço JONES".

No entanto, apresentam-se aqui duas dificuldades: os estudos feitos por Jones foram feitos em métricas e técnicas diferentes da PCU; o processo ao qual Jones dividiu o

percentual de tempo não é o processo unificado. O fato da tabela de Jones ter sido feita em pontos de função, não nos afasta da realidade, pois a técnica de pontos de casos de uso baseia-se nos mesmos princípios utilizados na técnica de pontos de função, fazendo uso inclusive dos mesmos critérios para os fatores ambientais.

Como as atividades constantes do processo proposto neste trabalho na Seção 4.2 são de engenharia de software, removemos o esforço sugerido por Jones, relacionado as atividades de gerenciamento de projetos, e distribuímos entre os demais tipos de atividades, resultando na coluna "% Esforço sem GP". Esta alteração se faz necessária, pois as atividades de gerenciamento de projetos não serão executadas pelos atores sintéticos, portanto não há a necessidade de sua estimativa.

Quanto as diferenças entre o processo utilizado por Jones e o proposto neste trabalho, procuramos mapear as naturezas das atividades do processo unificado nas constantes na Tabela 5.1. Por exemplo a atividade de "Levantamento de Requisitos do Sistema"envolve aspectos não apenas de requisitos mas também de documentação. Na Tabela 5.3 pode ser vista um exemplo do mapeamento das atividades nas classes de atividades que Jones apresenta em seu estudo. É importante citar que as atividades aqui apresentadas correspondem apenas a uma parte das atividades que compõem o projeto, que é composto por 85 atividades.

A Tabela 5.3 também apresenta a classificação de complexidade feita na classificação dos pontos de casos de uso e os pesos sugeridos por Karner.

A fim de calcular o peso real de cada atividade no projeto, buscou-se então ver o somatório dos pesos das atividades que eram classificadas em determinada categoria. A Tabela 5.4 apresenta em sua coluna "Soma dos Pesos"estes somatórios. Por exemplo para as atividades classificadas como envolvendo aspectos de requisitos, estas somaram 185 de peso. A coluna "Dist por Peso"apresenta a contribuição relativa de cada unidade de peso para cara tipo de atividade caracterizada por Jones. Ela foi obtida pela divisão dos valores da coluna "% Esforço sem GP"da Tabela 5.1 com a coluna "Soma dos Pesos"da Tabela 5.4.

Chegando a essa unidade de contribuição relativa de cada peso, tem-se uma unidade com a qual é possível ver a contribuição específica de cada atividade do projeto.

A Tabela 5.5 apresenta um exemplo das contribuições percentuais de cada atividade. Por exemplo, a atividade "Levantar Requisitos do Sistema"possui características 1 e 7, assim sendo deve receber as contribuições relativas das atividades tipo 1 (Requisitos) e tipo 7 (Documentação). Este valor somado é multiplicado pelo peso da sua complexidade, fazendo então com que a mesma tenha uma participação esperada no tempo do projeto

Cod	Atividade	% Esforco	%Esforco sem GP
1	Requisitos	3,66	4,11
2	Projeto Inicial	3,29	3,70
3	Projeto Detalhado	4,39	4,93
4	Codificação	18,29	20,55
5	Reutilização	0,32	0,36
6	Gerência de Configuração	1,32	1,48
7	Documentação	4,39	4,93
8	Testes de Unidade	16,46	18,49
9	Testes de Função	14,32	16,09
10	Testes de Sistema	13,17	14,79
11	Testes de Aceitação	9,41	10,57
12	Gerência de Projeto	10,98	
	Total	100,00	100,00

Tabela 5.1 Percentual estimado do projeto divido pelas atividades [Jon00]

SmartSim Economic Shopping	Mapeamento para Tabela de JONES	Complexidade da Atividade	
Concepcao			
Levantar Requisitos do Sistema	1,7	Complexo	15
Detalhar Especificacao de Caso de Uso			
Login	1,7	Medio	10
Cadastrar Usuario	1,7	Simples	5
Cadastrar Cesta de Compras	1,7	Medio	10
Pesquisar Melhor Estabelecimento par	1,7	Complexo	15
Pesquisar Menor Preco para Cesta de	1,7	Complexo	15
Alimentar Precos dos Produtos	1,7	Complexo	15
Receber Cotacao de Precos	1,7	Complexo	15
Estruturar Modelos de Casos de Uso			
Login	1,7	Medio	10
Cadastrar Usuario	1,7	Simples	5
Cadastrar Cesta de Compras	1,7	Medio	10
Pesquisar Melhor Estabelecimento par	1,7	Complexo	15
Pesquisar Menor Preco para Cesta de	1,7	Complexo	15
Alimentar Precos dos Produtos	1,7	Complexo	15
Receber Cotacao de Precos	1,7	Complexo	15
Elaboracao			
Estruturar Modelos de Casos de Uso			
Analisar Casos de Uso			
Login	2	Medio	10
Cadastrar Usuario	2	Simples	5
Cadastrar Cesta de Compras	2	Medio	10
Pesquisar Melhor Estabelecimento par	2	Complexo	15
Pesquisar Menor Preco para Cesta de	2	Complexo	15
Alimentar Precos dos Produtos	2	Complexo	15
Receber Cotacao de Precos	2	Complexo	15
Projetar Arquitetura	2,3	Complexo	15
Projetar Casos de Uso			
Login	3	Medio	10
Cadastrar Usuario	3	Simples	5
Cadastrar Cesta de Compras	3	Medio	10
...			

Tabela 5.2 Exemplo do mapeamento das atividades do projeto SES e as classes de atividades de Jones

SmartSim Economic Shopping	Mapeamento para Tabela de JONES	Complexidade da Atividade	
Concepcao			
Levantar Requisitos do Sistema	1,7	Complexo	15
Detalhar Especificacao de Caso de Uso			
Login	1,7	Medio	10
Cadastrar Usuario	1,7	Simples	5
Cadastrar Cesta de Compras	1,7	Medio	10
Pesquisar Melhor Estabelecimento par	1,7	Complexo	15
Pesquisar Menor Preco para Cesta de	1,7	Complexo	15
Alimentar Precos dos Produtos	1,7	Complexo	15
Receber Cotacao de Precos	1,7	Complexo	15
Estruturar Modelos de Casos de Uso			
Login	1,7	Medio	10
Cadastrar Usuario	1,7	Simples	5
Cadastrar Cesta de Compras	1,7	Medio	10
Pesquisar Melhor Estabelecimento par	1,7	Complexo	15
Pesquisar Menor Preco para Cesta de	1,7	Complexo	15
Alimentar Precos dos Produtos	1,7	Complexo	15
Receber Cotacao de Precos	1,7	Complexo	15
Elaboracao			
Estruturar Modelos de Casos de Uso			
Analisar Casos de Uso			
Login	2	Medio	10
Cadastrar Usuario	2	Simples	5
Cadastrar Cesta de Compras	2	Medio	10
Pesquisar Melhor Estabelecimento par	2	Complexo	15
Pesquisar Menor Preco para Cesta de	2	Complexo	15
Alimentar Precos dos Produtos	2	Complexo	15
Receber Cotacao de Precos	2	Complexo	15
Projetar Arquitetura	2,3	Complexo	15
Projetar Casos de Uso			
Login	3	Medio	10
Cadastrar Usuario	3	Simples	5
Cadastrar Cesta de Compras	3	Medio	10
...			

Tabela 5.3 Exemplo do mapeamento das atividades do projeto SES e as classes de atividades de Jones

Cod	Atividade	Soma dos Pesos	Dist por Peso
1	Requisitos	185	0,022
2	Projeto Inicial	100	0,037
3	Projeto Detalhado	185	0,027
4	Codificação	85	0,242
5	Reutilização	85	0,004
6	Gerência de Configuração	45	0,033
7	Documentação	270	0,018
8	Testes de Unidade	170	0,109
9	Testes de Função	285	0,056
10	Testes de Sistema	285	0,052
11	Testes de Aceitação	285	0,037

Tabela 5.4 Somatório dos pesos das atividades baseado na classificação de Jones

de 0.61%.

Levando em consideração que a estimativa em PCU aponta para um total de 3.388,84 pessoa-hora de esforço, então a coluna "Esforço Estimado em Horas"apresenta a estimativa de esforço para cada atividade no projeto SES.

Desta forma, tem-se então o esforço planejado para cada atividade. Este esforço será o parâmetro sobre o qual a simulação irá se basear. Estes valores serão carregados dos arquivos de configuração do jogo e serão utilizados para atribuir valor a propriedade "Size"do objeto "Atividade", constante do modelo de domínio definido na Seção 4.5

5.2.2 A Interface

A interface do VIRTUAL TEAM foi concebida dentro do contexto do projeto SMARTSIM com o objetivo de permitir o uso de elementos de interface que aumentassem a imersibilidade do jogador, ou seja, a sensação de realidade percebida pelo jogador. Através de pesquisas qualitativas junto ao público-alvo [Gur06], chegou-se a uma linguagem gráfica que permitisse ao jogo atingir a este objetivo, mas também, o de representar os as características de personalidade dos atores sintéticos. Nesta seção apresentaremos algumas telas do jogo, afim de que seja possível ter uma idéia clara da proposta do jogo VIRTUAL TEAM.

Figura 5.2 Menu Inicial do Jogo VIRTUAL TEAM

A Figura 5.2 apresenta o menu inicial do jogo onde é possível iniciar um jogo, ou

carregar um jogo ora iniciado, ou verificar a lista dos recordes de pontuações através do *Hall da Fama*.

O início de um novo jogo representa o início de um novo projeto, ou seja, uma nova tentativa de se executar o SES. Por uma questão de escopo, decidiu-se focar o jogo na gestão de pessoas, cujos processos estão concentrados na fase de execução. Assim sendo os processos próprios do planejamento ficaram restritos aos apresentados na Seção 4.3.

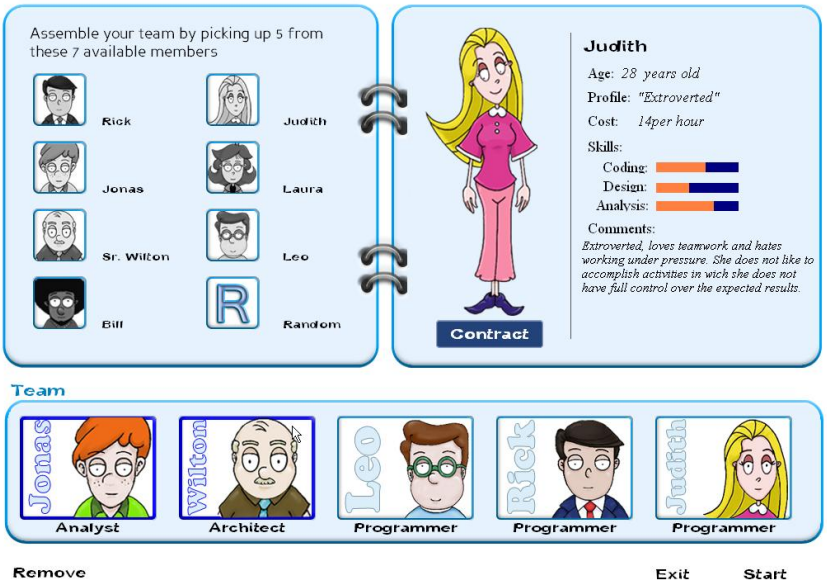

Figura 5.3 Tela de Seleção da Equipe

A seleção da equipe pode ser feita na tela apresentada na Figura 5.3. A equipe de desenvolvimento será formada por cinco desenvolvedores, sendo: um analista, um arquiteto e três programadores. Estes cinco desenvolvedores serão selecionados apartir de um grupo que é apresentado no painel superior esquerdo. Uma vez selecionado um desenvolvedor, suas características são apresentadas no painel logo a direita, sendo possível ver seu nome, idade, custo por hora, habilidades técnicas e características humanas. Uma vez feita a contratação o desenvolvedor passa a ocupar uma vaga na equipe, para o papel selecionado.

A Figura 5.4 apresenta a tela principal de execução do jogo, onde podem ser observados três painéis. O painel superior esquerdo apresenta o ambiente de trabalho da equipe. Nele é possível observar um prisma vermelho que indica o membro da equipe selecionado e sobre o qual deseja-se executar algum tipo de ação. O painel superior direito apresenta informações sobre o projeto, tanto planejamento quanto para acompanhamento. Já o painel inferior é reservado à apresentação das informações sobre os membros da equipe.

SmartSim Economic Shopping	Mapeamento para Tabela de JONES	% Esforco Estimado	Esforco Estimado em Horas
Concepcao			
Levantar Requisitos do Sistema	1,7	0,61	20,58
Detalhar Especificacao de Caso de Uso			
Login	1,7	0,40	13,72
Cadastrar Usuario	1,7	0,20	6,86
Cadastrar Cesta de Compras	1,7	0,40	13,72
Pesquisar Melhor Estabelecimento par	1,7	0,61	20,58
Pesquisar Menor Preco para Cesta de	1,7	0,61	20,58
Alimentar Precos dos Produtos	1,7	0,61	20,58
Receber Cotacao de Precos	1,7	0,61	20,58
Estruturar Modelos de Casos de Uso			
Login	1,7	0,40	13,72
Cadastrar Usuario	1,7	0,20	6,86
Cadastrar Cesta de Compras	1,7	0,40	13,72
Pesquisar Melhor Estabelecimento par	1,7	0,61	20,58
Pesquisar Menor Preco para Cesta de	1,7	0,61	20,58
Alimentar Precos dos Produtos	1,7	0,61	20,58
Receber Cotacao de Precos	1,7	0,61	20,58
Elaboracao			
Estruturar Modelos de Casos de Uso			
Analisar Casos de Uso			
Login	2	0,37	12,52
Cadastrar Usuario	2	0,18	6,26
Cadastrar Cesta de Compras	2	0,37	12,52
Pesquisar Melhor Estabelecimento par	2	0,55	18,79
Pesquisar Menor Preco para Cesta de	2	0,55	18,79
Alimentar Precos dos Produtos	2	0,55	18,79
Receber Cotacao de Precos	2	0,55	18,79
Projetar Arquitetura	2,3	0,95	32,34
Projetar Casos de Uso			
Login	3	0,27	9,03
Cadastrar Usuario	3	0,13	4,52
Cadastrar Cesta de Compras	3	0,27	9,03
...			

Tabela 5.5 Contribuições percentuais e estimativa de duração

Figura 5.4 Tela Principal de Execução do VIRTUAL TEAM

O painel superior esquerdo apresenta um menu suspenso, porém fixo, no qual é possível enviar comandos ao agente selecionado, tais como: associar tarefa, definir a carga-horária de trabalho, dar um *feedback* ao desenvolvedor sobre o seu trabalho, premiar o desempenho do desenvolvedor, alocar o desenvolvedor em treinamento e demitir o desenvolvedor da equipe.

No painel superior é possível encontrar um menu fixo, com informações sobre a data atual, botões para pausar e avançar o jogo, ter acesso a relatório na forma de emails que são enviados ao gerente e convocar reuniões. Opcionalmente pode ser apresentado um termômetro, que indica a proximidade ou afastamento dos objetivos do jogo. Na alocação de tarefas observa-se uma caixa de diálogo suspensa onde são apresentadas as atividades que compõem o processo. Ao selecionar a atividade ela é alocada ao desenvolvedor, e serão executadas segundo sua ordem de precedência e assim que o desenvolvedor já tiver terminado outra atividade. As atividades são executadas uma por vez por desenvolvedor, e assumem a restrição *ASAP - As Soon As Possible*, ou seja, o quanto antes possível.

O acompanhamento do projeto pode ser feito de várias formas. A Figura 5.6 apresenta uma possível forma de acompanhamento através do painel direito, na opção acompanhamento de projeto. Nele é apresentado um gráfico com três curvas, que apresentam as variáveis da análise de valor agregado (EVA) [Ins04]. Cada ponto selecionado na curva apresenta os valores das três variáveis no ponto da simulação selecionado e as variáveis de *forecasting* apartir daquele ponto.

No canto inferior direito do mesmo painel é possível encontrar um botão com uma seta vermelha. O seu acionamento faz aparecer um painel sobre o painel principal, com as informações da WBS (*Work Breakdown Structure*) [Ins04]. Neste painel é possível acompanhar o andamento das atividades como um todo, vendo o seu percentual de completude, sua data de início e o seu responsável.

A Figura 5.7 apresenta uma tela na qual é possível ver outra forma de fazer o acompanhamento de atividades, de forma individual por desenvolvedor, ou seja, nela é possível ver as todas as atividades alocadas para um desenvolvedor em particular, e além disso obter informações sobre essas atividades.

O painel inferior apresenta várias informações sobre o desenvolvedor, entre elas o perfil, com as mesmas informações apresentadas na tela de seleção da equipe, como pode ser observado na Figura 5.8, na qual são apresentado as habilidades técnicas do desenvolvedor. Mas também é possível acompanhar o índice de produtividade do desenvolvedor, o nível de satisfação dele para com o ambiente de trabalho e para com a tarefa.

Figura 5.5 Tela de Alocação de Tarefas

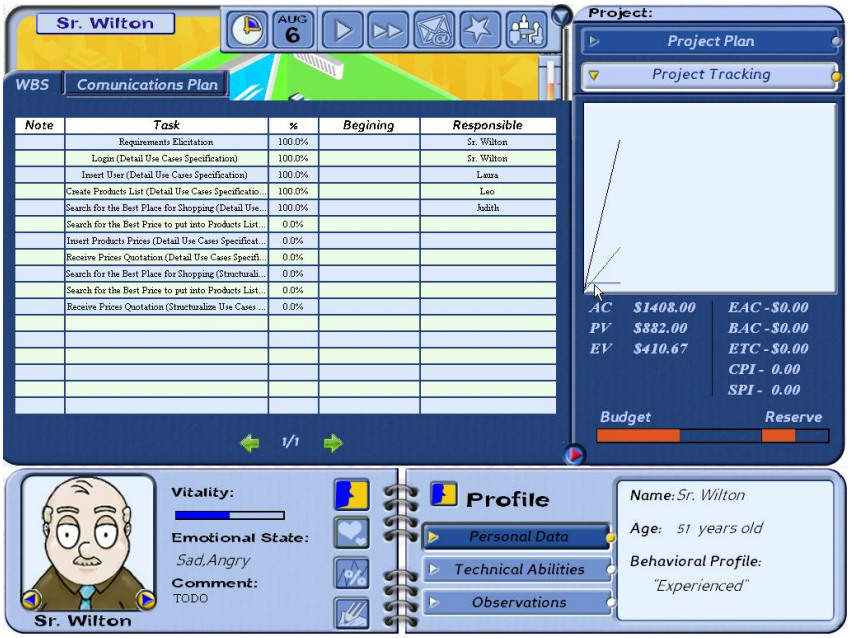

Figura 5.6 Tela de Acompanhamento de Atividades

Figura 5.7 Tela de Acompanhemento das Atividades de um Desenvolvedor

Figura 5.8 Tela de Apresentação dos Perfil do Desenvolvedor

5.3 A ARQUITETURA DO VIRTUAL TEAM

A arquitetura proposta para o Virtual Team visa permitir que haja a interação entre dois ambientes distintos: o ambiente de simulação baseada nos atores sintéticos e o ambiente de simulação baseada na dinâmica de sistemas.

A simulação baseada nos atores sintéticos destina-se a implementação dos aspectos individuais dos membros da equipe do projeto, como por exemplo: características de personalidade, motivação, relacionamento com outros membros da equipe, entre outros. A modelagem destes aspectos tem uma importante contribuição, na medida que aumentará o nível de realidade do jogo, e ainda permitirá explorar aspectos do relacionamento humano. Esta forma de modelagem assume especial importância também pela dificuldade de formação de um modelo sistêmico que comporte as teorias de personalidade, pois normalmente estas são baseadas no indivíduo e em suas características.

A simulação baseada na dinâmica de sistemas será responsável pela simulação do projeto em si e pelas variáveis mais diretas a ele relacionadas, como por exemplo: custo, tempo, erros, completude, EVA, entre outras.

No entanto, há a necessidade de interação entre estes dois ambientes, ou dois subsistemas, para que as informações necessárias ao projeto provindas dos membros da equipe sejam utilizadas e para que as informações resultantes da simulação sejam passadas aos atores sintéticos.

Não será objetivo aqui explicar toda a arquitetura do Virtual Team dado que muitos aspectos saem do escopo desta pesquisa, mas apenas aqueles relevantes para esta proposta de arquitetura de integração.

A arquitetura é aqui representada utilizando o diagrama de classes e de sequência da UML. A UML [RJB99] é a linguagem de modelagem atualmente utilizada para modelar sistemas orientados a objetos, caso do Virtual Team. Os diagramas de classe visam apresentar a estrutura estática da organização das principais classes que implementam a arquitetura sugerida. Na UML, uma estrutura particular assume um papel importante na organização das classes, são os pacotes, que funcionam como se fossem diretórios onde as classes e outras estruturas da UML podem ser agrupadas. Normalmente os critérios de agrupamento estão relacionados ao conceito de coesão. Para dar uma visão mais abrangente de como as classes foram organizadas é apresentada inicialmente um diagrama de classes com os pacotes do jogo. Já os diagramas de sequência apresentam uma visão dinâmica das classes, mostram as classes numa sequência de trocas de mensagens quando da prestação de um serviço especifico.

A linguagem de implementação utilizada foi C++ [Pre04] devido aos requisitos de alta

performance e baixo consumo de memória do jogo. A escolha de C++ também se deu em função da escolha de uso de um framework de construção de jogos chamado FORGE [Mad01], que permite controlar as iterações do jogador com os objetos de interface, como menus, avatares, gráficos, etc.

A arquitetura básica do jogo obedece a uma organização em camadas. O uso deste framework obriga algumas separações de conceitos mas principalmente o uso de um padrão de projeto *State* [G+98], onde ele caracteriza todos os objetos de uma tela como sendo parte de um estado.

A Figura 5.9 apresenta uma visão geral de como está organizada a arquitetura do jogo e a distribuição das camadas.

A camada de interface com o usuário (*UserInterface*) é representada pelo FORGE, componentes gráficos que estendem os componentes do FORGE e algumas outras classes que objetivam controlar o comportamento dos objetos na interface.

A camada de controle (*Controller*) é responsável pelo controle do laço de simulação do jogo, interagindo com a camada de modelo (*Model*) para obter as informações dos atores sintéticos e passar ao subsistema de simulação. A camada de controle também interage com camada de modelo (*Model*) para atualizar as informações do projeto e das atividades, a cada passo de simulação. O subsistema de simulação é responsável por interagir com a DLL que faz a simulação do modelo carregado do arquivo com as informações do metamodelo, conforme pode ser observado no A. A camada de modelo (*Model*) interage com o subsistema dos atores sintéticos, que é o responsável por implementar as regras de comportamento e os estados observados por cada agente ao longo de todo o jogo.

A Figura 5.10 apresenta os principais pacotes que compõem o jogo. Os três pacotes mais importantes são *virtualTeam.game, virtualTeam.simulation e virtualTeam.model*. O pacote *virtualTeam.game* contém toda a máquina do jogo, as classes de interface e as classes que modelam os comandos do jogador. Já o pacote *virtualTeam.model* contém as classes que modelam o projeto e as classes que representam os atores sintéticos, incluindo sua base de conhecimento e máquinas de inferência. No pacote *virtualTeam.simulation* encontram-se as classes responsáveis pela simulação do projeto na dinâmica de sistemas.

A Figura 5.11 apresenta a visão geral das classes de mais alto nível do jogo. Nela é possível observar algumas classes particularmente importantes em nosso contexto. Primeiramente a classe *VTGameManager* é responsável pelo laço principal do jogo, fazendo a troca dos estados exigidos pelo FORGE, que no caso são representados pela classe *VTGameState*. No entanto estas classes não possuem o gerenciamento direto das entidades que compõem o modelo do jogo, responsabilidade essa delegada ao *VTEntityManager*.

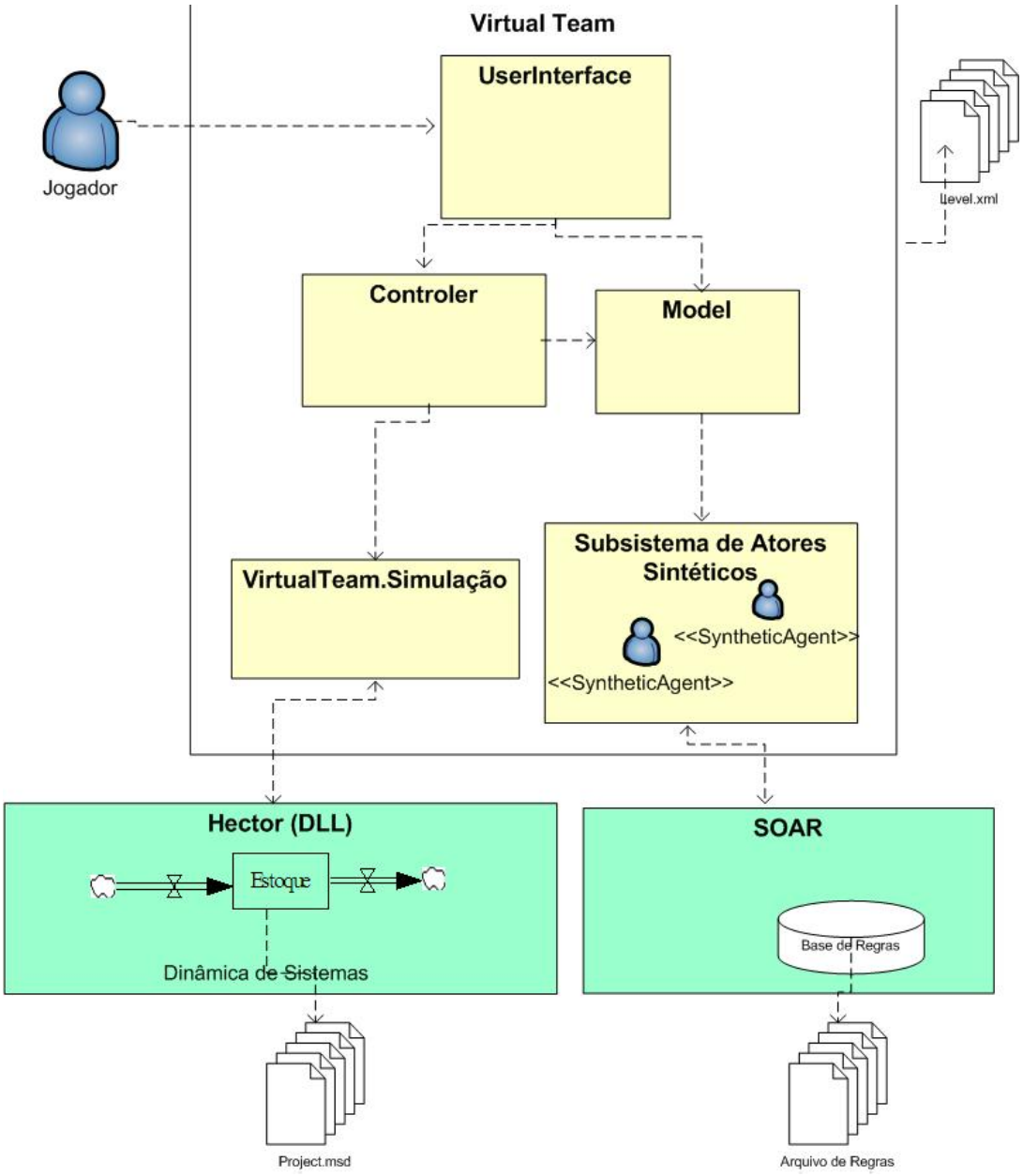

Figura 5.9 Visão Geral da Arquitetura

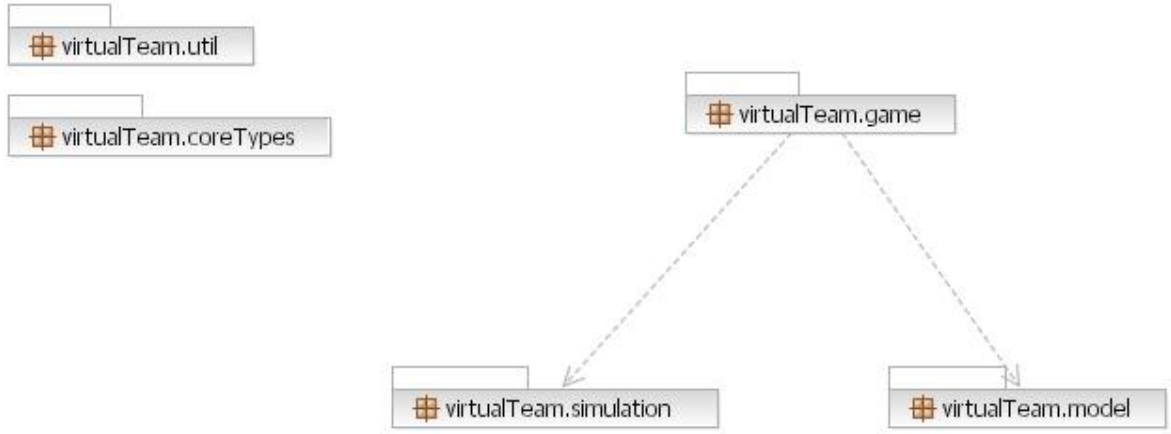

Figura 5.10 Pacotes que compõem o VIRTUAL TEAM

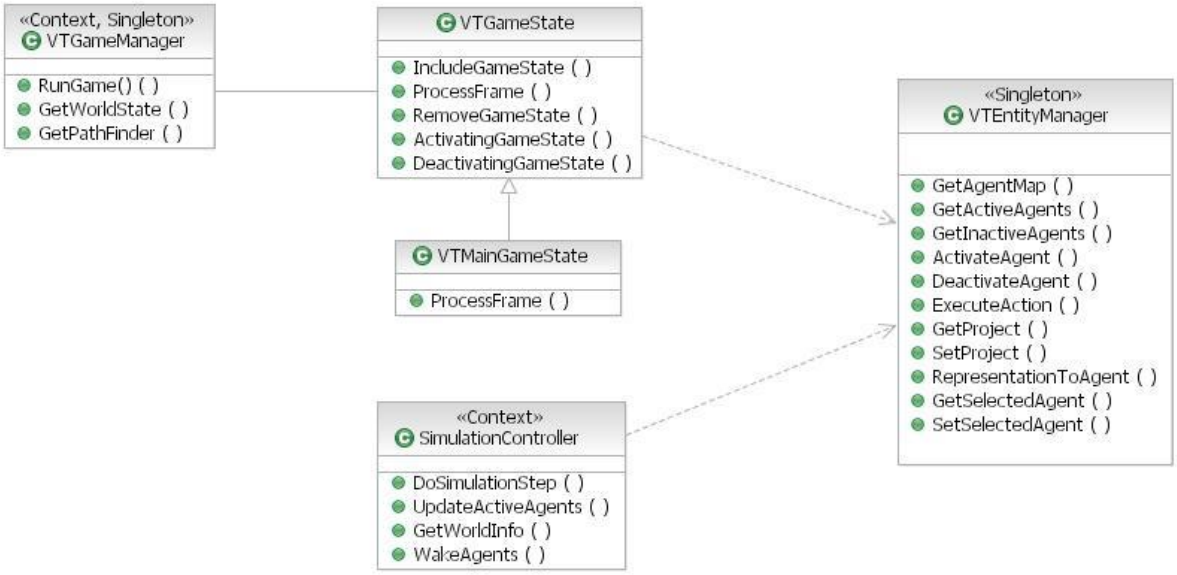

Figura 5.11 Principais classes do VIRTUAL TEAM

Os estados disponíveis no Virtual Team podem ser observados na Figura 5.12. São criadas classes que representam estados distintos do jogo, como por exemplo: a tela de menu inicial do jogo (*VTMainMenuState*, a tela de seleção dos membros da equipe (*VTSelectTeamState*), a tela que apresenta o ambiente de execução principal do jogo *VTMainGameState* e a tela de fim de jogo (*VTGameOverState*). Estas classes que mapeiam os estados funcionam como espécies de controladores do que acontece quando o jogo está naquele determinado estado. Desta forma cada um possui uma máquina de estados diferenciada.

A execução do projeto é gerenciada pelo *VTMainGameState*, que possui o laço principal que é responsável pela simulação do projeto e de interação com os agente inteligentes. A comunicação entre a classe de estado e a classe responsável pela simulação se dá através do acionamento de uma classe de *timer (VTMainTimer)*, que representa os *ticks* do relógio do jogo. Este controle é necessário para que o jogo tenha capacidade de ser acelerado, desacelerado ou pausado pelo jogador.

Na Figura 5.13 podemos ver a classe *SimulationController* que é responsável pela execução da máquina de simulação. Ela se relaciona com a classe de *timer* através de um padrão de projeto *Observer* [G+98]. Ela faz a interação com os objetos que compõem o modelo e que contém os dados do projeto e dos atores sintéticos através da *VTEntityManager* para obter as informações que serão necessárias à simulação.

O mecanismo de simulação encontra-se encapsulado no pacote *virtualTeam.simulation*. A principal classe desse pacote é a classe *SimulationEngine*, sendo responsável por simular uma unidade de tempo, no caso hora. No entanto, dado que os atributos relacionados aos atores sintéticos não estão no domínio da dinâmica de sistemas, e que alguns desses atributos são importantes para que o passo de simulação ocorra, há a necessidade de transitar informações entre o ambiente de simulação e a máquina do jogo. Para este fim foram criados dois *Value Objects* que irão conter dentro de si, todos os atributos necessários a esta troca de informação. O *SimulationInputData* e o *SimulationOutputData* funcionam então como *Value Objects*.

A classe *SumulationController* possui um processo de *cache* dos dados dos atores sintéticos, para que os dados sejam encapsulados de uma só vez e a invocação do passo de simulação ocorra com os dados de todos os agentes.

A Figura 5.13 também mostra a classe *VTEntityManager* contendo as associações para as classes que representam o projeto e os atores sintéticos.

A Figura 5.14 apresenta em detalhes as classes do pacote *virtualTeam.model*. Neste pacote encontram-se as classes do domínio do jogo, no caso gerenciamento de projetos.

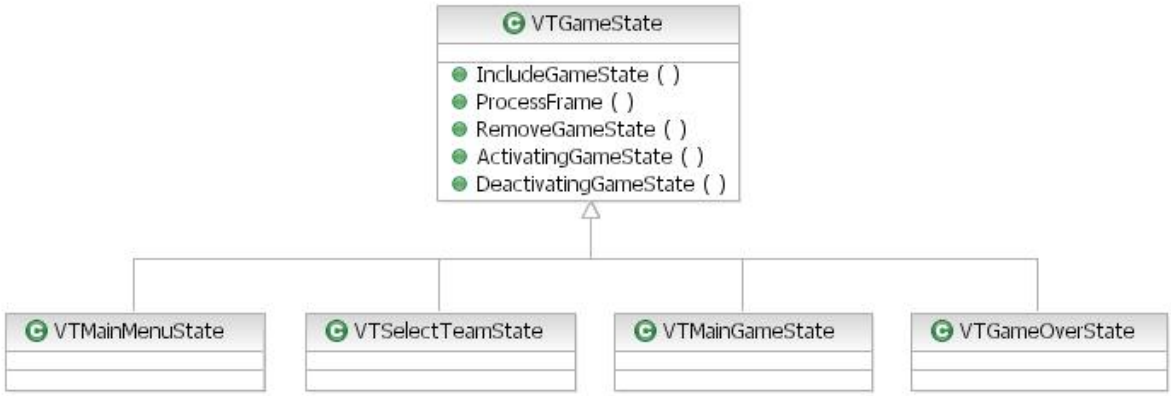

Figura 5.12 Hierarquia do VTGameState

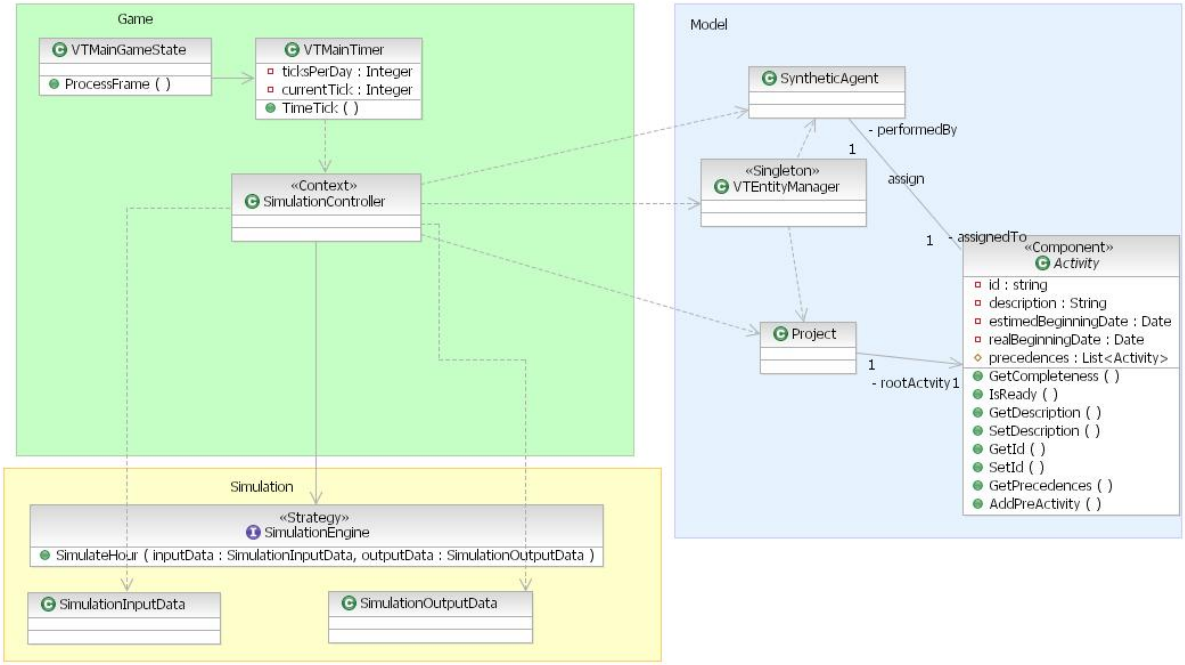

Figura 5.13 Visão geral da integração entre os pacotes

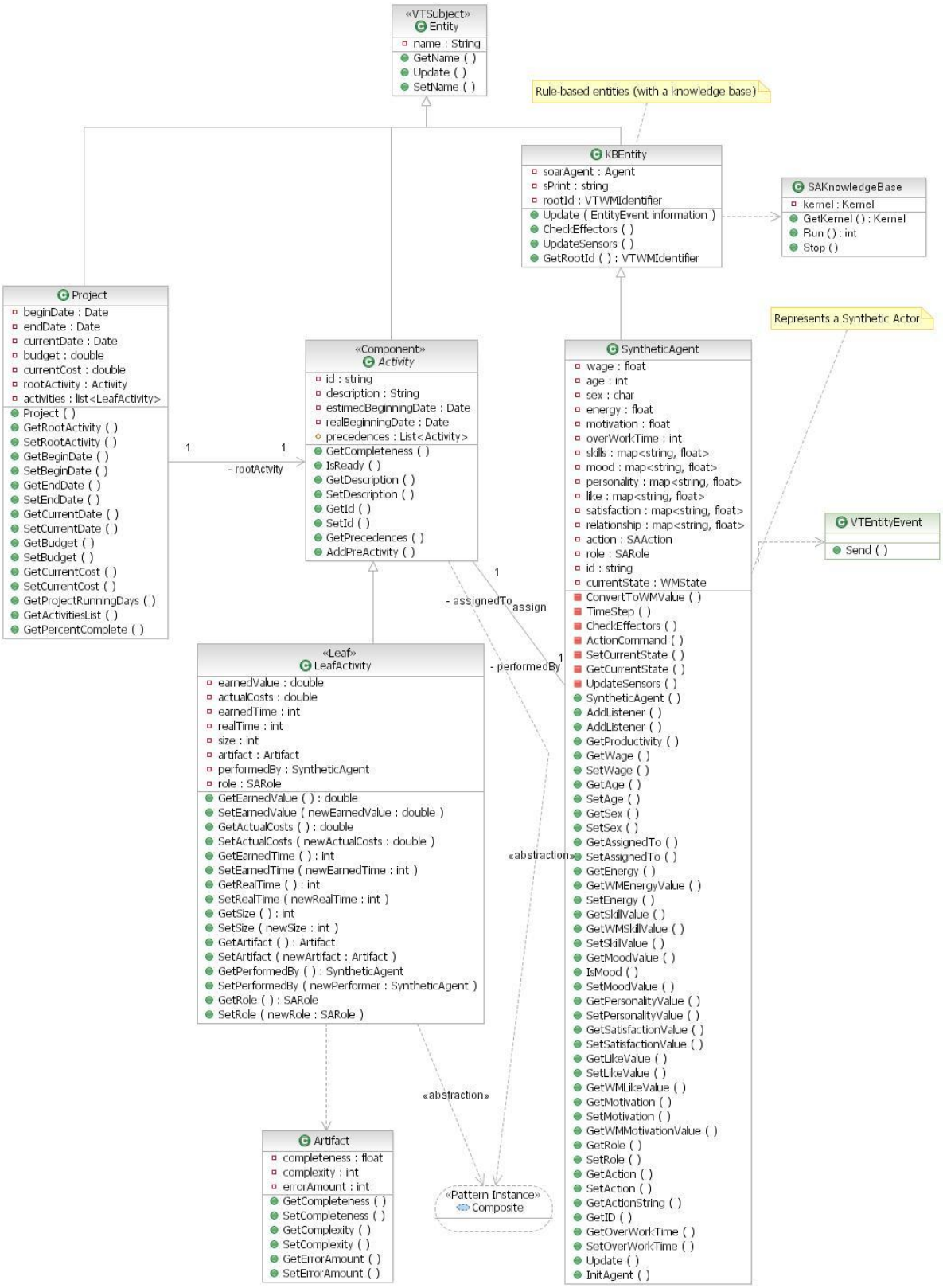

Figura 5.14 Classes do pacote virtualTeam.model

Todas as classes deste pacote herdam da classe *Entity*. Isso faz com que elas possam ser gerenciadas pelo *VTEntityManager*. A classe *Project* mapeia a instância do projeto em execução. Para a classe *Activity* foi aplicado um padrão de projeto chamado *Composite* [G+98], para suportar a relação hierárquica entre atividades, ou seja, guardar na mesma estrutura atividades e suas sub-atividades. Essas atividades também são reconhecidas na literatura como atividades sumário. Essas atividades não possuem recursos e produtos diretamente relacionados a ela, são formadas apenas por sub-atividades, que por sua vez podem ser atividades sumário ou atividades folha (*LeafActivity*). As classes *LeafActivity* representam as atividades que de fato consomem recursos e produzem artefatos. Elas possuem associadas a si os atores sintéticos responsáveis por sua execução. Estes atores sintéticos são representados por instâncias da classe *SyntheticAgent*. Os atores sintéticos, diferentemente das classes de *Project* e *Activity*, herdam da classes *KBEntity*. Esta classe foi criada para representar entidades que possuam alguma base de conhecimento associada. A base de conhecimento é representada pela classe *SAKnowledgeBase*.

A Figura 5.15 apresenta o pacote *virtualTeam.simulation*. Este pacote possui as classes responsáveis pela simulação. A classe *SimulationEngine* funciona como uma fachada para este pacote, sendo seu principal cliente a classe *SimulationController*. A troca de informações entre o ambiente de simulação e os atores sintéticos ocorre quando da execução do método *SimulateHour*. Este método recebe como parâmetros uma instância de *SimulationInputData*, que contém os dados a serem atualizados no ambiente de simulação, e uma instância da classe *SimulationOutputData*, que contém os dados do ambiente de simulação necessários ao projeto e aos atores. No caso específico deste projeto, os dados de entrada provém dos atores sintéticos, estando encapsulados na classe *SimulationAgentInputData* e dizem respeito a carga-horária, custo, capacidade produtiva e habilidades. Já os dados que interessam ao projeto e aos atores sintéticos dizem respeito ao andamento das atividades, encontrando-se encapsulados na classe *SumulationActivityOutputData*, contendo informações como erros, completude, custo atual e valor agregado de uma atividade.

A fim de gerar flexibilidade com relação a técnica de simulação, a implementação da classe *SimulationEngine* não é feita diretamente nela, mas através de algumas de suas classes "filhas". Aplica-se neste case o padrão de projeto *Strategy* [G+98]. Para o *VirtualTeam* foi criada uma classe para a simulação em dinâmica de sistemas, *SysDynSimulationEngine*.

A máquina de simulação utilizada é o HECTOR [Equ06], a mesma utilizada nos trabalhos de Barros [Bar01], Dantas [DBW04] e Veronese [Ver03]. Esta máquina de simulação encontra-se disponível através de uma DLL –*Dynamic Link Library*. As funções desta

Figura 5.15 Classes do pacote *virtualTeam.simulation*

DLL foram mapeadas na classe *MetaModeloProxy* que funciona como um *Proxy* para a DLL.

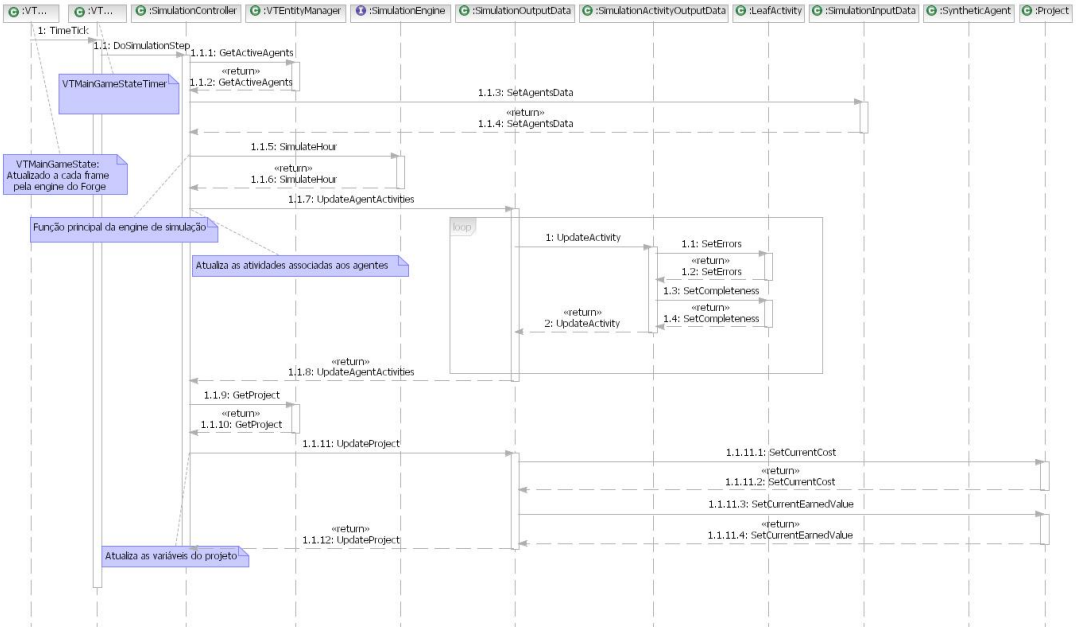

Figura 5.16 Looping principal responsável pela simulação

A Figura 5.16 apresenta o diagrama de sequência com as trocas de mensagens entre as classes apresentadas. Como ele é possível entender a seqüencias das ações no looping principal de simulação. O FORGE invoca a cada frame a classe *VTMainGameState*, que acontece várias vezes por segundo. Há um parâmetro que define uma razão entre o tempo passado no jogo e o tempo real, por exemplo, no VIRTUAL TEAM cada passo de simulação corresponde a 1 hora no tempo do jogo, mas no tempo real isto corresponde a 1 minuto em média. Com base na quantidade de vezes que ele é invocado pelo FORGE e na taxa de fps (*frames por segundo*) esta classe consegue determinar os momentos de invocar o método *TimeTick()* da classe *VTMainGameStateTimer*. Esta classe por sua vez invoca o método *DoSimulationStep* da classe *SimulationController*.

A *SimulatioControler* interage com a classe *VTEntityManager*, através do método *GetActiveAgents()*, afim de obter as informações sobre os atores sintéticos que serão utilizadas como parâmetro na simulação. Estes dados são encapsulados na classe *SimulationInputData* através do método *SetAgentsData()*. Logo após a classe *SimulationController* invoca o método *SimulateHour* da classe *SimulationEngine*, passando como parâmetros a classe *SimulationInputData*, com os dados dos atores sintéticos, e a classe *SimulationOutputData*, que conterá os dados de retorno da simulação.

A classe *SimulationEngine* encapsula dentro de si a chamada a DLL do HECTOR, responsável por executar o passo de simulação no ambiente da Dinâmica de Sistemas.

Após a execução do método a classe já contém todos os dados resultantes do novo passo de simulação e irá fazer a atualização dos atores sintéticos, no que diz respeito às suas atividades, fornecendo seus novos valores de erros e de completude das atividades.

Uma vez feito a atualização das atividades, através da comunicação com os atores sintéticos, é necessário atualizar os dados financeiros do projeto.

Uma vez terminada esta sequência, ela só será iniciada novamente, quando o *timer* indicar que é novamente o momento de atualização.

5.4 VALIDAÇÃO DO MODELO DE DOMÍNIO

No Capítulo 4 é apresentado um modelo do domínio de gerenciamento de projetos a ser utilizado pela simulação. Este modelo foi formalizado utilizando-se a linguagem definida por Barros [Bar01] para construção de meta-modelos. Barros [Bar01] e Veronese [Ver03] relatam que uma das principais dificuldades da simulação encontra-se na validação dos modelos apresentados.

As relações utilizadas para a criação do modelo de domínio aqui proposto na Seção 4.5, foram em parte retiradas de trabalhos correlatos como [Bar01], [AHM61], [Jon00] e parte retirada de suas definições no PMBOK™ *Guide* [Ins04]. No entanto, faz-se necessário confirmar se o comportamento deste modelo de domínio apresentar-se-á da forma esperada.

5.4.1 Restrições

O projeto SES, apresentado na Seção 5.2, possui ao todo 85 atividades de engenharia de software, que implementam o processo definido na Seção 4.2. São compostos por atividades que implementam a estrutura definida no modelo de domínio proposto. Em virtude da estrutura híbrida, em que parte do comportamento não ficará a cargo da simulação em dinâmica de sistemas, mas dos atores sintéticos, a estrutura do modelo de domínio apresentou apenas as classes *Project*, *Activity* e *HumanResource*, conforme pode ser observado na Seção 4.5. A classe *HumanResource* apresenta basicamente propriedades que são atribuídas a cada passo de simulação com os dados vindos dos atores sintéticos, portanto o seu comportamento foi suprimido do modelo de simulação.

O ambiente disponível para simulação é o HECTOR [Equ06], que fornece gráficos com o comportamento das variáveis ao longo dos passos de simulação. No entanto, o ambiente

de simulação não permite a interatividade, ou seja, não permite, por exemplo, que em determinado ponto da simulação haja uma parada, um parâmetro seja alterado, como a nível de produtividade, ou de habilidades, e a simulação seja continuada apartir do ponto de parada. Isto impõe limitação a validação do modelo a ser utilizado no jogo, na medida em que não será possível testar o modelo exatamente em suas condições de uso no jogo.

Barros [Bar01] criou em seu meta-modelo um mecanismo chamado de Cenário, no qual, é possível fazer a alterações nas equações do meta-modelo, mediante a ativação e desativação do cenário. O cenário uma vez ativado faz a modificação das equações até a sua desativação. No entanto, através do uso da interface gráfica do HECTOR é possível carregar e simular um modelo completo. Um modelo completo é formado pelo modelo de domínio e pelo modelo de projeto. A ativação do cenário é implementada no modelo de projeto, que deve estar completo no momento de carga do modelo, fazendo com que o cenário seja ativado desde o início da simulação. Desta forma não é possível pela interface gráfica do HECTOR a ativação de um cenário, que poderia provocar a alteração de propriedades ou equações no modelo de domínio em momentos específicos da simulação, o que poderia em parte simular o comportamento interativo do jogo.

5.4.2 Modelo Simplificado Equivalente

O modelo de domínio apresentado na Seção 4.5 é composto basicamente por três relações estruturais: *Project* → *Activity*; *Activity* → *Activity* e *Activity* → *HumanResource*. Desta forma é possível ao projeto saber as atividades que lhe pertencem, é possível à atividade saber seus predecessores e sucessores, é possível á atividade saber o seu responsável.

Para fins de teste do modelo de domínio, criou-se um modelo de projeto mais simplificado que mantivesse as mesmas características do modelo de projeto do SES. Para o modelo de domínio definido, basta que o modelo de projeto simplificado implemente as mesma relação estrutural, ou seja, basta que ele crie objetos que implementem as relações estruturais definidas. Neste caso o modelo mínimo necessário a esta validação é formado por duas atividades, que serão colocadas na relação de predecessora e sucessora uma da outra, um recurso-humano e um projeto.

```
DEFINE SistemaExemplo ProjectModel {

    EconomicShopping = NEW Project

    Rick = NEW HumanResource
```

```
        SET CostPerHour = 15;          # Custo por hora de trabalho
        SET Skill = 0;                 # Habilidade mínima
        SET CapProdIndex 1.0;          # Capacidade Produtiva

    LevantarUCs = NEW Activity
        LINK PerformedBy Rick;
        LINK ActProjRel EconomicShopping;

    LevantarAtores = NEW Activity
        SET PlannedStartTime = 50;
        LINK PerformedBy Rick;
        LINK Predecessors LevantarUCs;
        LINK ActProjRel EconomicShopping;

    ACTIVATE ProductivityLossDueAdminTask
        CONNECT AffectedActivity LevantarAtores;

};
```

São criadas duas atividades, na qual a "LevantarUCs"é predecessora de "LevantarAtores". Isso fará com que a atividades "LevantarAtores"só seja iniciada após o termino de sua predecessora. como não é atribuído a parte o valor do tamanho de cada atividade é de 100 pessoas-hora.

A observar as regras postadas no modelo de domínio, podemos ver que a completude da atividade é alterada a cada passo de simulação levando-se em consideração o índice de capacidade produtiva (CapProdIndex). Como este índice foi definido em 1,0 no modelo de projeto, então o comportamento apresentado na Figura 5.17 é o esperado.

Os erros são influenciados basicamente por dois fatores: *energy* e *skills*. A fim de ver o comportamento isolado das variáveis buscou-se definir a energia em um valor neutro de forma que sua influência no comportamento dos erros fosse anulada. Levando em consideração que:

$$EnergyFactor = 0.667 * (2 - PerformedBy.getEnergy)$$

e que $EnergyFactor$ deva ser igual a 1 para se atingir a neutralidade, então tem-se que: Energy deve ser igual a $0,5$.

Desta forma, o valor do *Skill* pode ser analisado de forma independente. A Figura 5.18 apresenta o comportamento do estoque *Error* da atividade $LevantarUC$ ao longo da simulação. Observa-se que o valor atinge o patamar de 13 pontos em 100 passos de simulação. Mais precisamente o valor atingido é de $13,34$, que corresponde ao valor

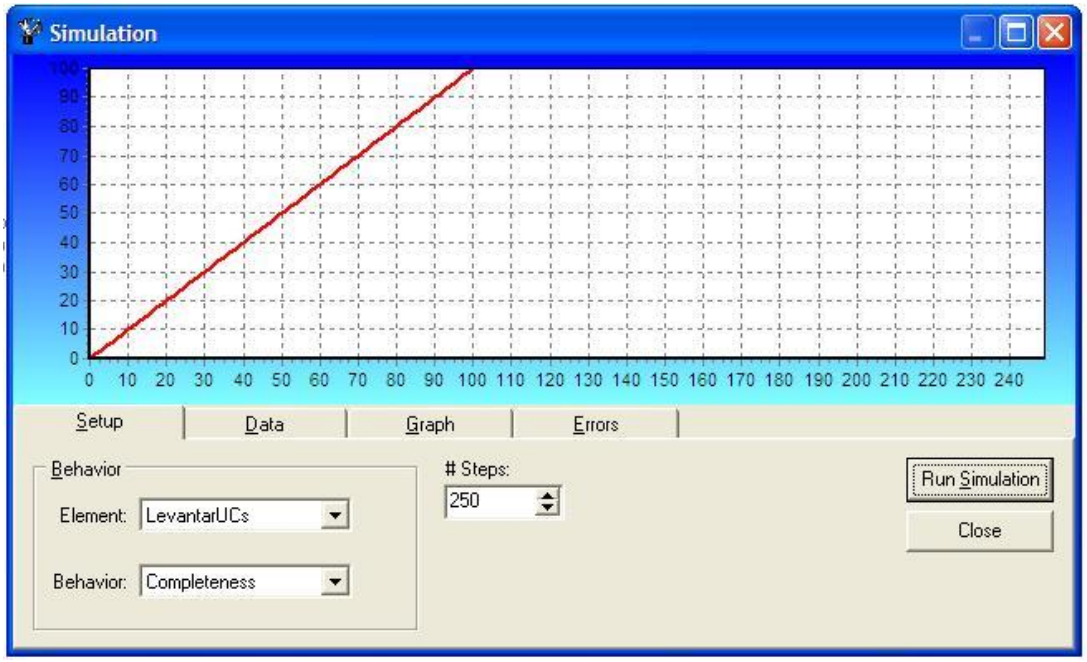

Figura 5.17 Comportamento da Completude do Caso de Uso "LevantarUC"na Simulação no HECTOR

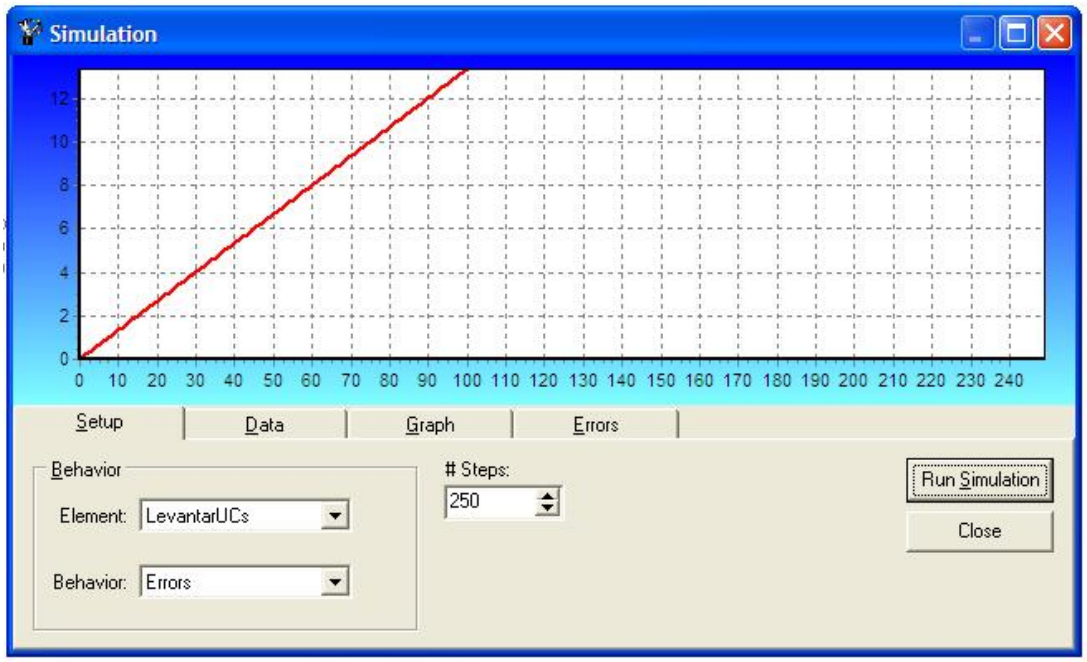

Figura 5.18 Comportamento dos Erros do Caso de Uso "LevantarUC"na Simulação no HECTOR

máximo definido pelo modelo para um desenvolvedor com habilidade mínima, ou seja, *Skill* zero.

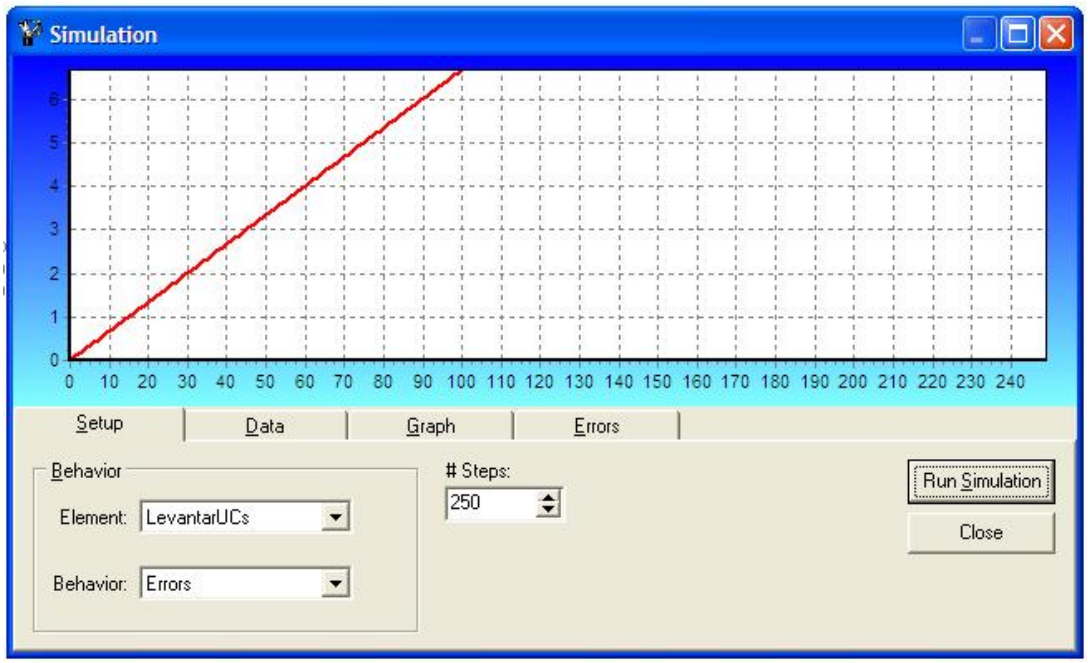

Figura 5.19 Comportamento dos Erros do Caso de Uso "LevantarUC"para o Skill Máximo na Simulação no HECTOR

A Figura 5.19 apresenta o comportamento do erro para o caso do *Skill* assumir valor máximo, ou seja, 1. Neste caso podemos observar que, conforme esperado, ele assume o patamar de 6 pontos, mais precisamente 6,67.

Na Figura 5.20 é apresentado o comportamento da variável completude para a atividade *LevantarAtores*. É importante observar que o seu início se dá apenas no passo 101, após o termino da anterior no passo 100. A esta atividade foi ativado um cenário, que reduz o valor da produtividade, devido às atividades administrativas. Por este motivo é possível observar que apenar de ter um *Size* de 100 ele leva mais de 120 passos para ser completa.

Os erros relacionados a esta atividade são semelhantes a sua predecessora, pois o cenário alterou apenas a capacidade produtiva, deixando o *Skill* e a *Energy* intactos.

O acompanhamento financeiro pode ser observador nas variáveis de análise de valor agregado que constam na classe *Project*.

O valor planejado reflete o fluxo de desembolso planejado para o projeto, desta forma ele leva em consideração a data planejada para o início da atividade. Caso a realização se dê após a data prevista observa-se atraso na atividade.

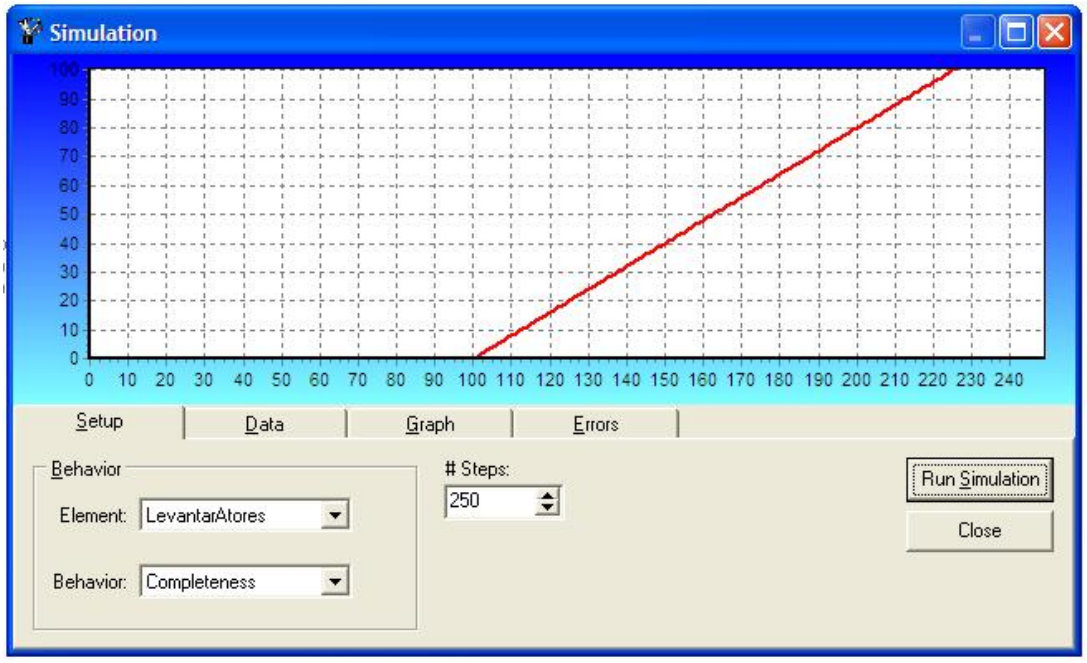

Figura 5.20 Comportamento da Completude do Caso de Uso "LevantarAtores"na Simulação no HECTOR

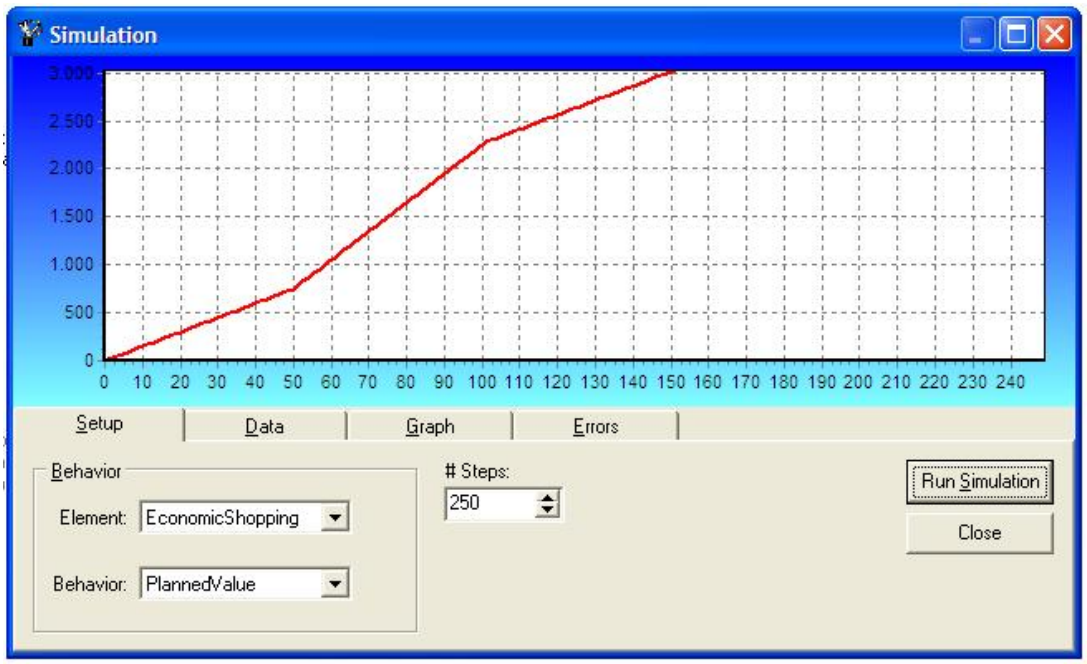

Figura 5.21 Comportamento do Valor Planejado na Simulação do HECTOR

A Figura 5.21 apresenta o comportamento da variável *PlannedValue* do projeto. No caso é possível observar que até o passo 50 há uma reta, que representa a contribuição da primeira atividade. No entanto a atividade *LevantarAtores* teve seu atributo *PlannedStartTime* fixado em 50, ou seja, planejou-se o seu início apartir do passo 50. Apartir deste ponto observa-se um aumento no ângulo de inclinação da reta, pois os valores passam a obter contribuição dos valores planejados para a segunda atividade. Esta inclinação se mantém até o passo 100, momento em que a primeira atividade acaba. A inclinação da reta volta ao valor anterior, pois trata-se do mesmo recurso alocado a execução da atividade, e este não teve seu custo por hora alterado durante a simulação. A simulação acaba no passo 150, pois por planejamento o tamanho da atividade seria de 100 passos. O valor planejado final é de $ 3.015,00.

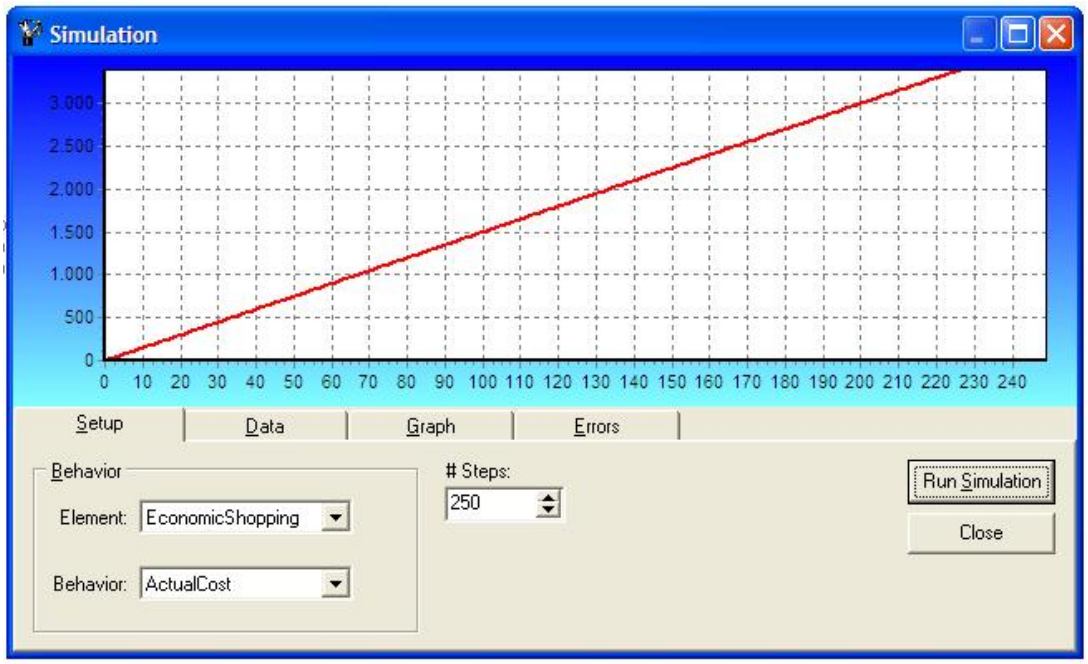

Figura 5.22 Comportamento do Custo Real na Simulação do HECTOR

A Figura 5.22 apresenta o comportamento da variável custo real. Ela representa o que de fato foi gasto com a alocação de recursos para a atividade, no caso deste modelo de domínio, é possível apenas a alocação de recursos-humanos. Ele não leva em consideração a produtividade do recurso, e sim a sua alocação. Por este motivo é possível ver os gastos do projeto sendo estendidos até que a segunda atividade termine. O custo real final é de $ 3.390,00.

A Figura 5.23 apresenta o comportamento da variável valor agregado. esta variável leva em consideração o que foi de fato produzido, ou seja, se desenvolvedor está com

o índice de capacidade produtiva menor que 1, então no próximo passo o avanço da atividade será menor. Por exemplo, um índice de produtividade em torno de 80% em um dia de trabalho irá agregar apenas 6,4 horas de valor agregado. É possível observar que a reta sofre um alteração em seu ângulo de inclinação apartir do passo 100. Isto acontece, pois na primeira atividade não há o acionamento do cenário que baixa a produtividade. Apartir da segunda atividade, o índice de capacidade produtiva é alterado, portanto a inclinação diminui.

Com esse pequeno modelo é possível perceber que o modelo de domínio criado representa a realidade de gerenciamento que se deseja representar, sendo porém de verificação limitada dada as restrições apresentadas na Seção 5.4.1.

5.5 COMPARAÇÃO COM O JOGOS ESTUDADOS

Na Seção 3.6 foram apresentados alguns jogos que tem objetivos muito próximos ao Virtual Team. Apesar desta proximidade é possível observar claras diferenças em seus objetivos e abordagens, resumidas a seguir:

- SESAM: Objetiva o ensino do gerenciamento de projetos, através de interface textual e disponibiliza uma linguagem gráfica para a construção das regras que farão parte do modelo;

- SimSE: Objetiva o ensino de engenharia de software, através de uma interface gráfica simples sem animações. Disponibiliza uma ferramenta para criação do modelo de engenharia de software a ser simulado. Os modelos criados apresenta grande proximidade com a realidade dos projetos de software, porém não aborda outras questões relacionadas a gestão de projetos;

- Tim: Objetiva o ensino de gerenciamento de projetos em interface gráfica simples, porém com animações. Utiliza a dinâmica de sistemas para a simulação, porém faz uso de uma meta-linguagem para criação do modelo a ser simulado. O modelo de engenharia de software utilizado é bastante simplificado, não sendo aplicável à realidade das organizações;

- Virtual Team: Objetiva o ensino de gerenciamento de projetos voltados a gestão de pessoas. Utiliza linguagem gráfica avançada, própria de jogos de entretenimento com animação dos personagens. Utiliza tecnologia híbrida de simulação, permitindo o uso de dinâmica de sistemas e de atores sintéticos. Processo de gerenciamento

de projetos focado nos processos de acompanhamento e controle, e processo de engenharia de software atual.

Desta forma é possível ter uma idéia do posicionamento do VIRTUAL TEAM em relação a ferramentas consideradas pares.

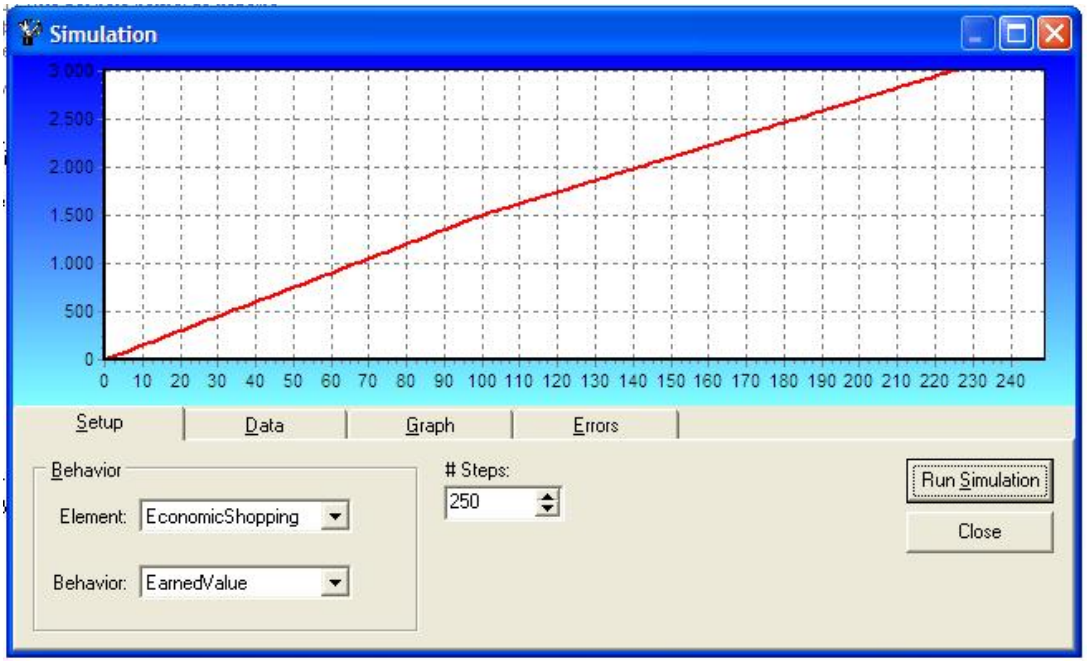

Figura 5.23 Comportamento do Valor Agregado na Simulação do HECTOR

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

A capacitação em gerenciamento de projetos vem sendo objeto de grande atenção por parte de empresas dos mais diversos setores, dado a comprovação dos resultados positivos que a utilização de pessoas bem qualificadas e processos bem definidos trazem para as organizações.

O uso de jogos de negócio em ambientes de capacitação empresarial tem sido cada vez mais freqüente. Uma das tecnologias mais utilizadas para a criação dos modelos utilizados nos jogos de negócios é a dinâmica de sistemas, pois com ela é possível representar e simular a lógica sistêmica de uma empresa ou de um segmento de mercado, por exemplo, dado que as teorias de administração modelam esses contextos de forma sistêmica. No entanto, a capacitação em gerenciamento de projetos faz uso de um corpo de conhecimento que aborda aspectos não apenas sistêmicos, mas também individuais e particulares, como a gestão de pessoas. Desta forma a criação de um jogo de negócio voltado a capacitação de gerentes de projetos deve levar em consideração não apenas aspectos macro ou genéricos de comportamento do projeto, mas também aspectos micro ou específicos, dado que esta é a realidade dos projetos.

A criação de mecanismos que aproximem o ambiente de experimentação vivenciado no jogo da realidade é importante para o uso dos jogos como ferramenta complementar de capacitação, pois aumenta a qualidade das decisões e ações tomadas durante as jogadas.

O uso de atores sintéticos como meio para se aproximar os jogos de negócio da realidade em casos como o de capacitação em gerenciamento de projetos permite que os aspectos individuais sejam modelados e apresentados na interface.

O uso combinado destas duas tecnologias permite atender aos requisitos existentes em um projeto real, que envolvem aspectos sistêmicos, modelados dentro da dinâmica de sistemas, e aspectos individuais, modelados dentro dos atores sintéticos. A utilização destas duas tecnologias só é possível caso haja uma arquitetura que permita a integração de informações entre os dois ambientes, sincronizada com as situações demandadas pelos cenários do jogo. Esta arquitetura foi definida e validada pelo jogo Virtual Team.

A aproximação do ambiente apresentado pelo jogo com a realidade não acontece apenas em função de seu modelo arquitetural ou tecnologia de implementação mas, princi-

palmente, pelos cenários e desafios apresentados ao jogador. Neste sentido, foi necessário criar um processo de software que seguisse metodologias de fato utilizadas no mercado de desenvolvimento de sistemas.

O processo de software proposto neste trabalho dá suporte à execução de um projeto, que será objeto de simulação por parte da equipe de desenvolvimento disponibilizada pelo VIRTUAL TEAM, no entanto as regras que nortearão a simulação necessitam estar descritas em um modelo de domínio. O modelo de domínio proposto, apresenta algumas simplificações em relação a trabalho correlatos, pois parte do comportamento do jogo foi retirada do modelo de domínio e colocada na base de regras dos atores sintéticos, mas também apresenta inovações como a inclusão das variáveis de análise de valor agregado do projeto.

A aproximação com a realidade está também relacionada às situações que são apresentadas ao jogador e às ações que este pode efetuar no jogo. Estas situações são modeladas por objetivos, desafios e comandos disponíveis no jogo. Com a finalidade de orientar a criação destes cenários e comandos, foi proposto um processo de gerenciamento de projetos, que reúne os processos [Ins04] aos quais o jogador, no papel de gerente de projetos, deve executar afim de que os objetivos do projeto sejam atingidos.

Desta forma, podemos pontuar as contribuições deste trabalho em:

- Criação de um modelo arquitetural de integração dos paradigmas de atores sintéticos e de dinâmica de sistemas;
- Criação de um modelo de domínio de gerenciamento de projetos de software que atendesse aos requisitos do projeto SmartSim e do jogo VIRTUAL TEAM;
- Definição de um processo de desenvolvimento de software baseado em metodologias atuais de forma a aproximar o jogo da realidade; e
- Definição de um processo de gerenciamento de projetos a ser utilizado no jogo VIRTUAL TEAM, de forma a capacitar o jogador nos processos nele incluso, através da orientação na criação de cenários, desafios e objetivos para o jogo.

6.1 LIMITAÇÕES

As limitações identificadas encontram-se principalmente relacionadas às dificuldades na validação do modelo. Não apenas as dificuldades já mencionadas com relação a verificação dos modelos de simulação, mas também pelo fato de que, dada a opção de utilização de atores sintéticos, como mecanismo de implementação dos membros da equipe, inseriu-se

características não determinísticas, ou seja, parte do comportamento do modelo, entendendo agora modelo como a soma dos conhecimentos modelados no modelo de domínio e da base de regras dos atores sintéticos, apresentará comportamento não preditivo devido às heurísticas e fatos existentes nos atores sintéticos. A verificação então deve acontecer através de pesquisas qualitativas junto ao público-alvo.

A proposta com o VIRTUAL TEAM é a de criação um protótipo de jogo de capacitação em gestão de pessoas com o uso de atores sintéticos, com o objetivo de gerar conhecimento para a criação de um *framework* para construção de jogos de negócios com uso de atores sintéticos. Por este motivo, a proposta de arquitetura de integração não pode ser considerada genérica, visto que sua aplicação está, até o presente momento, vinculada a criação do VIRTUAL TEAM. No entanto, ela representa uma proposta concreta e funcional da integração destas duas tecnologias de implementação de jogos.

Assim como todo modelo, que possui aplicação contextual dentro de uma realidade, os aqui propostos e utilizados na criação do VIRTUAL TEAM, não são de uso genérico, ou seja, suas construções obedeceram a características de um cenário específico. Portanto, o reuso destes modelos em outros ambientes, que não o do jogo, está condicionado a existência das mesmas condições nestes ambientes.

6.2 TRABALHOS FUTUROS

O VIRTUAL TEAM não se apresenta completo já seu objetivo inicial era a criação de um protótipo. No entanto, este protótipo continua em desenvolvimento, para o qual imagina-se possíveis caminhos abaixo relacionados:

- Extensão dos objetivos do jogo para exploração de cenários fora da gestão de pessoas, como por exemplo, comunicação, negociação, qualidade, entre outros;
- Implementação da fase de planejamento do projeto permitindo que o jogador não apenas seja responsável pela condução na execução do plano de projeto definido, mas também seja responsável pela criação do plano do projeto, com a definição das atividades constantes do projeto, suas estimativas, seus planejamentos de fases, etc. Esta fase de planejamento pode ser vista como uma espécie de editor de jogo, para a fase subseqüente, que é a de execução do projeto;
- Os processos de desenvolvimento de software e de gerenciamento de projeto criados para o VIRTUAL TEAM podem ser processos reais de uma empresa. A implantação de processos de software e de gerenciamento de projetos nas organizações vem obedecendo ao conceito de modelos de maturidade, na qual um modelo, normalmente

complexo vai sendo implantado de forma parcial e incremental. O processo de gerenciamento de projetos utilizados no jogo, pode estar relacionado aos processos reais e aos níveis de maturidade em gerenciamento de projetos, fazendo com que o jogo em um nível mais avançado considere por exemplo o cenário de ambientes multi-projetos, enquanto os níveis mais simples considera apenas uma equipe e com uma quantidade menor de processos [Ins04];

- Acompanhamento do projeto simulado no jogo através de ferramentas reais de acompanhamento de projetos através da integração do jogo com as bases de dados destas ferramentas;

- Criação de um subsistema de *log* das ações do jogador e do contexto em que estas ações foram tomadas, para que seja possível ao jogador refletir posteriormente sobre as ações e os efeitos da mesma no jogo;

- Criação de um tutor virtual que faça o papel do facilitador em sala de aula, permitindo o uso educacional da ferramenta em ambientes em que não haja um facilitador por perto, dentro de um contexto de educação a distância, a semelhança do agente VICTOR criado no projeto PMK por Torreão [Tor05].

- Criação de um agente baseado em conhecimento que seja capaz de capturar as experiências com as jogadas de diversos jogadores e que funcione como um consultor virtual. Este papel pode estar associado ao de tutor virtual ou não.

Este trabalho procurou apresentar mecanismos de evolução para a construção de uma ferramenta que contribua para a melhora do processo de capacitação em gestão de projetos, de forma a permitir que estes sejam cada vez mais eficazes, menos custosos, menos arriscados e de maior qualidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[A+04] A. Abran et al., editors. *Guide to the software engineering body of knowledge: SWEBOK*. IEEE Computer Society, Los Alamitos, California, 2004.

[AHM61] T. Abdel-Hamid and S.E. Madnick. *Software Project Dynamics: An Integrate Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1961.

[Ara95] R. Araújo. Construção gráfica de processos de desenvolvimento de software e geração de uma ontoloconstrução gráfica de processos de desenvolvimento de software e geração de uma ontologia de processos de softwaregia de processos de software, 1995. Trabalho de Graduação.

[Bar01] M. Barros. *Gerenciamento de Projetos Baseados em Cenários*. PhD thesis, UFRJ, December 2001.

[BKS03] Chrissis. M. B., M. Konrad, and S. Shrum. Cmmi guidelines for process integration and product improvement, 2003.

[Boe86] B. Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986.

[Cen99] Centre for Virtual Environment, University of Salford. Salford - United Kingdom. *Personality Centered-Agents for Virtual Computer Games. Proceeded in Workshop Intelligent Virtual Agents*, Proceeded in Workshop Intelligent Virtual Agents, September 1999.

[cha99] Chaos: A recipe for success. http://www.standishgroup.com/sample_research/PDFpages/chaos1999.pdf, 1999. Acessado em 12/06/2006.

[DBW04] A. Dantas, M. Barros, and C. Werner. Treinamento experimental com jogos de simulação para gerentes de projetos de software. volume 1 of *18o. Simposio Brasileiro de Engenharia de Software*, pages 32–38, 2004.

[DDL95] D. Drappa, A. Deininger, and M. Ludewig. Modeling and simulation of software projects. In *Proceedings of the Twentieth Annual Software*, pages 269 – 275, Greenbelt, MD (USA), 1995.

[Dug04] J. Duggan. System dynamics simulation for software development, 2004.

[Equ06] Equipe de Reutilização de Software. Hector - compilador do metamodelo da dinâmica de sistemas. http://reuse.cos.ufrj.br/site/pt/index.php?option=com_content&task=view &id=17&Itemid=27, 2006. Último acesso em 15/08/2006.

[Fal98] R. Falbo. *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*. PhD thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1998.

[For61] 1961 Forrester, J.W. *Industrial Dynamics*. MIT Press, Cambridge, MA, 1961.

[FSF99] Inc. Free Software Foundation. Gnu lesser general public license. http://www.gnu.org/licenses/lgpl.html, February 1999. Ultimo acesso em 10/07/2006.

[G+98] E. Gamma et al. *Design Patterns*. Addison Wesley Longman Inc., 1998.

[Gab06] L. O. S. Gabardo. Jogos de empresas - uma metodologia de utilização. *Revista Conhecimento Interativo*, 2(1):88–100, 2006.

[Gre88] C. S. Greenblat. *Designing Games and Simulations*. Number 2. Sage Publications, Inc., USA, 1988.

[Gro05] Object Management Group. Software process engineering metamodel specification. http://www.omg.org/technology/documents/formal/spem.htm, 2005. Especificação adotada como padrão da OMG em 2005.

[Gur06] I. Gurgel. Processo de criação de personagens: Um estudo de caso no jogo sério simgp, 2006. Projeto de Graduação em Design.

[Han96] G.A. Hansen. Simulating software development processes. *IEEE Computer*, 29(1):73–77, January 1996.

[HK89] W. S. Humphrey and M. I. Kellner. Software process modeling: principles of entity process models. In *ICSE '89: Proceedings of the 11th international conference on Software engineering*, pages 331–342, New York, NY, USA, 1989. ACM Press.

[Ins04] Project Management Institute, editor. *A guide to the Project Management body of knowledge: PMBOK Guide*. Project mamagement Institute, Inc, third edition, 2004.

[JB+98] I. Jacobson, , G. Booch, , and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1998.

[Joh04] S. Johnston. Rational uml profile for business modeling. http://www-128.ibm.com/developerworks/rational/library/5167.html, July 2004.

[Jon00] C. Jones. Software assessments, benchmarks, and best practices, 2000. Information Technology Series, Reading, MA: Addison-Wesley Publishing Co.

[Jär06] A. Järvinen. Games without frontiers:theories and methods for game studies & design. Último acesso em 05/08/2006, 2006.

[Kar93] G. Karner. Resource estimation for objectory projects. http://www.bfpug.com.br/Artigos/UCP/Karner - Resource Estimation.doc, September 1993. Último acesso em 31/05/2006.

[KMR99] M.I. Kellner, R.J. Madachy, and D.M Raffo. Software process simulation: Why? what? how? *Journal of Systems and Software*, 46(2/3), 1999.

[L+99] C.Y. Lin et al. Computer-aided softwar development process design. *IEEE Transactions on Software Engineering*, 15(9):1025–1037, 1999.

[Lud89] J. Ludewig. Modelle der software-entwicklung - abbilder oder vorbilder? softwaretechnik-trends. *Softwaretechnik-Trends*, 9(3):1–12, 1989. In German.

[Mad01] C. Madeira. Forge v8: Um framework para o desenvolvimento de jogos de-computador e aplicações multimídia. Master's thesis, Centro de Informática, Universidade Federal de Pernambuco, 2001.

[Mar97] L. A. Martin. The first step. Technical report, MIT System Dynamics Group, Cambridge, MA, December 1997.

[MB99] R.J. Madachy and B.W. Boehm. *Software Process Dynamics*. IEEE Computer Society Press, Los Alamitos, CA, 1999. Early Draft Version 4/99,disponível na URL http://www-rcf.usc.edu/ madachy/spd.

[Mer96] D. Merril. Training software development project managers with a software project simulator, 1996. Master of Science Thesis Proposal. Arizona State University. Tempe, USA.

[MGW98] A. I. Martínez-García and B. C. Warboys. From RADs to DESs: a mapping from process models to discrete event simulation. In *Proceedings of the Software Process Simulation Modeling (ProSim) Workshop'98*, Portland, OH, 1998.

[Min00] Henry Mintzberg. *Safári de estratégia: um roteiro pela selva do planejamento estratégico*. Bookman, 2000.

[OV06] S. R. B. Oliveira and A. M. L. Vasconcelos. Publicação do processo de software no imppros: Um ambiente de implementação progressiva do processo de software. *Revista Traços da UNAMA*, 10(16), 2006.

[P+03] D. Pfahl et al. An externally replicated experiment for evaluating the learning effectiveness of using simulations in software project management education. *Empirical Software Engineering*, 8:367 – 396, 2003.

[Per03] Henry. Perros. Computer simulation technics: The definitive introduction, 2003.

[Pre04] Microsoft Press, editor. *Microsoft Visual C++ 7.1 .NET Deluxe*. Microsoft, 2004.

[R+83] N. Roberts et al. *Introduction to Computer Simulation*. Addison-Wesley, Reading, MA, 1983.

[Rad] M. J. Radzicki. Introduction to system dynamics. http://www.albany.edu/cpr/sds/DL-IntroSysDyn/index.html. Consultado em 03/04/2006.

[Rat06] IBM Rational. Rational unified process. Base Plug-In 1.0, 2006.

[RCL98] I. Rus, J. Collofelo, and P. Lakey. Software process simulation for reliability strategy assessment. In *The Proceedings of the International Workshop on SoftwareProcess Simulation Modeling - ProSim'98*, Portland, OH, 1998.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[Rob05] S. P. Robbins. *Comportamento Organizacional*. PEARSON EDUCATION, 2005.

[Roy87] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87: Proceedings of the 9th International Conference on Software Engineering*, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[Rus95] S. J. Russell. *Artificial Intelligence: a modern approach*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1995.

[RW97] A. G. Rodrigues and T. M. Williams. System dynamics in software project management: Towards the development of a formal integrated framework. *European Journal of Information Systems,*, 6(1):51–66, March 1997.

[Sca99] W. Scacchi. Experience with software process simulation and modeling. http://www.usc.edu/dept/ATRIUM/Papers/JSS98/JSS98.html, 1999. to appear in Journal of Systems and Software, 1999.

[Sis98] T. Sisk. History of project management. http://office.microsoft.com/downloads/9798/projecthistory.aspx, 1998. Acessado em 25/01/2006.

[Sma06] SmartSim Research Group. Smartsim:business game com atores sintéticos. http://www.cin.ufpe.br/ smartsim/portugues.html, 2006. Consultado em 18/07/2006.

[Sof06] Softex. Mps.br Melhoria de Processo do Software Brasileiro: Guia geral (versão 1.1). http://www.softex.br/cgi/cgilua.exe/sys/start.htm?infoid=7629&sid=211, March 2006. Último acesso em 08/06/2006.

[Som04] I. Sommerville. *Software Engineering*. Addison Wesley, 7th edition, 2004.

[Sta88] J.D. Starman. A skepitc´s guide to computer models. Technical Report D-1401-1, School of Management, MIT, Cambridge, Massachussets, USA, 1988.

[Ste00] J. Sterman. *Business Dynamics: System Thinking and Modeling for a Complex World*. McGraw-Hill Higher Education, 2000.

[Tor05] P. Torreão. Project management knowledge learning environment: Ambiente inteligente de aprendizado para educação em gerencimento de projetos. Master's thesis, CIn, UFPE, Recife, Pernambuco, March 2005.

[vdH04] Emily Oh Alex Baker André van der Hoek. Teaching software engineering using simulation games. In *ICSE '04:Proceeding of School of Information and Computer Science*, 2004.

[Ver03] G. Veronese. Sistematização do desenvolvimento de jogos de simulação para treinamento. Master's thesis, Universidade Federal do Rio de Janeiro, COPPE, 2003.

[WP99] M. M. Woolfson and G. J. Pert. *An Introdution to Computer Simulation*. Oxford University Press, Department of Physics, University of York, 1999.

APÊNDICE A

MODELO DE DOMÍNIO EM GERENCIAMENTO DE PROJETOS UTILIZADO NA SIMULAÇÃO

Este apêndice apresenta o modelo de domínio em gerenciamento de projetos utilizado pelo VIRTUAL TEAM em sua totalidade. Sua escrita faz uso da linguagem de descrição de meta-modelo definida por Barros [Bar01].

```
ProjectModel {
    CLASS Project {
        STOCK ActualTime 0;
        PROC getActualTime ActualTime;
        RATE (ActualTime) RTActualTime DT;
        STOCK PlannedValue 0;
        STOCK EarnedValue 0;
        STOCK ActualCost 0;
    };

    CLASS Activity {
        PROPERTY Size 100;
        PROPERTY PlannedStartTime 0;
        PROC PlannedRunningTime ActProjRel.getActualTime - PlannedStartTime;
        RATE (ActProjRel.PlannedValue) RTPlannedValue
            IF(AND(PlannedRunningTime >= 0, PlannedRunningTime <= Size),
            #then
                DT * PerformedBy.getCostPerHour,
            #else
                0);
        STOCK Errors 0;
        # [mid: 1; min: 2/3(0.667); max: 4/3(1.334)]
        PROC EnergyFactor 0.667 * (2 - PerformedBy.getEnergy);
        PROC SkillFactor 0.667 * (2 - PerformedBy.getSkill);
        # [mid: 0.1; min: 0.045; max: 0.0178]
        PROC ErrGenFactor 0.1 * (EnergyFactor*SkillFactor);
        RATE (Errors) RTErrors
            IF(IsReady,
```

```
            #then
                ErrGenFactor,
            #else
                0);
        STOCK Completeness 0;
        PROC IsComplete Completeness >= Size;
        RATE (Completeness) RTCompleteness
            IF(IsReady,
            #then
                PerformedBy.getCapProdIndex,
            #else
                0);

        RATE (ActProjRel.EarnedValue) RTEarnedValue
            IF(IsReady,
            #then
                PerformedBy.getCapProdIndex * PerformedBy.getCostPerHour,
            #else
                0);
        #PROC getEarnedValue Completeness * PerformedBy.getCostPerHour;
        RATE (ActProjRel.ActualCost) RTActualCost
            IF(IsReady,
            #then
                PerformedBy.getCostPerHour,
            #else
                0);
        # Determine if the activity is concluded
        PROC IsConcluded Completeness >= 100;
        # Determine if precedent activities are concluded
        PROC IsAllDepsConcluded GroupMax (Predecessors, IsConcluded) >= 0;
        # Determine if the activity is ready to run
        PROC IsReady AND (IsAllDepsConcluded, NOT(IsConcluded));
    };

    CLASS HumanResource {
        PROPERTY WorkingHoursPerDay 8;    # Jornada diária de trabalho
        PROPERTY OverWorkTime 0;          # Horas Extras
        PROPERTY CostPerHour 10.00;       # Custo por hora normal de trabalho
        PROPERTY CapProdIndex 0.8;        # Capacidade Produtiva
        PROPERTY Skill 0.8;               # Habilidade

        PROC getWorkingHoursPerDay WorkingHoursPerDay;
        PROC getOverWorkTime OverWorkTime;
        PROC getCapProdIndex CapProdIndex;
```

```
        PROC getCostPerHour CostPerHour;
    };
# Relationships #
    RELATION PerformedBy Activity, HumanResource;
    MULTIRELATION Predecessors Activity, Activity (Successors);
    RELATION ActProjRel Activity, Project;
};
# Scenarios #
 SCENARIO ProductivityLossDueAdminTask ProjectModel {
    CONNECTION AffectedActivity Activity {
        PROC ProductivityLossFactor 0.2;   # 0.3 equal to 30%
        AFFECT RTCompleteness RTCompleteness * (1 - ProductivityLossFactor);
        AFFECT RTEarnedValue RTEarnedValue * (1 - ProductivityLossFactor);
    };
};

DEFINE SistemaExemplo ProjectModel {
};
```

APÊNDICE B

ESTRUTURA DE DEFINIÇÃO DO PROCESSO PADRÃO NO IMPPROS

Este apêndice apresenta o meta-modelo definido por Oliveira em [OV06] para a definição de processo no ambiente em ImpPros.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE ProcessoInstanciadoImpproS [

<!ELEMENT processoinstanciado (processo*, modelociclovida)>
   <!ATTLIST processoinstanciado
      nome         CDATA  #REQUIRED
      descricao    PCDATA #REQUIRED
      nivel        CDATA  #FIXED    "Processo Instanciado"
      situacao     CDATA  #REQUIRED
      organizacao PCDATA #REQUIRED
      version      CDATA  #REQUIRED
      data         CDATA  #REQUIRED>

<!ELEMENT processo (atividade*)> <!ATTLIST processo
      nome         CDATA  #REQUIRED
      descricao    PCDATA #REQUIRED
      origem       CDATA  #REQUIRED
      infoimpl     PCDATA #REQUIRED
      restricao    PCDATA #REQUIRED
      situacao     CDATA  #REQUIRED>

<!ELEMENT modelociclovida (fase*)> <!ATTLIST modelociclovida
      nome         CDATA  #REQUIRED
      tipo         CDATA  #REQUIRED>

<!ELEMENT fase (atividade*)> <!ATTLIST fase
      nome         CDATA  #REQUIRED
      ordem        CDATA  #REQUIRED
```

```
      niteracoes  CDATA  #REQUIRED>

<!ELEMENT atividade (superatividade, subatividade*, preatividade*, posatividade*,

artefatoentrada*, artefatosaida*, procedimento*, recursohardware*, recursosoftware*,

recursohumano*)> <!ATTLIST atividade
      nome        CDATA  #REQUIRED
      descricao   PCDATA #REQUIRED
      tipo        CDATA  #REQUIRED
      origem      CDATA  #REQUIRED
      restricao   PCDATA #REQUIRED>

<!ELEMENT superatividade CDATA>

<!ELEMENT subatividade CDATA>

<!ELEMENT preatividade CDATA>

<!ELEMENT posatividade CDATA>

<!ELEMENT artefatoentrada CDATA>

<!ELEMENT artefatosaida CDATA>

<!ELEMENT procedimento CDATA>

<!ELEMENT recursohardware CDATA>

<!ELEMENT recursosoftware CDATA> <!ELEMENT recursohumano CDATA>

]>
```

APÊNDICE C

XML *SCHEMA* DE DEFINIÇÃO DO PROCESSO PARA O *VIRTUAL TEAM*

Este apêndice apresenta o XML *Schema* de definição do processo utilizado no VIRTUAL TEAM. Ele representa uma extensão em relação ao modelo de Oliveira [OV06], apresentado no Apêndice B. O detalhamento das estruturas estendidas e suas justificativas são apresentadas na Seção 4.4.

```
<?xml version="1.0"
encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="id" type="xs:ID"/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attributeGroup name="id_name">
    <xs:attribute ref="id" use="required"/>
    <xs:attribute ref="name" use="required"/>
  </xs:attributeGroup>

  <xs:element name="processoinstanciado">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="processo"/>
          <xs:element ref="modelociclovida"/>
      </xs:sequence>
      <xs:attribute name="nome" use="required"/>
      <xs:attribute name="descricao" use="required"/>
      <xs:attribute name="nivel" type="xs:string" fixed="Processo Instanciado"/>
      <xs:attribute name="situacao" use="required"/>
      <xs:attribute name="organizacao" use="required"/>
      <xs:attribute name="version" use="required"/>
      <xs:attribute name="data" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="processo">
    <xs:complexType>
```

```
    <xs:sequence>
      <xs:element ref="artefato" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tech-skill" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="role" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element minOccurs="0" maxOccurs="unbounded" ref="atividade"/>
    </xs:sequence>
    <xs:attribute name="nome" use="required"/>
    <xs:attribute name="descricao" use="required"/>
    <xs:attribute name="origem" use="required"/>
    <xs:attribute name="infoimpl" use="required"/>
    <xs:attribute name="restricao" use="required"/>
    <xs:attribute name="situacao" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="modelociclovida">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="fase"/>
    </xs:sequence>
    <xs:attribute name="nome" use="required"/>
    <xs:attribute name="tipo" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="fase">
  <xs:complexType>
     <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="atividade-ref"
        type="xs:IDREF"/>
     </xs:sequence>
     <xs:attribute name="nome" use="required"/>
     <xs:attribute name="ordem" use="required"/>
     <xs:attribute name="niteracoes" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="atividade">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="1" ref="superatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="preatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="posatividade"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="artefatoentrada"/>
```

```
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="artefatosaida"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="procedimento"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursohardware"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursosoftware"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="recursohumano"/>
      </xs:sequence>
      <xs:attributeGroup ref="id_name"/>
      <xs:attribute name="descricao"/>
      <xs:attribute name="tipo"/>
      <xs:attribute name="origem"/>
      <xs:attribute name="restricao"/>
   </xs:complexType>
</xs:element>

<!-- adicionado elemento artefato para relacionamento com as atividades-->
<xs:element name="artefato">
  <xs:complexType>
      <xs:attributeGroup ref="id_name"/>
        <xs:attribute name="size" type="xs:byte" use="required"/>
      <xs:attribute name="complexity" type="xs:byte" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="superatividade" type="xs:IDREF"/>
<xs:element name="subatividade" type="xs:IDREF"/>
<xs:element name="preatividade" type="xs:IDREF"/>
<xs:element name="posatividade" type="xs:IDREF"/>
<xs:element name="artefatoentrada" type="xs:IDREF"/>
<xs:element name="artefatosaida" type="xs:IDREF"/>
<xs:element name="procedimento" type="xs:string"/>
<xs:element name="recursohardware" type="xs:string"/>
<xs:element name="recursosoftware" type="xs:string"/>
<!-- detalhamento deste elemento para inclusão de Roles -->
<xs:element name="recursohumano">
  <xs:complexType>
      <xs:sequence>
         <xs:element name="tech-skill-level" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:attribute name="tech-skill-ref" type="xs:IDREF" use="required"/>
                  <xs:attribute name="level" type="xs:byte" default="3"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="role-ref" type="xs:IDREF"/>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name="tech-skill">
     <xs:complexType>
        <xs:attributeGroup ref="id_name"/>
     </xs:complexType>
  </xs:element>

  <xs:element name="role">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="tech-skill-ref" type="xs:IDREF" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="id_name"/>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

APÊNDICE D

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DO *VIRTUAL TEAM*

Este apêndice apresenta a formalização, em formato XML, do processo de desenvolvimento de software que foi apresentado na Seção 4.2. Esta estrutura segue o meta-modelo apresentado no Apêndice C.

```
<?xml version="1.0" encoding="UTF-8"?>
    <processoinstanciado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="procInstImpprosSmartSim2.xsd"
    nome="SmartSim"
    descricao="SmartSim Software Process Instance"
    situacao=""
    organizacao="Cin - UFPE"
    version=""
    data="2006-03-22">

    <processo nome="??" descricao="??" origem="" infoimpl="" restricao="" situacao=""

      <artefato id="DocReq" name="Documento Resquisitos" size="45" complexity="80"/>

        <tech-skill id="AnaSkill" name="Analysis Skill"/>
        <tech-skill id="DesignSkill" name="Design Skill"/>
        <tech-skill id="ImplSkill" name="Implementation Skill"/>

        <role id="ReqAna" name="Requirements Analyst">
            <tech-skill-ref>AnaSkill</tech-skill-ref>
            <tech-skill-ref>DesignSkill</tech-skill-ref>
        </role>
        <role id="SysAna" name="System Analyst">
            <tech-skill-ref>AnaSkill</tech-skill-ref>
            <tech-skill-ref>DesignSkill</tech-skill-ref>
            <tech-skill-ref>ImplSkill</tech-skill-ref>
        </role>
        <role id="Developer" name="Developer">
```

```
        <tech-skill-ref>AnaSkill</tech-skill-ref>
        <tech-skill-ref>DesignSkill</tech-skill-ref>
        <tech-skill-ref>ImplSkill</tech-skill-ref>
    </role>
    <role id="Tester" name="Tester">
        <tech-skill-ref>DesignSkill</tech-skill-ref>
        <tech-skill-ref>ImplSkill</tech-skill-ref>
    </role>
    <role id="Arch" name="Architect">
        <tech-skill-ref>AnaSkill</tech-skill-ref>
        <tech-skill-ref>DesignSkill</tech-skill-ref>
        <tech-skill-ref>ImplSkill</tech-skill-ref>
    </role>

    <atividade id="LevReq" name="Levantar Requisitos">
        <artefatoentrada>DocReq</artefatoentrada>
        <artefatosaida>DocReq</artefatosaida>
        <recursosoftware>Case Tool</recursosoftware>
        <recursohumano role-ref="ReqAna">
            <!-- level tá variando de 1 a 5 equivalente a lowest,
                 low, medium, high e highest
            -->
            <tech-skill-level tech-skill-ref="AnaSkill" level="5"/>
        </recursohumano>
    </atividade>

    <atividade id="DetEspCasUso" name="Detalhar Especificacao de Casos de Uso">
        <preatividade>LevReq</preatividade>
        <artefatosaida>DocReq</artefatosaida>
        <recursohumano role-ref="ReqAna">
            <tech-skill-level tech-skill-ref="AnaSkill" level="5"/>
        </recursohumano>
    </atividade>

    <atividade id="EstModCasUso" name="Estruturar Modelos de Casos de Uso">
        <preatividade>DetEspCasUso</preatividade>
        <artefatosaida>DocReq</artefatosaida>
        <recursohumano role-ref="ReqAna">
            <tech-skill-level tech-skill-ref="AnaSkill" level="3"/>
        </recursohumano>

    </atividade>

    <atividade id="ProInt" name="Prototipar Interface">
```

```
        <preatividade>EstModCasUso</preatividade>
        <artefatosaida>DocReq</artefatosaida>
        <recursohumano role-ref="ReqAna">
            <tech-skill-level tech-skill-ref="AnaSkill" level="3"/>
        </recursohumano>
    </atividade>

    <atividade id="AnaCasUso" name="Analisar Caso de Uso">
        <preatividade>EstModCasUso</preatividade>
        <recursohumano role-ref="ReqAna">
            <tech-skill-level tech-skill-ref="AnaSkill" level="5"/>
        </recursohumano>
    </atividade>

    <atividade id="ProjArq" name="Projetar Arquitetura">
        <preatividade>AnaCasUso</preatividade>
        <recursohumano role-ref="Arch">
            <tech-skill-level tech-skill-ref="DesignSkill" level="5"/>
        </recursohumano>
    </atividade>

    <atividade id="ProjCasUso" name="Projetar Casos de Uso">
        <preatividade>ProjArq</preatividade>
        <recursohumano role-ref="SysAna">
            <tech-skill-level tech-skill-ref="DesignSkill" level="4"/>
        </recursohumano>
    </atividade>

    <atividade id="ProjSubSis" name="Projetar SubSistema">
        <preatividade>ProjCasUso</preatividade>
        <recursohumano role-ref="SysAna">
            <tech-skill-level tech-skill-ref="DesignSkill" level="3"/>
        </recursohumano>
    </atividade>

    <atividade id="EstModImpl" name="Estruturar Modelo de Implementacao">
        <preatividade>ProjSubSis</preatividade>
        <recursohumano role-ref="Arch">
            <tech-skill-level tech-skill-ref="DesignSkill" level="4"/>
        </recursohumano>
    </atividade>

    <atividade id="PlanInteg" name="Planejar Integracao">
        <preatividade>EstModImpl</preatividade>
```

```
        <recursohumano role-ref="Developer">
            <tech-skill-level tech-skill-ref="ImplSkill" level="4"/>
        </recursohumano>
    </atividade>

    <atividade id="ImplComp" name="Implementar Componentes">
        <preatividade>PlanInteg</preatividade>
        <recursohumano role-ref="Developer">
            <tech-skill-level tech-skill-ref="ImplSkill" level="3"/>
        </recursohumano>
    </atividade>

    <atividade id="TestesUnit" name="Realizar Testes Unitarios">
        <preatividade>ImplComp</preatividade>
        <recursohumano role-ref="Developer">
            <tech-skill-level tech-skill-ref="ImplSkill" level="3"/>
        </recursohumano>
    </atividade>

    <atividade id="CorrDefeitos" name="Corrigir Defeitos">
        <preatividade>ImplComp</preatividade>
        <recursohumano role-ref="Developer">
            <tech-skill-level tech-skill-ref="ImplSkill" level="3"/>
        </recursohumano>
    </atividade>

    <atividade id="IntegSubSis" name="Integrar Sistema e SubSistemas">
        <preatividade>CorrDefeitos</preatividade>
        <recursohumano role-ref="SysAna">
            <tech-skill-level tech-skill-ref="DesignSkill" level="4"/>
        </recursohumano>
    </atividade>

    <atividade id="ElabPlTestes" name="Elaborar Plano de Testes">
        <preatividade>AnaCasUso</preatividade>
        <recursohumano role-ref="SysAna">
            <tech-skill-level tech-skill-ref="ImplSkill" level="4"/>
        </recursohumano>
    </atividade>

    <atividade id="ProjTestes" name="Projetar Testes">
        <preatividade>ElabPlTestes</preatividade>
        <recursohumano role-ref="Tester">
            <tech-skill-level tech-skill-ref="ImplSkill" level="4"/>
```

```
            </recursohumano>
        </atividade>

        <atividade id="ImplTestes" name="Implementar Testes">
            <preatividade>ProjTestes</preatividade>
            <recursohumano role-ref="Tester">
                <tech-skill-level tech-skill-ref="ImplSkill" level="3"/>
            </recursohumano>
        </atividade>

        <atividade id="ExecTestes" name="Executar Testes">
            <preatividade>ImplTestes</preatividade>
            <preatividade>IntegSubSis</preatividade>
            <recursohumano role-ref="Tester">
                <tech-skill-level tech-skill-ref="ImplSkill" level="2"/>
            </recursohumano>
        </atividade>

        <atividade id="AvalTestes" name="Avaliar Testes">
            <preatividade>AvalTestes</preatividade>
            <recursohumano role-ref="Tester">
                <tech-skill-level tech-skill-ref="ImplSkill" level="3"/>
            </recursohumano>

        </atividade>

        <atividade id="ImplantProd" name="Implantar Produto">
            <preatividade>AvalTestes</preatividade>
            <recursohumano role-ref="Developer">
                <tech-skill-level tech-skill-ref="ImplSkill" level="2"/>
            </recursohumano>

        </atividade>
    </processo>

    <modelociclovida nome="RUP" tipo="">
        <fase nome="Concepcao" ordem="" niteracoes="1">
            <atividade-ref>LevReq</atividade-ref>
            <atividade-ref>DetEspCasUso</atividade-ref>
            <atividade-ref>EstModCasUso</atividade-ref>
        </fase>
        <fase nome="Elaboracao" ordem="" niteracoes="1">
            <atividade-ref>AnaCasUso</atividade-ref>
            <atividade-ref>ProjArq</atividade-ref>
```

```
            <atividade-ref>ProjCasUso</atividade-ref>
            <atividade-ref>ProjSubSis</atividade-ref>
            <atividade-ref>EstModCasUso</atividade-ref>
            <atividade-ref>PlanInteg</atividade-ref>
        </fase>

        <fase nome="Construcao" ordem="" niteracoes="1">
            <atividade-ref>ProjCasUso</atividade-ref>
            <atividade-ref>ProjSubSis</atividade-ref>
            <atividade-ref>ImplComp</atividade-ref>
            <atividade-ref>TestesUnit</atividade-ref>
            <atividade-ref>CorrDefeitos</atividade-ref>
            <atividade-ref>IntegSubSis</atividade-ref>
            <atividade-ref>ElabPlTestes</atividade-ref>
            <atividade-ref>ProjTestes</atividade-ref>
            <atividade-ref>ImplTestes</atividade-ref>
            <atividade-ref>ExecTestes</atividade-ref>
            <atividade-ref>AvalTestes</atividade-ref>
        </fase>

        <fase nome="Transicao" ordem="" niteracoes="1">
            <atividade-ref>ImplTestes</atividade-ref>
            <atividade-ref>ExecTestes</atividade-ref>
            <atividade-ref>AvalTestes</atividade-ref>
            <atividade-ref>ImplantProd</atividade-ref>
        </fase>
    </modelociclovida>
</processoinstanciado>
```