UNIVERSIDADE FEDERAL DO PARÁ
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Breno Bernard Nicolau de França

Proposta de um Modelo de Simulação de Processo de Software para o Ambiente WebAPSEE

Belém
2007

UNIVERSIDADE FEDERAL DO PARÁ
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Breno Bernard Nicolau de França

Proposta de um Modelo de Simulação de Processo de Software para o Ambiente WebAPSEE

Trabalho de Conclusão de Curso apresentado para obtenção do grau de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Rodrigo Quites Reis

Belém
2007

UNIVERSIDADE FEDERAL DO PARÁ
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Breno Bernard Nicolau de França

Proposta de um Modelo de Simulação de Processo de Software para o Ambiente WebAPSEE

Trabalho de Conclusão de Curso apresentado para obtenção do grau de Bacharel em Ciência da Computação.

Data da defesa:
Conceito:

Banca Examinadora

Prof. Dr. Rodrigo Quites Reis
Depto. De Informática/UFPA - Orientador

Prof. Dr. Elói Luiz Favero
Depto. De Informática/UFPA - Membro

Prof. MSc. Paulo Roberto Bastos de Almeida
Unama - Membro

Aos meus pais, Mair Raimundo Souza de França e Maria do Carmo Nicolau Rossy, meu irmão Bruno Hamon Nicolau de França e a Deus.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela finalização deste trabalho.

Aos meus pais Maria do Carmo e Mair Raimundo que sempre me deram condições de continuar com meus estudos e sonharam com a realização desta etapa. Além de todo carinho, formação moral e apoio financeiro. Muito obrigado!

Ao meu irmão Bruno Hamon pela compreensão e incentivo durante o período do curso.

Aos meus orientadores acadêmicos Prof. Rodrigo Quites Reis e Profa. Carla Alessandra Lima Reis pelas oportunidades, formação acadêmica e ensinamentos repassados.

Aos meus amigos de trabalho do Laboratório de Engenharia de Software pelas contribuições em discussões e companheirismo pelos dois anos em que trabalhei neste.

A todos meus amigos que me deram força, incentivo e acreditaram na minha capacidade.

Aos professores do Colegiado de Ciência da Computação pelo conhecimento adquirido nas disciplinas ministradas no decorrer do curso.

Ao pesquisador alemão Heribert Schlebbe pelos ensinamentos técnicos e o convite para cooperação em pesquisa na Universidade de Stuttgart.

À empresa Eletronorte pelo apoio financeiro através de bolsa durante o desenvolvimento deste trabalho.

"A Simulação nos leva além do reino da especulação e nos aproxima do mundo da predição."

Lori Monson

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

BFPUG – Brazilian Function Point Users Group

CASE – Computer-Aided Software Engineering

CMM – Capability Maturity Model CMM

COCOMO II – Constructive Cost Model Versão 2

COPLIMO – Constructive Product Line Investment Model

COTS – Commercial off the shelf

CSE-USC – Center for Software Engineering at University of Southern California

ECA – Evento-Condição-Ação

IFPUG – International Function Point Users Group

LABES-UFPA – Laboratório de Engenharia de Software da Universidade Federal do Pará

NASA – National Aeronautics and Space Administration

PML – Process Modeling Language

PSEE – Process-Centered Software Engineering Environment

RBC – Raciocínio Baseado em Casos

SGBD – Sistema Gerenciador de Banco de Dados

SESAM – Software Engineering Simulation by Animated Models

TIM – The Incredible Manager

UML – Unified Modeling Language

RESUMO

A dificuldade na tomada de decisões em projetos de software e na detecção de falhas em um processo antes de sua execução são desafios que ocorrem no dia-a-dia de profissionais envolvidos em atividades de gerência e engenharia do processo de software. É neste contexto que atuam as ferramentas de apoio à tomada de decisão gerencial a partir da análise do processo de software, as quais podem influenciar em aspectos como tempo e custo, fatores determinantes tanto em pesquisas quanto na indústria de desenvolvimento de software. Poucas ferramentas neste nível se mostraram eficazes nos últimos anos. Em virtude disso, experimentos utilizando simulação de processo de software mostraram que esta técnica é promissora e pode obter melhores resultados.

Este trabalho propõe um modelo de simulação de processo de software para incorporar ao ambiente WebAPSEE tal funcionalidade a fim de determinar "a priori" o que poderá ocorrer durante a execução de um processo. Permitir a avaliação de modelos de processo em termos de estrutura e instanciação é o objetivo deste modelo de forma a descobrir possíveis anomalias quanto à alocação de recursos (materiais e humanos), definição de prazos, paralelismo de atividades entre outros. Essa proposta tem como base a utilização do conhecimento contido nas informações coletadas durante execuções passadas através da técnica de Raciocínio Baseado em Casos. Além desse conhecimento, o modelo aqui proposto utiliza também o modelo matemático de estimativa de software COCOMO II (*COnstructive COst MOdel*) para complementar as situações em que não houver informação suficientemente semelhante no repositório do WebAPSEE.

O trabalho é concluído com a discussão dos aspectos relacionados com a implementação do modelo e a apresentação de um protótipo limitado, o qual permite a discussão de um estudo de caso. Por fim, a proposta é discutida levando-se em consideração aspectos como as técnicas utilizadas, a capacidade de utilizar este modelo em outros cenários e o grau de acurácia do modelo.

PALAVRAS-CHAVE: Processo de Software, Tecnologia de Processo de Software, Simulação de Processo de Software, Raciocínio Baseado em Casos, COCOMO-II.

ABSTRACT

The difficulties on software project decision taking and on the early detection of possible failures prior to the actual enactment of process models challenge the professionals involved on both project management and process engineering activities. In the context of decision support, specific software tools were developed based on software processes analysis techniques, and may influence costs and schedule issues, which constitute important quality factors, recognized both by research and the software industry. However, a small number of tools in this context have achieved efficient results recently. At this point, experiences using software process simulation tools have shown that this technique is a promising field of study that can achieve better results.

This work proposes a software process simulation model in order to extend the functionality of the existing WebAPSEE environment to foresee what can happen during the process enactment. The target of this model is to allow the evaluation of process models concerning their structure and instantiation, discovering possible deficiencies in resource and personal allocation, schedule definition, activity parallelism and others. This proposal is based on the knowledge gathered from past enactments through the Case-Based Reasoning technique. Besides this knowledge, the proposed model also uses the COCOMO II (*COnstructive COst MOdel*) mathematical software estimation model to complement on situations where there is no similar information in the WebAPSEE repository.

This work discusses about the implementation of the proposed model. It also presents a limited prototype, which makes possible the analysis of a case study. Finally, the proposal is discussed with respect to the selected techniques, the opportunities of using this model in others domains, and the actual model accuracy degree.

KEYWORDS: Software Process, Software Process Technology, Software Process Simulation, Case-Based Reasoning, COCOMO-II.

1. INTRODUÇÃO

O desenvolvimento de software durante muitos anos foi marcado por um cenário caótico (Humphrey 1988), onde a falta de organização e de gestão eram características predominantes. Segundo Pressman (2000) este cenário foi considerado como uma crise por muitos autores na área de Engenharia de Software, a "crise do software". Isto decorria da dificuldade de medir, estimar o software e entender o processo de desenvolvimento de software. É neste contexto que os processos, métodos e ferramentas de engenharia de software concentraram seus esforços para aumentar a qualidade do software produzido.

Buscando o aumento da qualidade, técnicas como Ocultamento de Informações de Parnas (1985) e notações como a Linguagem de Modelagem Unificada (UML – *Unified Modeling Language*) proporcionaram avanços consideráveis para modularização e manutenção do software e na documentação padrão para o paradigma de orientação a objetos respectivamente, acarretando no aumento da qualidade do software produzido. No que diz respeito a planejamento e gerência de processos, outros importantes avanços foram alcançados, tais como o Processo Unificado (Humphrey 1995) e o Processo Pessoal de Desenvolvimento de Software (Jacobson 1999). Entretanto, a automação dessas práticas se faz necessário em resposta ao contínuo aumento da complexidade e do tamanho do software construído atualmente.

Nas seções seguintes é apresentado o contexto deste trabalho, suas motivações, os objetivos gerais da proposta e a organização do texto.

1.1 CONTEXTO DO TRABALHO

Dentre os avanços recentes na Engenharia de Software, foi citada a necessidade de automatizar alguns processos e metodologias. Hoje em dia, algumas ferramentas são propostas para realizar essa automação seja em nível mais baixo, que é o caso das ferramentas CASE (*Computer-Aided Software Engineering*) ou em mais alto nível de abstração, caso dos PSEEs (*Process-Centered Software Engineering Environment*), ambientes para gerência de processo de software.

Este trabalho busca contribuir com avanços no escopo da área denominada Tecnologia de Processo de Software (ver seção 2.1), cujo foco está na automação de processos de software do ponto de vista da gerência e da engenharia de processos. Tanto na aplicação cotidiana de processos de software em organizações quanto em pesquisas voltadas ao estudo dessa área.

Os avanços esperados consistem na possibilidade de “previsão” de ocorrências e experimentações em curto prazo através da construção de uma proposta de um modelo de simulação de processo de software, que reúne abordagens apresentadas na literatura, conforme os trabalhos relacionados e o estado da arte apresentados no capítulo 3, e uma abordagem própria para a solução do problema.

1.2 MOTIVAÇÕES

A necessidade de pesquisas com base em experimentações na área de processos de software é apresentada por Fuggetta (2000) ao discutir sobre a importância de descobertas significativas e a obtenção de maior acurácia dos estudos sobre processos, do ponto de vista prático. Entretanto, algumas experimentações podem durar meses para chegar a alguma conclusão.

De forma análoga ao que acontece com as experimentações, a execução de processos reais em uma organização pode levar meses para descobrir possíveis causas de falhas de projetos, causando grandes prejuízos à mesma.

Com experimentos baseados em simulação é possível reduzir o tempo e o custo da pesquisa, assim como os desperdícios em uma organização com a realização de um projeto cujo processo foi mal definido ou a alocação de pessoas e recursos foi mal feita, por exemplo.

Os ambientes e modelos de simulação propostos atualmente possuem muitas simplificações, gerando resultados que não condizem com a realidade das organizações de software hoje em dia. Além disso, poucas abordagens voltadas para simulação de processos em geral são feitas. Assim, essas limitações fazem com que muitos modelos de simulação sejam voltados para um modelo de processo de software adotado por uma organização em específico (ver capítulo 3).

A próxima seção descreve os objetivos gerais e a metodologia utilizada neste trabalho.

1.3 OBJETIVOS GERAIS E METODOLOGIA

Este trabalho consiste na construção de um modelo de simulação de processos de software que permite avaliar, com base no conhecimento de uma organização de desenvolvimento de software, modelos de processo de software no contexto organizacional.

O modelo aqui proposto leva em consideração as características e funcionalidades já incorporadas em um Ambiente de Gestão de Processos real desenvolvido na Universidade

Federal do Pará (WebAPSEE) a fim de permitir sua implementação futura de forma integrada ao ambiente, reutilizando o conhecimento por este adquirido em execuções de processos passados.

É importante considerar que o produto final deste trabalho é um modelo inédito de simulação de processo de software que será aplicado ao ambiente WebAPSEE e não um (sistema) simulador de processo de software propriamente dito. O caráter inédito do trabalho se dá em função da abordagem mista para o modelo de simulação que utiliza como base teórica a técnica de Raciocínio Baseado em Casos (von Wangenheim 2003) e o modelo COCOMO II (Boehm 2000). Alguns resultados desse modelo de simulação são apresentados no capítulo 6, onde são discutidos alguns indícios que demonstram a viabilidade de utilização e um exemplo de aplicação.

Quanto à metodologia para estruturação do estudo e o desenvolvimento do trabalho foi utilizado o método dedutivo, buscando a validação da hipótese (proposta). O modelo é proposto a partir de requisitos levantados por pesquisas bibliográficas - tanto recentes quanto clássicas - e heurísticas a respeito de simulação de processos de software.

Primeiramente foram realizados estudos comparativos a respeito do modelo de simulação já proposto de acordo com os existentes na literatura, analisando e comparando as soluções existentes será realizada para buscar a melhor abordagem a ser desenvolvida no trabalho; então, algumas funcionalidades foram projetadas e implementadas. Entretanto, o modelo será validado de forma concreta com futuras experimentações em ambientes industriais reais.

1.4 ORGANIZAÇÃO DO TEXTO

Este trabalho está organizado como segue. O capítulo 2 apresenta os conceitos importantes que envolvem gestão de processo de software, o ciclo de vida dos processos e a área denominada tecnologia de processo de software, com ênfase na execução e sua importância para a simulação de processo de software. O capítulo 3 trata os conceitos e os desafios em simulação de processo de software são tratados, assim como alguns trabalhos relacionados ao aqui proposto. O capítulo 4 detalha o modelo de simulação proposto neste trabalho, suas particularidades e limitações. O capítulo 5 aborda aspectos de implementação do modelo como o modelo de dados, ferramentas utilizadas e os algoritmos importantes. O capítulo 6 apresenta um exemplo de simulação e suas vantagens. Finalmente, o capítulo 7 apresenta as conclusões e trabalhos futuros são apresentados e discutidos.

2. GESTÃO DO PROCESSO DE SOFTWARE

Este trabalho está inserido no contexto de tecnologias utilizadas para o auxílio à gerência e definição de processos de software. Com isso, algumas definições são apresentadas neste capítulo a fim de situar o leitor no cenário de gestão do processo de software, proporcionando um melhor entendimento da proposta deste trabalho.

Os estudos e propostas sobre Processo de Software constituem uma linha de pesquisa das mais importantes em Engenharia de Software na atualidade devido ao estreito relacionamento entre a qualidade do produto e a qualidade do processo (Fuggetta 2000).

De acordo com Pressman (2005), um processo de software é definido como "um *framework* para as tarefas que são requeridas a fim de construir software de alta qualidade" e tem um modelo de processo para representar o processo de forma abstrata, apresentando a descrição deste, dada uma perspectiva particular. Outra definição, por Humphrey (1988), diz que um processo de software é definido como uma seqüência de tarefas que, quando executadas de maneira adequada, produzem os resultados esperados. A definição de um processo de software deve considerar ainda os relacionamentos de todas as tarefas, ferramentas e métodos utilizados, políticas organizacionais e habilidades, treinamentos e motivação das pessoas envolvidas. Além de todos esses fatores processos de software devem considerar a complexa inter-relação de fatores organizacionais, econômicos, culturais e tecnológicos.

Humphrey (1988) cita, ainda, que o comportamento caótico no processo de desenvolvimento da maioria das organizações atuais de desenvolvimento de software deu-se em função da falta de experiência para entender as conseqüências deste comportamento. Além disso, no desenvolvimento de software, as atividades de codificação e teste são aquelas onde a visualização do produto se torna mais efetiva, entretanto, sempre há um grande risco de irem à direção errada pela falta de planejamento e análise.

Segundo Fuggetta (2000), durante os vinte primeiros anos de pesquisa em processos de software, consideráveis conquistas foram alcançadas. A busca pela melhoria da qualidade do processo – consequentemente do produto - levou pesquisadores e estudiosos a propor modelos, métodos, metodologias e *frameworks* com esse propósito. Por exemplo, no *framework* definido em Humphrey (1988), a melhoria da capacidade e maturidade em desenvolvimento de software de uma organização, é dada em cinco passos: (1) entender o estado atual de seu processo de desenvolvimento, (2) desenvolver uma visão do processo desejado, (3) estabelecer uma lista de

ações de melhorias do processo requeridas em ordem de prioridade, (4) produzir um plano para cumprir estas ações, e (5) investir em recursos para executar o plano. Mais ainda, a definição de processos ajuda os profissionais a entender seu trabalho e como melhorá-lo.

Quanto à natureza, modelos de Processo de Software são diferentes dos processos industriais por envolverem pessoas trabalhando em atividades criativas (Lima Reis 2003). A natureza dinâmica do desenvolvimento de software implica na impossibilidade de prever todas as características dos modelos de processo e produtos de software resultantes. Assim, modelos de processo freqüentemente evoluem durante o desenvolvimento do software, o que demanda flexibilidade durante a execução do processo. Além disso, modelos de processo não evoluem de maneira certa e determinística. Escolhas entre caminhos alternativos devem ser permitidas, e estas escolhas podem depender dos resultados das atividades anteriores (Heinl *et al*. 1999).

Este capítulo é dividido em quatro seções principais. A seção 2.1 descreve o ciclo de vida do processo de software, o qual contribui para o entendimento das várias etapas onde ferramentas de simulação podem auxiliar na tomada de decisão. A seção 2.2 apresenta o contexto da área denominada Tecnologia de Processo de Software e seus principais conceitos. A seção 2.3 mostra um paralelo entre a Execução do Processo de Software e a Simulação de Processo de Software, identificando as questões em comum entre estas fases do ciclo de vida do processo. Finalmente, a seção 2.4 apresenta um resumo com as considerações finais deste capítulo.

2.1 META-PROCESSO DE SOFTWARE: O CICLO DE VIDA DOS PROCESSOS

O ciclo de vida dos processos de software é apresentado nesta seção para contextualizar a importância da simulação na definição de processos. Embora os desafios atuais relacionados à simulação de processos de software sejam apresentados no capítulo 3, esta seção adianta o assunto em busca da elucidação dos aspectos temporais que indicam os momentos em que a simulação é utilizada durante o ciclo de vida dos processos.

Além da já citada natureza dinâmica da execução dos processos de software, a concepção e utilização de um processo de software possuem, também, características evolutivas, ou seja, também há preocupação explícita de forma permitir contínuas mudanças e melhorias.

Durante o ciclo de vida do processo, existem atividades a serem realizadas, assim como em um processo de software. Essas atividades denominam-se meta-atividades, como descrito em Lima Reis (2003). A figura 1 apresenta um esquema deste ciclo de vida.

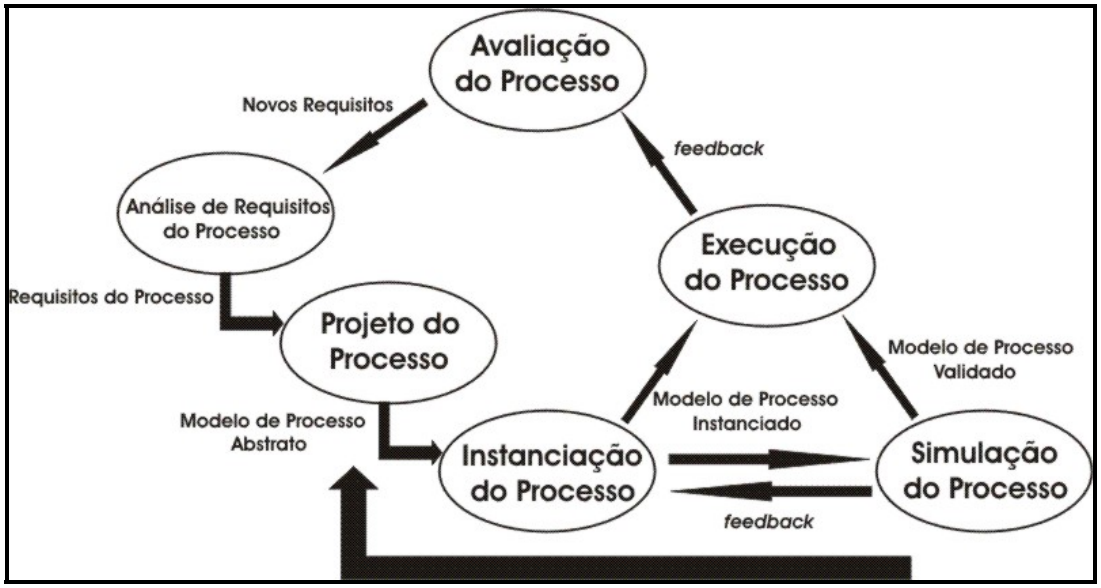

Figura 1. Ciclo de Vida do Processo de Software [adaptado de Lima Reis (2003)]

A figura 1 mostra as meta-atividades (elipses) do ciclo de vida do processo de software e seu fluxo (setas). O início deste ciclo é a meta-atividade **Análise de Requisitos de Processo,** que resultam na elaboração dos requisitos identificados do processo. Estes requisitos servem de entrada para a meta-atividade de **Projeto do Processo**, onde será definida a arquitetura geral e detalhada do processo e através da linguagem de modelagem de processos. A meta-atividade **Instanciação do Processo** utiliza o modelo de processo abstrato, resultado do projeto, para acrescentar detalhes concretos acerca do processo, como prazos, custos e pessoas das atividades a serem realizadas, tendo como resultado o modelo de processo instanciado. Na meta-atividade de **Simulação do Processo** (foco deste trabalho) é possível validar, encontrar anomalias e oportunidades de melhora no processo antes deste entrar em execução, retornando resultados (*feedback*) para atividades de projeto e instanciação. Já a **Execução do Processo**, utiliza o resultado da instanciação do processo e, através de ferramentas, métodos e metodologias, ocorre a realização do processo no mundo real. Por fim, a cada execução (durante e ao fim), métricas e estimativas, ou seja, dados quantitativos e qualitativos sobre o processo, são obtidos e consumidos pela meta-atividade de **Avaliação do Processo**. Os resultados da Avaliação do Processo são retornados à atividade de **Análise de Requisitos de Processo** para gerar novos requisitos.

2.2 TECNOLOGIA DE PROCESSO DE SOFTWARE

Grande investimento vem sendo feito pela Academia e Indústria na construção de ferramentas que auxiliem o gerenciamento do processo de desenvolvimento de software, constituindo uma área de pesquisa denominada Tecnologia de Processo de Software (Gruhn 2002).

A Tecnologia de Processo de Software teve início com o estudo de Osterweil (1987), onde estabelece um paralelo entre processo de software e software propriamente dito. Isto é, o autor afirma que é possível construir um processo, como um software, a partir de uma linguagem de programação de processos (do termo *process programming*) definida sobre um formalismo. Entretanto, a Tecnologia de Processo de Software ganhou maior atenção a partir da ênfase dada ao uso de modelos como o *Capability Maturity Model* (CMM) – descrito por Paulk *et al*. (1991) e utilizado para avaliação da qualidade de software a partir da maturidade dos processos organizacionais adotados pelos desenvolvedores. Assim, com o uso de Ambiente de Desenvolvimento de Software Centrado em Processo [mais conhecido pela sua sigla em inglês PSEE de *Process-Centered Software Engineering Environment* (Gimenes 1994)], é possível coordenar atividades de equipes dispersas geograficamente, acompanhar os prazos e consumo de recursos, além de facilitar a reutilização de boas práticas gerenciais por diferentes projetos adotados. As soluções desenvolvidas nesta área devem levar em consideração elementos específicos do contexto de processos de software, tais como: o caráter criativo do processo, a grande tendência a mudanças, a impossibilidade de se reutilizar imediatamente processos executados e a falta de visibilidade do produto resultante.

Um PSEE integra: 1) ferramentas como editores de modelagem de processos baseados em uma ou mais Linguagens de Modelagem de Processos (ou PML – *Process Modeling Language*); 2) máquinas de execução (que "executam" de forma automatizada o modelo de processo descrito com PMLs), e; 3) repositórios de dados (frequentemente fornecidos por ferramentas para controle de versão) para ajudar os gerentes do processo nas tarefas que podem ser automatizadas. O centro de controle de um PSEE é sua Máquina de Execução, o qual desempenha função análoga à de um *kernel* de sistema operacional de computador, sendo responsável por coordenar os elementos do modelo de processo. Execução de processos de software é um tema melhor explorado pela seção 2.3. Para uma discussão mais detalhada sobre o assunto, o leitor deve consultar Lima Reis (2003).

A falta de flexibilidade e dinamismo no apoio fornecido para gerência de processo, PMLs

excessivamente detalhadas e o alto grau de intrusão dos ambientes têm sido reconhecidas por serem umas das mais importantes razões para a baixa aceitação dos PSEEs existentes (Fuggetta 2000). Melhorias na flexibilidade de execução de processos de software foi o tema do trabalho apresentado em (Lima Reis 2002a), onde é descrita a primeira versão do ambiente *APSEE*.

Nos últimos anos, os esforços da comunidade de processo de software foram direcionados para a construção de sistemas envolvendo um único formalismo para modelagem e execução de processos. Esta abordagem foi adotada por um grande número de sistemas monolíticos como SPADE, Oz, JIL, *APSEE* e outros descritos em Finkelstein (1994) e Wang (2002). Como resultado, o apoio automatizado para processos de software não atingiu um nível suficiente de maturidade para ser usado no mundo real: a maioria dos sistemas propostos são apenas protótipos acadêmicos. Abordagens mais recentes - tais como adotadas pelo ambiente WebAPSEE (França et al 2006) - atenuam os problemas por serem baseadas em plataformas abertas (por exemplo., Java) e permitem a modelagem de processos segundo vários formalismos especializados e complementares entre si (Reis 2002b).

2.3 EXECUÇÃO E SIMULAÇÃO DE PROCESSOS

A coleta de métricas durante a execução do processo é um requisito muito citado para PSEEs (Bandinelle 1994) e para desenvolvimento de software em geral. Entretanto, o uso eficiente deste recurso nem sempre tem sido tratado, o que tem causado uma falta de interesse em manter os esforços de coletar métricas. Este conhecimento poderia ser utilizado para ajudar na tomada de decisões para alocar pessoas e recursos no processo, além disso, pode ser usado para simulação de processo (Scacchi 1999).

Para simulação de processo de software é importante dar um enfoque maior na meta-atividade de execução de processos, pois se trata do comportamento que a simulação se propõe a "imitar". Portanto, os próximos parágrafos serão dedicados a este assunto.

A capacidade de execução do processo de software é o núcleo de um PSEE, pois se trata do componente voltado ao acompanhamento e coordenação das atividades e processos desempenhados pelas pessoas durante o desenvolvimento de software. Em virtude da natureza dinâmica e criativa dos processos adotados na construção do software atual, a coordenação das atividades deve ser bastante flexível de modo que durante a execução alterações sejam toleradas. Ou seja, no momento em que o processo está em execução, é possível alterar o fluxo desta e as pessoas, custos e prazos definidos em momentos anteriores. Estas alterações são comumente

chamadas de modificações *ad-hoc* (Aalst 1999). O desafio ao lidar com modificações *ad-hoc* é garantir a consistência (i.e., evitar *deadlocks* ou efeitos nas atividades passadas) após serem realizadas.

Este acompanhamento e coordenação das atividades que são realizados durante a execução de processos devem, também, ser feitos ao simular um processo. Ou seja, toda a sincronização do estado do processo com os estados das atividades deve ser considerada no momento da simulação.

Muitos dos problemas que surgem no decorrer da execução de um processo de software podem ser visualizados durante a simulação, como o aumento demasiado do custo com o passar do tempo da realização das atividades e do consumo dos recursos, por exemplo.

O componente responsável pela execução em um PSEE - freqüentemente chamado de Máquina ou Motor de Execução - interage com os outros componentes do ambiente sincronizando e coordenando o estado do processo (e suas atividades).

O estado do processo é mantido por meios de *feedback* da realização do processo obtidos durante a execução, mantendo a consistência entre o estado da execução e o estado da realização. De acordo com Lima Reis (2003) este *feedback* pode ser obtido de diversas formas: (1) o usuário pode agir explicitamente de modo a "avisar" que concluiu uma atividade; (2) questionamentos podem ser feitos para o usuário sobre o andamento do processo; (3) uma ferramenta pode gerar um evento automaticamente; (4) uma consulta ao banco de dados do ambiente pode obter informações sobre artefatos criados; ou (5) podem ser gerados eventos que sinalizem o andamento das atividades.

A figura 2 apresenta a arquitetura de um PSEE segundo Lima Reis (2003). Neste diagrama, o componente central é, também, o mecanismo de execução de processos que interpreta o modelo de processo instanciado de acordo com a semântica da linguagem de modelagem, além disso, interage com ferramentas externas e com o usuário. A figura contém ainda repositório de informações, do histórico e das métricas dos processos executados, além da interação com dados organizacionais e as ferramentas para definição (modelagem) de processos de software executáveis.

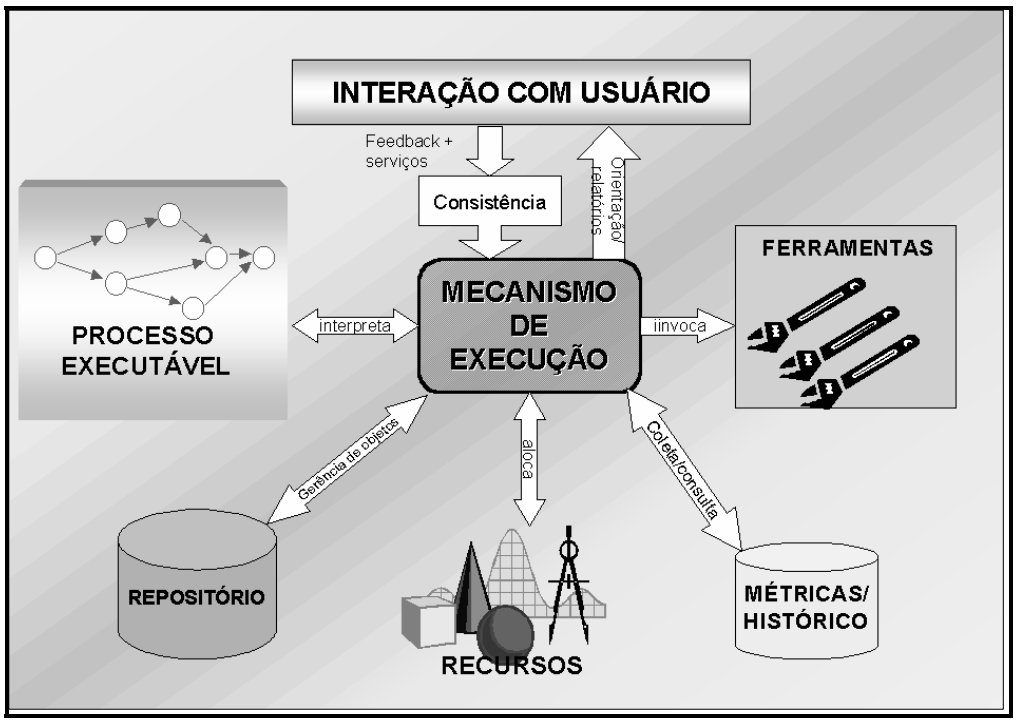

Figura 2. Arquitetura de um PSEE (Lima Reis 2003).

Entidades organizacionais como recursos e o próprio histórico das execuções passadas dos processos de software adotados, muitas vezes consolidado pela organização, são importantes ao se considerar à sua interação nesta arquitetura.

A simulação de processos deve, assim como a execução, considerar aspectos organizacionais, tais como: pessoas envolvidas, recursos utilizados, habilidades dos agentes, afinidade entre os agentes ao realizarem tarefas em grupo, políticas da organização, entre outros.

2.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foram apresentados conceitos fundamentais e problemas relacionados a Processos de Software que dizem respeito ao contexto deste trabalho.

Entender como a tecnologia de processo de software influencia no ciclo de vida deste, mais especificamente a simulação, é de grande importância para o acompanhamento e compreensão do texto seguinte. Falar de ambientes de gerência de processo de software (PSEEs) e como seus mecanismos de execução atuam é, também, crucial para situar o estado da arte da automação de processos de desenvolvimento de software. Além disso, é importante citar, também, os problemas relacionados com a execução de processos, que são motivadores para que processos sejam simulados antes de executá-los. Este capítulo serve de base para as discussões apresentadas nos capítulos posteriores.

3. SIMULAÇÃO DE PROCESSO DE SOFTWARE

Simulação de processos de software é um assunto importante na comunidade de Engenharia de Software. Isto é evidenciado pela existência de um evento internacional exclusivamente dedicado ao tema, denominado ProSim (iniciado em 2003).

Embora exista alto número de ferramentas eficazes no apoio à tomada de decisão em áreas como planejamento operacional e decisões de negócios, na Engenharia de Software, ainda hoje em dia, ferramentas com esse propósito são pouco eficazes. Isto ocorre segundo Donzelli (2006) provavelmente pela complexidade inerente da área ao lidar com naturezas sociais e tecnológicas.

A simulação é uma técnica promissora e que tem alcançado bons resultados no contexto de processo de software quando se trata de planejamento e gerência de riscos (Münch 2003). O principal benefício esperado pela simulação é fornecer para o gerente e para a equipe de desenvolvimento um diagnóstico “a priori” sobre possíveis ocorrências, tais como: atrasos, falha na alocação dos profissionais às tarefas do processo, insuficiência de recursos, riscos entre outras.

Em um processo de software real os recursos (materiais e humanos) são essenciais e muitas vezes escassos, assim como o cronograma a ser cumprido e os riscos que envolvem o projeto. Neste cenário, o desperdício de ambos deve ser evitado ao máximo. Ao passo que um processo de software demora muito tempo para ser executado e recursos que podem ser desperdiçados com falhas no projeto, na simulação isso pode ser realizado em questão de minutos, no máximo horas e sem gastos reais.

Durante e após a simulação o processo pode ser analisado, ajudando a alocação de recursos e pessoas, estabelecer prazos, e até mesmo a melhorar o processo. Isto contribui para o trabalho tanto de engenheiros de quanto de gerentes de processo ao avaliarem os impactos de um determinado cenário, como uma ferramenta de suporte a decisão (Donzelli 2006). Por exemplo, a simulação pode auxiliar na análise do impacto da adoção de novas tecnologias, métodos e metodologias. Uma contribuição pouco visível como benefício da simulação, mas apresentada por Raffo (1999), é o refinamento dos objetivos para a coleta de métricas, focando no apoio à predição da performance do processo.

Várias abordagens vêm sendo experimentadas pela literatura, com diferentes propósitos, desde o auxílio no diagnóstico “a priori” (Mi e Scacchi 1990) (Silva 2000) até no

desenvolvimento de jogos interativos que auxiliam no treinamento de gerentes novatos ou estudantes (Drappa 2000) (Dantas 2004).

A seção 3.1 apresenta os principais desafios em Simulação de Processo de Software e suas particularidades e a seção 3.2 apresenta os trabalhos relacionados ao aqui proposto são apresentados em um total de seis modelos de simulação e descritos para mostrar suas abordagens e limitações. Finalmente, a seção 3.3 apresenta as considerações finais do capítulo.

3.1 DESAFIOS

Diversas barreiras são encontradas quando no projeto de um simulador de processo de software. Nesta seção algumas dessas dificuldades são citadas e discutidas.

A primeira dificuldade é manipular a grande quantidade de informação utilizada e gerada durante a execução real de um processo de software. Incorporar ao modelo de simulação variáveis como: estado do processo e de cada atividade; artefatos consumidos, transformados e produzidos; custo de recursos materiais (financeiros, tecnológicos, entre outros tipos) e humanos; dados organizacionais, ferramentas utilizadas e o relacionamento entre elas geram uma enorme e complexa rede de informações a manter.

Existem, ainda hoje, poucas abordagens integradas aos PSEEs. Portanto, valiosa oportunidade é desperdiçada, pois o que poderia ser uma vantagem, na medida em que o ambiente já possui uma gerência das informações acerca do processo e da organização.

Outro problema com ferramentas de simulação é a falta de um ambiente real de desenvolvimento de software (Dantas 2004). Ou seja, muitas ferramentas simplificam demasiadamente os processos a serem simulados e as variáveis a serem consideradas. Dentre estas, as exceções são os modelos de simulação voltados para a simulação de um modelo de processo em particular, como apresentado por Münch (2003).

Apesar de todas as outras dificuldades, considerar o caráter criativo das atividades seja, talvez, a maior dificuldade entre elas. Os atores do processo de software realizam tarefas de natureza extremamente criativa, por exemplo, codificar algoritmos complexos, projetar interfaces gráficas, projetar a arquitetura de um sistema e lidar com novas tecnologias. O autor deste trabalho acredita que uma forma de amenizar este problema (existe na maioria dos simuladores) é reutilizando conhecimento das execuções passadas de processos das organizações para compor uma base de conhecimento para simulação. Colabora com esta intenção do autor a experiência realizada no Laboratório de Engenharia de Software da NASA apresentada em (Donzelli 2006),

esta concluiu que um dos fatores que beneficiaram sua abordagem de simulação foi reutilizar conhecimento disponível da organização.

3.2 TRABALHOS RELACIONADOS

Nesta seção são apresentados trabalhos relacionados ao aqui desenvolvido e que contribuíram para o avanço do estado da arte em simulação de processos de software.

3.2.1 SESAM

O projeto **SESAM (*Software Engineering Simulation by Animated Models*)** apresentado em Drappa (2000) é um simulador de processo de software em formato de jogo que tem como idéia principal criar um modelo de processo de desenvolvimento de software e executá-lo usando um sistema de simulação com a finalidade de treinar gerentes de processos. O projeto inicial é sempre o mesmo, onde as restrições de prazos, custos e qualidade (número de defeitos) são fornecidas ao usuário. Além disso, os dados organizacionais e os tipos de atividades (especificação de requisitos, codificação entre outros) são sempre os mesmos, isto é, os profissionais a serem contratados, recursos disponíveis e cargos são pré-definidos e não representam a realidade das organizações de software.

Com o desenrolar de uma sessão de simulação, os atributos do processo e do produto (o software a ser produzido) começam a ser instanciados, sendo atribuídos valores de acordo com os profissionais alocados, com a utilização dos recursos, e com as tarefas realizadas.

Ao fim da simulação um relatório é gerado mostrando se o software produzido foi aceito ou não pelo cliente de acordo com as restrições impostas inicialmente. Caso o cliente não concorde com a qualidade do produto, suas justificativas são apresentadas.

A figura 3 mostra a tela de apresentação inicial do projeto ao usuário (gerente) com as restrições citadas previamente. A barra situada na porção inferior da figura 3 é o local onde usuário interage com o sistema através de comando pré-definidos no campo de texto, podendo ainda avançar no tempo através do botão ***Proceed***. Em geral, o sistema aceita comandos escritos na língua inglesa que freqüentemente estão associados com ações de contratar, demitir e delegar tarefas aos atores do processo.

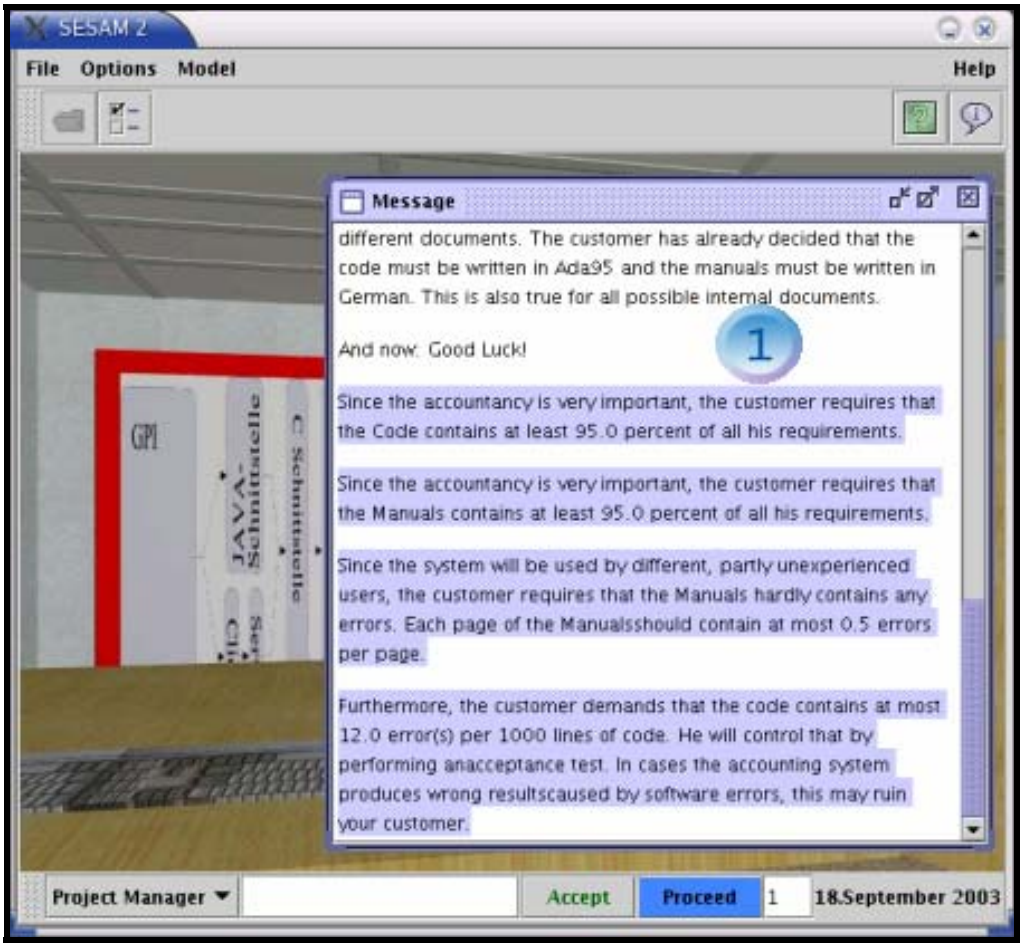

Figura 3. Tela de Apresentação do Projeto ao usuário do SESAM.

Internamente, o simulador SESAM utiliza a primeira versão do modelo COCOMO (Boehm 1981), calibrado de acordo com as métricas pré-definidas de seus agentes e recursos, para cálculo de estimativas de custo, tamanho e prazos, além de regras empíricas inseridas no modelo através de uma linguagem própria do simulador para representá-las.

3.2.2 TIM

Outro simulador baseado em jogo com objetivos educacionais é o **TIM (*The Incredible Manager*)** apresentado por Dantas (2004). Na figura 4, a interface gráfica do jogo é apresentada.

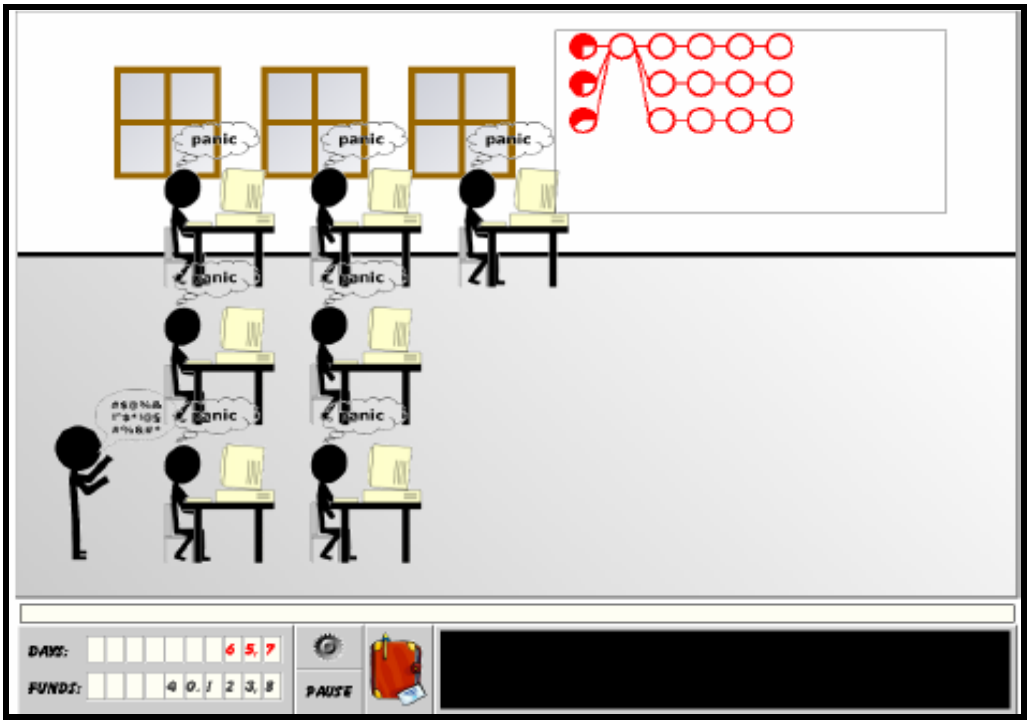

Figura 4. Tela do jogo TIM (Dantas 2004).

Conforme a figura 4 nota-se a presença dos atores do processo em frente aos computadores e a figura do gerente comandado a execução do processo. À direita e acima é possível ver a realização das tarefas, ou seja, o andamento do processo. Na porção inferior são apresentados os dias decorridos e os custos do processo.

O simulador TIM utiliza, internamente, o modelo *System Dynamics* definido em (Forrester 1961) para conduzir a simulação do processo. Segundo Dantas (2004), este modelo é baseado em uma visão holística para descrever e avaliar o comportamento visível presente em um sistema, este comportamento é determinado pela estrutura dos elementos que participam do sistema e o relacionamento entre eles. Este modelo é utilizado também no modelo de simulação apresentado em Raffo e Wakeland (2003) para prever o impacto de atividades independentes de verificação e validação no processo de desenvolvimento de software da NASA.

3.2.3 SimSE

Já o simulador definido por Navarro (2005), denominado **SimSE**, utiliza apenas regras empíricas, quais chama de "*The Fundamental Rules of Software Engineering*" (NAVARRO 2002), para conduzir a simulação e baseia-se totalmente no modelo de processo Cascata. Sendo

assim, não é possível simular processos iterativos, tampouco modelar um processo nesta ferramenta.

Como se pode ver na figura 5, o ambiente tem como ponto forte uma interface completamente gráfica. Nesta tela podemos ver os atores do processo em seus locais de trabalho, e o envio de mensagens (através de balões) as quais descrevem o estágio de andamento de suas tarefas.

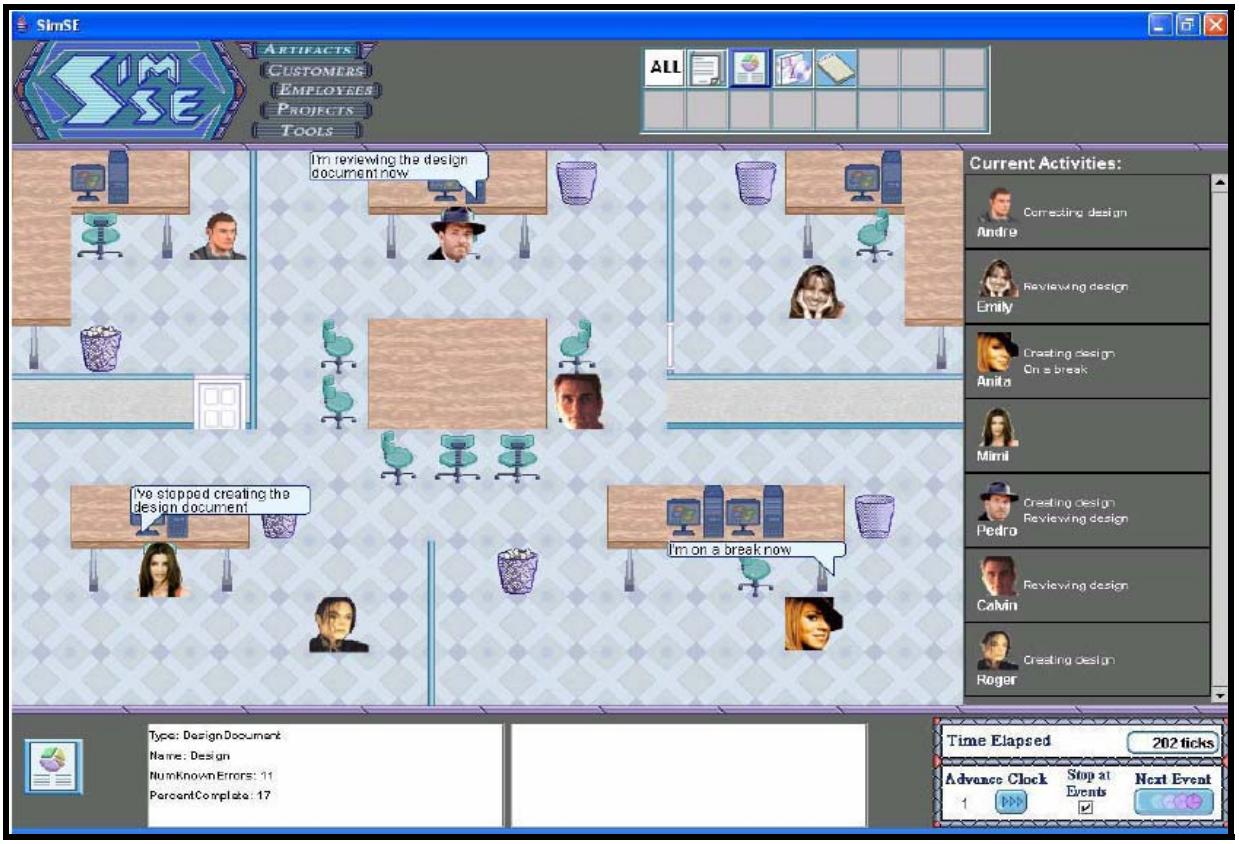

Figura 5. Ambiente gráfico do SimSE (Navarro 2005).

3.2.4 Simulação de Linha de Produto de Software

Contrariamente às abordagens baseadas em jogo anteriormente citadas, o modelo de simulação definido por Chen (2004) utiliza os conceitos de simulação Evento-Discreta e de Linha de Produto de Software. De acordo com Clements (2005), Linha de Produto de Software (do termo em inglês *Software Product Line*) trata do conjunto de sistemas de software que compartilham um núcleo comum de características, satisfazendo uma necessidade específica de um segmento particular do mercado, que são desenvolvidos a partir de um conjunto de recursos

centrais previamente definidos. Neste modelo, assim como no anterior, cada modelo de simulação é aplicável a apenas um modelo de processo, o qual descreve as etapas voltadas à inspeção de documentos de especificação de requisitos.

Este modelo utiliza ainda o modelo COPLIMO (Boehm 2000) para estimativas de custo e prazos. Sendo assim, para cada novo processo a ser simulado, são necessárias novas variáveis e uma nova calibragem do modelo COPLIMO (Constructive Product Line Investment Model).

3.2.5 Articulator

Dentro da categoria de simuladores integrados à PSEEs, um dos trabalhos pioneiros é definido por Mi e Scacchi (1990). O **Articulator** (como é chamado o protótipo desenvolvido por Mi) é um ambiente com apoio a modelagem e simulação de processos. A abordagem utilizada no Articulator é baseada em conhecimento. Na figura 6 é mostrada a arquitetura deste sistema.

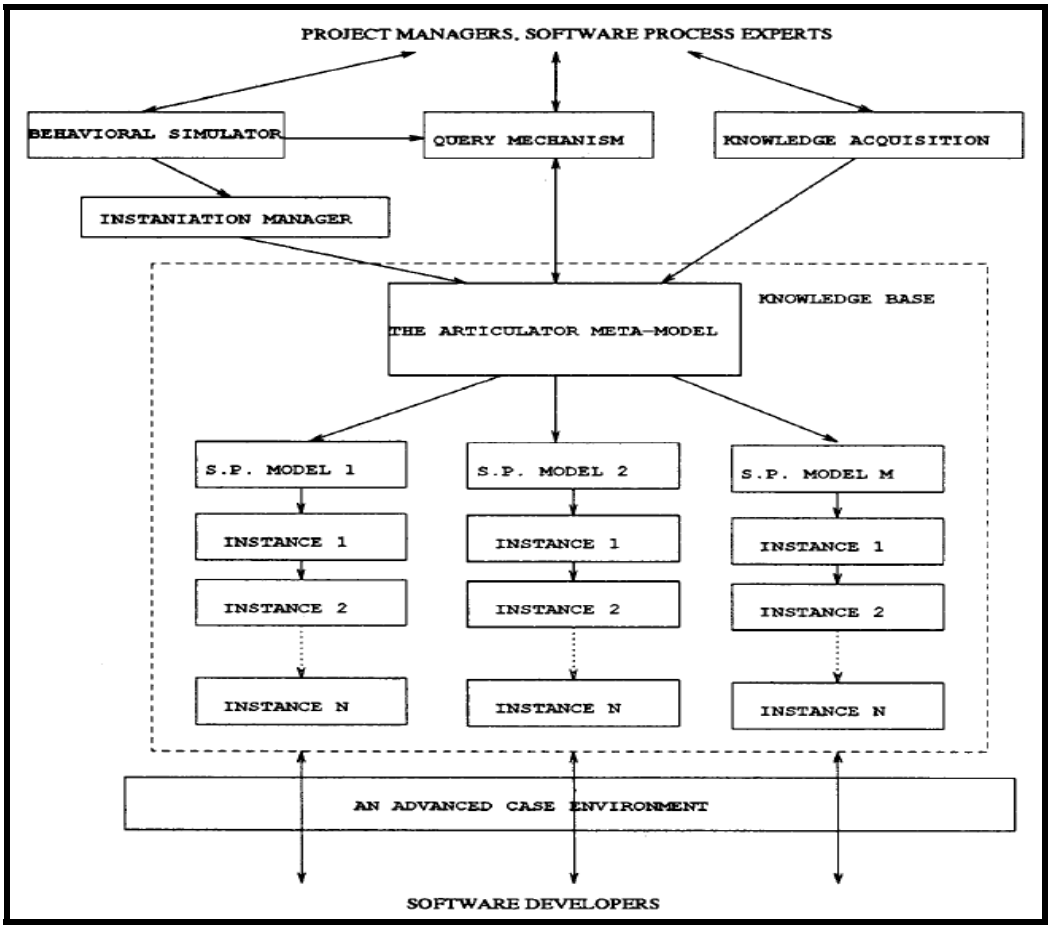

Figura 6. Arquitetura do Articulator (Mi e Scacchi 1990).

A arquitetura do simulador Articulator é dividida, basicamente, em cinco componentes:

1. A base de conhecimento (***knowledge base***), que trata do meta-modelo do ambiente, ou seja, uma "teia de recursos e situações" [expressão usada em Mi e Scacchi (1990)] onde os modelos de processos são definidos e contém a representação das habilidades dos agentes do processo ao realizarem tarefas.
2. O gerente de instanciação (***Instantiation Manager***), responsável por estabelecer o limite entre os modelos de processo e suas instâncias. Estes modelos de processo são abstrações das instâncias dos processos que são realmente simulados.
3. O simulador comportamental (***behavioral simulator***) controla a simulação de um dado modelo de processo, criando a seqüência de passos a ser seguida durante o período de desenvolvimento. Este componente funciona como um verdadeiro interpretador da linguagem de modelagem (definição) de processos.
4. O mecanismo de consultas (***query mechanism***) é responsável por realizar de forma otimizada consultas acerca das informações do modelo de processo, da base de conhecimento e das instâncias do processo desejadas pelo usuário.
5. E, finalmente, o gerente de aquisição de conhecimento (***knowledge acquisition***) que traduz uma descrição estruturada dos recursos, agentes e tarefas em um modelo de processo.

3.2.6 AgentProcess

Por fim, o simulador ***AgentProcess*** (Simulação de Processo de Software baseado em Agentes Cooperativos) é uma ferramenta de simulação integrada ao ambiente Prosoft-APSEE (Nunes 1992) baseado em conhecimento que utiliza agentes inteligentes para representar o comportamento dos desenvolvedores envolvidos em um projeto de desenvolvimento de software. Este trabalho proposto por Silva (2000) calcula o tempo simulado em função das habilidades dos agentes do processo e a afinidade entre os mesmos conforme as tabela 1 e 2 apresentadas a seguir.

Tabela 1. Cálculo da variação do tempo em função da habilidade do agente (Silva 2000).

Habilidade Geral do Agente (HAB)	***Tempo_Hab =***
HAB <= 0.3	Acréscimo de 30% no tempo previsto.
HAB > 0.3 & HAB <= 0.4	Acréscimo de 20% no tempo previsto.
HAB > 0.4 & HAB <= 0.5	Acréscimo de 10% no tempo previsto.
HAB > 0.5 & HAB <= 0.6	Tempo igual ao previsto.
HAB > 0.6 & HAB <= 0.7	Decréscimo de 10% no tempo previsto.
HAB > 0.7 & HAB <= 0.8	Decréscimo de 20% no tempo previsto.
HAB > 0.8 & HAB <= 1	Decréscimo de 30% no tempo previsto.

Tabela 2. Cálculo da variação do tempo em função da afinidade do agente (Silva 2000).

Afinidade Geral do Agente (AFIN)	***Tempo_Afin =***
AFIN <= 0.5	Acréscimo de 30% no tempo previsto.
AFIN > 0.5 & AFIN <= 0.6	Acréscimo de 10% no tempo previsto.
AFIN > 0.6 & AFIN <= 0.7	Tempo igual ao previsto.
AFIN > 0.7	Decréscimo de 10% no tempo previsto.

Nas tabelas 1 e 2 nota-se que de acordo com os graus de habilidade e afinidade dos agentes o tempo necessário para o(s) agente(s) realizar(em) as tarefas pode aumentar, diminuir ou permanecer o mesmo do tempo previsto pelo gerente (usuário).

3.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Simulação de processos de software é uma abordagem que tem tido sucesso em muitas de suas aplicações como mostrado nos trabalhos relacionados. Isto ocorre com muita freqüência quando se trata do estudo de um processo de software em específico, como é o caso de Donzelli (2006) ao realizar experimentos no *Software Engineering Laboratory* (SEL) da NASA. Entretanto, ainda existem muitos desafios nesta área (mais acentuados em propostas que visam simulação de processos de software não específicos) que levam a simplificações que comprometem a utilização desses modelos em organizações reais.

Muitos dos trabalhos relacionados contribuíram para o desenvolvimento do modelo proposto neste trabalho. No que diz respeito à fundamentação teórica, algumas das abordagens utilizadas pelos modelos descritos na seção anterior inspiraram algumas decisões e a utilização de uma abordagem híbrida de simulação, que incorpora diversas técnicas e metodologias a um só

modelo de simulação a fim de aproximar os resultados da realidade. Um exemplo de técnica utilizada é o modelo COCOMO, porém em sua versão dois.

Além disso, alguns modelos de simulação apresentam resultados alcançados pela sua adoção, por exemplo, o impacto de atividades de verificação e validação no processo de desenvolvimento de software (Rus 2002).

4. PROPOSTA DE UM MODELO DE SIMULAÇÃO DE PROCESSO DE SOFTWARE

Este capítulo apresenta o problema e o resultado da pesquisa realizada para construção de um modelo de simulação de processo de software baseado em conhecimento.

A seguir, são apresentados os pontos que levaram à construção desse modelo. O problema é discutido na seção 4.1, enquanto que o contexto para qual o modelo de simulação é proposto, ou seja, o ambiente WebAPSEE é brevemente descrito na seção 4.2. Os objetivos específicos do trabalho são tratados na seção 4.3 como requisitos do modelo. Na seção 4.4 o modelo propriamente dito é apresentado e na seção 4.5 as principais decisões tomadas para a definição do modelo são apresentadas. As limitações do modelo são apresentadas na seção 4.6. Finalmente, a seção 4.7 apresenta as considerações finais do capítulo.

4.1 O PROBLEMA

Alguns simuladores como (Mi e Scacchi 1990) conseguem reproduzir de forma fidedigna um modelo de processo a partir de dados inseridos na sua base conhecimento. Entretanto, um grande problema destes simuladores é a extração de conhecimento de organizações reais para compor a base de conhecimento para simulação. Neste ponto o modelo aqui proposto se distancia das abordagens similares.

No trabalho aqui apresentado, a intenção é construir um modelo de simulação de processo de software que, a partir de um processo modelado e instanciado no ambiente WebAPSEE[1], simule a execução do mesmo através da análise de experiências passadas, ou seja, baseada em ocorrências em processos executados anteriormente. A idéia de adotar o ***log*** estruturado dos passos de execução de processos no WebAPSEE para fins de descoberta de conhecimento e simulação foi proposta por Paxiúba *et al* (2005). O ***log*** registra os eventos do processo, tais como o início e término de atividades, a ocorrência de atrasos, entre outros. Assim, o ***log*** possui um conhecimento importante acerca de uma organização de desenvolvimento de software que tem o potencial de ser útil para alimentar o mecanismo de simulação.

Na figura 7 é apresentada a estrutura do ***log*** de eventos do ambiente WebAPSEE.

[1] O ambiente WebAPSEE está em desenvolvimento pelo Laboratório de Engenharia de Software da UFPA (LABES-UFPA). Trata-se de um ambiente que permite a definição e acompanhamento do processo de software a partir do gerente e todos os demais profissionais participantes (ver seção 4.2).

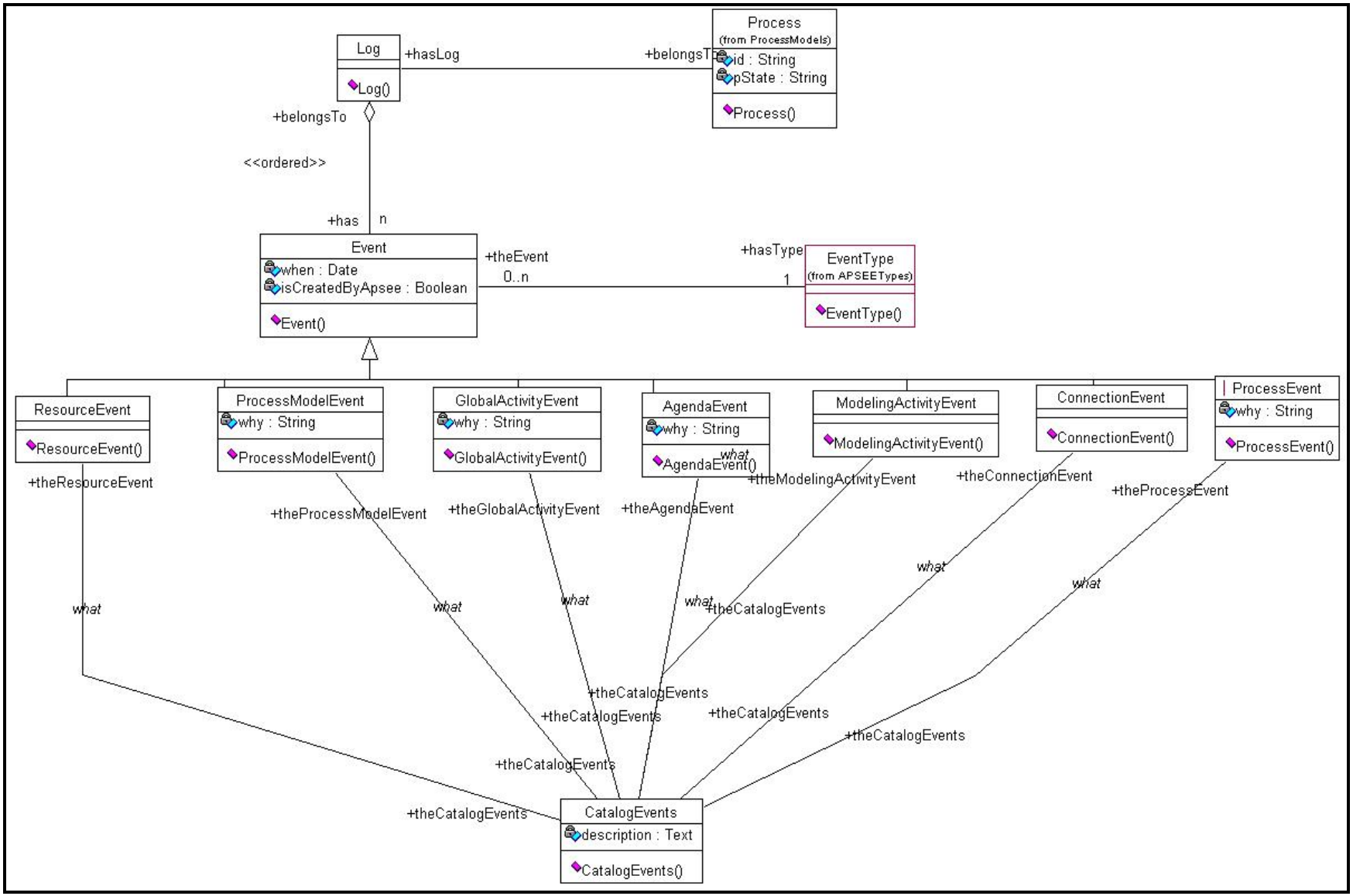

Figura 7. Estrutura do *log* de eventos do ambiente WebAPSEE (Paxiúba *et al* 2005).

Conforme descrito em LABES-UFPA (2006), o pacote Log compreende todos os eventos do WebAPSEE. Seu objetivo é armazenar informação sobre o que aconteceu (atributo *What*), quando aconteceu (atributo *When*), quem foi o responsável, neste caso pode ser o mecanismo de execução (atributo *isCreatedByApsee* indica se o responsável é o APSEE Manager), o agente ou a atividade (atributo *Who* indica o responsável), e a razão da ocorrência do evento (atributo *Why*). O valor do atributo *What* está relacionado com uma instância específica do catálogo de eventos (classe EventsCatalog) ou pode armazenar a identificação da regra que foi responsável pela ocorrência. Cada processo de software no ambiente APSEE tem um *log* (ou seja, uma instância da classe **Log**), que é composta por eventos (classe **Event**). Eventos estão relacionados aos seguintes componentes do WebAPSEE: recursos, modelos de processos, atividades (visão global, visão do desenvolvedor e eventos de modelagem), conexões entre atividades, e o próprio processo de software.

Baseado neste modelo, a ferramenta possibilita realizar experimentos de baixo custo e tempo dentro do computador, oferecendo orientações aos engenheiros e gerentes de processos de software de como prosseguir, ou seja, como dar o andamento necessário ao processo em avaliação (simulado). Além de contribuições na indústria, a ferramenta tem o potencial de originar inúmeras contribuições na área acadêmica com a possibilidade de estudos e avaliações (em um curto período de tempo) de modelos de processos de software (novas propostas e comparações) e das respectivas influências que estes podem sofrer dadas diversas alterações.

Nas tabelas 3 e 4 são apresentadas uma análise e algumas definições em alto nível das necessidades e características do simulador. O foco desta análise é no problema e nas necessidades dos interessados na ferramenta e dos usuários da mesma e porque estas necessidades existem. A ferramenta tem como objetivo fornecer um ambiente de simulação de processo de software que utilize o conhecimento obtido dos processos executados no ambiente WebAPSEE.

Tabela 3. Descrição do Problema.

Problema	Falta de ferramentas de apoio a decisão para os gerentes de processo de software.
Pessoas Afetadas	Todos os envolvidos no processo de desenvolvimento de software na organização.
Impacto	Dificuldade de gerenciamento do processo de software. Falta de conhecimento dos responsáveis pela gerência dificultam a tomada de decisões.
Solução/Benefícios	Facilitar o Gerenciamento de Riscos. Detectar possíveis anomalias no processo antes que este seja executado. Avaliação dos modelos de processo de software.

Tabela 4. Descrição do Produto.

Para:	Organizações de desenvolvimento de software.
Necessidade:	Auxiliar o gerente de processo na tomada de decisões.
O que:	<NOME DO SIMULADOR>.
Benefícios?	Simular processos de software instanciado, procurando gerenciar riscos e detectar inconsistências no mesmo que possam gerar problemas durante a sua execução, como aumento de custos, má alocação de pessoas, má distribuição de tarefas e comprometimento da qualidade do(s) produto(s) obtido(s).

Todas as pessoas envolvidas no processo de desenvolvimento de software de uma organização que utilize a ferramenta podem ter seu trabalho afetado pela adoção do mesmo. O gerente/engenheiro de processos constitui o principal usuário potencial da ferramenta, que utilizará a mesma para avaliação prévia dos modelos de processo de software. Os desenvolvedores, também, podem ser beneficiados (de maneira indireta) pela adoção de processo que levam em consideração suas habilidades, afinidades e experiência adquirida em processos anteriores. Por exemplo, um gerente pode descobrir que a carga de trabalho de um determinado desenvolvedor está demasiadamente alta em alguns trechos do processo, podendo assim equilibrar sua carga de trabalho a partir de resultados da simulação. Uma outra possibilidade ainda considerando os desenvolvedores é quanto às suas habilidades. Segundo Boehm (1981) um desenvolvedor tem uma produtividade maior quando há um casamento entre as habilidades requeridas pela atividade e as habilidades do desenvolvedor.

A seguir, a tabela 5 apresenta alguns benefícios esperados com a utilização da ferramenta.

Tabela 5. Benefícios esperados pela utilização da ferramenta.

Benefícios	Características de Suporte
O histórico de processos semelhantes é utilizado para avaliar novos processos.	A ferramenta utiliza o histórico de eventos de processos executados para simular a execução de novos processos.
Auxiliar o Gerente de Processo na tomada de decisão.	O gerente de processo pode simular diversas alternativas para o seu novo cenário.
Validação do Modelo de Processo.	O processo modelado pode ser "validado"[2] pela ferramenta através da análise dos resultados da simulação.

4.2 O AMBIENTE WebAPSEE

O ambiente WebAPSEE é um PSEE baseado em tecnologias de distribuição que provê serviços via internet para seus clientes para gerenciar e desenvolver processos de software. Este ambiente é baseado na noção de que para se prover flexibilidade na execução de processo, um PSEE tem que conformar os padrões abertos.

Além da utilização de software livre, o ambiente tem compromisso por flexibilizar a execução do processo dinamicamente, ou seja, em tempo de execução é possível que se altere o processo, excluindo atividades, deslocando pessoas e recursos de atividades e inserindo em outras, por exemplo.

Este ambiente possui duas visões: a do gerente e a do desenvolvedor. A figura 8 mostra a visão do gerente (*Manager Console*) como parte mais ampla da figura e a visão do desenvolvedor (*Task Agenda*) na parte inferior direita da imagem.

[2] O termo validado está no sentido de que uma simulação foi realizada e o processo conseguiu atingir um estado final de sucesso, ou seja, sem ultrapassar restrições de custo e prazo.

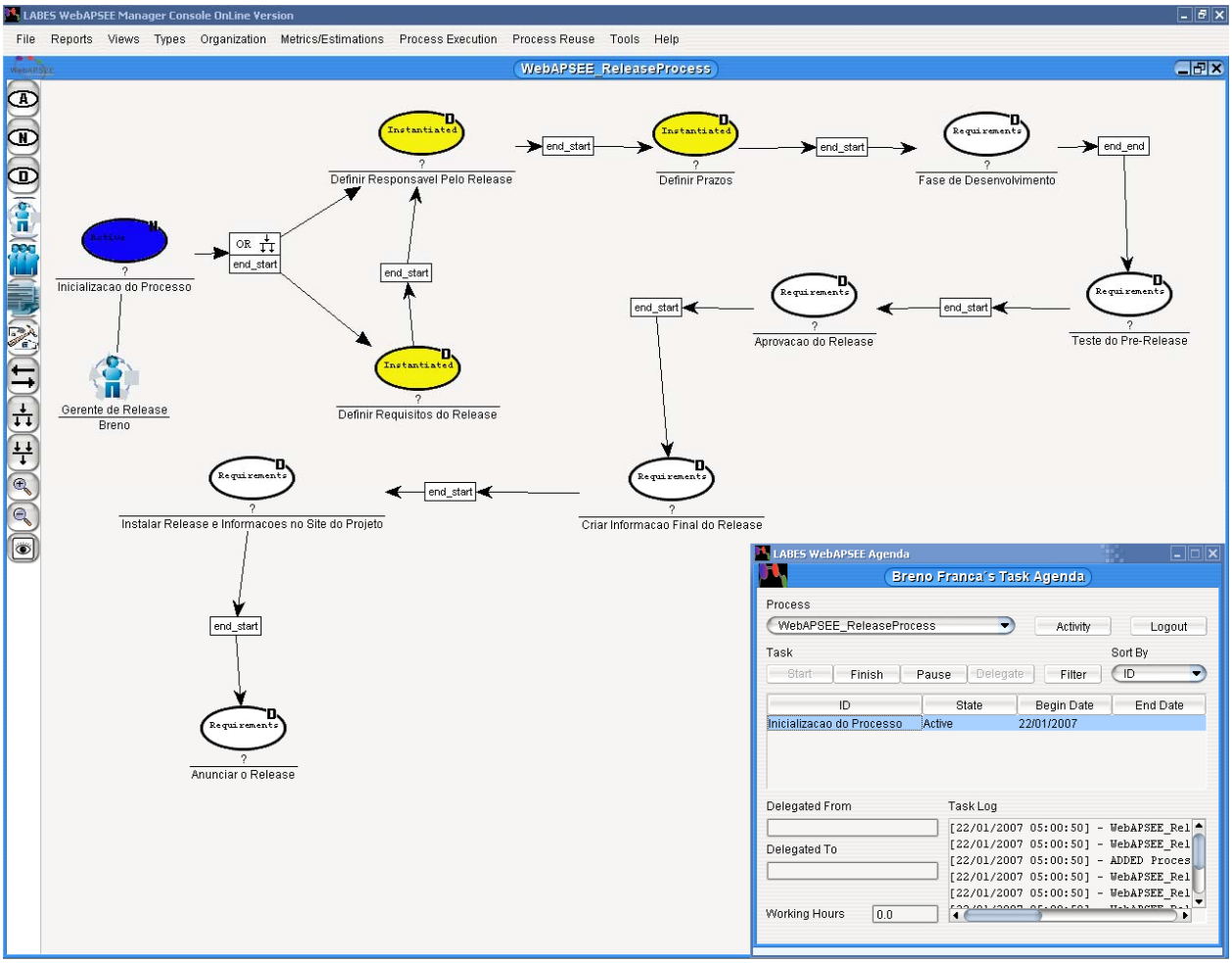

Figura 8. Interface Gráfica do Ambiente WebAPSEE.

No WebAPSEE o gerente é responsável, juntamente com os engenheiros de processo, pela modelagem e instanciação do processo, ou seja, definir o fluxo das tarefas a serem realizadas, assim como seus prazos, custos, estimativas e métricas a serem utilizadas, artefatos (artefatos de software) produzidos e consumidos e pessoas e grupos envolvidos. Tudo isto pode ser feito através do *Manager Console*, principal interface que contém um editor de processos, juntamente com formulários para cadastro e inserções de informações sobre a organização e o processo como um todo.

Na visão do desenvolvedor, o processo é visto como uma Agenda onde tem suas tarefas a serem realizadas, assim como os artefatos (documentos, códigos, entre outros dados concretos geralmente voltados à documentação de um projeto) a serem produzidos e os recursos a serem utilizados. O agente por meio dessa Agenda fornece o *feedback* para o gerente sobre o estado de

suas atividades. Esse *feedback* é imediatamente visualizado no editor de processos do gerente, por onde este também pode acompanhar o andamento do processo.

4.3 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são apresentados a partir dos principais requisitos que devem ser encontrados na ferramenta. Estes requisitos foram obtidos a partir de consultas à literatura, dentre elas é importante citar (Silva 2000) e (Mi e Scacchi 1990), restrições relacionadas ao ambiente WebAPSEE e a partir da utilização de ferramentas de simulação de propósito geral (Freitas Filho 2001). Estes requisitos são listados na tabela 6.

Tabela 6. Requisitos do Simulador.

Requisito	Descrição
Simular a execução do processo até a atividade final	A partir do modelo de um processo a ferramenta deve ser capaz de simular a execução deste até a atividade final.
Simular o comportamento dos agentes desenvolvedores	A ferramenta deve considerar características como afinidade, habilidade, custo, produtividade dos desenvolvedores e atividades passadas.
Permitir o acompanhamento do progresso da simulação	A ferramenta deve exibir de algum modo à evolução da simulação para que esta possa ser acompanhada pelo gerente de processo.
Detectar anomalias no processo	A ferramenta deve comparar o processo que está sendo simulado com os demais processos executados anteriormente e ser capaz de exibir relatório detectando eventuais anomalias como má alocação de pessoas, má estimativa de prazos, distribuição de tarefas inadequadas.
Fornecer o grau de acurácia[3] da simulação	A ferramenta deve atribuir grau de acurácia da simulação executada, medida em função da média e desvio padrão das incertezas acumuladas ao determinar a duração das atividades.
Permitir a simulação de tarefas concorrentemente	A ferramenta deve permitir que tarefas concorrentes sejam simuladas.
Permitir regular a velocidade da simulação	A ferramenta deve permitir que a velocidade da simulação possa ser ajustada de acordo com a necessidade do gerente.
Emitir relatório do resultado da simulação	A ferramenta deve emitir relatório do resultado obtido com a simulação do processo, indicando possíveis anomalias e estatísticas coletadas durante a simulação. Este relatório deve ser o mais completo e inteligível possível.
Fácil utilização	A ferramenta deve possuir alto grau de usabilidade para facilitar sua aceitação pelos gerentes de processo.

Simular a execução completa do processo significa que a partir do modelo de processo instanciado (com atributos como prazos, pessoas, recursos e artefatos definidos) o funcionamento do simulador é baseado no envio de chamadas à máquina de execução de processos do ambiente WebAPSEE com base nas informações mantidas pelo controlador da simulação. Este controlador mantém informações sobre o cálculo e andamento do tempo de realização do sistema, gerência das restrições do projeto, entre outros.

[3] A palavra acurácia é utilizada neste trabalho com o sentido de proximidade entre o valor obtido experimentalmente e o valor verdadeiro na medição de uma grandeza física, de acordo com o dicionário *Houaiss*.

Quanto ao comportamento dos agentes, este é simulado através do grau de suas habilidades e o histórico de sua produtividade, ou seja, é considerada a produtividade de quando o agente realizou tarefas nos moldes da tarefa atual. Este requisito foi levado em consideração a partir da utilização da habilidade dos agentes como parâmetro importante para definição do tempo necessário para o agente realizar a atividade no modelo definido por Silva *et al* (2000).

O andamento da simulação deve ser visualizado através do *Manager Console* de forma semelhante como o gerente atualmente acompanha o decorrer de uma execução real de processo. Isto contribui para que o usuário tenha um acompanhamento familiar àquilo que é fornecido pelo ambiente WebAPSEE quando ocorre a execução de um processo.

A detecção de eventuais anomalias como má alocação de pessoas, má estimativa de prazos, distribuição de tarefas inadequadas pode ser feita ao comparar os dados correntes da simulação com as restrições fornecidas pelo usuário no início da simulação. Sendo assim, custos elevados, pessoas sobrecarregadas de tarefas e prazos não cumpridos são automaticamente registradas. E, ao final da simulação, relatórios e estatísticas pode ser gerados para documentação e facilitar o processo de análise.

O grau de acurácia tem como base o grau de acurácia mínimo exigido da simulação (fornecido pelo usuário no início da simulação via formulários para configuração dos parâmetros da simulação) e é calculado pela média e o desvio padrão dos graus de certeza obtido para cada atividade. Este grau mínimo de acurácia é fator-base utilizado no cálculo da duração das atividades e geração de relatórios e estatísticas. Para mais detalhes sobre os parâmetros da simulação, ver seção 4.4.1 e 4.4.2.

Simular tarefas concorrentes foi um dos problemas encontrados durante a especificação do modelo e sua solução proposta não foi baseada em nenhuma outra da literatura. Esta solução é descrita em detalhes na seção 4.5.

Durante a simulação do processo, é possível alterar a velocidade da simulação para melhor acompanhar as ocorrências no processo e verificar os pontos exatos onde podem ocorrer possíveis falhas ou melhorias no processo. O ponto máximo deste controle de velocidade é a pausa da simulação, quando as variáveis de estado são todas guardadas para uma continuação em um momento futuro.

Ferramentas de simulação comumente geram as saídas da simulação, isto é, seus resultados no formato de relatórios (Freitas Filho 2001) (Drappa 2000). Nesta proposta um dos

requisitos para o usuário é que sejam emitidos relatórios contendo gráficos, tabelas comparativas, estatísticas, assim como o resultado final da simulação quanto à violação de restrições de prazos e custos.

A partir dos requisitos coletados para o simulador de processos de software, o funcionamento, em alto nível, deste simulador segue o diagrama UML de casos de uso mostrado na figura 9.

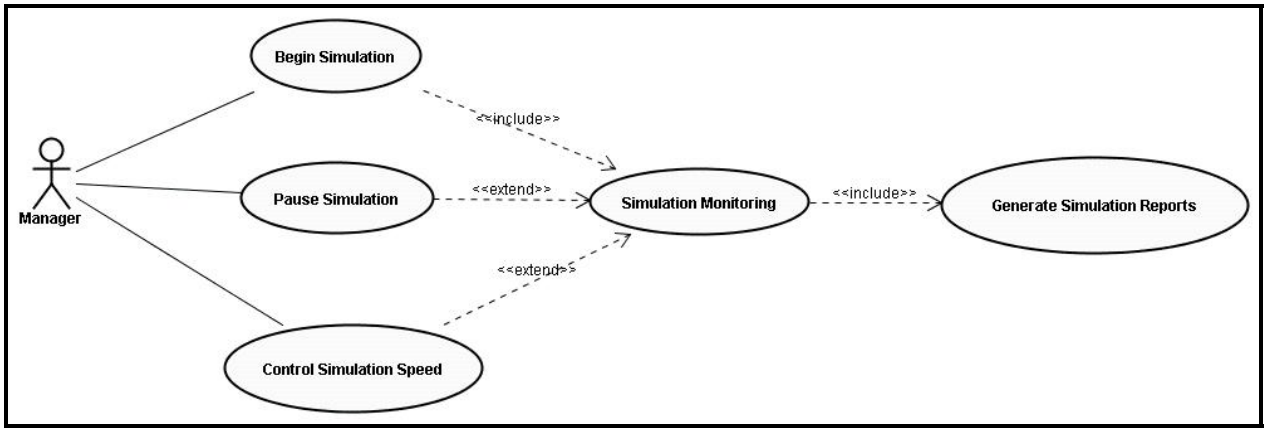

Figura 9. Diagrama UML de Casos de Uso para o Simulador de Processos.

A figura do *Manager* (gerente) como único ator da ferramenta não representa que este seja o único beneficiado, somente que é o usuário direto da ferramenta. O caso de uso ***Begin Simulation*** (Iniciar Simulação), que inclui ***Simulation Monitoring*** (Monitorar Simulação), denota a ação do gerente para iniciar a simulação a partir de um dado modelo de processo com suas atividades, prazos, recursos, pessoas e artefatos definidos. Ao fim desta simulação relatórios são gerados como mostra o caso de uso ***Generate Simulation Reports*** (Gerar Relatório do Resultado da Simulação). Deve-se notar que a simulação pode, ainda, ser pausada no caso de uso ***Pause Simulation*** (Pausar Simulação) e ter sua velocidade de execução regulada no caso ***Control Simulation Speed*** (Regular Velocidade de Simulação), isto é, aumentada ou diminuída.

Este diagrama, assim como outros que compõem a documentação do modelo de simulação foram escritos no idioma inglês devido à cooperação em pesquisa realizada no período de outubro a meados de novembro de 2006 com o *Abteilung Software Engineering, Institut für Software-technologie* da *Universität Stuttgart*.

4.4 MODELO DE SIMULAÇÃO

O modelo de simulação aqui proposto é classificado com um **Modelo de Simulação Orientado a Eventos.** Outros mecanismos de simulação foram considerados, uma discussão acerca da decisão tomada é apresentada na seção 4.5. Na figura 10 o Diagrama de Atividades UML define a seqüência de passos a ser seguida no decorrer da simulação.

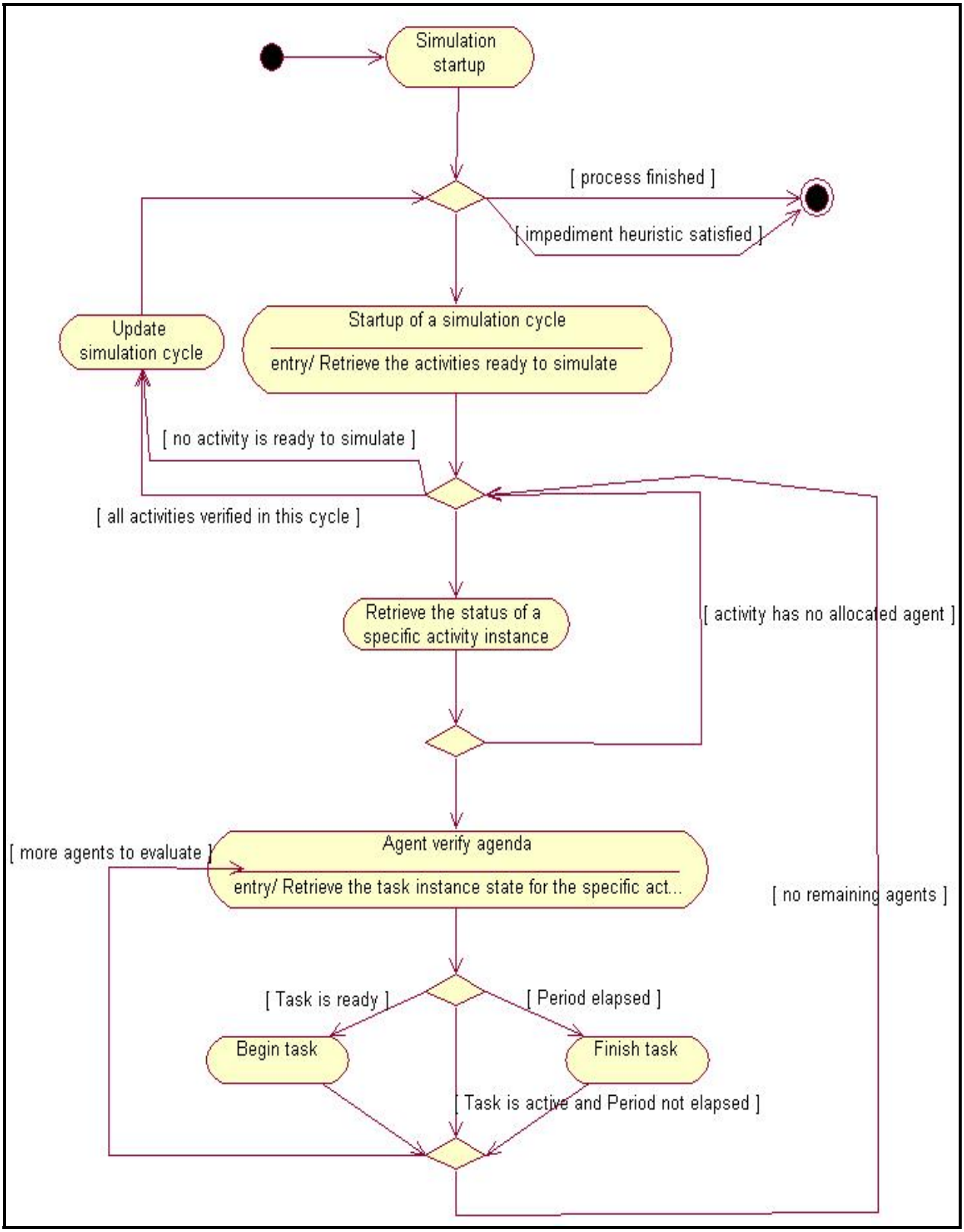

Figura 10. Diagrama de Atividades do Modelo de Simulação.

Ao usuário iniciar a simulação, o modelo da Figura 10 é disparado. Tem início o primeiro ciclo com o nó ***Simulation Startup*** (Inicialização da Simulação), onde é realizada a inicialização das variáveis e o relógio da simulação. Uma avaliação é realizada logo no início, onde um teste é realizado para saber se a simulação deve prosseguir, isto é, se haverá um novo ciclo de simulação. A pergunta a ser feita neste instante é se todas as atividades do processo foram concluídas ou há alguma situação de impedimento de continuidade da simulação na condição ***Process Finished*** ou ***Impediment Heuristic Staisfied*** (Processo Terminado ou Heurística de Impedimento Satisfeita), por exemplo, término de recursos, prazos ultrapassados entre outros. Caso a resposta seja sim, a simulação deve prosseguir. Caso contrário, a simulação é terminada. Então, as atividades que estão prontas para simulação são selecionadas no momento em que é alcançado o nó ***Startup of a Simulation Cycle*** (Inicialização de um Ciclo de Simulação).

Prosseguindo a simulação, deve ser realizado o teste para saber se todas atividades deste ciclo já foram verificadas. Caso a resposta seja verdadeira ou não exista atividade(s) a ser(em) simuladas no ciclo, o ciclo de simulação é atualizado no nó ***Update Simulation Cycle***. Já enquanto houver atividade a ser simulada será realizada uma iteração simular o ciclo para cada atividade. Neste ponto será feita uma iteração dos agentes envolvidos para cada atividade. Para cada agente o nó ***Agent Verifies Agenda*** (Agente Verifica Agenda) retorna uma tarefa especifica e verifica seu estado para tomar uma determinada ação. Então, há uma ramificação em três caminhos possíveis para prosseguir:

1. Caso a tarefa esteja pronta, então: inicia-se a tarefa através do nó ***Begin Task*** (Iniciar Tarefa) e continua-se a verificação para os agentes restantes.
2. Caso a tarefa esteja ativa, porém sua duração não tenha terminado (i.e, o número de ciclos de simulação não tenha sido ultrapassado), então: continua-se a verificação para os agentes restantes;
3. Caso a duração calculada para a tarefa tenha terminado, então: finaliza-se com o nó ***Finish Task*** (Terminar Tarefa) a tarefa e continua-se a verificação para os agentes restantes.

   Este procedimento é realizado até que o teste da condição inicial seja satisfeito.

A duração das tarefas é calculada através de uma função que recebe como entrada os agentes envolvidos, os artefatos consumidos (entrada) e a serem produzidos (saída), os recursos a serem utilizados, e o tipo da atividade. É realizada, então, uma busca na base histórica a fim de

determinar o grau de semelhança entre essa atividade e outras atividades passadas. A partir desse grau de semelhança e do conhecimento extraído das métricas do repositório do WebAPSEE são feitas inferências para determinar a duração (em horas) da atividade.

Com o objetivo de calcular a duração requerida para realizar uma tarefa deve-se que considerar duas situações:

1. A primeira diz respeito ao conhecimento já acumulado pela organização que utiliza o ambiente WebAPSEE. Então, uma diversidade de dados da execução está envolvida tais como *log* de eventos, duração das atividades, artefatos de software envolvidos, recursos e agentes de software. Considerando esse cenário, uma abordagem chamada RBC[4] (Raciocínio Baseado em Casos - *Case-Based Reasoning*) é adotada, devido esta abordagem levar em consideração **casos** passados para prever o que ocorre no **caso** (situação) presente. O caso (ou conjunto de casos) que tiver o maior grau de similaridade – calculado por uma função de similaridade – deve, então, ser adaptado para a situação atual.
2. A outra situação refere-se ao caso em que a organização não possui dados de execuções passadas, isto é, a organização acabou de instalar o ambiente e tenha, talvez, poucos processos executados (1 ou 2) processos. Esta situação se aplica também quando a função do caso RBC acima retorne resultados da base histórica que tenham similaridade muito baixa, isto é, não atinjam um limiar mínimo de similaridade (definido pelo usuário) para que os dados sejam úteis, o que provavelmente iria implicar em menor acurácia do resultado da simulação. Neste caso, conhecimento empírico sobre processos de software em geral são utilizados para determinar a duração das atividades. Sendo assim, o modelo COCOMO II (*COnstructive COst MOdel*) (Boehm 2000) será adotado para estimar a duração e esforço necessários para executar as tarefas.

Adicionalmente, algumas heurísticas podem ser incorporadas ao modelo em formato de regras. Estas regras devem ser aplicadas para aumentar ou diminuir indicadores como esforço, tempo de duração, custo e/ou qualidade (porcentagem), isto é, sentenças que informam em quanto o modelo de simulação deve aumentar (ou diminuir) estas variáveis. Em resumo, estas regras empíricas são utilizadas para inserir realismo ao modelo de simulação [fatos reais

[4] Ver seção 4.4.1.

descobertos em estudos de gerência de processo de software, por exemplo, a Lei de Brooks (Brooks 1995)].

4.4.1 Raciocínio Baseado em Casos (RBC)

von Wangenheim (2003) define Raciocínio Baseado em Casos como:

> "Uma técnica de inteligência artificial que consiste em um enfoque para solução de problemas e para o aprendizado baseado em experiências passadas. RBC resolve problemas ao recuperar e adaptar experiências passadas – chamadas casos – armazenadas em uma base de casos. Um novo problema é resolvido com base na adaptação de soluções de problemas similares já conhecidas".

Na definição de um modelo RBC, quatro etapas são necessárias: (1) a modelagem/representação dos casos, (2) a função de similaridade, (3) o método de recuperação de casos e (4) a o método de adaptação dos casos. Uma última etapa é a retenção de novos casos, entretanto, esta não é obrigatória segundo a literatura (von Wangenheim 2003) e não será aplicada neste modelo porque os casos gerados com a simulação não são fatos concretos, o que poderia acarretar em um maior grau de incerteza nas inferências tomadas a partir dos resultados da simulação. A decisão acerca dos métodos utilizados em cada etapa é detalhada na seção 4.5.

4.4.1.1 Representação de Casos

Para representar os casos é utilizada a abordagem orientada a objetos, visto que o sistema e seu modelo de dados já apresentam as informações neste paradigma. Nessa abordagem, cada caso é representado como um objeto (ou vários objetos em associação) contendo as informações relevantes para determinar uma situação. Um caso é dividido em **sintoma/problema** e **solução**, onde **sintoma/problema** a parte do caso que denota a descrição do problema, e a **solução** as ações tomadas para resolver este problema. Assim, a estrutura do caso:

- **Sintoma/Problema:** são atividades normais[5] com estado "terminada" (*finished*). Estas atividades possuem como atributos o nome, o tipo da atividade, a coleção de recursos requeridos para realizá-la, a coleção de recursos pessoas (agentes ou grupos) e os artefatos consumidos e produzidos pela mesma.
- **Solução:** A solução incorpora o tempo e o custo necessários para executar a atividade em questão.

Esta descrição do problema coincide com o modelo de classes das atividades planas do

[5] Atividades normais são atividades planas de um processo de software, ou seja, não possuem decomposição em sub-processos e necessitam de recursos, agentes e envolvem artefatos de entrada e saída para realizá-la.

ambiente WebAPSEE. Sendo assim, os casos utilizam a mesma estrutura do ambiente para representar os casos (figura 11).

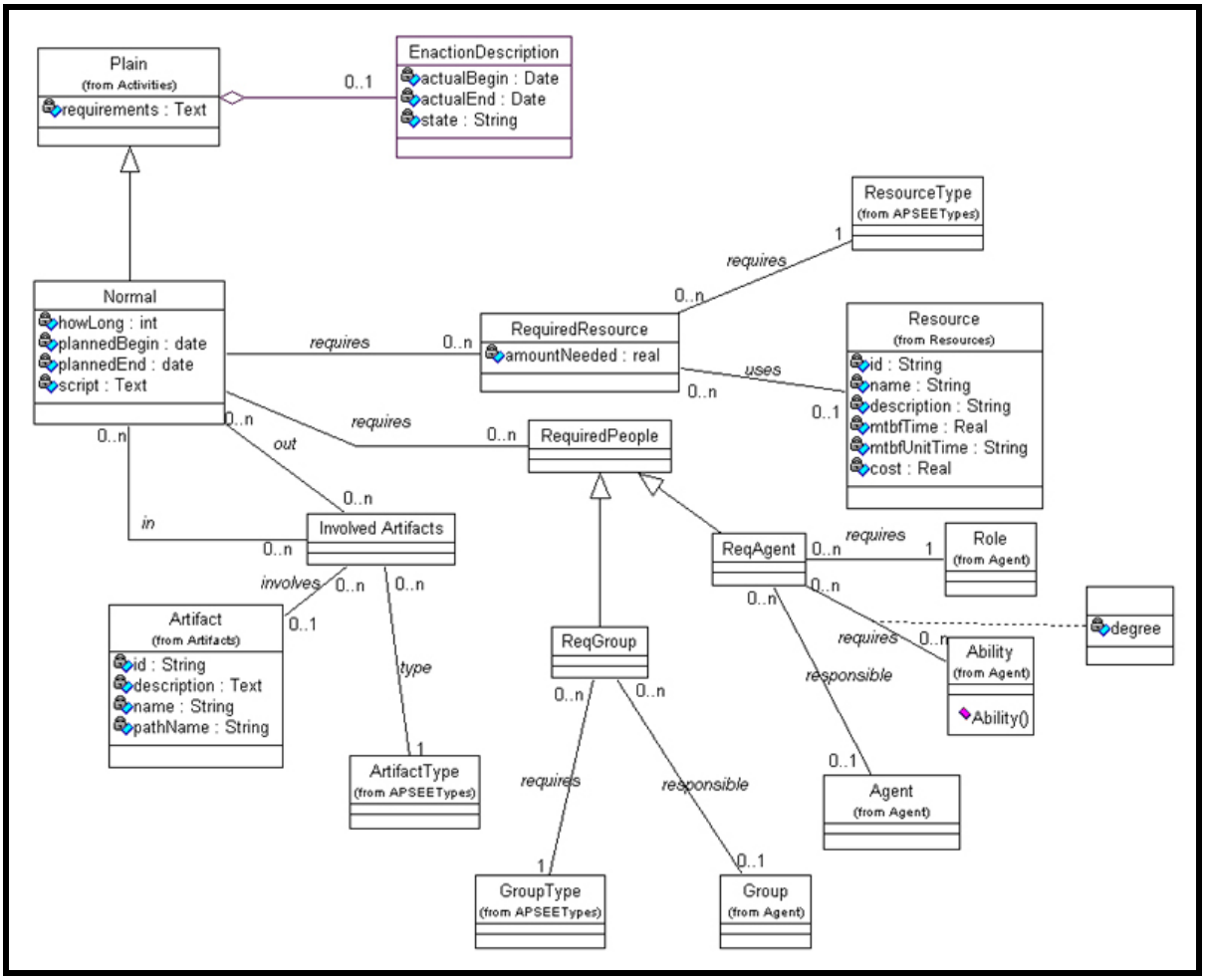

Figura 11. Diagrama de Classes de atividades planas, pacote *plainActivities* (Lima Reis 2003).

O diagrama de classes do pacote *PlainActivities* é mostrado na Figura 11 e detalha a definição de atividades simples (*Plain*) como normais. As atividades normais (classe *Normal*) necessitam de recursos (classe *RequiredResource*), agentes e/ou grupos (classe *RequiredPeople*) e envolvem artefatos de entrada e saída (classe *InvolvedArtifacts*), os quais são descritos de forma abstrata (tipos) e instanciada (identificadores dos componentes). Ambos os tipos de atividade possuem uma descrição de sua execução (classe *EnactionDescription*) que indica o estado da atividade, suas datas de início e fim.

A idéia de armazenar os tipos necessários na descrição da atividade serve para aumentar a flexibilidade da definição do processo. O projetista de processo pode, durante a modelagem, definir apenas o tipo de recurso necessário e decidir qual recurso vai ser efetivamente utilizado no

início da execução da atividade. Atividades normais possuem cronograma abstrato com duração em dias (*how long*), datas de início e fim planejadas (*planned_begin* e *planned_end*), o *script* com os objetivos da atividade; e pré e pós-condições para a sua execução.

O estado da atividade (*state*) na classe *EnactionDescription* é o atributo que armazena a situação de cada atividade e serve de referência para a transição de estados de todo o processo de software. Os possíveis estados de uma atividade simples são descritos a seguir:

- ***Waiting:*** as dependências da atividade ainda não estão satisfeitas;
- ***Ready:*** pronta para começar;
- ***Active:*** a atividade está sendo realizada pelos agentes responsáveis (pelo menos um agente está trabalhando na atividade);
- ***Paused:*** todos os agentes solicitaram pausa da atividade;
- ***Finished:*** a atividade foi concluída. Se a atividade for cooperativa, significa que todos os agentes concluíram;
- ***Cancelled:*** a atividade foi cancelada antes de iniciar;
- ***Failed:*** a atividade falhou após o início por decisão dos agentes ou do gerente.

4.4.1.2 Função de Similaridade

A função de similaridade é uma função complexa composta por diversas outras funções de similaridades. Ou seja, para calcular a similaridade global entre duas atividades é necessário calcular a afinidade entre os componentes da atividade-caso e a atividade-consulta (problema).

Considerando **atividade-caso** como sendo uma atividade contida na base de casos (base histórica) e **atividade-consulta** a atividade em questão para a qual se procura obter uma similar. Por exemplo, dadas duas atividades normais N1 e N2 é preciso calcular a similaridade entre seus atributos [Id, e Name (String) - somente atividades terminadas] e seus relacionamentos (tipo da atividade, agentes/grupos requeridos, recursos e artefatos). Recursivamente, os atributos e relacionamentos têm funções próprias para o cálculo de similaridade local levando em consideração as particularidades do tipo relacionado (comparação de *strings*, inteiros, instâncias da classe APSEEType, entre outros). Sendo assim, a similaridade global (entre atividades) é uma soma ponderada das similaridades locais normalizada para o intervalo [0 ; 1].

Assim,

$$\Sigma\ sim_{local\ i}\ (N1.atributo/relacionamento,\ N2.atributo/relacionamento)\ x\ W_i$$

Onde W_i é o peso atribuído ao atributo ou relacionamento.

Os pesos por atributo ou relacionamento estão distribuídos de acordo com a tabela 7.

Tabela 7. Distribuição de Pesos para os atributos e relacionamentos das atividades.

Atributo / Relacionamento	Peso	
	Global	Local
Nome	1	-
Tipo da Atividade	2	-
Coleção de Recursos	1	-
Recurso	-	4
Id	-	6
Custo	-	2
Classe	-	2
Tipo de Recurso	-	6
Coleção de Pessoas Requeridas	3	-
Agente Requerido	-	-
Agente	-	3
Id	-	8
Custo por hora	-	2
Cargo	-	6
Id	-	3
Subordinado à	-	2
Tipo de Cargo	-	5
Coleção de Habilidades	-	1
Habilidade	-	3
Grau	-	2
Tipo de Habilidade	-	5
Grupo Requerido	-	-
Instância do Grupo	-	4
Tipo do Grupo	-	6
Coleção de Artefatos	3	-
Instância do Artefato	-	4
Tipo do Artefato	-	6

Estes pesos foram distribuídos inicialmente de acordo com a experiência do autor em modelagem de processos de software e com algumas informações da literatura. Por exemplo, Pressman (2005) afirma que as pessoas são componentes-chave para a realização do processo, o que contribui para que o atributo "Coleção de Pessoas Requeridas" tenha peso mais alto. Os valores reais (finais) devem ser atualizados depois de realizada um ampla avaliação do modelo de simulação através de experimentações em bases históricas reais.

4.4.1.3 Método de Recuperação

O método de recuperação utilizado no modelo proposto é o de **Recuperação em Dois Níveis** (von Wangenheim 2003). No primeiro nível são recuperados os casos que obedecem a uma determinada heurística. Para o modelo proposto são selecionadas as atividades dadas como terminadas (*finished*), ou seja, as atividades executadas com sucesso e que sejam do mesmo tipo (ou subtipo) da atividade em questão.

O ambiente WebAPSEE possui uma hierarquia de tipos que define de forma abstrata os componentes do processo denominada *APSEE Types*. Esta hierarquia é apresentada no diagrama de classes UML da figura 12.

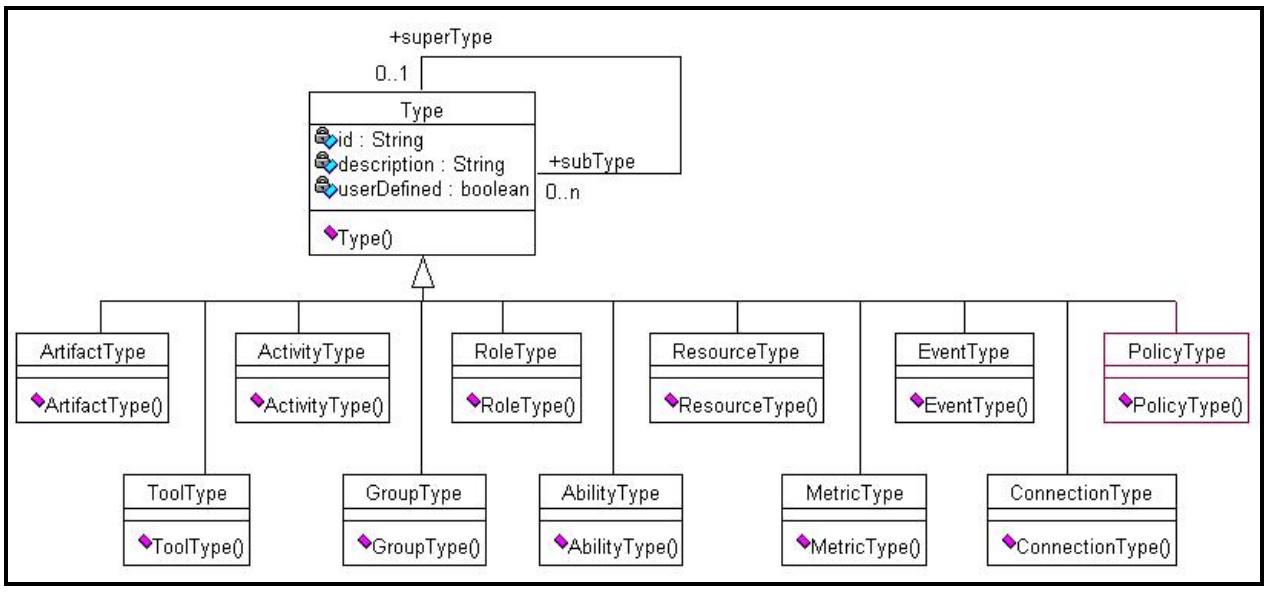

Figura 12. Hierarquia de tipos WebAPSEE (LABES-UFPA 2006).

O pacote *Types* da figura 12 contém as hierarquias de tipos para os componentes do WebAPSEE. Neste é apresentada a hierarquia de tipos principais fornecidas juntamente com o modelo, mas o usuário do ambiente poderá definir novos tipos e instâncias a partir desta hierarquia de acordo com as necessidades da organização ou do processo.

A hierarquia de tipos apresentadas na Figura 12 é decomposta em outras hierarquias, são elas: recursos, cargos, grupos, habilidades, métricas, atividades, conexões, políticas, artefatos, ferramentas e eventos. Estas hierarquias de tipos relacionadas aos componentes do WebAPSEE permitem que sejam descritos processos abstratos que podem ser refinados conforme necessário para execução ou usados para reutilização e ainda apóiam a descrição de mecanismos que lidam com elementos genéricos de processos (LABES-UFPA 2006). Essa estrutura de tipos aumenta a flexibilidade do modelo facilitando o processo de extensão do mesmo.

No segundo nível da recuperação de casos, os casos resultantes da primeira seleção são classificados pelo grau de similaridade. O caso com maior grau de similaridade é adotado como solução a ser considerada.

4.4.1.4 Método de Adaptação

O método de adaptação a ser utilizado será o de **Adaptação Nula**. Nesta abordagem, o problema atual terá sua solução baseada na solução do caso mais similar contido na base de casos. A adaptação da solução ocorrerá da seguinte forma:

Sendo a solução composta pelo **custo** e **tempo** necessários para executar a atividade, o custo será a soma dos custos dos recursos requeridos com o custo por hora dos agentes envolvidos; e o tempo será o tempo da atividade mais similar. Entretanto, o grau de semelhança influenciará no grau de acurácia da simulação. Por exemplo: considerando-se uma **atividade-problema** (atual) e uma **atividade-caso** (passada) com grau aferido de 90% de semelhança, a duração estimada da atividade-problema será igual à duração da atividade-caso e com o grau de acurácia de 90%. O grau de acurácia total da simulação é dado pela média e desvio padrão de todas as atividades do processo.

A solução parte da idéia de que problemas similares possuem soluções similares, ou seja, o tempo necessário para executar a nova atividade será similar ao tempo referente às atividades similares (se houver).

4.4.2 Modelo COCOMO II

O modelo COCOMO II (*COnstructive COst Model* versão 2) proposto por Boehm (2000) consiste em um modelo voltado para realizar estimativas de custo e prazos aplicados aos processos de desenvolvimento atuais. Para assim prover um framework analítico quantitativo e um conjunto de ferramentas e técnicas para avaliação dos efeitos do aumento da qualidade que a tecnologia de software proporciona nos custos e prazos do ciclo de vida do software.

Esta segunda versão ocorre em função da primeira versão do modelo contemplar apenas processos Cascata e não considerar aspectos inerentes à organização que desenvolve o software, assim como os aspectos sociais inerentes ao desenvolvimento de software.

Nesta versão do modelo são suportados modelos de processos bem mais avançados e atuais. Estes processos contemplam outros fatores, como por exemplo: COTS (*Commercial off the shelf*), disponibilidade de software reutilizável; grau de entendimento da arquitetura e dos requisitos; visão do mercado; tamanho do software; e confiabilidade requerida.

A maior diferença entre a primeira versão do COCOMO para a segunda está nos efeitos relacionados ao tamanho envolvendo reutilização e reengenharia, modificações nos efeitos de escala e modificações nos indicadores de custo do modelo.

4.4.2.1 Métricas utilizadas no COCOMO II

Entre outros parâmetros, o modelo COCOMO II utiliza o **tamanho do produto** esperado para estimar o esforço e os prazos necessários para construí-lo. Sendo assim, este tamanho pode ser medido com três métodos diferentes: *Object Points*, Pontos de Função Não-ajustados e Linhas de Código.

Para fins de entendimento, é suficiente saber que o modelo de simulação proposto neste trabalho utiliza Pontos de Função Não-ajustados. Entretanto, cada um destes métodos é descrito em (Boehm *et al.* 1995).

A utilização de Pontos de Função Não-ajustados deu-se em função da grande utilização e interesse por parte da comunidade, os quais podem ser evidenciados pelo grande número de associados ao IFPUG (*International Function Point Users Group*), inclusive a participação expressiva do grupo brasileiro oficial de representação do IFPUG, o BFPUG (*Brazilian Function Point Users Group*), criado desde 1998, com mais de centenas de associados (Vazquez, Simões e Albert 2003).

Outro fator considerado para adoção deste método são algumas deficiências no método de contagem por linhas de código, como abordado por Vazquez, Simões e Albert (2003): (1) a falta de padronização, (2) a falta de significado para os clientes usuários do objeto de medição, (3) a dificuldade de aplicação nas fases iniciais do processo e (4) a dependência da tecnologia utilizada.

E a utilização de *Object Points* não foi considerada devido a sua utilização no modelo COCOMO II ser aplicada apenas ao setor denominado pelo autor de *Application Composition*,

que consiste em produzir aplicações que são diversificadas o bastante para ser tratadas por soluções com pacotes pré-prontos, mas são suficientemente simples para rapidamente serem construídas a partir de componentes interoperáveis (Boehm *et al.* 1995).

4.4.2.2 Estimativa de Custo

De acordo com Boehm (1995), modelos de estimativa de custo de software frequentemente têm um fator exponencial para explicar as relativas economias ou deseconomias de escala encontradas quando um projeto de software aumenta de tamanho. Este fator é geralmente representado como o expoente *B* na equação:

$$\text{Effort} = \text{A (Size)}^{\text{B}}$$

Se $B < 1.0$, o projeto apresenta economia de escala. Se o tamanho do produto dobra, o esforço do projeto é menor que o dobro. A produtividade do projeto aumenta de acordo com o aumento do tamanho do produto.

Se $B = 1.0$, as economias e as deseconomias de escala estão balanceadas.

Se $B > 1.0$, o projeto apresenta deseconomia de escala. Isto acontece, geralmente, devido a dois principais fatores: aumento da comunicação interpessoal e aumento da integração de sistemas de grande porte. Projetos maiores terão mais custos pessoais adicionais, e assim mais caminhos de comunicação interpessoal. Integrar um pequeno produto como parte de um maior requer não só o esforço de desenvolver esse produto pequeno, mas um custo adicional de esforço para projetar, manter, integrar, e testar suas interfaces.

O esforço (*PM*), em pessoas-mês (*person-months*), é função do tamanho e dado pela fórmula:

$$PM = A \times \text{Size}^{E} \times \prod_{i=1}^{n} EM_i ,$$

onde o expoente *E* é dado por:

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

Os valores adotados para os coeficientes *A* e *B* da equação acima são 2,94 e 0,91, respectivamente, de acordo com a calibragem 2000 do COCOMO II (Boehm 2000). Já os valores para EMi e SFj dizem respeito aos multiplicadores de esforço (*effort multipliers*) e aos fatores de escala (*scale factors*), respectivamente.

Cinco fatores de escalas são considerados na versão 2 do modelo COCOMO. Estes fatores são índices responsáveis por tratar as seguintes questões: Precedência (*Precedenteness*), flexibilidade no desenvolvimento (*Development flexibility*), resolução de risco / arquitetura

(*Architecture / risk resolution*), coesão do time (*Team Cohesion*) e maturidade do processo de desenvolvimento (*Process Maturity*).

Multiplicadores de esforço são componentes do modelo utilizados para ajustar a estimativa de pessoas-mês obtida a partir do tamanho do projeto e indicadores exponenciais. Em sua totalidade, o modelo COCOMO II possui 18 multiplicadores de esforço que tratam dos seguintes aspectos: o produto, a plataforma onde o produto será implantado, pessoais (organizacional) e o projeto em si. Para mais detalhes sobre os multiplicadores de esforço são encontrados em (Boehm 2000).

4.4.2.3 Estimativa de Prazo

O tempo de desenvolvimento estimado, em meses, a partir do esforço é dado pela fórmula:

$$TDEV = C \text{ x } (PM)^{F},$$

onde o expoente F é dado pela fórmula

$$F = D + 0.2 \text{ x } (E - B)$$

e os valores de C e D são, respectivamente, 3,67 e 0,28 de acordo com a calibragem COCOMO II.2000. Os valores de E e B são os mesmo das fórmulas apresentadas na seção anterior.

Estas variáveis podem assumir valores como *Very Low*, *Low*, *Nominal*, *High*, *Very High*, *Extra High*, e para cada uma delas o modelo COCOMO possui valores tabelados a atribuir. Por exemplo, EM1 (RELY – *Required Software Reliability*) tem um multiplicador de esforço de 0.82 para o nível *Very Low* e 1.10 para o nível *High*.

4.4.2.4 Calibragem do modelo COCOMO II

Calibrar o modelo significa ajustar os valores, tanto dos coeficientes das fórmulas (*A, B, C,* e *D*) quanto dos multiplicadores de esforço e fatores de escala, com os dados e condições particulares de cada processo, produto a ser desenvolvido e organização.

Como mencionado anteriormente, a calibragem utilizada pelo modelo de simulação é a calibragem 2000 apresentada em (Boehm 2000). Esta foi baseada em 161 projetos norte-americanos de software, entre eles projetos de médio e grande porte.

O motivo de utilizar a calibragem padrão do modelo COCOMO II se deve justamente ao fato de que este é usado em situações onde a base histórica está vazia ou insuficientemente populada, ou seja, quando não estão disponíveis dados para calibrar o modelo.

4.4.2.5 Considerações na aplicação do modelo COCOMO II ao WebAPSEE

Para aplicar o COCOMO II ao modelo de simulação foi necessário estabelecer tipos primitivos às atividades do processo de software. Os tipos permitidos são: **Requisitos**, **Projeto** (*Design*), **Codificação**, **Teste**, **Gerência**, **Verificação e Validação**, **Documentação** e um tipo arbitrário que não seja aplicável a nenhum dos outros tipos.

Destes tipos apenas Projeto, Codificação e Teste são considerados no escopo de atuação do modelo COCOMO II. Sendo assim, para os casos em que o grau de similaridade não for suficiente (de acordo com o limiar estabelecido pelo usuário) estes três tipos serão calculados de acordo com as equações do modelo COCOMO II. Boehm (2000) define que para as atividades de requisitos (tais como entrevistas com usuários, especificação de requisitos, entre outras) pode ser atribuído de seis a dez por cento do tempo total estimado para o processo.

Os detalhes relacionados aos outros tipos de atividade que não se encaixam nestas situações estão explicados na seção 4.5.

4.4.3 Dados Empíricos

De acordo com Münch (2003), dados empíricos obtidos a partir de estudos em engenharia de software são uma importante fonte de criação de modelos de simulação exatos[6]. Este autor afirma ainda que descobertas empíricas em conjunto com princípios de Engenharia de Software ajudam imensamente a atingir objetivos de tempo, custo e qualidade.

Em função disso, um modelo para incorporar regras empíricas foi proposto no formato ECA (Evento-Condição-Ação), como segue a gramática na listagem 1.

[6] Do termo em inglês *accurate*. Entretanto, não necessariamente no sentido estrito da palavra “exato”, mas bem perto da realidade.

```
S → <EVENT> if <COND> then <ACTION> ;

EVENT → On Execute Process | On Beginning Task | On Finishing Task |
        On Canceling Activity | On Failing Activity |
        On Creating New Activity | On Defining Artifacts |
        On Defining Artifact Connections | On Adding People (agent or group) |
        On Removing People (agent or group) | On Adding Resource |
        On Removing Resource

COND → <EXPRESSION> | <EXPRESSION><CONNECTION>
EXPRESSION → <OPERAND><RELATION>
CONNECTION → and <COND> | or <COND>
RELATION → (> | < | >= | <= | = | <> | contains | not_contains | sub_type_of) <OPERAND>
OPERAND → Object | Object <OPERATOR>
OPERATOR → MethodOperator | <ASSOCIATION> |
<RESERVED_WORD><OPERATOR> | MethodOperator <OPERAND>
ASSOCIATION → (union | intersection) <OPERAND>
RESERVED_WORD → all | any | no

ACTION → <VARIABLE> = <VARIABLE> + <VARIABLE> * Percentual |
         <VARIABLE> = <VARIABLE> - <VARIABLE> * Percentual
```

Listagem 1. Gramática da Linguagem de Regras Empíricas.

Como exemplo de regra, pode-se ter os elementos descritos na Tabela 8. Neste caso a Lei de Brooks é apresentada. Então, esta é aplicada às questões de tempo. É necessária uma melhor descrição para saber em quanto o tempo deve aumentar. A regra pode ser melhor interpretada como descrito no item Detalhamento. Finalmente, uma descrição ainda mais rigorosa é fornecida na sintaxe da linguagem de regras empíricas.

Tabela 8. Exemplo de regra empírica com a Lei de Brooks.

Descrição textual	Se um projeto está atrasado e mais pessoas são adicionadas, o projeto ficará mais atrasado ainda
Detalhamento	Ao adicionar mais pessoas em uma atividade, o esforço será maior (devido à comunicação e treinamento) impactando em um aumento no tempo necessário para realiza-la.
Especificação	On Adding People if act.number_involved_agents() > 1 and act.is_late() then _SCHEDULE = _SCHEDULE + _SCHEDULE * X/100.

Outro exemplo de regra abordando produtividade é apresentado na tabela 9.

Tabela 9. Exemplo de regra empírica em relação à produtividade.

Descrição textual	Em media, empregados recém contratados têm a produtividade de 50% em relação a um empregado experiente ("*The average newly hired employee is about ½ as productive as an experienced employee*").
Detalhamento	A idéia é que empregados novos não possuem conhecimentos sobre normas e a estrutura da organização, bem como o processo de desenvolvimento e tecnologias utilizadas pela mesma.
Especificação	On Beginning Task **if** ag.get_plays_role_since_date ("Programmer") < X **then** _SCHEDULE = _SCHEDULE - _SCHEDULE * 0.5

Estas regras têm dois escopos: regras aplicadas ao processo como um todo e regras associadas a uma única atividade. Algumas condições são associadas a estas regras. Para dar ao usuário (gerente do processo, engenheiro de processo ou até mesmo um especialista) a oportunidade de criar algumas regras, deve ser possível realizar algumas consultas considerando atributos e componentes de atividades/processos (agentes envolvidos, artefatos, recursos requeridos, métricas e outros). Alguns métodos são providos para realizar estas consultas. As tabelas 10, 11, 12, 13, 14 e 15 apresentam listagem dos métodos classificados em relação aos tipos principais do ambiente.

Tabela 10. Métodos aplicados às atividades.

Nome do Método	Parâmetros	Retorno	Descrição
Get_name	----------	*String*	Retorna o nome da atividade.
duration_in_hours	----------	*Integer*	Retorna o número de horas requeridas para executar a atividade.
input_artifacts	----------	*Set of artifacts*	Retorna os artefatos de entrada envolvidos na atividade.
output_artifacts	----------	*Set of artifacts*	Retorna os artefatos de saída envolvidos na atividade.
number_involved_agents	----------	*Integer*	Retorna o número de agentes envolvidos nesta atividade.
get_involved_agents	----------	*Set of Agents*	Retorna os agentes envolvidos na atividade.
get_required_roles	----------	*Set of Roles*	Retorna os cargos requeridos para a atividade.
get_required_abilities	----------	*Set of Abilities*	Retorna as habilidades requeridas para a atividade.
get_required_resources	----------	*Set of Resources*	Retorna os recursos requeridos para a atividade.
get_size	metric_name	*Float*	Retorna o tamanho da atividade em pontos por função ou KLOC.
get_type	----------	*String*	Retorna o nome do tipo da atividade.
is_late	----------	*Boolean*	Retorna se uma atividade está atrasada ou não.
effort_needed	----------	*Float*	Retorna o esforço necessário para realizar a atividade
Cost	----------	*Float*	Retorna o custo para realizar a atividade.

Tabela 11. Métodos aplicados a processos.

Nome do Método	Parâmetros	Retorno	Descrição
get_name	----------	*String*	Retorna o nome do processo.
duration_in_hours	----------	*Integer*	Retorna a duração do processo em horas.
get_size	metric_name	*Float*	Retorna o tamanho do processo em pontos por função ou KLOC.
is_late	----------	*Boolean*	Retorna se um processo está atrasado ou não.
effort_needed	----------	*Float*	Retorna o esforço necessário para realizar a atividade
overall_cost	----------	*Float*	Retorna o custo total do processo.

Tabela 12. Métodos aplicados a tarefas.

Nome do Método	Parâmetros	Retorno	Descrição
duration_in_hours	activity_id, agent_id	*Float*	Retorna a duração em horas de uma atividade no ponto de vista de um agente (Tarefa).

Tabela 13. Métodos aplicados aos agentes.

Nome do Método	Parâmetros	Retorno	Descrição
affinity_degree_with	Other_agent_id	*Integer*	Retorna o grau de afinidade com outro agente.
plays_role_since_date	role_id	*Date*	Retorna a data do primeiro dia do agente como esse cargo.
ability_degree	ability_id	*Integer*	Retorna o grau de determinada habilidade.
cost_per_hour	----------	*Float*	Retorna quanto o agente recebe por hora.

Tabela 14. Métodos aplicados a cargos.

Nome do Método	Parâmetros	Retorno	Descrição
needs_abilities	----------	*Set of abilities*	Retorna um conjunto de habilidades necessárias para este cargo.
is_subordinated_to	Role_id	*boolean*	Retorna se um cargo é subordinado a outro.
Commands	----------	*Set of roles*	Retorna todos os cargos imediatamente abaixo deste.

Tabela 15. Métodos aplicados a artefatos.

Nome do Método	Parâmetros	Retorno	Descrição
get_type	----------	*String*	Retorna o tipo do artefato.
get_name	----------	*String*	Retorna o nome do artefato.

4.5 DECISÕES NA PROPOSTA DO MODELO

O ponto inicial de definição do modelo de simulação foi a escolha do mecanismo de avanço de tempo para a simulação. Foram considerados os mecanismos a seguir, os quais são apresentados em (Freitas Filho 2001):

- **Simulação Orientada a Atividade** avalia o modelo em intervalos constantes de tempo, sem levar em consideração as alterações ocorridas no mesmo.
- **Simulação Orientada a Evento** avalia o modelo somente quando algum parâmetro modelado tem seu valor alterado. A vantagem sobre a Simulação Orientada a Atividade é que avaliações irrelevantes serão evitadas.
- **Simulação Orientada a Processo** provê uma ampla visão do sistema, permitindo que o projetista modelar cada componente isoladamente. Os eventos internos de cada módulo são guardados isolados dos eventos ocorridos em outros módulos.

De acordo com essa classificação do mecanismo de tempo em simuladores, quanto à lógica de simulação, tem-se utilizado principalmente os dois tipos descritos abaixo:

- **Simulação com Lista de Eventos Fixa** implementa um mecanismo de avanço de tempo orientado a atividade.
- **Simulação com Lógica dirigida a Eventos** abrange a visão orientada a evento de simulação.

Dos mecanismos supracitados, o adotado para o simulador em estudo foi o de **simulação orientada a evento**, devido à característica temporal do processo de desenvolvimento de software, e por ser não determinístico, isto é, a partir dos mesmos dados de entrada (instanciação do modelo de processo) é possível se obter saídas diferentes (projeto concluído com sucesso ou não). Além disso, a natureza dinâmica dos processos de software não permite a utilização de uma lista fixa de eventos, o que levou a adoção da **Simulação com Lógica dirigida a Eventos.**

Seguindo esta decisão, o modelo foi proposto e os eventos que alteram os estados da simulação também: início da execução, início de uma tarefa e fim de uma tarefa.

No cálculo da duração das atividades foi considerada a idéia de utilizar o conhecimento da organização obtido a partir de execuções passadas (seção 4.1). A técnica de RBC foi utilizada devido ao fato de ser adequada em situações que se deseja realizar ações com base em experiências passadas.

Para a definição do modelo RBC foi utilizada a abordagem orientada a objetos para representar os casos, pois as informações do sistema e seus relacionamentos foram definidos em modelos orientados a objetos. Esta abordagem facilita a manipulação das informações do sistema, já que não haverá mudança de paradigma.

A Recuperação em Dois Níveis foi utilizada pela possibilidade de definir heurísticas para minimizar o custo computacional na busca aos casos, considerando que outros métodos como busca seqüencial, árvores k-d e redes semânticas (von Wangenheim 2003) demandam alto poder computacional para a geração da estrutura de consulta com uma grande massa de dados. A geração dessas estruturas seria um processo repetido inúmeras vezes devido às consultas serem diferentes para cada atividade a ser simulada. Essa otimização é necessária em função da grande massa de dados recuperada em um processo e a cada novo processo executado cresce significativamente.

Dentre os métodos de adaptação abordados na literatura (von Wangenheim 2003) (Maher e Silva Garza 1997), o utilizado foi a **Adaptação Nula** devido à dificuldade se atribuir um valor da solução (neste caso o tempo estimado para atividade) ou um método que determine em quanto aumentar ou diminuir o tempo sem uma prévia experimentação deste método.

O modelo COCOMO II foi utilizado como alternativa ao modelo RBC cálculo por não precisar obrigatoriamente de uma base histórica, adotando assim sua calibragem 2000 (ano). Como dito anteriormente (seção 4.4.2.3), essa calibragem, ou seja, os dados que estabelecem

valores aos coeficientes das equações do modelo conta com uma base histórica de 161 diferentes projetos de software, dando assim uma confiabilidade aceitável para a simulação.

Algumas considerações acerca do escopo de atuação e dos tipos de atividades suportados pelo COCOMO II foram abordados na seção 4.4.2.4. Entretanto, outros tipos de atividade não foram considerados e por isso algumas simplificações foram realizadas, são eles: Gerência, Validação e Verificação, Documentação e o tipo designado para qualquer outro tipo que não se encaixe em nenhum dos outros definidos, chamado de ***Default Activity Type***.

Para o tipo Gerência foi considerado que o tempo necessário para realizar atividades de gerência (por exemplo, gerência de requisitos) corresponde ao tempo necessário para executar o processo todo, por ser uma atividade guarda-chuva do processo (Pressman 2005).

Já para os tipos de Verificação e Validação, o tempo de simulação atribuído corresponde a uma porcentagem do tempo necessário para realizar a atividade que produziu o artefato a ser verificado e/ou validado.

Apesar do conhecimento que atividades de documentação ocupam um tempo significativo no processo de software, este tipo de atividade não terá um valor atribuído para o tempo de simulação da atividade. Porém, este esforço necessário para realizar a documentação é considerado no modelo COCOMO II como um multiplicador de esforço denominado ***DOCU*** (Boehm 2000) o qual é distribuído pelas atividades do processo.

E, por fim, atividades do tipo *Default Activity Type* não serão consideradas na simulação.

É importante ressaltar que estas situações dos parágrafos relativos aos tipos de atividade não suportados pelo COCOMO II serão utilizadas somente nas situações em que o grau de similaridade não for suficientemente bom.

4.6 LIMITAÇÕES DO MODELO

Algumas situações no processo de simulação ainda não possuem uma solução totalmente automatizada. Nesta seção serão apresentadas estas questões, até o momento, parcialmente respondidas (solucionadas):

1. **Como determinar o término de tarefas concorrentes alocadas para um mesmo agente?**
2. **Como determinar a duração de tarefas realizadas por um grupo de pessoas?**

Na literatura investigada através de portais de grande importância na área de Computação (incluindo as bibliotecas digitais mantidas pelo *Institute of Electrical and Electronics Engineers*, Sociedade Brasileira de Computação e *Association for Computing Machinery*), nada de concreto foi encontrado a respeito.

A solução proposta para a primeira pergunta se dá em duas etapas:

- **Quando uma atividade pronta deverá ser iniciada por um agente?**

A resposta a esta pergunta pode ser deixada à preferência do usuário nas configurações da simulação onde três opções são possíveis: 1) sempre que uma atividade estiver pronta pode ser iniciada, ou seja, todas as atividades prontas; 2) um número limitado de atividades pode ser iniciado, sendo este número determinado aleatoriamente; e 3) somente quando não houver outra atividade ativa.

A solução 1 causa um problema de concorrência entre as atividades, isto é, gera uma nova pergunta: "como o agente deve se comportar ao ter duas tarefas ativas em sua agenda?".

A solução 2 causa o mesmo problema da solução 1 e incorpora incertezas à simulação, devido o uso de um número aleatório.

A solução 3 não causa problemas de concorrência, devido sua característica seqüencial, entretanto, expressa com menos realidade o processo de desenvolvimento de software, visto que, em muitas organizações,

- **Como o agente deve escalonar o tempo de cada tarefa concorrente?**

Um algoritmo para escalonar esse tempo foi proposto e é explicado a seguir:

Considerando-se um desenvolvedor com duas tarefas **T1** e **T2** a serem realizadas. Se estas tarefas fossem realizadas seqüencialmente, **T1** teria seu tempo final após 5 horas (300 min) de duração e **T2** em 2 horas (120 min). Deve-se supor que a duração estimativa (T1 e T2) foram calculados utilizando-se a função citada anteriormente.

Como simplificação, foi estabelecido que ao realizar um conjunto de tarefas uma delas terminará na soma das suas durações. Para o exemplo em questão, o tempo final de uma delas será após 7 horas (5h + 2h). Já o tempo da outra atividade será determinado de acordo com o esforço percentual que o agente dedicará para cada tarefa em situação de concorrência. Este percentual será perguntado ao usuário da ferramenta sempre que o simulador detectar essas situações.

Adotando os valores fornecidos pelo usuário como 90% de dedicação para **T1** e 10% de dedicação para **T2**, devido critérios que o usuário adotou (prazos, importância, dependências, entre outros), a tabela 16 apresenta o andamento das duas tarefas a cada ciclo da simulação.

Tabela 16. Andamento das tarefas.

Ciclo	Tempo Percorrido de T1	Tempo Percorrido de T2	Tempo Total Percorrido
1	54 min	6 min	1 hora = 60 min
2	108 min	12 min	2 horas = 120 min
3	162 min	18 min	3 horas = 180 min
4	216 min	24 min	4 horas = 240 min
5	270 min	30 min	5 horas = 300 min
6	300 min	60 min	6 horas = 360 min
7	300 min	120 min	7 horas = 420 min

Considerando cada ciclo com a duração de uma hora, isto é, a unidade de tempo em horas, e como foram dedicados 90% do tempo para **T1** (54 minutos de cada ciclo) e 10% do tempo para **T2** (6 minutos cada ciclo), então seria obtido um acréscimo de 54 minutos para o tempo percorrido de **T1** a cada ciclo e de 6 minutos para **T2**. Do ciclo 1 ao 5, são apresentados os acréscimos definidos realizados para cada tarefa. Entretanto, no ciclo 6, uma situação diferente é mostrada: não é necessária a utilização dos 54 minutos de dedicação para T1, tendo em vista que só 30 minutos (300 min – 270min = 30 min) são necessários para completar a tarefa. Sendo assim, utilizam-se apenas 30 minutos para **T1** e os 30 minutos restantes são utilizados para **T2**. E, finalmente, no ciclo 7 somente a **T2** necessita de dedicação do agente, ocupando o ciclo inteiro (1 hora = 60 min) e terminando sua duração.

A solução para a segunda pergunta não foi encontrada completamente como a primeira.

Sabe-se que é necessário levar em consideração o número de pessoas (agentes) envolvidas na atividade, pois segundo Boehm (1981), quanto mais agentes trabalham em um documento, mais esforço é necessário para alcançar o resultado devido o crescente esforço da comunicação entre as pessoas. Boehm mostra também que existe um limite máximo para o tamanho da equipe e um limite mínimo para a duração em todo projeto de software. É necessário também levar em consideração a afinidade entre os desenvolvedores, por exemplo, se dois desenvolvedores já atuaram junto em tarefas parecidas no passado é mais provável que tenham sucesso nesta nova tarefa.

Além destas dificuldades, algumas outras limitações como, por exemplo, lidar com similaridades em processos iterativos e impactos na qualidade do produto ainda estão em investigação.

4.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou os aspectos específicos do modelo de simulação assim como a justificativa para a escolha dos métodos utilizados. O problema a ser resolvido pelo modelo proposto foi também apresentado, discutindo a abordagem baseada em conhecimento escolhida para resolvê-lo e os benefícios esperados.

Neste capítulo foi apresentada, também, uma visão geral do ambiente WebAPSEE com seus principais objetivos e visões. Assim como os objetivos específicos e requisitos do modelo de simulação, os mecanismos de simulação evento-discreta e os métodos de estimativa (COCOMO II) e inteligência artificial utilizados para o cálculo de estimativa de duração das atividades a serem simuladas, buscando o aumento do realismo da simulação.

Embora algumas simplificações tenham sido feitas, estudos e experimentos continuam sendo realizados para complementar as lacunas deixadas por estas simplificações.

5. PROJETO DO SIMULADOR

Nesta seção são apresentados detalhes que dizem respeito ao projeto deste modelo de simulação.

Primeiramente, uma visão geral da arquitetura é mostrada na seção 5.1, onde são apresentados seus componentes e o relacionamento entre estes. Em seguida, na seção 5.2 são apresentados os aspectos de implementação desses componentes, tais como: organização em pacotes, diagramas de classe, algoritmos importantes e as principais tecnologias utilizadas. E na seção 5.3 são apresentadas algumas considerações finais do capítulo.

5.1 VISÃO GERAL DA ARQUITETURA PROPOSTA

Após a definição dos requisitos e casos de uso, a atividade de projeto teve início, onde a arquitetura de software foi proposta como na figura 13.

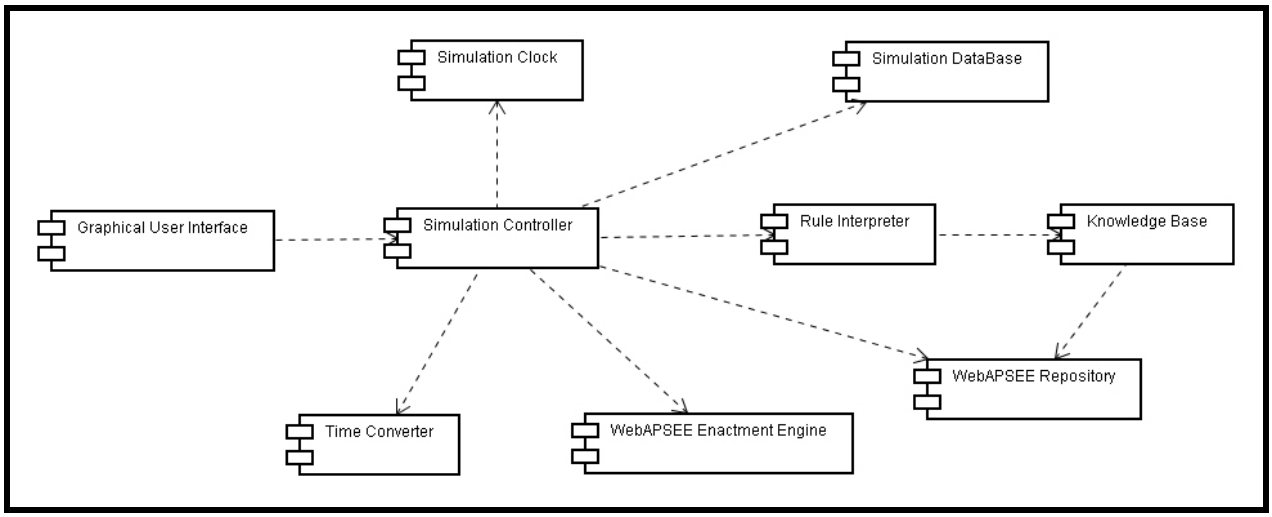

Figura 13. Diagrama de componentes da arquitetura proposta.

Do ponto de vista de proximidade com o usuário, o primeiro componente é o de **Interface com o Usuário (*Graphical User Interface*)** da ferramenta, por onde este pode controlar a simulação. Os eventos (pedido para pausar a velocidade da simulação, por exemplo) gerados por esta interface são tratados pelo **Controlador da Simulação (*Simulation Controller*)** que, dependendo da informação solicitada, invocará serviços de outros componentes para dar andamento à simulação. Este componente contém a implementação do *loop* principal da simulação apresentado na seção 4.4.

A simulação tem uma duração diferente da duração de projetos reais, e é chamada de tempo de simulação. Esta abstração é feita através do componente **Relógio da Simulação**

(*Simulation Clock*). Como o objetivo é simular a execução de um processo de software, o simulador deve ser integrado ao ambiente. Mais especificamente, a **Máquina de Execução (*WebAPSEE Enactment Engine*)** de processos do WebAPSEE deve ser reutilizada, tendo em vista a economia de tempo e a confiabilidade, pois se trata de um componente já disponível e testado. Entretanto, como esta Máquina de execução trabalha com o tempo real do sistema, há a necessidade de um **Conversor de Tempo** (denominado aqui de ***Time Converter***) para a conversão do tempo do sistema para o tempo de simulação. Portanto, ao invés de se utilizar o relógio físico do computador para gestão de eventos de processo, o simulador usará um relógio lógico, virtual, baseado nos ciclos da simulação.

As informações sobre o processo a ser simulado estão armazenadas no **Repositório do WebAPSEE (*WebAPSEE Repository*)** que possui, além de informações sobre o processo a ser simulado, informações sobre execuções passadas deste processo e de outros processos, servindo com uma base histórica. Essas informações contidas no repositório do WebAPSEE constituem fonte de conhecimento que pode ser consultada diretamente ou através de mecanismos mais sofisticados, como Mineração de Dados, por exemplo. Entretanto, a descoberta de conhecimento não é escopo deste trabalho, somente a interpretação das regras geradas. Esse conhecimento é então armazenado em uma **Base de Conhecimento (*Knowledge Base*)** em forma de regras, que são utilizadas pelo **Interpretador de Regras (*Rule Interpreter*)**, mecanismo de tomada de decisão para os eventos da simulação.

Por fim, os dados da simulação são armazenados na **Base de Dados da Simulação (*Simulation DataBase*)**, servindo como fonte para gerar relatórios e análises. É importante ressaltar que esta base não faz parte da base de dados do WebAPSEE. São estruturas de fato diferentes! A utilização de bases distintas se faz necessária por dois motivos: o primeiro diz respeito ao fato de que os dados de simulação não são dados reais, podendo comprometer a base histórica da organização; e segundo pela característica do simulador como uma ferramenta opcional ao ambiente.

5.2 PROJETO DETALHADO DA ARQUITETURA

Esta seção trata dos aspectos de projeto detalhado do modelo de simulação. Nas subseções seguintes os principais componentes e algoritmos do modelo de simulação serão apresentados.

5.2.1 Pacotes

Na figura 14 o diagrama de pacotes mostra a visão geral dos dados e suas relações de dependência.

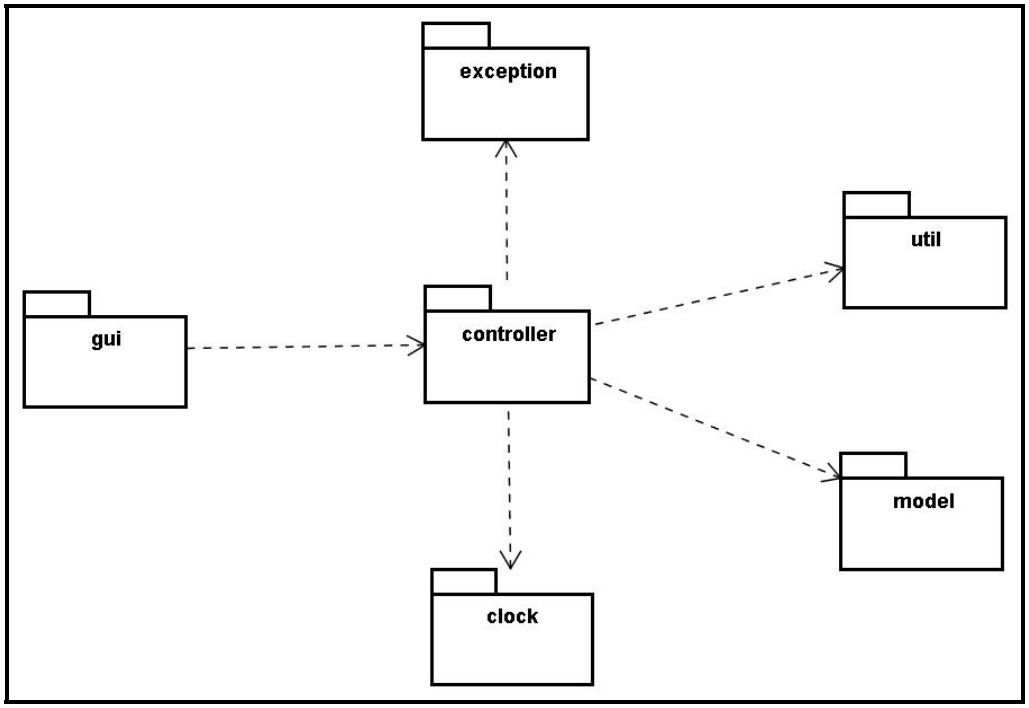

Figura 14. Diagrama de pacotes.

O pacote central ***controller*** corresponde ao componente ***Simulation Controller*** apresentado na arquitetura e engloba as classes apresentadas no diagrama de componentes da figura 13. Este pacote é o núcleo do simulador e depende diretamente de outros quatro pacotes: ***exception***, ***util***, ***model*** e ***clock***. O pacote ***gui*** representa as classes que implementam as interfaces gráficas com o usuário, tais como formulários e interfaces de controle da simulação. As classes que implementam as interfaces é quem invocam o ***controller***, por isso a dependência.

5.2.2 Modelo de Dados

O modelo de dados foi baseado em metodologias orientadas a objetos e alguns aspectos funcionais foram baseados em orientação a aspectos.

A figura 15 apresenta as classes relacionadas ao controle central da simulação.

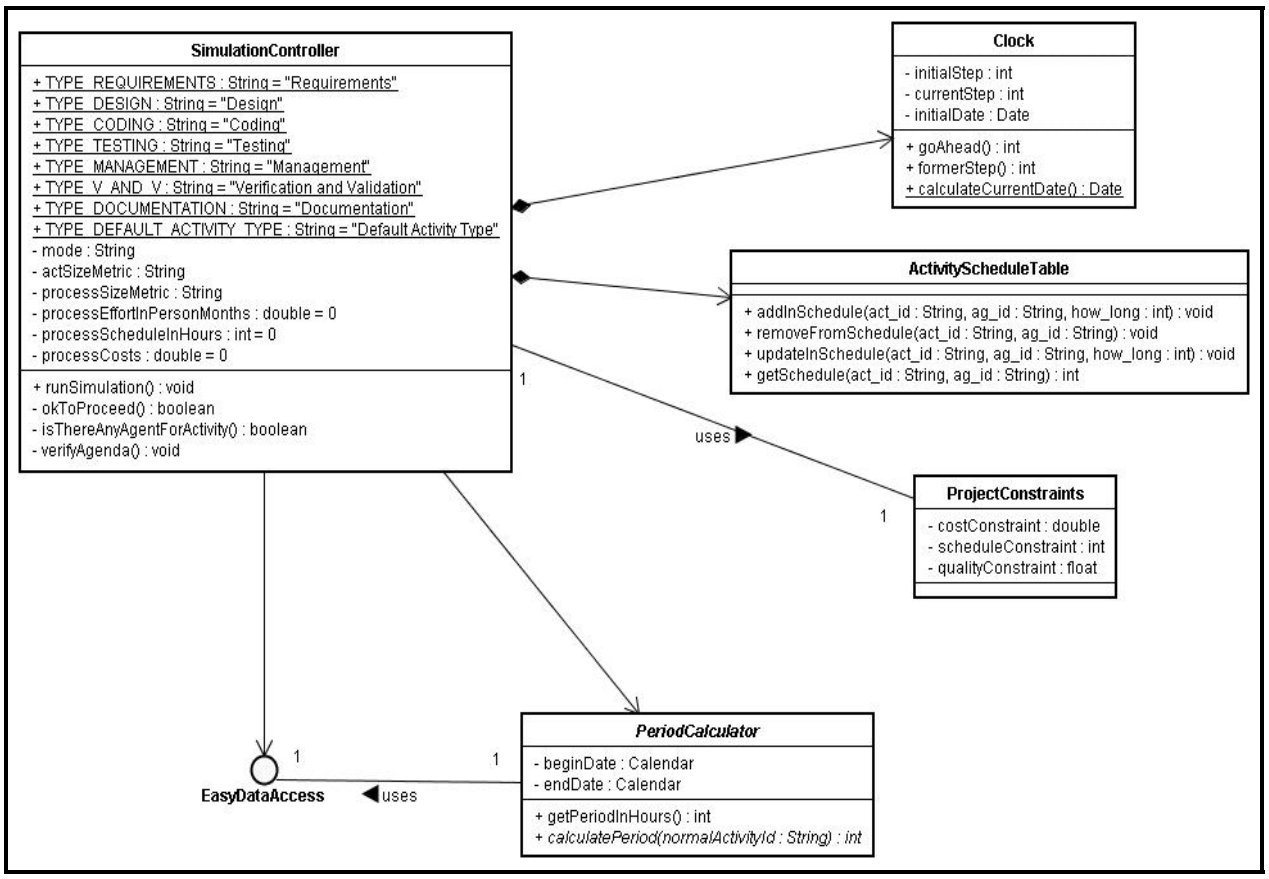

Figura 15. Diagrama de classes do pacote *controller*.

A classe ***SimulationController*** possui constantes (atributos escritos em caixa alta) para designar os tipos de atividades permitidos pelo modelo de simulação. Estes tipos foram definidos para identificar o escopo de atuação do modelo COCOMO II (ver seção 4.4.2.5), conforme definido por Boehm (2000): Projeto, Codificação e Testes. O atributo ***mode*** é usado para sinalizar qual modo de operação (ver seção 4.5) da simulação[7]. Os atributos ***actSizeMetric*** e ***processSizeMetric*** são usados para recuperar os nomes das métricas que definem o tamanho da produção do processo e das atividades.

Ao início da simulação são calculadas estimativas de esforço em pessoas-mês (***person-months***), de prazo em horas (***hours***) e de custo financeiro para basear a simulação. Estes valores são armazenados nos atributos ***processEffortInPersonMonths, processScheduleInHours*** e ***processCosts***, respectivamente.

Quanto às operações:

[7] Os três modos de operação possíveis são: **serial**, **aleatório** e **quando pronto**

- A operação ***runSimulation*** inicia a simulação;
- A operação ***okToProceed*** é responsável para testar se a simulação deve prosseguir em termos de estado do processo, custos e/ou prazos estourados;
- O método ***isThereAnyAgentForActivity*** verifica a presença de agentes na atividade;
- ***verifyAgenda*** é o método que analisa a agenda para cada agente, decidindo se uma atividade deve ser iniciada ou terminada de acordo com o tempo previsto para sua simulação.

Uma instância de ***SimulationController*** está associada com objetos de outras classes, conforme descrito a seguir:

- Dois relacionamentos são estabelecidos para representar a relação de composição entre objeto da classe ***SimulationController*** e os objetos das classes ***Clock*** e ***ActivityScheduleTable***, que representam o relógio da simulação e a tabela contendo tempo restante (em ciclos) que falta para cada atividade simulada terminar a simulação, respectivamente;
- Relacionamento com ***ProjectConstraints***, que define as restrições de custo, prazo e qualidade iniciais do projeto fornecidas pelo usuário;
- ***PeriodCalculator***, a qual é responsável pelo cálculo do período de duração das atividades, seja via similaridade nos casos RBC ou via modelo COCOMO II;
- E a interface ***EasyDataAccess*** para facilitar o acesso aos dados com consultas complexas.

A figura 16 apresenta o pacote *Clock*. Neste diagrama é usada a extensão à UML proposta por Lucrédio *et al* (2004) para representar a participação de Aspectos em um modelo de Objetos.

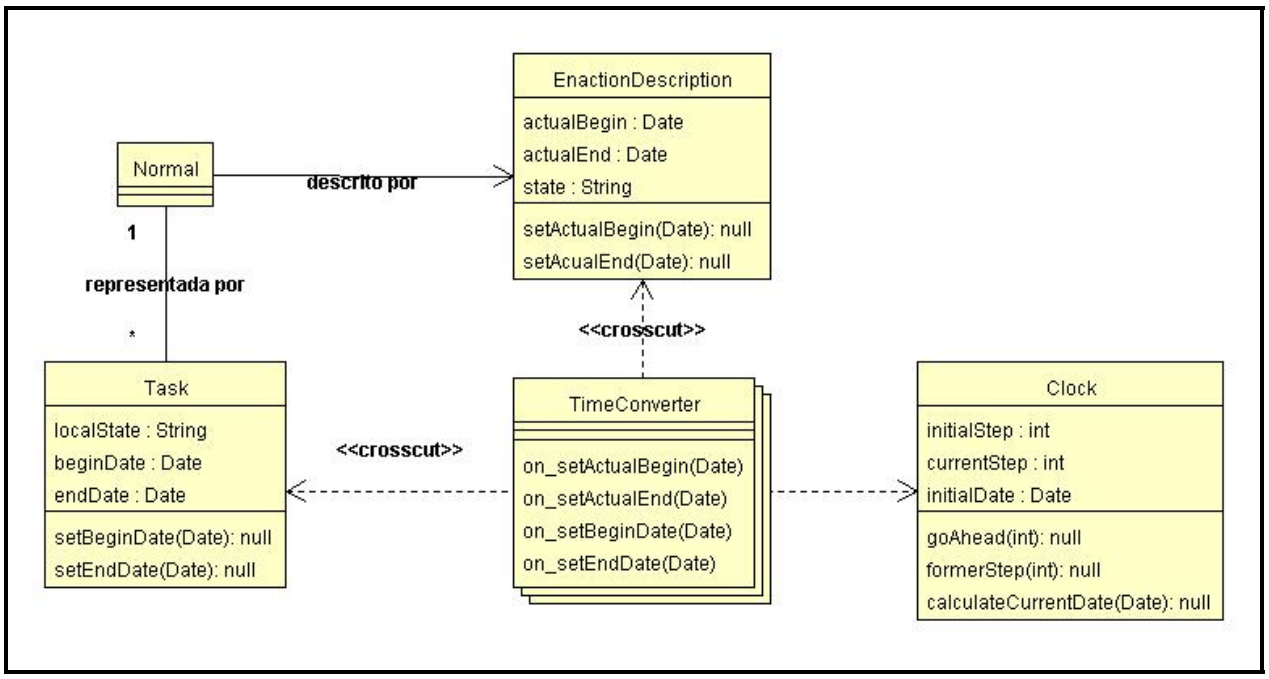

Figura 16. Diagrama de classes e aspectos do pacote *clock*.

A classe **Normal** representa as atividades normais do ambiente WebAPSEE (ver seção 4.4.1.1) que tem sua execução descrita pela classe **EnactionDescription.** Nesta classe são armazenados o estado e as datas de inicio e fim da atividade e são representadas pela classe **Task** na visão dos agentes (França *et al* 2006). Tanto na classe ***EnactionDescription*** quanto na classe **Task** existem operações que manipulam datas no contexto da máquina de execução, entretanto, estes métodos recebem como parâmetro datas atuais do sistema. É neste contexto que o aspecto ***TimeConverter*** atua nesses métodos antes de sua execução alterando o parâmetro pelo tempo utilizado no relógio da simulação (classe **Clock**) através do método ***calculateCurrentDat*e**, que calcula a data atual da simulação com base da data de início da simulação (atributo **initialDate**) e no passo (ciclo) corrente da simulação (atributo **currentStep**).

No pacote ***exception*** é definida a hierarquia de exceções do sistema, conforme a figura 17.

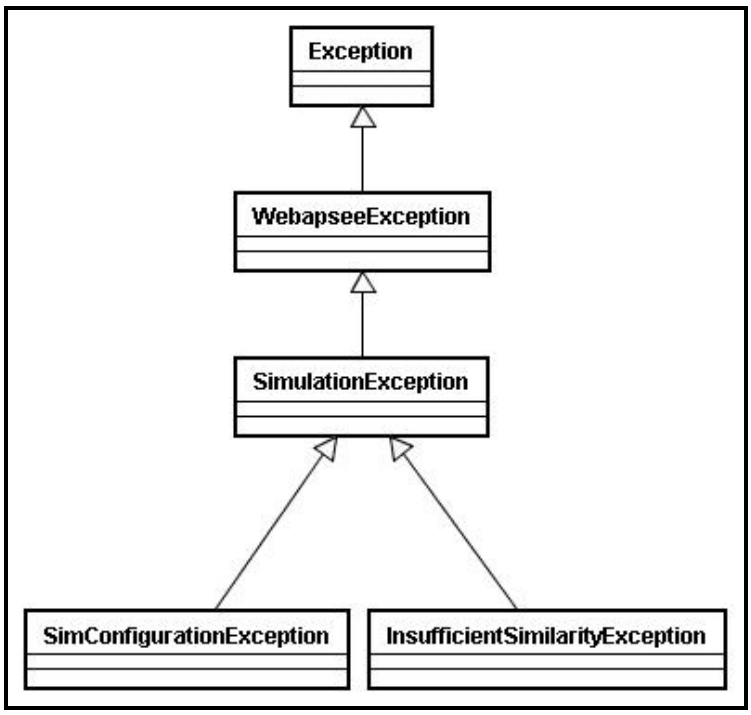

Figura 17. Diagrama de classes do pacote *Exception*.

Todas as classes de exceção herdam, primeiramente, da classe ***Exception*** do pacote **java.lang** da linguagem Java. A classe de exceção do sistema WebAPSEE, ***WebapseeException***, é a subclasse seguinte. Então, uma exceção geral de simulação é declarada ***SimulationException*** e suas subclasses ***SimConfigurationException*** e ***InsufficientSimilarityException*** que tratam de qualquer erro relativo ao acesso à informações disponíveis no arquivo de configuração que salva os parâmetros da simulação e grau de similaridade insuficiente entre os casos RBC, respectivamente.

O pacote ***Util*** é referente às classes utilitárias de implementação. São classes que não possuem relacionamentos com outras classes, elas possuem funções secundárias. Por exemplo, a classe **Configure** que é responsável pelo acesso aos dados contidos no arquivo de configuração do simulador e a interface **EasyDataAccess** e a classe que a implementa **EasyDataAccessImpl,** que são responsáveis por métodos que facilitam o acesso a dados que necessitam de consultas complexas para recuperá-los. Por exemplo, neste componente está disponível o método responsável pela recuperação em dois níveis dos casos (ver seção 4.4.1.3).

Finalmente, o pacote ***Model*** que contém as classes do modelo de simulação. Este pacote se subdivide em três outros pacotes independentes: **cbr**, **cocomoii** e **knowledge**.

No pacote cbr estão as classes relativas ao módulo RBC. Este é o responsável pelo cálculo de similaridade entre os casos usando a técnica de RBC (figura 18).

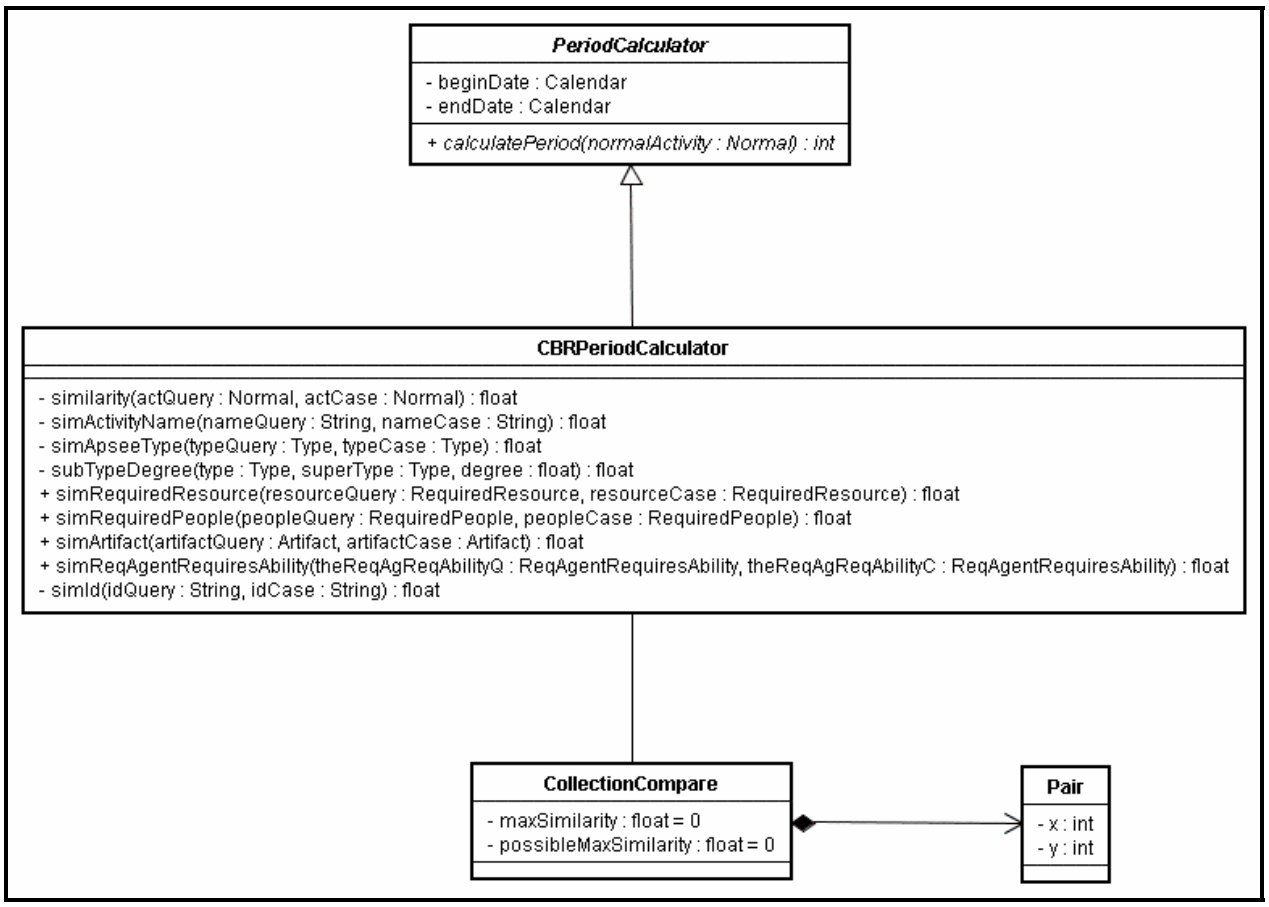

Figura 18. Diagrama de classes do pacote cbr.

Na figura 18, a classe **PeriodCalculator** foi definida como abstrata a fim de estabelecer a estrutura e os métodos necessários para o cálculo da duração de atividades independente do método a ser utilizado. A classe **CBRPeriodCalculator** é subclasse da classe **PeriodCalculator** e implementa o método de cálculo do período de duração necessário pra executar a atividade com base na similaridade entre a atividade em questão e os casos (atividades) passados. Este cálculo é fornecido pelo método público **calculatePeriod**, o qual invoca o método **similarity** para comparar cada caso com a atividade em questão. Então, métodos locais são chamados para o cálculo da similaridade local de cada componente, isto é, nome da atividade (**simActivityName**), o tipo da atividade (**simApseeType**), recurso requerido (**simRequiredResource**), pessoas requeridas (**simRequiredPeople**), artefatos de entrada e saída (**simArtifact**) e as habilidades requeridas para cada agente (**simReqAgentRequiresAbility**).

Muitos dos relacionamentos das instâncias de (Atividade) Normal são “um-para-muitos”. Sendo assim é necessária a comparação entre coleções e para isso a classe **CollectionCompare** foi definida, que por sua vez é composta por coleções de pares (classe **Pair**) x e y. Na seção a seguir (seção 5.2.3) será explicada a utilização destes pares (x,y).

5.2.3 Algoritmos Importantes

Esta seção apresenta dois algoritmos importantes para o modelo de simulação: o **algoritmo de comparação de coleções** (determina o grau de similaridade entre coleções) e o **algoritmo de recuperação dos casos em dois níveis** nas seções 5.2.3.1 e 5.2.3.2 respectivamente.

5.2.3.1 Algoritmo de Comparação de Coleções

O **algoritmo de comparação de coleções** possui particularidades que influenciam diretamente na sua complexidade. A idéia é, por exemplo, comparar duas coleções de agentes requeridos por uma atividade. Então, admitem-se duas coleções A e B com os seguintes elementos:

A = {josé, maria, joão} e B = {maria, carlos},

Neste caso, A e B são coleções de tamanho diferente, o que é muito comum no domínio do trabalho aqui proposto (por exemplo, em equipes de uma organização de desenvolvimento de software). As coleções nos casos a serem comparados (recursos, pessoas, artefatos e habilidades) são conjuntos, isto é, não possuem elementos repetidos e a ordem que estão armazenados não possui importância semântica ao contexto. Desta forma, um conjunto {maria, carlos} é idêntico ao conjunto {carlos, maria}. Esta característica impede que seja utilizado, por exemplo, métodos otimizados de comparação de *strings* (conjuntos de caracteres), onde técnicas de programação dinâmica são utilizadas (Setubal e Meidanis 1997), devido a estes considerarem as coleções como listas e não conjuntos, ou seja, a ordem é importante no contexto.

Sendo assim, o método proposto para calcular a similaridade dos conjuntos consiste em gerar todas as combinações possíveis de pares de elementos e, para cada combinação, realizar a soma da similaridade desses pares, retirando ao final o maior valor entre as somas.

Inicialmente gerar essas combinações parece ser computacionalmente inviável, entretanto, foi utilizada uma adaptação do algoritmo das oito rainhas apresentado em (Skiena e Revilla 2003). O algoritmo das oito rainhas utiliza a técnica de *backtracking* para posicionar oito rainhas

em um tabuleiro 8 x 8 de tal forma que quaisquer duas rainhas não possam se atacar. Isto significa que duas rainhas não podem se posicionar na mesma linha, coluna ou diagonal, ou seja, deve-se gerar todas as permutações possíveis para as posições [oito posições (x,y) em cada solução] em que as rainhas não se ataquem.

A analogia feita com este problema no contexto da comparação de coleções consiste no fato que deve ser gerado um conjunto de vários (todas as combinações possíveis) somatórios com pares de elementos (similaridade entre o par). Este conjunto de equações foi abstraído para uma matriz onde cada linha seria uma equação e, para cada linha, a coluna representa a parcela do somatório. Uma única diferença na adaptação do algoritmo é que diagonais não são consideradas, somente as linhas e colunas. Pode-se imaginar o problema das rainhas, porém sem a opção de atacar na diagonal.

A solução descrita acima gera todas as combinações possíveis de pares de elementos entre os conjuntos, sendo que um par nunca deve conter elementos do mesmo conjunto e uma equação não pode conter elementos repetidos. Por exemplo, considerando o exemplo fornecido na narrativa acima, (josé, joão) não é um par válido pois possui elementos do mesmo conjunto e (josé, maria) + (josé, carlos) não é uma equação válida pois o elemento "josé" está em duas parcelas na mesma equação. O fim dessa analogia é dado pelo par (x,y), que, no problema das oito rainhas, refere-se à posição de cada rainha no tabuleiro e, na similaridade das coleções, refere-se a posição do elemento em seu conjunto.

Para os conjuntos A e B definidos anteriormente, a matriz gerada com os resultados é apresentada na tabela 17.

Tabela 17. Resultado do algoritmo de *backtracking*.

(josé,maria) +	(maria,carlos) +	(joão,0)
(josé,carlos) +	(maria,0) +	(joão,maria)
(josé,0) +	(maria,maria) +	(joão,carlos)
(josé,maria) +	(joão,carlos) +	(maria,0)
(josé,carlos) +	(maria,maria) +	(joão,0)
(josé,0) +	(maria,carlos) +	(joão,maria)

O próximo passo consiste na soma das linhas e retirada do maior valor. Neste momento é feita uma otimização: quando em um par de elementos um destes é nulo (0) sua similaridade é zero. Sendo assim, não é necessário realizar todas as somas. No caso exemplo, a maior similaridade possível é dois (2), tendo em vista que a similaridade está no intervalo [0;1] e todas as linhas possuem pelo menos um par onde a similaridade é nula. Assim, a primeira soma que tiver uma similaridade igual a dois, o processamento é terminado e esta será a similaridade da coleção. No pior caso, realizará todas as somas e retirará o maior valor como similaridade.

5.2.3.2 Algoritmo de Recuperação em Dois Níveis

O segundo algoritmo considerado importante é o de **recuperação em dois níveis dos casos**. Este algoritmo consiste em uma consulta via linguagem de consultas ao repositório do WebAPSEE que retorna todas as atividades normais que tenham sido terminadas, isto é, com o estado igual a *finished*. Isso faz com que o número de casos seja reduzido consideravelmente e garante que os casos a serem retornados sejam casos de sucesso, por se tratar de atividades que conseguiram ser terminadas.

A segunda etapa é a classificação do tipo da atividade que está sendo recuperada. Serão considerados casos somente as atividades que sejam do mesmo tipo ou subtipo da atividade problema. Assim, um método recursivo foi implementado para percorrer a hierarquia de tipos do ambiente WebAPSEE a fim de descobrir se o tipo da atividade recuperada como caso é subtipo do tipo da atividade em questão. Este atributo foi selecionado por possuir um peso importante no grau de similaridade (ver seção 4.4.1.2). Desta forma, se essa similaridade local for igual a zero, a similaridade global será, no máximo, 0,79 não satisfazendo o grau mínimo aceito pela simulação (80%). Este grau está definido em um intervalo de 80% a 100%, que pode ser nivelado pelo usuário.

5.2.4 Tecnologias Utilizadas na Implementação do Protótipo

Como o projeto do simulador é baseado em um sistema existente (WebAPSEE), é importante ressaltar a adoção de algumas tecnologias utilizadas.

O modelo de simulação foi implementado completamente utilizando a linguagem de programação orientada a objetos Java. Para o mapeamento objeto-relacional e a persistência dos dados foi utilizado *framework* Hibernate e o Sistema Gerenciador de Banco de Dados (SGBD) MySQL.

A conversão do tempo do sistema, utilizado pela **Máquina de Execução do Ambiente WebAPSEE**, para o tempo de simulação, baseado no tempo do **Relógio da Simulação**, foi implementada utilizando a linguagem orientada a aspectos AspectJ. Utilizando essa tecnologia foi possível reutilizar a **Máquina de Execução do Ambiente WebAPSEE** sem alterar nenhuma linha de código.

5.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou os aspectos relativos à implementação do modelo de simulação para o ambiente WebAPSEE. Detalhes acerca de dados, algoritmos e tecnologias utilizadas foram mostrados.

Para a especificação deste projeto foi utilizada a notação UML para descrição dos componentes da arquitetura, classes e seus relacionamentos.

A arquitetura foi proposta com o intuito de estabelecer os componentes e interfaces bem definidos, provendo baixo acoplamento e maior flexibilidade para futuras manutenções. Os componentes foram então refinados através da definição de classes e a interface com a máquina de execução do WebAPSEE foi mantida sem alterá-la externa ou internamente.

Os algoritmos importantes foram selecionados pelo fator performance, considerando que desempenham um papel significativo e muitas vezes executados na simulação. E a adoção das tecnologias utilizadas foi influenciada em maior parte pela interoperabilidade com o WebAPSEE.

6. EXEMPLO DE SIMULAÇÃO

Nesta seção será apresentado um exemplo de uma simulação realizada após a implementação do modelo. Trata-se de um processo para desenvolvimento de portais *web*. Este processo é utilizado no Laboratório de Engenharia de Software da universidade Federal do Pará para testes do ambiente WebAPSEE devido a utilização diversos componentes do processo (recursos, artefatos) e várias dependências entre conexões (conexões múltiplas, simples e de *feedback*). Convém ressaltar que a base histórica utilizada é uma base fictícia, o que não invalida o exemplo, pois mostra para quais situações este modelo funciona.

6.1 A BASE HISTÓRICA

Como mencionado anteriormente, o processo a ser simulado é um processo para construção de portais *web*. Sendo assim, a base histórica contém a execução de processos similares ao mostrado a seguir. O processo está descrito em um alto nível de abstração como mostra a figura 19.

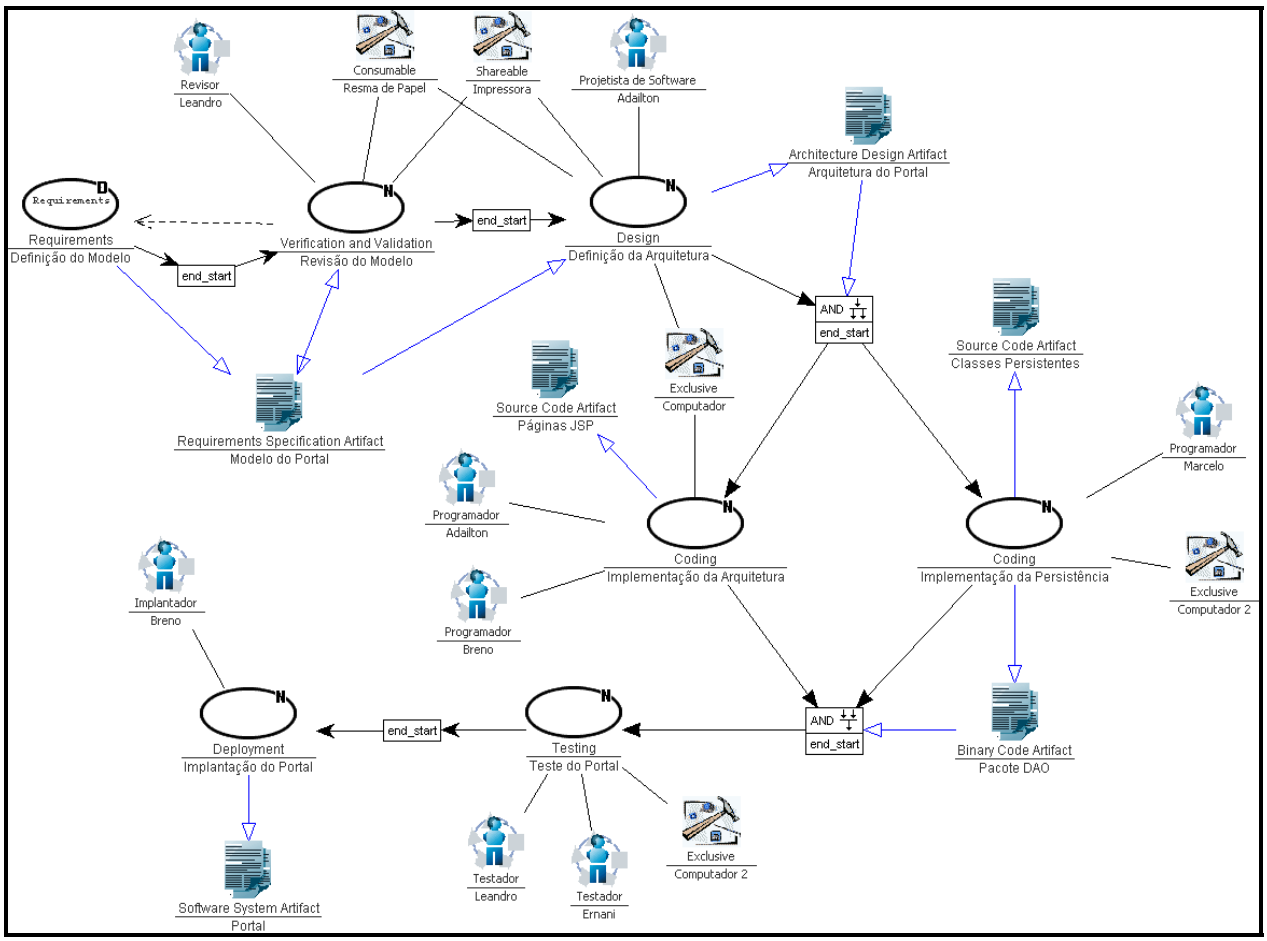

Figura 19. Processo de desenvolvimento de portais *web*.

No processo da figura 19 as elipses representam as atividades a serem realizadas pelos agentes do processo, representados pela figura de um boneco cercado por um círculo. As atividades chamadas **normais** (que são realizadas por atores do processo) são caracterizadas pela letra "N" no canto superior direito da elipse e as atividades decompostas, que representam um sub-processo (um detalhamento da atividade), são caracterizadas pela letra "D" no canto superior direito da elipse. Os recursos utilizados são representados pelo ícone do martelo e os artefatos pelo ícone de dois documentos sobrepostos.

As setas com a orientação preenchida representam o fluxo do processo, ou seja, a ordem em que as atividades devem ser realizadas. Já as setas com a orientação não preenchida representam o fluxo dos artefatos, isto é, quando um artefato é de entrada (consumido) ou de saída (produzido).

A primeira atividade é uma atividade de requisitos e trata da **Definição do Modelo**. Esta atividade é uma atividade decomposta. Seu sub-processo é detalhado na figura 20 a seguir.

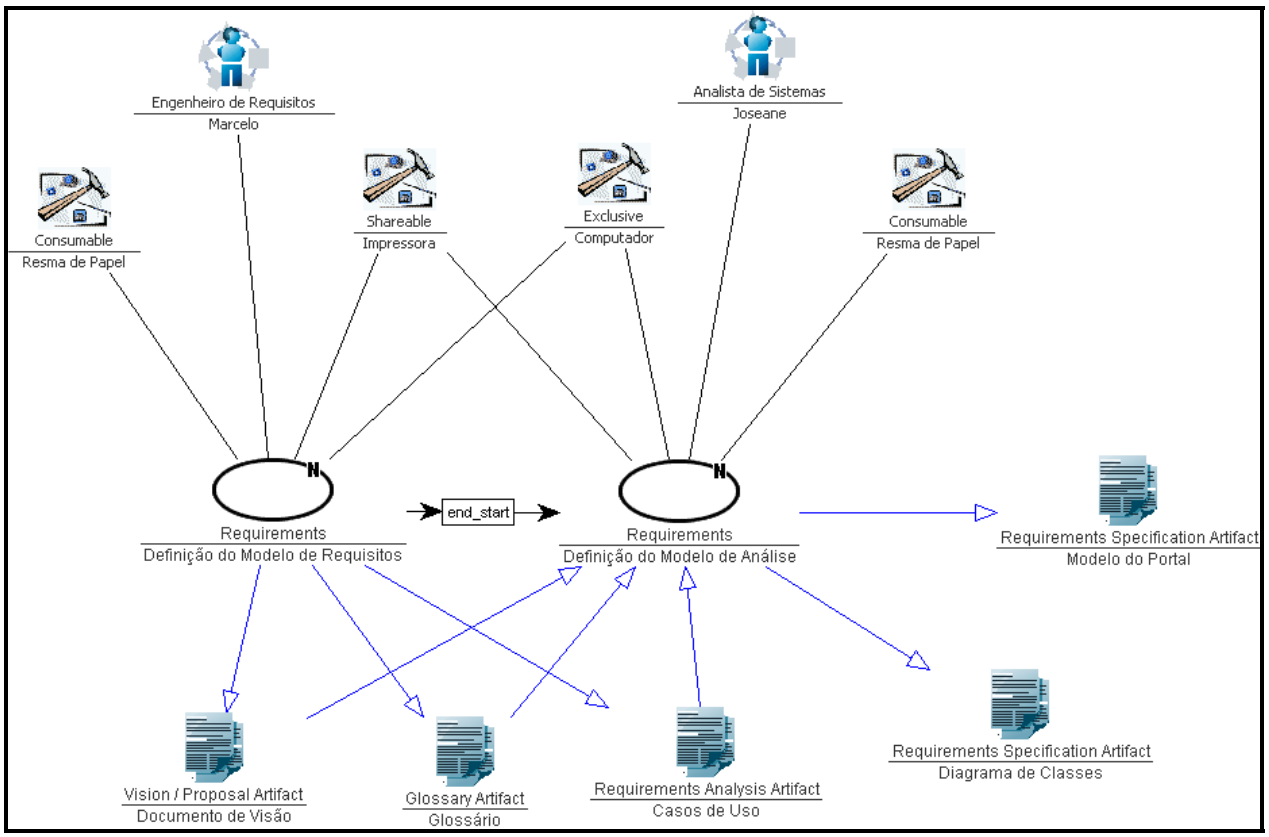

Figura 20. Atividade decomposta Definição do Modelo.

Neste sub-processo a primeira atividade de requisitos é a atividade de **Definição do Modelo de Requisitos**, onde os artefatos **Documento de Visão**, **Glossário** e os **Casos de Uso** serão produzidos. Esta atividade é realizada pelo agente **Marcelo** (**Engenheiro de Requisitos**) e consome os recursos **Resma de Papel**, **Impressora** e **Computador**. Ao fim desta atividade, a atividade **Definição do Modelo de Análise** poderá ser iniciada, de acordo com a conexão simples *end-start* estabelecida entre elas. Nesta outra atividade, os artefatos produzidos pela atividade **Definição do Modelo de Requisitos** são consumidos e os mesmos recursos são utilizados. Os artefatos produzidos por esta atividade são: o **Modelo do Portal** e os **Diagramas de Classes**. Com isto a macro atividade **Definição do Modelo** é terminada.

Voltando ao processo raiz, o fluxo segue com a atividade de verificação e validação de **Revisão do Modelo** sobre o artefato **Modelo do Portal**, realizada pelo **Revisor Leandro**. Neste ponto existe uma conexão caracterizada pela linha tracejada, ela é responsável pela possibilidade de iteração entre partes do processo dada uma condição, neste caso uma condição referente à qualidade (quantidade de erros encontrados). Entretanto, o mecanismo de condições não está

integrado ao ambiente WebAPSEE e o modelo aqui proposto aborda somente questões de custo e tempo, não trata qualidade.

Com o fim da atividade de revisão, a **Definição da Arquitetura** é iniciada com o agente Adailton como **Projetista de Software** e os recursos **Computador, Resma de Papel** e **Impressora**. Nesta atividade, o artefato da Arquitetura do Portal é produzido com base no **Modelo do Portal**. Realizada a tarefa de Arquitetura, são iniciadas as tarefas de codificação, tanto a **Implementação da Arquitetura** quanto a **Implementação da Persistência**. Estas atividades ocorrem em paralelo, produzindo como resultado final "páginas web JSP" com suas devidas lógicas envolvidas e as classes persistentes com seus respectivos DAOs (*Data Access Object* - padrão de projeto que mascara o acesso aos dados na camada de persistência) respectivamente.

Posteriormente, os testes são realizados na atividade **Teste do Portal**, seguindo com a implantação na última atividade **Implantação do Portal**.

6.2 O PROCESSO EM QUESTÃO

A fim de diminuir os custos e prazos, o gerente da organização realiza algumas alterações no modelo. Neste contexto, a simulação é útil para verificar os impactos que estas mudanças terão na execução, ratificando ou não as suposições do gerente.

O modelo de processo com as modificações realizadas sé mostrado na figura 21.

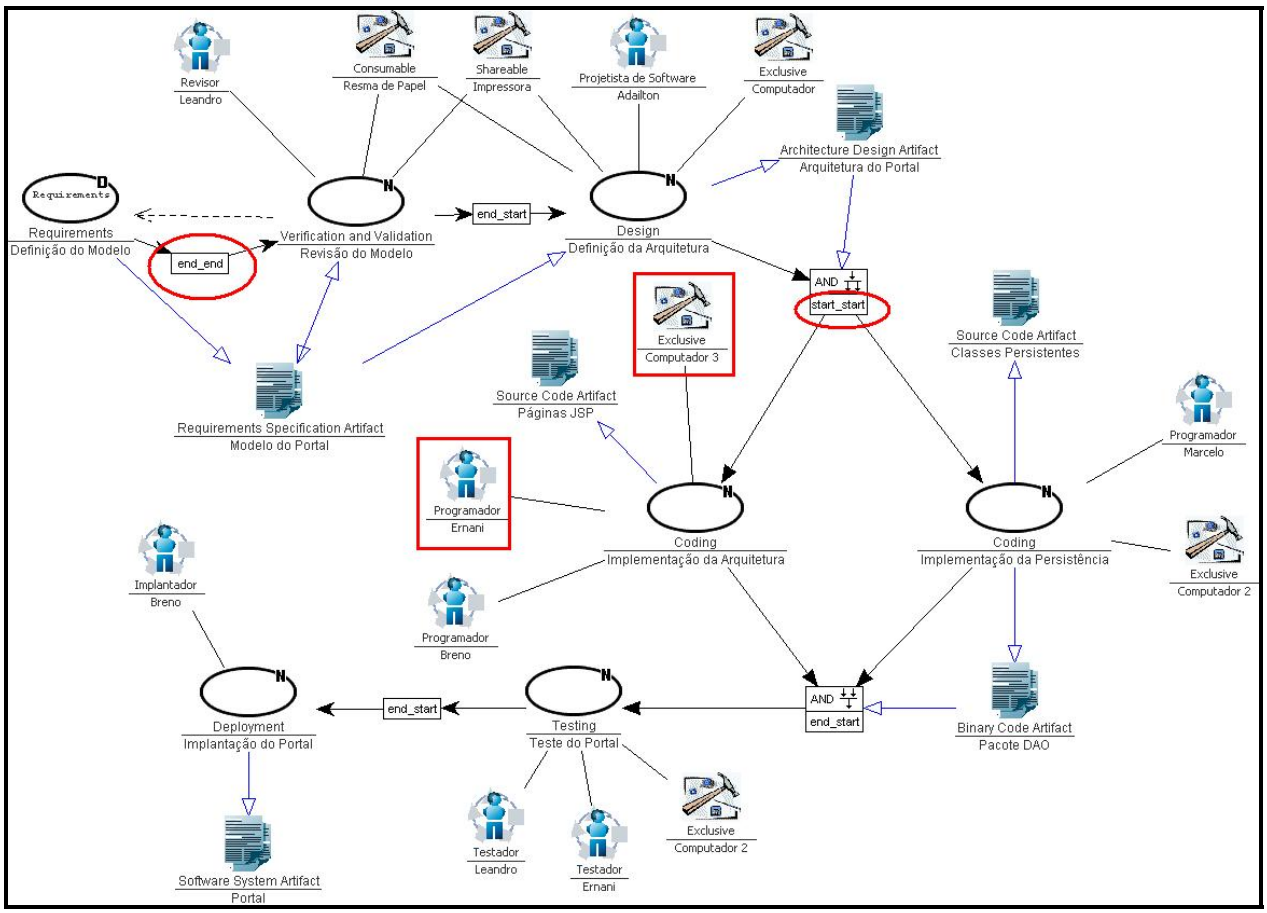

Figura 21. Modelo de Processo a ser simulado.

As modificações no modelo de processo foram realizadas com o intuito de diminuir o tempo de execução do processo. Para isso, A definição do modelo foi paralelizada com a revisão do mesmo, ou seja, ambas ocorreram simultaneamente, porém, a revisão só poderá terminar quando a definição do modelo terminar (***end-end***). Da mesma forma, a **Definição da Arquitetura** é dependência apenas para iniciar as atividades de codificação (**Implementação da Arquitetura** e **Implementação da Persistência**), isto é, a conexão de ramificação (chamada de *Branch*) é disparada ao início da atividade anterior. Sendo assim, a conexão ***Branch start-start*** permitirá que a implementação comece juntamente com a definição da arquitetura de maneira paralela.

Desta forma, o recursos exclusivo que existia para as atividades **Definição da Arquitetura** e **Implementação da Arquitetura** teve que ser alocado a somente uma delas, pois sendo um recurso exclusivo, impossibilitaria as atividades de ocorrerem em paralelo. Assim como o programador **Ernani**, que foi alocado para a atividade **Implementação da Arquitetura**

devido ao agente **Adailton** (definido anteriormente) estar alocado, durante este mesmo período, para a atividade **Definição da Arquitetura**.

6.3 RESULTADOS DA SIMULAÇÃO

Nesta seção serão apresentados os resultados e algumas considerações acerca da simulação realizada.

A tabela 18 apresenta os dados referentes à execução passada do processo mostrado na seção 6.1.

Tabela 18. Base Histórica.

Atividade	Início	Fim	Custo (Reais)
Definição do Modelo de Requisitos	2007-02-02 19:01:14	2007-02-07 19:01:17	630.00
Definição do Modelo de Análise	2007-02-07 19:01:41	2007-02-12 19:01:43	630.00
Revisão do Modelo	2007-02-12 19:02:14	2007-02-14 19:02:18	175.00
Definição da Arquitetura	2007-02-14 19:02:46	2007-02-19 19:03:08	510.00
Implementação da Arquitetura	2007-02-19 19:03:19	2007-02-27 19:05:07	1536.00
Implementação da Persistência	2007-02-19 19:05:36	2007-02-24 19:05:39	600.00
Teste do Portal	2007-02-27 19:05:56	2007-03-14 19:06:31	2400.00
Implantação do Portal	2007-03-14 19:06:53	2007-03-19 19:06:56	480.00
Total	45 dias = 360 horas (8 horas/dia)		R$ 6961.00

Na tabela 18 são listadas as atividades e seus respectivos dias e horas de início e fim, assim como o custo para cada atividade. Sendo o custo a soma dos custos dos recursos consumíveis e das horas trabalhadas pelos agentes. Além disso, a última linha apresenta a totalização da duração em dias e dos recursos da execução.

Após a simulação do modelo mostrado na figura 20, a tabela com os valores referentes à nova “execução” são apresentados na tabela 19.

Tabela 19. Resultados da Simulação.

Atividade	Início	Fim	Custo (Reais)
Definição do Modelo de Requisitos	2007-02-06 15:25:00	2007-02-11 16:25:00	630.00
Definição do Modelo de Análise	2007-02-11 17:25:00	2007-02-16 18:25:00	630.00
Revisão do Modelo	2007-02-06 15:25:00	2007-02-16 18:25:00	815.00
Definição da Arquitetura	2007-02-16 19:25:00	2007-02-21 20:25:00	510.00
Implementação da Arquitetura	2007-02-16 20:25:00	2007-02-24 21:25:00	1408.00
Implementação da Persistência	2007-02-16 20:25:00	2007-02-21 21:25:00	600.00
Teste do Portal	2007-02-24 22:25:00	2007-03-12 15:25:00	2400.00
Implantação do Portal	2007-03-12 16:25:00	2007-03-17 17:25:00	480.00
Total	40 dias = 320 horas (8 horas/dia)		R$ 7473.00

Na tabela 19 são apresentados os resultados da simulação realizada e algumas divergências dos dados em relação à execução passada são visíveis.

As duas primeiras atividades da tabela (**Definição do Modelo de Requisitos** e **Definição do Modelo de Análise**) mantiveram suas perspectivas de tempo (cinco dias cada) e custo (seiscentos e trinta reais cada), tendo em vista que o sub-processo da atividade decomposta **Definição do Modelo** não foi alterado. Entretanto, a tentativa de paralelizar a atividade de **Revisão do Modelo** com sua definição aumentou a carga horária do revisor, assim, aumentando também o custo da atividade, mas diminuiu o tempo total entre as atividades de definição e revisão de doze dias para dez dias.

Com o início da **Definição da Arquitetura** juntamente com as atividades de codificação, todo código que poderia ser implementado durante a especificação da arquitetura foi implementado. Assim, reduzindo o tempo das atividades de projeto e implementação para oito dias no total ao invés de treze. Houve ainda um ganho no custo da atividade de **Implementação da Arquitetura** devido ao custo por hora do agente Ernani ser menor em relação ao custo do agente anterior, Adailton.

Em resumo, essas modificações resultaram em uma redução de 11% do tempo anterior (cinco dias) e um acréscimo de 7% (R$ 512,00) no custo total do projeto.

A contribuição maior destes resultados não se limita aos números aqui apresentados, mas sim na possibilidade de experimentos em curto tempo que respondem a perguntas do tipo “o que aconteceria se ...?”, como no exemplo aqui apresentado: “O que aconteceria se as atividades de **Definição do Modelo** e a **Revisão do Modelo** fossem realizadas paralelamente?”, ou ainda “O que aconteceria se modificasse os agentes alocados para a atividade de **Implementação da Arquitetura**”. Essas e outras perguntas teriam respostas, mesmo que incertas, que indicam qual o caminho que o fluxo, custo e prazo do processo tomariam. Porém, o modelo não contempla questões de qualidade, isto é, o impacto que estas modificações teriam na qualidade da produção de cada atividade e no produto final.

7. CONCLUSÕES

Neste trabalho foi apresentada uma proposta de um modelo de simulação de processo de software baseado em conhecimento. Esta proposta objetiva utilizar o conhecimento de execuções passadas armazenado no ambiente WebAPSEE para fins de análise e estudo de processos de software.

O modelo proposto é baseado em uma abordagem mista, onde cálculo da duração é feito ora com base histórica da organização, ora com base no modelo COCOMO II. Ambas as abordagens acontecem inseridas em um contexto maior da simulação baseada em técnicas de simulação evento-discreta.

Na seção 7.1 são listadas as principais contribuições de forma resumida. A seção 7.2 apresenta uma análise crítica do modelo proposto. A seção 7.3 faz uma comparação com os trabalhos relacionados. A seção 7.4 aponta e discute as questões em aberto e os trabalhos futuros e a seção 7.5 apresenta as considerações finais deste trabalho.

7.1 RESUMO DAS CONTRIBUIÇÕES

A seguir as principais contribuições presentes neste trabalho são apresentas:

- Foi utilizada a técnica de avanço de tempo baseada em simulação orientada a eventos com um fluxo bem definido dos passos da simulação que apresenta situações de decisão, onde dependendo da resposta eventos diferentes serão disparados e o tempo (ciclo) da simulação é avançado (seção4.4);

- A abordagem RBC combinada com o modelo COCOMO II propõem uma maneira complementar para o cálculo da duração estimada das atividades, onde tanto o conhecimento obtido em execuções passadas dos processos de software da organização quanto o conhecimento advindo experiências de projetos internacionais são incorporadas para o cálculo da duração estimada das atividades. Sendo assim, se uma atividade semelhante foi realizada nesta organização, uma estimativa com base nessa experiência é utilizada. Entretanto, caso a organização nunca tenha realizado nada suficientemente semelhante para comparar, o modelo de estimativa COCOMO II é utilizado, dando confiabilidade aos resultados devido

a grande base de experiências e métricas incorporadas neste modelo pela calibragem 2000 (seção 4.4.2.4).

- O bom nível de detalhamento dos casos RBC para determinar a similaridade entre uma atividade atual e uma atividade passada é um elemento de destaque na arquitetura proposta. O grande número de atributos e relacionamentos envolvidos (seção 4.4.1.2) para determinar a semelhança, que são fornecidos pelo modelo de dados do ambiente WebAPSEE (seção 4.4.1.1), ajudam a aumentar a precisão da função de similaridade (von Wangenheim 2003);
- O desenho da arquitetura proposta é minimamente invasivo em relação ao sistema WebAPSEE. O uso do paradigma de Orientação a Aspectos na modificação do mecanismo de execução original do ambiente para fornecer conversão do tempo real do sistema para o tempo de simulação facilitou sobremaneira a reutilização do componente existente. Além disso, esta alternativa fornece um caminho para integração do ambiente com suas extensões em trabalhos futuros com necessidades semelhantes.

7.2 ANÁLISE DO MODELO PROPOSTO

Nesta seção é feita uma análise crítica sobre o modelo proposto considerando as técnicas utilizadas, a acurácia do modelo e a possibilidade de utilizá-lo em outros cenários. Esta análise aponta dificuldades, limitações e vantagens do modelo de acordo com os aspectos mencionados.

7.2.1 Considerações sobre as Técnicas Utilizadas

A utilização da técnica de inteligência artificial **Raciocínio Baseado em Casos** já foi considerada no domínio da Engenharia de Software (Maiden e Sutcliffe 1993) e também, mais especificamente, no domínio de Processos de Software (Reis 2002c). Entretanto, no contexto de Simulação de Processo de Software não foi encontrada nenhuma evidência da utilização desta técnica, apesar da existência de outros modelos baseados em conhecimento (Mi e Scacchi 1998).

Embora esta técnica seja aplicada às experiências passadas, a dificuldade de lidar com informações em alto nível de abstração, isto é, no nível de abstração de modelos de processo de software foi um ponto chave na definição da representação dos casos. Isto ocorre em função do tamanho do domínio que o valor cada atributo pode assumir, por exemplo, a quantidade de tipos de atividades que podem ser atribuídos aos processos. A hierarquia de tipos do ambiente

WebAPSEE contribuiu para a organização e classificação dos atributos, pois esta limita de certa forma o escopo destes valores.

O modelo COCOMO II abrange parte dos tipos de atividade permitidos pela simulação (seção 5.2.2). Entretanto, segundo Boehm (2000), em um projeto real de desenvolvimento de software é significativo o percentual de atividades pertencentes a estes tipos (projeto, codificação e teste) quando se trata de um desenvolvimento de um software desde a concepção. Para os outros tipos de atividades não contemplados pelo COCOMO II foram utilizadas algumas heurísticas e outros tipos não foram considerados na simulação (atribuindo ao tempo para simular a atividade o valor zero). Por exemplo, as atividades de gerência foram atividades consideradas importantes em todo o andamento do processo, sendo assim o tempo necessário para realizá-las será o tempo necessário para realizar todo o processo. As abstenções (não considerar uma atividade na simulação) são somente para situações de exceção do modelo, ou seja, quando não houver um método recomendado para realizar o cálculo.

Algumas exigências devem ser atendidas pelo usuário do modelo, como por exemplo: não faz sentido estimar esforço e prazos via COCOMO II para atividades onde não foi informado o tamanho estimado para o produto (artefato de saída) desta, seja em pontos por função ou em linhas de código. Outra exigência é a obrigatoriedade da atribuição de tipos para as atividades, pois este é um atributo chave para tomar decisões de como calcular o tempo de simulação (em ciclos) para cada atividade.

7.2.2 Acurácia do Modelo

Embora o modelo RBC aplicado ao modelo de simulação proposto nesse trabalho seja bem detalhado no que diz respeito ao número de atributos e relacionamentos dos casos, o seu mecanismo de adaptação fornece apenas uma atribuição da solução (tempo de duração da atividade) do caso passado para o novo caso. Isto influencia no grau de acurácia global da simulação, que será representado pela média e desvio padrão das similaridades de cada atividade. Não é realizado nenhum melhoramento ou adequação à solução do caso atual em relação aos casos passados. Em termos práticos, isto significa que os valores de atividades similares são retornados sem qualquer adaptação para o simulador.

Sobre o modelo COCOMO II.2000, em uma amostra de 30% dos projetos reais da base utilizada para calibrar o modelo, a estimativa de tempo acertou por volta de 72% do tempo real,

segundo o CSE-USC (2007). Alguns outros dados de sucesso são mostrados também em (Chulani, Boehm e Steece 1999).

O fato do modelo de simulação aqui apresentado utilizar dados reais de simulações passadas como fonte de conhecimento é, também, uma vantagem no que diz respeito à acurácia, pois se trata de fatos ocorridos no passado e não proposições ou incertezas.

Melhores resultados do modelo de simulação proposto neste trabalho tem o potencial de serem alcançados quando experimentações forem realizadas na indústria.

7.2.3 Adaptabilidade

Ao projetar um modelo de simulação, as preocupações estão voltadas ao domínio do sistema que se deseja simular. Desta forma, aspectos acerca da área de Processo de Software foram considerados na concepção deste modelo. Entretanto, abstraindo-se as questões relativas à processo de software e considerando o modelo de simulação como sendo constituído somente pelo modelo RBC, é possível simular processos de outros domínios, como por exemplo, processos de *Workflow*. Para isto, é necessário que se criem outros tipos de atividade, outros tipos de métricas, entre outros atributos que devem ser configurados de acordo com o domínio a ser aplicado. Esta característica ocorre em função da utilização do modelo de dados do ambiente WebAPSEE, como mostrado em Lima Reis (2003).

Algumas novas simplificações e heurísticas deveriam ser consideradas, pois as aqui adotadas referem-se ao domínio de processo de software. Além disso, as questões tratadas pelo modelo COCOMO II deveriam contar com um outro método de estimativa relativo ao novo domínio.

7.3 TRABALHOS RELACIONADOS

Muitos dos trabalhos relacionados não têm o mesmo objetivo deste. Portanto, uma comparação tentando responder às questões como “qual é o melhor modelo?” não é o foco desta seção, mas sim considerações acerca do que é abordado em um modelo e que não é tratado em outro.

Dos modelos com objetivos pedagógicos, isto é, modelos destinados ao ensino e treinamento de engenheiros de software, o modelo que mais se assemelha ao aqui proposto é o utilizado no simulador SESAM devido à utilização do modelo COCOMO para a estimativa de esforço e prazo para as atividades. Entretanto, no SESAM os tipos de atividade e de artefato são restritos e não representam a quantidade real de atividades envolvidas nos processos de software

atuais, tais como o Processo Unificado (Jacobson, Booch e Rumbaugh 1999). Quanto a suas restrições ainda, o SESAM não possui variedade nos dados organizacionais. São disponibilizados sempre os mesmo profissionais e com o mesmo perfil (cargo, habilidades e métricas).

O trabalho apresentado em Mi e Scacchi (1990), denominado *Articulator*, não possui integração com um ambiente real que proporcione dados de execuções passadas de processos. Apesar de ter uma interface bem definida entre os modelos de processo e suas instancias. Todo conhecimento que é alimentado em sua base de conhecimento (*knowledge base*) advém de estratégias pré-definidas no modelo atribuídas aos agentes do processo e alimentação por dados gerados em simulações passadas, ou seja, dados que originalmente já incorporam incertezas. Diferente disso, o modelo proposto no presente trabalho utiliza dados de execuções passadas reais, isto é, parte de fatos para atingir um maior grau de acurácia.

As desvantagens das simplificações aqui adotadas já foram citadas neste trabalho. Mas quando se trata de simular processos desconhecidos, ou seja, qualquer processo de software que o usuário venha a definir (situação no modelo aqui descrito), estas se tornam inevitáveis. As abordagens de modelos de simulação para um processo em uma organização em específico são mais ricas em detalhes. Por exemplo, o trabalho apresentado em Raffo e Wakeland (2003), citado anteriormente, na previsão do impacto de atividades de verificação e validação independentes.

No trabalho de Raffo e Wakeland também é citado como trabalho futuro a inserção de outros modelos de processo para o simulador. Entretanto, ainda assim serão modelos de processo conhecidos em suas essências e poderão "prever" de maneira mais precisa o impacto de certas modificações. Isto ocorre devido ao fato que informações como características organizacionais e padrões de qualidade adotados serem conhecidos pelo modelo de simulação, ou seja, enquanto as informações que o modelo aqui proposto busca de forma abstrata na base de execuções passadas, nesta outra abordagem já são incorporadas ao modelo.

7.4 QUESTÕES EM ABERTO E TRABALHOS FUTUROS

Este trabalho propõe uma nova abordagem para simulação de processos no sentido de fornecer uma real utilização do histórico de execuções passadas de uma organização. Entretanto, algumas questões não foram totalmente solucionadas. Aqui nesta seção serão mencionados algumas questões e trabalhos futuros a partir dos avanços alcançados nesse trabalho.

Nenhuma solução em simulação de processos da literatura trata de um modelo capaz de simular um **processo iterativo** qualquer, isto é, nenhum modelo foi construído para lidar

qualquer que seja o processo de software iterativo. Um caminho apontado pela literatura (Raffo e Wakeland 2003) (Dantas 2004) é o da abordagem de *Software Project Dynamics* proposta por Abdel-Hamid e Madnick (1991), onde é analisado o comportamento evolutivo dos processos.

Um outro aspecto ainda em aberto é o das atividades com os tipos **Documentação** e **Verificação e Validação**. Estas ainda não possuem no modelo um método satisfatório para o calculo de suas durações em situações onde a base histórica não for suficiente.

O mecanismo atual de adaptação utilizado no modelo RBC também foi simplificado propositalmente em função da falta de experimentação. Uma alternativa vem sendo avaliada que se trata de utilizar o método de Estimativa por Analogia apresentado em McConnell (2006), onde um projeto é estimado em função de seus semelhantes através da multiplicação dos atributos como tamanho e número de pessoas envolvidas por um fator (a razão) entre o novo projeto e o antigo semelhante. Entretanto, experimentos necessitam ser feitos para a adoção deste método.

O texto - seção 4.4.1.2 – menciona que a calibragem dos pesos deve ser determinada a partir de experimentações realizadas com dados históricos de processos reais.

Em um aspecto mais geral, o modelo de simulação é extensível para incorporar questões de qualidade de software, isto é, além de considerações acerca de tempo e custo, futuramente pretende-se que este modelo possa considerar questões de qualidade, como número de defeitos inseridos por atividade, cumprimento dos requisitos, entre outros.

Por fim, o mecanismo utilizado na interpretação das regras empíricas apresenta uma sintaxe inspiradas no modelo de políticas do WebAPSEE (Lima Reis 2003), entretanto com a semântica diferente. Existe a idéia de juntar esse mecanismo ao de políticas para complementar as situações que, atualmente, são bem resolvidas com o mecanismo de políticas.

7.5 CONSIDERAÇÕES FINAIS

O propósito deste trabalho é contribuir com a análise de modelos de processo de software quanto a sua estrutura e instanciação antes da execução, prevendo o impacto no tempo e custo necessários para executá-los.

Este capítulo apresentou as principais contribuições deste trabalho, assim como uma análise do modelo proposto e uma comparação com os trabalhos relacionados. Apesar dos avanços disponíveis na literatura, muitas questões estão em aberto na área de simulação de processo de software, principalmente no que diz respeito a processos evolutivos.

Com este modelo espera-se agregar um valor ainda maior ao ambiente WebAPSEE, associando ao ambiente de modelagem e execução de processos a capacidade de simulação. Desta forma, é mais uma meta-atividade do ciclo de vida dos processos de software apoiada pelo ambiente.

8. REFERÊNCIAS BIBLIOGRÁFICAS

AALST, Wil van der. Generic Workflow Models: How to handle dynamic change and capture management information. In: FOURTH IECIS INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, 1999, Edinburger. **Proceedings of…** Edinburger: 1999.

ABDEL-HAMID, Tarek; MADNICK, Stuart. **Software Project Dynamics**: An Integrated Approach. Englewood Cliffs, New Jersey: Prentice-Hall Software Series, 1991.

BANDINELLI, Sergio et al. Coupled vs Decoupled User Interaction Environments in PSEEs. In: 9th IINTERNATIONAL SOFTWARE PROCESS WORKSHOP, 10, 1994, Reston, USA. **Proceedings of…** Reston, USA: 1994.

BANDINELLI, Sergio; FUGGETTA, Alfonso; GHEZZI, Carlo. Software Process Model Evolution in the SPADE Environment. **IEEE Transactions on Software Engineering**, v. 19, n. 12, dec. 1993.

BOEHM, Barry et al. **Software Engineering Economics.** 1.ed. Prentice Hall, 1981.

BOEHM, Barry et al. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. In: SOFTWARE ENGINEERING SPECIAL VOLUME ON SOFTWARE PROCESS AND PRODUCT MEASUREMENT, 1995, Amsterdam, Netherlands: **Annals of**…, Amsterdam, Netherlands, 1995.

BOEHM, Barry et al. **Software cost estimation with Cocomo II**. 1.ed. Prentice Hall, 2000.

BROOKS, Frederick. **The Mythical Man Month**. Anniversary ed. Addison-Wesley, 1995. Cap 2. p. 13-29.

CHEN, Yu; GANNOD, Gerald C.; COLLOFELLO, James S.; SARJOUGHIAN, Hessam S. Using simulation to facilitate the study of software product line evolution. 7th IINTERNATIONAL WORKSHOP ON PRINCIPLES OF SOFTWARE EVOLUTION, 2004. **Proceedings of**…2004. p. 103- 112.

CHULANI, Sunita; BOEHM, Barry; STEECE, Bert. Bayesian Analysis of Empirical Software Engineering Cost Models. **IEEE Transactions on Software Engineering**, v. 25, Issue 4. p: 573 – 583. 1999.

CLEMENTS, Paul C. et al. Project management in a software product line organization**. Software, IEEE**. v. 22, Issue 5, p. 54 – 62, set.-out. 2005.

CSE-USC. Center for Software Engineering at University of Southern California. **COCOMO II Status**. Disponível em: <http://sunset.usc.edu/affiliates/cocomo/index.html>. Acessado em 09 de fevereiro de 2007.

DANTAS, Alexandre R.; BARROS, Márcio O.; WERNER, Cláudia. M. L. A Simulation-Based Game for Project Management Experiential Learning. In: INTERNATIONAL CONFERENCE

ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 08, 2004, Banff, Canada. **Proceedings of**… Banff: 2004.

DONZELLI, Paolo. A Decision Support System for Software Project Management. **IEEE Software**, v. 23, Issue 4, p. 67 – 75, jul. 2006.

DRAPPA, Anke; LUDEWIG, Jochen. Simulation in Software Engineering Training. In: 22nd IINTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE '00, 2000. **Proceedings of**…2000. p. 199.

LABES-UFPA. Laboratório de Engenharia de Software da Universidade Federal do Pará. **Documentação de Referência do Sistema WebAPSEE 1.0**. Belém, 2006.

FINKELSTEIN, Anthony; KRAMER, Jeff; NUSEIBEH, Bashar. **Software Process Modeling and Technology**. v. l. John Willey and Son Inc. Research Study Press, 1994.

FORRESTER, Jay Wright. **Industrial Dynamics**. Cambridge: The MIT Press, 1961.

FRANÇA, Breno et al. Gerência Flexível de Processos de Software com o Ambiente WebAPSEE. In: 19º. SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 10, 2006, Florianópolis. **Sessão de Ferramentas...**Florianópolis: 2006.

FREITAS FILHO, Paulo José de. **Introdução à Modelagem e Simulação de Sistemas com Aplicações em Arena**. Florianópolis: Visual Books, 2001.

FUGGETTA, Alfonso. Software Process: A Roadmap. In: 22th INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE'2000, Ireland. **Proceedings of**… Ireland, ACM Press, 2000.

GIMENES, Itana. O Processo de Engenharia de Software: Ambientes e Formalismos. In: XIII JAI, XIV Congresso da SBC, Caxambu-MG. **Anais**…Caxambu: 1994.

GRUHN, Volker. Process-Centered Software Engineering Environments: A brief history and future Challenges. **Annals of Software Engineering**, Kluwer, v. 14, p. 363-382. 2002.

HEINL, Petra. et al. A comprehensive approach to flexibility in workflow management systems. **Software Engineering Notes**, New York, v.24, n.2, p. 79-88. 1999.

HUMPHREY, Watts S. Characterizing the Software Process: A Maturity Framework. **Software, IEEE,** p73-79, mar. 1988.

HUMPHREY, Watts S. **A discipline for SoftwareEngineering.** Addison-Wesley Publishing Company, 1995.

JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. **The Unified Software Development Process**. Massachusetts: Addison Wesley Longman, Inc., 1999.

LIMA REIS, Carla Alessandra; REIS, Rodrigo Quites; ABREU, Marcelo M.; Schlebbe, Heribert. Nunes, Daltro J. Flexible Software Process Enactment Support in the *APSEE* Model. In: IEEE

INTERNATIONAL SYMPOSIUM ON HUMAN-CENTRIC COMPUTING LANGUAGES AND ENVIRONMENTS (HCC'2002), Arlington, USA, 9, 2002. **Proceedings of**…Arlington, USA: IEEE CS Press, Los Alamitos, set 2002a.

LIMA REIS, Carla Alessandra. Resource Instantiation Policies in Software Process Environments. In: 26th ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC'02), Oxford, England, 2002. **Proceedings of**...Oxford: IEEE CS Press, Los Alamitos, 2002b.

LIMA REIS, Carla Alessandra. **Uma Abordagem Flexível para Execução de Processos de Software Evolutivos**. 2003. Tese de Doutorado – PPGC, Universidade Federal do Rio Grande do Sul, Porto Alegre.

LUCRÉDIO, Daniel et al. Uma ferramenta CASE para o Desenvolvimento de Software Orientado a Aspectos. In: SESSÃO DE FERRAMENTAS DO SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 11. **Anais**... Brasília: SBC, out. 2004.

MAHER, Mary Lou; SILVA GARZA, Andrés Gómez. Case-based reasoning in design. **Expert, IEEE**. v. 12, Issue: 2, p. 34-41, mar/apr. 1997.

MAIDEN, Neil; SUTCLIFFE, Alistair. Case-based reasoning in software engineering. In: IEE COLLOQUIUM ON CASE-BASED REASONING, 2, 1993. **Proceedings of**…p.2/1 - 2/3, feb. 1993.

McCONNELL, Steve. **Software Estimation:** Demystifying the Black Art (Best Practices). Microsoft Press. mar, 2006.

MI, Peiwei.; SCACCHI, Walt. A Knowledge-Based Environment for Modeling and Simulating Software Engineering Process. **Knowledge and Data Engineering, IEEE Transactions**, v. 2, p. 283-289, sep. 1990.

MÜNCH, Jürgen.; ARMBRUST, Ove. Using Empirical Knowledge from Replicated Experiments for Software Process Simulation: A Practical Example. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, ISESE, 2003. **Proceedings of…**p. 18- 27, oct. 2003.

NAVARRO, Emily O. The Fundamental Rules of Software Engineering. Disponível em: <http://www.ics.uci.edu/~emilyo/SimSE/se_rules.html>, 2002. Acessado em 20 de Janeiro de 2007.

NAVARRO, Emily O.; VAN DER HOEK, A.; Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model**.** In: 18th CONFERENCE IN SOFTWARE ENGINEERING EDUCATION & TRAINING (CSEET'05), 2005. **Proceedings of**…2005.

NUNES, Daltro José. Estratégia Data-Driven no Desenvolvimento de Software. In: ASIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 6. (SBES'1992), Gramado, 1992. **Anais**...Gramado: SBC, Porto Alegre, 1992, p. 81 - 95.

OSTERWEIL, Leon. Software processes are software too. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Monterey, California, United States, 1987. **Proceedings of**…Monterey, California, United States: p. 2 – 13. 1987.

PARNAS, Dewayne. L.; CLEMENTS, P. C.; WEISS, D. M. The Modular Structure of Complex Systems. **IEEE Transactions on Software Engineering**, v. 11, Issue 3, mar. 1985. p. 259 – 266.

PAULK, Mark C. et al. **Key Practices of the Capability Maturity Model**. 1991

PAXIÚBA, Carla. et al. Towards an Event Recording Mechanism for a Process-based Environment. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 32, 2005. **Anais**…São Leopoldo: SBC, 2005.

PRESSMAN, Roger. S. **Software Engineering**: A practitioners approach. 6.ed. McGraw Hill. 2005.

ProSim 2006. Workshop on Software Process Simulation and Modeling. Disponível em <http://www.cnsqa.com/spw2006/jsp/html/spw/index.jsp>. Acessível em 28 de janeiro de 2007.

RAFFO, David. Getting the Benefits from Software Process Simulation. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING (SEKE'99), Held in Kaiserlautern, Germany. **Proceedings of**… Held in Kaiserlautern, jun, 1999.

RAFFO, David. E WAKELAND, Wayne. Assessing IV & V benefits using simulation. In: 28th ANNUAL NASA GODDARD SOFTWARE ENGINEERING WORKSHOP, p. 97 – 101, 2003. **Proceedings of…**2003.

REIS, Rodrigo Quites et al. Automatic Verification of Static Policies on Software Process Models. **Annals of Software Engineering**. Special Volume on Process-Based Software Engineering. v. 14. Kluwer Academic Publishers, 2002a.

REIS, Rodrigo Quites et al. Towards an Aspect-Oriented Approach to Improve the Reusability of Software Process Models In: INERNATIONAL WORKSHOP ON EARLY ASPECTS, 2002. **Proceedings of...**: ACM SigSoft, 2002b.

REIS, Rodrigo Quites. **APSEE-Reuse: Um Meta-Modelo para Apoiar a Reutilização de Processos de Software**. 2002c, 214 p. Tese de Doutorado – PPGC, Universidade Federal do Rio Grande do Sul, Porto Alegre.

RUS, Ioana; BIFFL, Stefan; HALLING, Michael. Systematically combining process simulation and empirical data in support of decision analysis in software development. In: 14th INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, Ischia, Italy, 2002. **Proceedings of**…Ischia: p. 827 – 833. 2002.

SCACCHI, Walt. Experience with software process simulation and modeling. **Journal of Systems and Software.** 1999.

SETUBAL, João; MEIDANIS, João. **Introduction To Computational Molecular Biology**. PWS Publishing Company, 1997.

SILVA, Fábio *et al*. SimAgentProcess: Uma Ferramenta para Simulação de Processos de Software Baseada em Conhecimento. In: Anales de Terceras Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes Software (IDEAS'2000), Cancún, México, 2000. **Anales**...Cancún: 2000.

SKIENA, Steven; REVILLA, Miguel A. **Programming Challenges**: The Programming Contest Training Manual. Springer. 2003.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Análise de pontos de função**: medição, estimativas e gerenciamento de projetos de software. São Paulo: Érica, 2003.

VON WANGENHEIM, Christiane Gresse; VON WANGENHEIM, Aldo. **Raciocínio Baseado em Casos**. Editora Manole Ltda., 2003.

WANG, Yingxu; BRYANT, Antony. (Ed.). *Editors' Introduction:* Process-Based Software Engineering: Building the Infrastructures. Annals of Software Engineering. Special volume on Process-based Software Engineering, Boston, MA, v. 14, Oct. 2002.