UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Um Modelo de Simulação de Processos de Software Baseado em Conhecimento para o Ambiente PROSOFT

por

FÁBIO AUGUSTO DAS DORES SILVA

Dissertação submetida à avaliação,
como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Daltro José Nunes
Orientador

Porto Alegre, Janeiro de 2001

CIP – Catalogação na Publicação

Silva, Fábio Augusto das Dores

Um modelo de Simulação de Processos de Software Baseado em Conhecimento para o Ambiente PROSOFT/ por Fábio Augusto das Dores Silva. – Porto Alegre: PPGC da UFRGS, 2001.

123 p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2001. Orientador: Nunes, Daltro José.

1. Simulação de Processos de Software 2. Simulação Baseada em Conhecimento3. Agentes Inteligentes. I. Nunes, Daltro José. II. Título.

Dedico este trabalho aos meus pais

Agradecimentos

Agradeço a Deus, de fundamental importância na realização desse trabalho, principalmente nos momentos mais difíceis.

Agradeço a proteção e as luzes enviadas do céu por meu pai Ivo Silva, que me inspiraram na realização desse trabalho.

A força dada por minha mãe, que sempre me incentivou a seguir a diante, acreditando em mim e no meu trabalho. Aos meus irmãos e sobrinhos pelo carinho trocado nos telefonemas de domingo.

Agradecimento especial a Juliana, minha namorada, sempre caminhando junto comigo. Obrigado pelo apoio e compreensão em todos os momentos.

Agradeço ao meu orientador, Prof. Dr. Daltro José Nunes, pela disposição para ajudar, também pela compreensão e confiança no meu trabalho. Agradecimentos especiais aos companheiros de trabalho Rodrigo e Carla, que contribuíram diretamente na realização desse trabalho.

Agradeço à CAPES que através do PPGC forneceu a ajuda financeira que permitiu o desenvolvimento desse trabalho.

Agradeço também o apoio e solidariedade dos amigos Alexandre, Luciana, Adagenor, Jaime, Bruno, Otávio, Ronnie, Edson, Andréa, João Paulo, Evandro e a todos aqueles que de uma forma ou de outra ajudaram na realização dessa dissertação de mestrado.

Sumário

Lista de Abreviaturas

ADS	Ambiente de Desenvolvimento de Software
ATO	Ambiente de Tratamento de Objetos
CMM	*Capability Maturity Model* - Modelo de Capacidade e Maturidade
CSCW	*Computer-Supported Cooperative Work*
DSS	*Decision Support System*
ICS	Interface de comunicação do Sistema
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
PML	*Process Model Language*
PSEE	*Process-centered Software Engineering Environment*
SEI	*Software Engineering Institute* - Instituto de Engenharia de Software
SESAM	*Software Engineering by Simulation of Animated Models*
SPICE	*Software Process Improvement and Capability Determination* – Determinação da capacidade e melhoria do processo de software
UML	*Unified Method Language*

Lista de Figuras

Lista de Tabelas

Resumo

Construção de software com qualidade tem motivado diversas pesquisas na área de Engenharia de Software. Problemas como a grande complexidade requerida pelas aplicações atuais e a necessidade de gerenciamento de um número cada vez maior de pessoas envolvidas em projetos são obstáculos para serem transpostos.

Trabalhos relacionados a tecnologia de processos de software aparecem como uma proposta para se obter maior controle das atividades realizadas com o intuito de se obter maior qualidade. A simulação de processos de software, através da representação dos passos definidos em um modelo, tem sido utilizada no auxílio a gerentes de projetos de sistemas para fornecer-lhes informações preciosas sobre o desenvolvimento de um sistema especificado. A representação de conhecimento a respeito das características relacionadas a um ambiente de desenvolvimento ajuda na obtenção de simulações mais realísticas. A partir do modelo, o simulador obtém uma descrição do ambiente em que deve atuar, baseado no conhecimento que se tem a respeito do ambiente.

Esse trabalho apresenta um modelo de simulação de processos de software baseado em conhecimento para ser inserido em um ambiente de engenharia de processos de software. A função do modelo é simular um processo de software instanciado, procurando detectar inconsistências no mesmo que possam gerar problemas durante a sua execução, como aumento de custos e comprometimento da qualidade do(s) produto(s) obtido(s). Após a simulação o projetista pode constatar a necessidade de se refazer o modelo, ajustar parâmetros ou executar o processo de software. O objetivo da simulação, nesse trabalho, é auxiliar as pessoas responsáveis por um ambiente de desenvolvimento a obter modelos de processos validados.

O modelo de simulação foi definido para ser utilizado no ambiente PROSOFT, que é um ambiente de desenvolvimento que permite a integração de novas ferramentas para desenvolvimento de software. O ambiente PROSOFT vem recebendo propostas de extensão que tem contribuído para o seu aprimoramento, fornecendo para seus usuários uma quantidade cada vez maior de ferramentas de auxílio a construção de artefatos de software. As propostas mais recentes foram um modelo para construção de sistemas especialistas, a definição de um ambiente cooperativo e um gerenciador de processos de software.

ATOs algébricos (construções do PROSOFT) são utilizados para especificar formalmente o modelo de simulação definido neste trabalho. A validação é realizada através de um modelo em UML (*Unified Method Language*) que foi utilizado como base para a construção de um programa implementado usando a linguagem Java. Isso ocorre porque a ferramenta do PROSOFT (implementada em Java) que seria utilizada para validar as especificações algébricas ainda não está finalizada.

Palavras-chave: Ambientes de engenharia de processos de software, simulação de processo de software, validação de processos de software, simulação baseada em conhecimento, modelo de simulação e agentes inteligentes.

Title: “Knowledge-Based Software Process Simulation Model for PROSOFT Environment.”

Abstract

Research in the Software Engineering area has been motivated by the need to improve software quality. Problems like the complexity of the current applications and the difficulty in managing many people involved are obstacles to be transposed.

The software process technology research intends to improve the software quality through better activities control. Software processes simulation has been used by managers in the aid of systems design to supply them with precious information on the behavior of the development of a specified system. The knowledge representation regarding the features of a development environment brings more realistic simulations. The simulator, based in a model, gets a description of the environment where it must act, established in the knowledge that it has about the environment.

This work presents a knowledge-based software processes simulation model to be inserted in a process-centered software engineering environment. The model objective is to simulate a software process in order to detect inconsistencies on it. Those inconsistencies could generate problems during process execution, as high costs and low quality products as result. After the simulation, the designer can evidence the need to refine the model, adjusting parameters in order to execute the software process. This research aims to help people responsible for software development environment to get validated processes models.

The simulation model was specified in PROSOFT environment. This environment allows new tools integration for software development. PROSOFT environment has been receiving various proposals for extension that has contributed for its improvement, supplying its users with tools to help software construction. Most recent proposals are an expert systems construction model, a definition of a cooperative environment and software processes manager.

Algebraic ATOs (PROSOFT constructions) are used to specify the formal simulation model defined in this work. The validation is carried through a model in UML that was used as base for the construction of a program implemented with Java language. This occurs because the PROSOFT tool (Java implementation) that would be used to validate the algebraic specifications is still not finished.

Keywords: Process-centered software engineering environment, software process simulation, and software processes validation, knowledge-based simulation, simulation model and intelligent agents.

1 Introdução

A obtenção de qualidade de software é um dos principais objetivos da Engenharia de Software. Diversas abordagens e ferramentas vem sendo propostas com o intuito de se obter produtos que funcionem corretamente e atendam aos requisitos dos usuários. Os dois principais focos de qualidade estão em se medir os produtos obtidos e o processo de desenvolvimento [RUS98].

O foco na qualidade dos produtos é uma abordagem tradicional, que procura identificar características que um produto deve apresentar, para ser considerado de qualidade. O problema dessa abordagem está no fato de ser estática, já que pouca coisa pode ser feita se forem identificados problemas nos produtos analisados. O foco na medida de qualidade de processos é mais recente e atua durante o desenvolvimento. Existem alguns modelos de padrão de qualidade definidos por organismos internacionais. Os padrões mais representativos são o modelo CMM (Capability Maturity Model) [HUM99], que foi proposto no SEI (Software Engeneering Institute - Carnegie Mellon University) e o modelo SPICE (Software Process Improvement and Capability Determination) [PAU95], que é uma norma ISO/IEC 15504 baseado no modelo CMM e na norma ISO9001/9000-3.

Problemas comuns no desenvolvimento de software incluem a complexidade dos produtos de software requeridos, as dificuldades no estabelecimento dos requisitos, a estabilização dos requisitos e a grande dificuldade de manutenção de software [LIM98]. Além disso, de acordo com [COL98], existem três variáveis importantes no desenvolvimento de software: tempo, custo e qualidade. Estas variáveis são inter-relacionadas e infortunamente, abordagens que têm efeito positivo em uma das variáveis, frequentemente produzem impactos negativos nas outras. Daí a necessidade de ambientes que auxiliem gerentes no desenvolvimento de forma global, obtendo ganhos de tempo, custo e qualidade do software produzido.

A tecnologia de processos de software busca dar suporte à construção de ambientes que permitam a definição formal dos passos que devem ser realizados durante o desenvolvimento de software. Representar os passos necessários na construção de um software faz com que eles não fiquem implícitos e tenham significado a partir de uma visão global. A essa representação dá-se o nome de modelagem de processos de software. Com os passos representados em um modelo é possível a gerência automatizada da realização dos passos de um processo, atividade denominada de execução de processos de software.[1]

Pesquisas atuais na área de processos de software incluem a busca por linguagens visuais para modelagem de processos com semântica bem definida, ferramentas de auxílio à reutilização de processos, ferramentas de auxílio à instanciação de processos de software, validação dos processos de software antes de serem executados e execução de processos de forma mais flexível.

---

[1] O termo frequentemente usado na literatura especializada é *software process enaction*, pois o processo é executado por máquinas + pessoas. Neste trabalho usaremos o termo execução com o sentido de *enaction*.

Nesse capítulo é apresentada uma breve descrição dos conceitos fundamentais para esse trabalho. A seguir são descritos o ciclo de vida do processo, a simulação de processos de software e o ambiente PROSOFT. Por fim serão apresentados o objetivo e a organização do trabalho.

1.1 Ciclo de Vida do Processo

Modelos de processos são evolutivos e necessitam de atualização contínua. Existe um ciclo de vida para processos de software que é análogo ao ciclo de vida de software. A figura 1.1 apresenta um ciclo de vida de processos de software considerado nesse trabalho, baseado em [LIM98]. Na figura, os retângulos representam atividades que constituem o meta-processo de software e as setas indicam o fluxo de modelos de processo em diferentes níveis de detalhe. Nesse ciclo, a execução de processos opera sobre um modelo previamente validado e que tenha sido continuamente aperfeiçoado desde a análise dos seus requisitos até a sua instanciação. Durante a execução, o processo pode ser alterado ou pode estar incompleto, então as fases de projeto, instanciação e simulação podem ocorrer diversas vezes com o objetivo de aperfeiçoar o processo adotado.

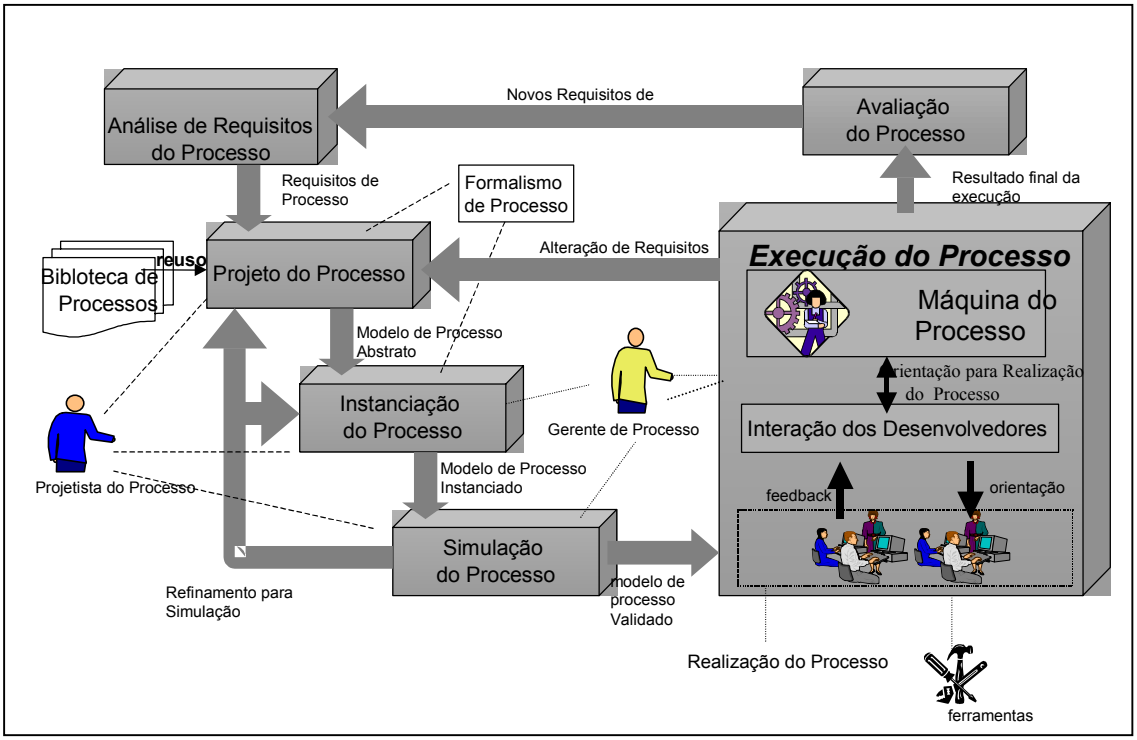

FIGURA 1.1 - Meta-modelo de processo de software (baseado em [LIM98])

A validação e verificação de modelos de processos de software são problemas que vem merecendo grande interesse por parte da comunidade que trabalha com processos de software. Uma questão é como se obter produtos de qualidade com a menor quantidade possível de realizações do ciclo do modelo, reduzindo assim gastos.

Apesar da tecnologia de processos de software fornecer uma base formal para representação de atividades de desenvolvimento, ajudando na comunicação entre os envolvidos, não é possível prever apenas com uma avaliação informal, as implicações que o modelo terá no desenvolvimento. Como garantir que a distribuição de tarefas

entre os desenvolvedores seja a mais apropriada e que os prazos previstos sejam alcançados, evitando prejuízos financeiros devido a ocorrência de atrasos?

Uma proposta é a utilização de modelos de simulação de processos de software. Através desses modelos é possível representar o comportamento de gerentes e desenvolvedores em um ambiente de desenvolvimento de software baseado em um modelo de processos, dessa forma é possível prever de forma aproximada as consequências que a aplicação de um modelo terá para um ambiente de desenvolvimento.

A possibilidade de representar os passos de desenvolvimento, utilizando modelos de processo de software e a possibilidade de testar esses modelos antes que eles sejam efetivamente implantados, através da simulação de projetos de software, permite a validação do que está sendo representado e a descoberta de possíveis inconsistências, que poderiam acarretar grandes prejuízos se ocorressem em um projeto real.

1.2 Simulação de Processos de Software

Simulação vem sendo usada por diversas áreas de conhecimento para entender o comportamento de sistemas reais (atuais ou planejados). Por ser uma ferramenta utilizada para uma grande quantidade de propósitos, existem diversos trabalhos caracterizando simulação.

Para [MAD99], simulação é desenvolver um modelo de sistema e realizar experimentos com ele. O propósito desses experimentos é representar como funciona a realização de sistemas propostos ou reais, para predizer os efeitos de modificações no sistema com a progressão de tempo. Simulação é uma eficiente ferramenta de comunicação para mostrar como processos trabalham, enquanto se pensa de que forma eles podem ser melhorados. Simulação também pode ser usada em treinamento individual, através da interação dos participantes com modelos executando em tempo real, para perceberem os efeitos de suas decisões.

Para [STR84] simulação é utilizada para descrever o funcionamento dos sistemas. A descrição do sistema, a fim de satisfazer as necessidades de estudo e análise, consiste em escolher adequadamente procedimentos e formas de representar componentes e expressar as relações entre os mesmos por expressões lógicas ou matemáticas. Este processo é conhecido como abstração, o resultado expresso é conceituado como modelo do sistema, ideia, fenômeno ou instrumento em estudo.

A simulação de processos, inserido em um ambiente de desenvolvimento de software é utilizada para aumentar o grau de interação entre as fases de modelagem e execução, pois permite que o modelo seja testado e aperfeiçoado, baseado em informações obtidas da simulação, antes de ser executado. A simulação pode ser útil também para validar os modelos de processo, para realização automática de escalonamentos e testes e no treinamento de gerentes de desenvolvimento.

1.3 Ambiente PROSOFT

O PROSOFT [NUN94] é um ambiente de desenvolvimento de software que integra ferramentas de apoio ao desenvolvimento e baseia-se na definição formal de software, particularmente no método algébrico. Trata-se de um ambiente que integra ferramentas para desenvolvimento de software, permitindo também que novas ferramentas sejam adicionadas ao mesmo. As ferramentas do PROSOFT visam auxiliar o desenvolvedor de software desde a especificação de requisitos até a implementação do sistema.

Recentemente foi proposto um gerenciador de processos [LIM98] ao conjunto de ferramentas do PROSOFT, fazendo com que ele passasse a apresentar características de um ambiente de desenvolvimento de software orientado a processo (PSEE – *Process-centered Software Engineering Environment*). Esse gerenciador permite a execução de um modelo de processo de software, previamente modelado.

A adoção de um ciclo de vida de processo utilizando a simulação, como o ciclo apresentado na seção 1.1, permitirá ao gerente de um projeto que esteja utilizando o PROSOFT validar modelos de processo antes da execução, avaliando possíveis consequências do que está representado.

Além do gerenciador de processos outras ferramentas foram definidas para o ambiente, tais como ferramentas de auxílio a construção de interfaces e de diagramas de fluxo de dados. Cita-se também a definição de um ambiente para construção de sistemas especialistas [MOR97], que permite a representação de conhecimento a respeito de qualquer domínio de informação e a definição de um ambiente cooperativo [REI98].

1.4 Objetivos do Trabalho

Nesse trabalho é proposto um modelo de simulação de processos de software, para ser integrado ao ciclo de vida do processo descrito na seção 1.1 e ser adicionado ao ambiente PROSOFT. Será apresentada uma especificação formal para esse modelo e também um modelo UML que serviu de base para a construção de um protótipo do simulador.

Para a especificação do simulador foi utilizada uma funcionalidade equivalente ao gerenciador de processos de software proposto por [LIM98], que é utilizado para representar e executar modelos de processo. O simulador tem como entrada um modelo de processo de software e como saída, avaliações realizadas acerca do desempenho desse modelo. As avaliações são obtidas da simulação, que utiliza conhecimento a respeito de um ambiente de desenvolvimento de software real, armazenado em uma base de conhecimento. Essas avaliações podem ser utilizadas pelo projetista para melhorar o modelo, criando uma iteração que só terminará quando o gerente de desenvolvimento considerar que o modelo está pronto para execução. Para enriquecer o modelo são utilizados conceitos da Inteligência Artificial – IA, pois sistemas multiagentes são utilizados para modelar a interação com os desenvolvedores na realização de tarefas, através de uma abordagem cognitiva. O sistema baseado em conhecimento é utilizado para representar conhecimento sobre o ambiente (heurísticas).

A ferramenta construída usando como base o modelo, que será apresentado, é utilizada para realizar experimentos em modelos de processos de software que representam o funcionamento de um ambiente real ou o funcionamento esperado em um ambiente proposto. Os experimentos permitirão validações no modelo, testando se o mesmo está de acordo com os requisitos levantados.

A validação do modelo submetido à simulação será possível através da análise das informações de saída da ferramenta, permitindo ao gerente verificar informações como: tamanho das filas para utilização de recursos, tempo de ociosidade de desenvolvedores, ocorrência de *deadlock* na execução de processos, etc. O modelo estará pronto para executar quando o gerente considerar que o escalonamento de desenvolvedores e recursos na realização de atividade permite que os prazos apresentados sejam possíveis de serem obtidos.

Definem-se como objetivos deste trabalho o estudo e avaliação da aplicação de técnicas de simulação na Engenharia de Software: verificação da possibilidade de se utilizar simulação como uma técnica de validação de modelos de processo; compreensão da utilidade da aplicação de técnicas de Inteligência Artificial na Engenharia de Software; realização da proposta de um modelo de simulação baseada em conhecimento contribuindo com o ambiente PROSOFT através da integração de uma nova ferramenta e por fim construção de um protótipo para validar os conceitos apresentados.

1.5 Organização do Trabalho

A apresentação deste trabalho está organizada assim:

No **segundo capítulo** são apresentados conceitos importantes da tecnologia de processos de software. Esses conceitos formam uma base importante para o trabalho, já que eles são objetos de estudo e são constantemente referenciados.

No **terceiro capítulo** são apresentados conceitos de simulação. O objetivo é apresentar um levantamento a respeito de simulação, avaliando de que forma a tecnologia de simulação pode ser útil ao ser integrada a tecnologia de processo de software São apresentados os conceitos da inteligência artificial que são importantes para o modelo de simulação.

No **quarto capítulo** é apresentado o estado da arte da tecnologia de simulação de processos de software. São apresentadas ambientes e ferramentas que serviram de base na definição do modelo de simulação proposto neste trabalho.

No **quinto capítulo** é apresentado o modelo proposto, que é um modelo de simulação de processos de software que utiliza agentes inteligentes.

O **sexto capítulo** apresenta uma especificação formal das ferramentas utilizadas para implementar o funcionamento do simulador de processos do PROSOFT.

No **sétimo capítulo** é apresentada uma representação dos passos da simulação.

O **oitavo capítulo** apresenta as conclusões do trabalho e trabalhos futuros.

O **anexo 1** apresenta operações formais complementares às operações apresentadas no sexto capítulo.

2 Tecnologia de Processos de Software

O modelo apresentado neste trabalho fará uso de conceitos da tecnologia de processos de software e ambientes de desenvolvimento de software. Através de uma representação formal, obtida por um modelo de processo de software, será possível simular fatos importantes que ocorrem em um ambiente de desenvolvimento.

Nesse capítulo serão estudados conceitos de processos de software, identificando como é feita a representação de elementos importantes no contexto de desenvolvimento de software. Serão apresentadas as fases principais da tecnologia que são a modelagem e a execução e também serão apresentados os três domínios relacionados à tecnologia de processos de software: o domínio da definição do processo, o domínio da realização do processo e o domínio da execução da definição do processo. Por fim serão apresentados alguns trabalhos recentes desenvolvidos nessa área.

2.1 Conceitos

De acordo com [GAR96], qualquer disciplina requer um conjunto de conceitos definidos precisamente em termos, para permitir comunicação e comparação de idéias. Neste trabalho um processo de software é visto como um conjunto de atividades, métodos, práticas e transformações usadas por pessoas para desenvolver software. Essa é uma definição geral definida e comumente aceita no SEI CMM [PAU95]. As atividades definidas no processo são realizadas para se obter um conjunto concreto de objetivos.

Para [ALO99], essa definição pode ser estendida com a observação de que um processo é uma sequência de programas de computador heterogêneos e distribuídos que trocam dados entre eles. Um sistema que suporta definição, execução e monitoramento de processo é um *sistema de gerenciamento de processos*. O que ocorre em muitas aplicações é que o objetivo dos projetistas é implementar em software, um processo da vida real (um experimento científico, uma série de transformações de dados e processos de negócio, etc). Em geral, um processo é uma série de passos que usam algum conjunto de entradas e transformam essas entradas em itens de saída úteis. Um processo não necessita envolver transformação física [RUS98].

Ambientes de Engenharia de Software centrados em processos (PSEE - *Process-centered Software Engineering Environment*) guiam pessoas envolvidas na execução de atividades cooperativas pertencentes ao domínio complexo do desenvolvimento de software. Para isso, PSEEs exploram formas de modelos de processos. Um modelo de processo é uma descrição formal dos vários passos que precisam ser realizados pelos agentes que são membros da organização de desenvolvimento de software. [BEC99]

PSEEs são apontados por [GAR96] como a nova geração de ambientes de Engenharia de Software, que apóiam não apenas a tarefa de desenvolvimento de software, mas também o gerenciamento associado e as funções de garantia de qualidade.

Um problema crítico para PSEE é a definição de uma interação adequada entre o sistema e os usuários. Em alguns casos, PSEEs não são completamente aceitos em ambientes industriais, por serem considerados muito prescritivos e limitados para representar a autonomia dos atores humanos. A representação da interação do usuário realiza um papel importante no estabelecimento desta percepção, fazendo do sistema um sucesso se a representação for intuitiva, fácil de aprender e precisa, fornecendo aos atores uma visualização efetiva do estado do processo de software. Em outras palavras, é preciso ter usuários cientes do processo e o ambiente de interação precisa ser flexível suficiente para suportar todos os pontos de vista que os diferentes usuários têm do processo.

2.2 Modelagem de Processos

Qualquer representação de um processo é um modelo desse processo. Para se ter mais precisão sobre modelagem de processo são definidos os seguintes conceitos:

- Um **desenvolvedor** é uma entidade que realiza um passo do processo. Um agente pode ser humano ou um computador que executa os passos do processo.
- Um **recurso** é uma entidade requirida para realizar um passo do processo. Alguns autores consideram que recurso é sempre uma entidade não humana, enquanto outros consideram que um desenvolvedor pode ser um recurso.
- Um **artefato** é uma saída obtida da realização do processo.

Usando esses conceitos, [GAR96] define modelo de processo em termos de um conjunto de atividades (ou passos) que são realizadas por agentes usando recursos para se obter um prouto.

Modelar o processo de software permite entender e apoiar o desenvolvimento de grandes sistemas de software. O processo de software é a coleção de atividades relacionadas, visto como um processo coerente sujeito a argumentação, utilizado na produção de um sistema de software [MI90]. A modelagem é um método para caracterizar a estrutura de um processo, qualquer processo.

O **modelo de processos de software** é uma representação formal (abstração) dos elementos envolvidos no processo de software, sendo que esses elementos podem ser executados por pessoas ou máquinas. A maioria das descrições de ciclo de vida representam um modelo extremamente abstrato de desenvolvimento de software, não fornecem orientação sobre como integrar os vários passos do processo que são realizados pelas pessoas envolvidas. Um dos objetivos da modelagem de processo é decompor estas descrições em detalhes suficientes para orientar a execução do processo [GIM94].

Outro componente da modelagem de processo de software envolve a representação de atributos de processos e de produtos do processo. Atributos descrevem o estado dos processos e seus produtos de trabalho associados. Atributos de processos incluem horas gastas, tempo de desenvolvimento, eficiência na remoção de defeitos e o número de processos alterados. Atributos de produtos típicos incluem: tamanho (número de objetos ou linhas de código), complexidade e número de defeitos. Esses atributos são trilhados quando um modelo é executado, para ajudar na validação do

modelo e para identificar áreas onde o processo pode ser melhorado.

2.2.1 Itens Relacionados com Modelagem de Processo

O trabalho [TIA99] apresenta uma breve história da modelagem de processos e seus limites. Essa história é apresentada em três itens relacionados que são: (1) assistência e controle, (2) reuso e interoperabilidade e (3) mobilidade e delegação.

Assistência e controle de modelagem de processo servem para automatizar o processo de desenvolvimento de software, construindo sistemas hábeis a assistir e guiar a produção de software baseados em modelos de processo. Reuso e interoperabilidade na modelagem de processo trabalha com a heterogeneidade entre os modelos. Mobilidade e delegação no contexto de modelagem de processo são usadas para propor mecanismos que permitam migração de componentes de processo de um ambiente para outro.

2.2.2 Linguagens de Modelagem

De acordo com [FRA99] um importante esforço vem sendo realizado nos últimos anos com o objetivo de definir linguagens de modelagem de processos (PML - *Process Model Language*) adequadas para processos de software. Como resultado, algumas importantes características foram adicionadas: orientação a objetos aparece como uma forma natural de modelar a parte estrutural dos processos, ao mesmo tempo em que obtém um certo grau de reuso e modularidade; avanço das linguagens de programação de processos para notação gráfica, de paradigmas de controle imperativo para reativo, de sistemas orientados a documentos para sistemas orientados a atividade; o nível de abstração dos sistemas aumentou, o que tem tornado mais fácil seu uso; entre outras características.

Entretanto, existem outros aspectos em modelagem de processos usando PMLs que ainda necessitam de estudos mais detalhados. Entre eles [FRA99] menciona que: muitas PMLs são difíceis de usar para descrever fina granularidade de processos enquanto conservam uma notação de alto nível; ainda que algumas PMLs sejam difundidas na comunidade, nenhuma delas se apresenta como um padrão para modelagem de processos; muitos dos modelos de processos são limitados para tratar com algumas mudanças e tomadas de decisão em tempo de execução.

Por outro lado, PMLs normalmente não fornecem representações para expressar formalmente algumas propriedades para serem usadas em tempo de execução ou para garanti-las. As habilidades e reuso fornecidos pelas PMLs não são poderosas o suficiente para tratar com a complexidade de processos de software.

Os aspectos apresentados por [FRA99] que são levados em consideração neste trabalho são: padronização, expressividade da parte dinâmica do modelo, modularidade, flexibilidade e aspectos formais.

2.2.3 Características da Modelagem

De acordo com [CUR92] e [ARM92] são objetivos da modelagem de processos de software:

- Facilitar a comunicação e compreensão entre as pessoas;
- Facilitar o aperfeiçoamento do processo;
- Prover orientação automatizada do processo;
- Suportar execução automatizada;
- Permitir reutilização;
- Suportar a gerência do processo.

Para que os processos sejam descritos adequadamente, muitas formas de informação devem ser integradas. As linguagens de modelagem utilizadas para representar modelos de processos de software diferem na forma de representação, utilizando uma ou mais perspectivas diferentes. De acordo com [CUR92], as perspectivas mais comumente utilizadas são: funcional, comportamental, organizacional e informacional.

Modelagem de processo de software é útil por um número de razões. A construção de um modelo força a documentação do processo como ele realmente existe. Isso pode demonstrar discrepâncias entre os processos definidos e os processos atuais. O modelo, uma vez desenvolvido, representa os processos da organização, podendo ser comunicado para todos do time de desenvolvimento tal que eles tenham um entendimento comum do processo. Isso é muito importante porque a maioria das organizações nunca tem seus processos definidos e processos atuais em sincronia.

2.3 Execução de Processos de Software

A fase de execução de um processo de software utiliza o modelo executável, obtido da instanciação dos conceitos abstratos do modelo de processo de software construído na fase de modelagem e pode ser caracterizada como a realização automatizada da construção de um software.

Quando o objetivo é executar um modelo de software, deve haver a preocupação em se coordenar múltiplos usuários que estão trabalhando em um ambiente e também as ferramentas automatizadas que devem estar disponíveis para um determinado usuário ou grupo de usuários.

A execução do modelo garante o gerenciamento do que está sendo realizado e o que será realizado. O que acontece de fato é uma interpretação automática do que deve ser feito em um ambiente de desenvolvimento para se alcançar os produtos desejados, gerando orientação e suporte para os desenvolvedores das atividades.

Automatizar o gerenciamento de uma execução de processos de software requer a utilização de ferramentas que irão manipular a descrição do processo. Têm-se para isso um **mecanismo de execução de processo**, que é responsável pela coordenação das atividades que devem ser realizadas por pessoas e outras ferramentas automatizadas.

O suporte a automatização de processos requer uma base computacional para controlar o comportamento dentro do ambiente automatizado.

Uma máquina de execução de processos deve tratar das questões: automação de processo, quando existirem atividades automáticas; trabalho cooperativo, para dar suporte a coordenação e cooperação entre pessoas trabalhando em um projeto de software; monitoração, para prover diferentes visões do estado da execução do processo; e registro da história do processo, para coletar dados da evolução do processo. [BAN92]

Os requisitos de execução que devem ser satisfeitos pelo mecanismo de execução e pelo ambiente de desenvolvimento que suporta o processo, apresentados por [FRO94] são: Fluxo de controle (sequência), iteração, interação humana, gerência de informação, métricas, adaptação dinâmica, execução de ferramentas e alocação de recursos.

O trabalho [FER93] apresenta um modelo de execução de processos de software dividido em quatro partes:

- Criação de uma instância do modelo: Uma instância é uma cópia executável do modelo.
- Fornecimento de valores para parâmetros da instância: O modelo pode conter variáveis de instância, que são assinaladas em diferentes instâncias do modelo.
- Início da execução: Esse passo é semelhante a "carregar e rodar" de um programa de computador. As instâncias executáveis começam a funcionar sobre o controle de uma máquina de processo.
- Modificação dinâmica de um modelo executável: Processos e modelos de processos não são entidades estáticas e em geral não podem ser completamente definidas em todos os níveis de detalhe *a priori*.

A simulação é uma abordagem para realizar a execução do modelo de processos de software sem utilizar recursos e ferramentas reais. Com o modelo pronto, usando uma ferramenta de software, o comportamento do modelo pode ser simulado. Quando o modelo é executado via simulação, a dinâmica do modelo causa alteração dos valores dos atributos sobre o tempo, durante o período de execução. Os atributos (custo, escalonamento, qualidade, etc.) podem ser graficamente plotados, de forma que um gerente possa avaliar a performance do modelo relativo ao projeto real. [RUS98]

Um conceito comumente utilizado para representar a execução de processos de software do ponto de vista do desenvolvedor é a agenda. A agenda é um ferramenta genérica para gerenciar trabalho individual. Uma agenda contém todas as atividades realizadas, atividades sendo realizadas e atividades prontas para serem realizadas. Exemplos da utilização de agendas na representação da execução de processos podem ser encontrados em [FER93] e [LIM98]. A figura 2.1 [LIM98] é um exemplo de uma representação da utilização de agendas.

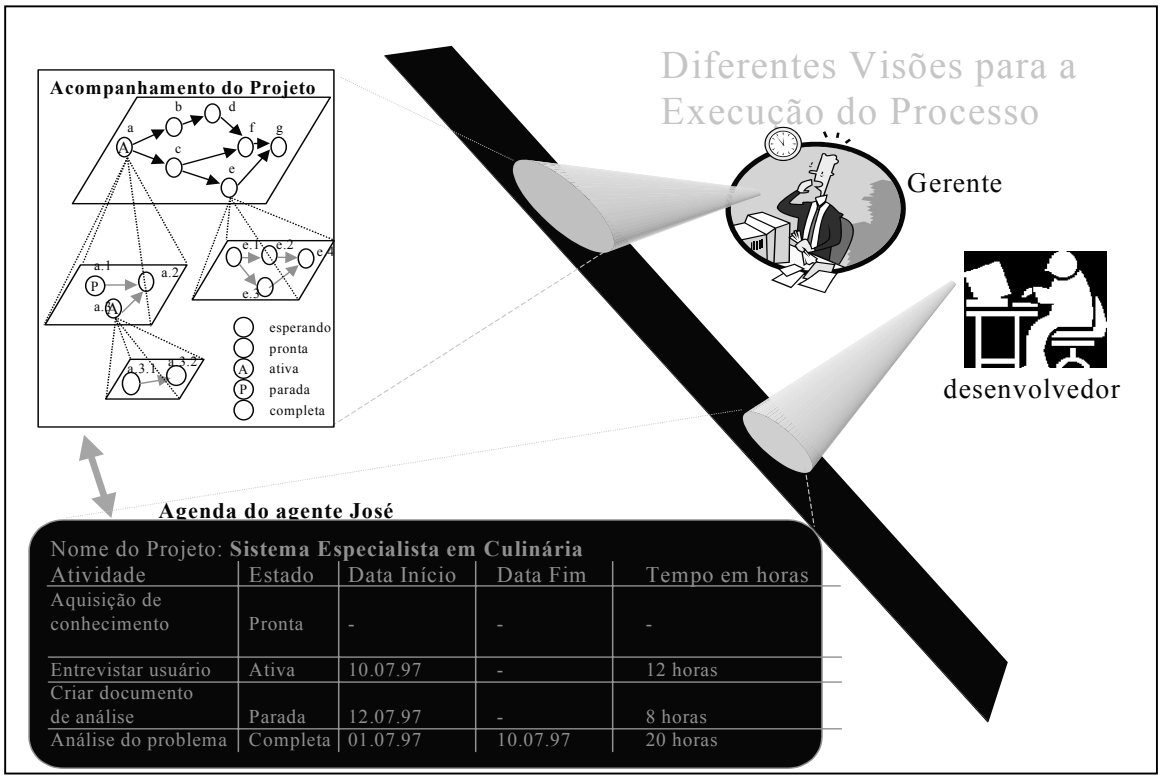

FIGURA 2.1 - Utilização de agendas na execução de processos [LIM98]

2.4 Domínios de Processo de Software

Em [DOW94] são apresentados três diferentes domínios a serem reconhecidos em processos de software: o domínio das definições do processo, o domínio da realização do processo e o domínio da execução da definição do processo.

O domínio da definição do processo contém características de processos ou fragmentos de processos, expressados em alguma notação, em termos de como eles podem ou devem ser realizados. Definição de processos executáveis inclui: informações para guiar a realização do processo, tal como descrições de como os passos podem ser realizados, definições de várias restrições em vários aspectos da realização ou nos produtos produzidos pelo processo e declarações de objetivos que a realização de um processo deverá alcançar.

O domínio da realização do processo indica atividades ou ações conduzidas por agentes humanos e agentes não humanos no desenrolar de um projeto de software. Todas as atividades do processo são parte da realização do processo. Algumas destas atividades serão conduzidas utilizando ferramentas de software ou então outras facilidades do ambiente.

O domínio da execução do processo definido é composto por aquilo que faz parte de um ambiente centrado em processos para suportar a realização de processos governado pela definição. Uma execução é iniciada carregando-se uma cópia de uma definição de processo executável ou um fragmento de definição para um mecanismo de execução. A figura 2.2 representa os três domínios e o fluxo de informações e *feedback* entre eles.

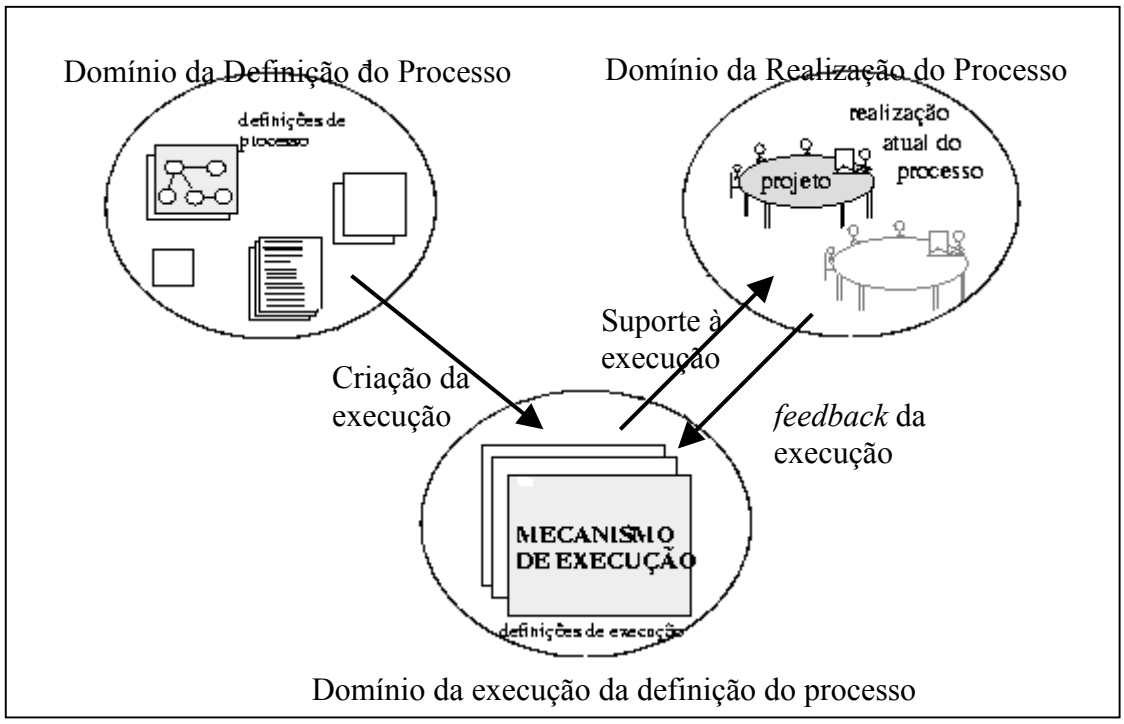

FIGURA 2.2 - Domínios de processos de software [DOW94]

2.5 Áreas Relacionadas

Existem diversas pesquisas correntes em várias áreas da ciência da computação eem que suas aplicações estão relacionadas com a tecnologia de processos de software. Desenvolvimentos em quatro áreas específicas são de especial interesse: trabalho cooperativo suportado por computador (CSCW- *computer-supported cooperative work*), banco de dados, automação de escritórios e *workflow*.

Pesquisas em CSCW exploram o uso do suporte do computador para facilitar e intensificar o trabalho realizado por um grupo de pessoas. Como no caso de PSEEs, pesquisas em CSCW possuem uma natureza interdisciplinar combinando questões em construção de sistemas, comportamento humano e estruturas organizacionais. Produtos que apóiam o trabalho em grupo aparecem rapidamente nos ambientes de desenvolvimento para dar suporte a coordenação e comunicação de grupos de trabalho cada vez mais distribuídos. Correio eletrônico foi um dos precursores de tais sistemas. A tecnologia produzida por sistemas CSCW pode formar uma importante parte da infra-estrutura na qual será construído um PSEE.

A tecnologia de banco de dados realiza um papel importante em qualquer sistema que gerencia uma grande quantidade de dados e informações. Ambientes de Engenharia de Software armazenam e gerenciam acesso para muitos objetos de diferentes tipos, tais como documentos de projeto, código fonte, código objeto e manuais de usuário. PSEEs adicionalmente gerenciam processos e relacionamentos entre objetos. Bancos de dados tradicionais não estão aptos a suportar as necessidades de ambientes de software que necessitam de objetos grandes com diferentes padrões de acesso. A pesquisa corrente em banco de dados está buscando soluções para tratar esses requisitos não tradicionais, em muitos casos construindo banco de dados especializados ou customizando os existentes.

Trabalhos em automação de escritórios e computação organizacional combinam várias facetas interdisciplinares de interesse a PSEEs. De forma simplista, automação de escritórios inicia com o esforço para automatizar tarefas rotineiras. Pesquisas recentes indicam que automação só será útil se processos de trabalho, de negócio e organizacionais estiverem integrados e bem entendidos. A simples tarefa de introduzir computadores em um ambiente de trabalho está sendo substituída pela busca de formas de usar o computador e a automação para integrar, dar suporte e reconstruir os processos da organização.

Sistemas de gerenciamento de *workflow* são sistemas usados especificamente para dar suporte aos processos organizacionais. A premissa desses trabalhos é que as unidades básicas de qualquer organização são os processos de trabalho realizados dentro dela. Processos de trabalho são compostos de comunicação entre os requisitantes e atendentes e ações realizadas por um responsável. Um modelo de processo de trabalho pode ser representado por uma rede conectando requisitantes a atendentes formando os nodos. O *workflow* é representado por arcos na rede. Uma rede representando todos os processos organizacionais torna possível analisá-lo para identificar coisas como comunicação redundante.

2.6 Ambientes Orientados a Processo

Existem diversos ambientes com suporte à execução de processos de software que resolvem as questões inerentes a área de diversas formas, com diferentes abordagens e objetivos diversos. Tais ambientes são resultantes da constante pesquisa nesta área. São exemplos de ambientes orientados a processo: EPOS, Endeavors, *ProcessWeaver* e OPERA.

O EPOS [CON91] [CON94] é um exemplo de ambiente de desenvolvimento de software que permite modelagem de processos e gerência de configuração. Sua arquitetura contém uma interface comum com o usuário e várias ferramentas e um gerente de atividades, dentre outras ferramentas. EPOS esá baseado em uma linguagem de processo, chamada SPELL, a qual consiste de uma extensão do sistema de banco de dados orientado a objetos.

EPOS consiste basicamente de um modelo de atividades, um modelo de produtos, um modelo de cooperação e o modelo de projetos que sumariza todos os modelos. Existem dois tipos de tarefas: ***Interactor***, com interação humana e ***Deriver***, a qual invoca uma ferramenta e é executada automaticamente.

Endeavors é um ADS orientado a processos com enfoque para questões de distribuição, comunicação e coordenação [TAY96]. O projeto do ambiente enfoca mecanismos para apoiar a distribuição de processos e usuários, integração de ferramentas, adoção incremental de novas tecnologias, customização e reutilização de componentes de processo e suporte à mudança dinâmica de processo.

O predecessor do Endeavors é o sistema Teamware [YOU94], a partir do qual foram herdadas diversas características. A linguagem Teamware foi criada para aumentar a coordenação de equipes durante o processo de software e aumentar o controle do processo através de uma representação mais precisa. Nesta linguagem o processo é definido através de uma rede de atividades e o interpretador de processos é o componente do ambiente responsável por guiar sua execução.

O ambiente *ProcessWeaver,* apresentado em [FER93] e [CHR95], provê

ferramentas para gerenciar tarefas individuais e automação do processo. Para o usuário do ambiente é provida uma janela chamada *Agenda*, através da qual as tarefas são enviadas para os usuários. Existem ainda outras ferramentas para o suporte à execução de processos, tais como: *Method Editor, Activity Editor, Cooperative Procedure Editor* e *Work Context Editor*. O *Method Editor* permite a definição das hierarquias de atividades, enquanto o *Activity Editor* permite especificação das entradas, saídas e cargos para as atividades. Através do *Cooperative Procedure Editor* podem ser modelados os processos detalhados de cada atividade utilizando Redes de Petri. E o *Work context Editor* permite que sejam definidas as janelas onde os agentes executam suas atividades.

O ambiente permite delegação de tarefas através da ferramenta Agenda. Quando uma tarefa (ícone) é selecionada na Agenda, os detalhes da tarefa são mostrados na janela *Work Context*. Essas duas janelas provêem uma visão do usuário final. O desenvolvedor de processo é suportado pelo *Activity Editor*, *Cooperative Procedure Editor* e *Method Editor*.

Quando um agente termina uma tarefa, pode pressionar o botão *Done* na janela *Work Context*. Isto causa uma mudança de estado a qual é detectada pelo ambiente permitindo o encadeamento do processo.

Em *ProcessWeaver*, os modelos de processo são desenvolvidos através de uma notação gráfica que permite a decomposição de atividades de alto nível e de processos de baixo nível (através do *Cooperative Procedure Editor*). Estas visões gráficas também são suportadas por informações textuais armazenadas em formulários. Além disso, existe a possibilidade de armazenar processos para reutilização.

A execução do processo de software em *ProcessWeaver* inicia a partir da instanciação do processo. Entretanto, o ambiente não provê mecanismos para permitir modificação dinâmica do processo (durante a execução), nem suporta o registro da história do processo e coleta de métricas.

O OPERA [ALO99] é um sistema de suporte a processos genérico que pode ser usado em uma variedade de aplicações como comércio eletrônico, gerenciamento de dados genéticos e sistemas de informações geográficas. Foi desenvolvido seguindo a idéia de que as diversas tecnologias estão convergindo e que os conceitos de processos estão cada vez mais genéricos. O OPERA é uma ferramenta de suporte a processo, com um *kernel* que fornece suporte a funcionalidade do processo [ALO97]. Isso não significa que ele seja um sistema completo e sim uma plataforma sobre a qual o sistema de suporte a processos atuais pode ser construído. Como um conceito de *kernel*, necessita ser extendido e adaptado para ser usado em aplicações concretas, implementando funcionalmente o que as futuras aplicações serão.

A arquitetura do OPERA é organizada em torno de três camadas de serviços: serviços de banco de dados, serviços de processos e serviços de interface. Cada uma dessas camadas representa um aspecto básico da funcionalidade que o OPERA fornece: persistência, execução e desenvolvimento.

A **camada de serviço de banco de dados**, contém a camada de armazenamento usada para persistência e a camada de abstração de banco de dados. A camada de abstração de banco de dados implementa o mecanismo necessário para tornar o sistema de banco de dados independente, tal que uma variedade de SGBDs (Sistemas de Gerencia de Banco de Dados) possam ser utilizados. A **camada de serviços de processo** contém todos os componentes requeridos para coordenar e monitorar a

execução de processos. Por fim, a **camada de serviços de interface** garante todos os mecanismos que permitem ao OPERA interagir com diferentes plataformas de hardware e software. Eles são implementados em uma CASE por uma base, toda vez que o OPERA é utilizado em um ambiente diferente. Também inclui as interfaces dos usuários para definição e monitoramento do processo.

Além desses, cita-se como importantes ambientes de processo: Adele [BEL91], Merlin [EMM91], Oikos [AMB90] e SPADE [BAN92].

3 Simulação de Processos de Software

Simulação é um processo de imitar a realidade com modelos. Tais modelos podem conservar ou não as características físicas ou lógicas do sistema real imitado. No caso em que as características são conservadas, há um processo de miniaturização ou representação parcial. Quando o modelo não conserva as características, teremos o caso da simulação simbólica. Nesse caso, a parte lógica do sistema real é conservada e expressa através de várias equações matemáticas, onde as variáveis representam os componentes do sistema [SHI75].

Neste trabalho, será estudada a aplicação a simulação, no contexto dos ambientes de desenvolvimento de software, como uma ferramenta de auxílio a validação de modelos de processo de software e definição de estimativas de projeto. Nesse caso, simulação pode ser vista como uma eficiente ferramenta de comunicação, para mostrar como os processos trabalham, enquanto estimula a criatividade do gerente para pensar em como podem ser melhorados [MAD99].

A simulação é normalmente utilizada em planejamentos ou antes da implantação de políticas de desenvolvimento de grandes protótipos, testes ou implementações em um ambiente operacional. Ela é utilizada para representar em detalhes as consequências e implicações de um curso de ações propostas [RUS98].

O trabalho [SHI75] apresenta as principais atividades em que a simulação é utilizada:

- Experimentação e avaliação, para prever as consequências de mudança de políticas, condições ou métodos, sem ter de realiza-las no sistema real, com grandes gastos e riscos de se obter resultados inesperados;
- Estudo de novos sistemas, a fim de reprojetá-los ou refiná-los;
- Ferramenta para familiarizar equipes com equipamentos ou sistemas ainda não operacionais;
- Estudo de processos transitórios ou intermediários, pois, a simulação permite o estudo de estados transitórios ou intermediários em qualquer ponto que se desejar.
- Verificação ou demonstração de uma nova idéia, sistema ou maneira de resolução de um problema;
- Como meio de projeção no futuro, isto é, como ferramenta de previsão e planejamento quantitativo.

Existem muitos benefícios práticos em se realizar simulação em ambientes organizacionais. Em planejamentos de projetos individuais, simulação pode ajudar na avaliação de investimentos com retornos demorados e de tecnologias estratégicas. Ela pode suportar aprendizado organizacional, tornando modelos explícitos em um ambiente de grupo, onde todos os participantes podem contribuir com o modelo. [MAD99]

3.1 Simulação Baseada em Computador

Com o advento da informática, a construção de modelos para serem simulados em um ambiente computacional também cresceu. Os modelos baseados em computador passaram a ser de fundamental importância, onde eles descrevem, explicam e predizem o comportamento de um sistema real.

A utilização de um simulador de sistemas baseado em computador permite realizar experimentos no modelo de sistemas utilizado, para saber qual pode ser a melhor política a ser adotada. O computador torna o trabalho mais dinâmico, permitindo a definição e realização de uma grande quantidade de experimentos.

Dentro da simulação baseada em computador existem duas áreas distintas: a simulação continua e a simulação discreta. **Simulação continua** descreve sistemas por conjuntos de equações para serem resolvidas numericamente, de forma algébrica ou diferencial, onde geralmente existe o tempo como uma variável independente. Já a **simulação discreta** descreve um sistema em termos de relações lógicas, que causam alterações de estado em pontos discretos, ao invés de atuar continuamente sobre o tempo. [RUS98].

3.2 Modelos de Simulação

Os modelos de simulação são abstrações do sistema real ou do sistema conceitual e são usados como base para experimentação de baixo custo [MAD99]. A simulação é fortemente dependente dos modelos construídos, como abstração de um sistema real, por isso a qualidade do modelo é essencial para a qualidade da simulação.

Usando um modelo de simulação como um veículo experimental é importante para a comunidade de Engenharia de Software. Nessa área é fácil propor hipóteses e difícil testá-las. No modelo do sistema, o efeito de alterar um fator pode ser observado enquanto todos os outros fatores permanecem inalterados. Tal experimentação revelará novas informações dentro das características do sistema que o modelo representa [RUS98].

De acordo com [MEN99] modelos de simulação são geralmente utilizados para otimização, isto é, ao invés de executar o modelo uma vez para obter resposta, o modelador realiza múltiplas execuções, ajustando parâmetros de entrada entre repetições, em um esforço para produzir resultados que podem ser julgados de acordo com critérios especificados. Assim, seguindo a terminação da execução de uma simulação, o modelador frequentemente estudará os dados de saída da simulação que caracterizam a execução, tentando identificar os fatores que limitaram a *performance* observada do sistema, para fazer ajustes que podem melhorá-la.

Vários mecanismos razoáveis podem ajudar o modelador nessa tarefa. Ele pode analisar os dados de saída e sugerir possíveis causas de gargalos para considerações do modelador. Para auxiliar nessa tarefa existem mecanismos automatizados baseados em heurísticas definidas em [LAW84], por exemplo. Esses mecanismos podem também sugerir reconfigurações que podem liberar os gargalos. Os parâmetros a serem alterados e o grau no qual os valores precisam ser alterados precisam ser identificados para a próxima simulação rodar.

Dados de uma série de simulações podem ajudar o modelador a entender a

estrutura do comportamento do sistema. O modelador pode avaliar as características comportamentais do sistema, dessa forma encontrar a melhor configuração do sistema obtendo aumento de *performance*. Um conjunto bem planejado de execuções, exigirá consideravelmente mais informações que um igual número de execuções realizadas com alterações de parâmetros *ad-hoc*.

De acordo com [SCH96] fazem parte de um modelo de execução a definição dos experimentos, das replicações e execuções. Sendo que um projeto de simulação é composto de experimentos. Um experimento é diferenciado dos demais pelo uso de alternativas na lógica e/ou nos dados de um modelo. Uma parte alternativa em uma sequência de regras pode ser experimentada ou a quantidade de recursos disponíveis pode ser variada.

Durante a execução o *clock* da simulação (um dado armazenado e gerenciado internamente) representa a passagem do tempo. Em uma simulação contínua o *clock* é incrementado independente dos acontecimentos da simulação, na simulação discreta, após todas ações possíveis terem sido realizadas, em um dado tempo simulado, o *clock* é avançado para o mais próximo evento escalonado.

A figura 3.1 representa um possível modelo de simulação. O esquema representa a inicialização do ambiente e depois ocorre uma iteração onde os eventos são chamados sequancialmente e o controle é feito pela variável “I”. O último evento do esquema imprime o sumário dos eventos.

3.3 Simulação Discreta

Simulação discreta examina problemas nos quais o ordenamento e o tempo dos eventos é o principal foco de interesse. Nesses sistemas os eventos válidos estão relacionado ao tempo no qual alguma atividade inicia ou termina. Por exemplo, simulando uma rede de computadores para estimar a capacidade efetiva do sistema ou tamanho das filas, podemos estar interessados no tempo inicial e duração do processamento de um trabalho, ao invés de detalhes da transmissão de sinais através da rede. Nesses problemas não é eficiente avançar o tempo em pequenos passos fixados e sim avançar o tempo para o próximo evento de interesse. Em geral, eventos podem ocorrer em qualquer tempo, o avanço do tempo não é uniforme e pode ser alternadamente grande ou pequeno.

Para [SCH96] a natureza da simulação evento-discreta é a de que o estado do modelo altere somente em conjuntos de pontos discretos, mas possivelmente randômicos. Duas ou mais unidades de tráfego (transações), sempre serão manipuladas em um mesmo ponto de tempo. Tais movimentos "simultâneos" de tráfego em um ponto de tempo é possibilitado pela manipulação de unidades de tráfego serialmente naquele ponto. Isso sempre permitirá o tratamento de complexidades lógicas em simulação evento-discreta.

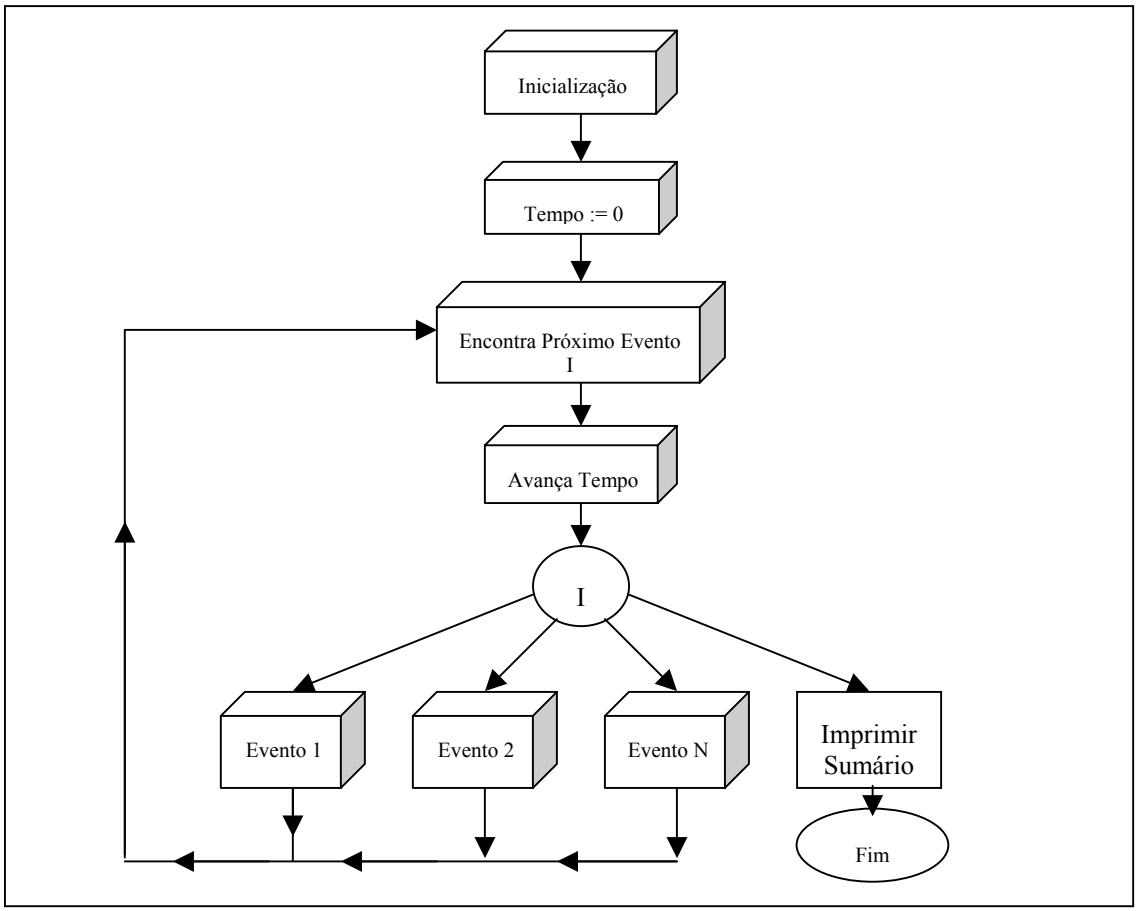

FIGURA 3.1 - Esquema de simulação [DAV79]

Entende-se por unidades de tráfego a visão básica da simulação evento-discreta. Nessa visão, um sistema consiste de unidades discretas de tráfego que movem-se de um ponto para o outro no sistema enquanto competem com cada outra pelo uso de recursos.

Relações lógicas são usadas para descrever a dinâmica do sistema e sua evolução entre eventos. Os resultados são obtidos medindo-se os tempos entre eventos e compilando-se estatísticas quantitativas de atividades ou inatividades das entidades envolvidas.

Usar simulação evento-discreta especializa o processo para encontrar alternativas de otimização. Similarmente quando frequência repetitiva de processos está sendo estudada, simulação evento-discreta fornece uma base para avaliar e validar a replicabilidade de processo, para uma experiência atual. Ela pode ser utilizada quando dados e eventos puderem ser medidos ou capturados empiricamente.

Dentro da simulação discreta, os conceitos de tempo e estado são de fundamental importância. [NAN81] identifica as seguintes primitivas que permitem realizar um relacionamento entre esses conceitos:

- Um **instante** é um valor de tempo do sistema, no qual o valor de ao menos um atributo de um objeto pode ser alterado.
- Um **intervalo** é a duração entre dois sucessivos instantes.

- Um ***span*** é a sucessão contígua de um ou mais intervalos.
- O **estado de um objeto** é a enumeração de todos os valores de atributos do objeto em um instante particular.

Essas definições fornecem a base de alguns conceitos de simulação largamente utilizados em [PAG94]:

- Uma **atividade** é um estado de um objeto sobre um intervalo.
- Um **evento** é a alteração no estado do objeto.
- Uma **atividade objeto** é o estado do objeto entre dois eventos, descrevendo sucessivas alterações de estado para aquele objeto.
- Um **processo** é a sucessão de estados de um objeto sobre um *span*.

Evento, atividade e processo por sua vez são a base dos três conceitos primários dentro da simulação evento-discreta:

- Em uma representação baseada em eventos, o modelador identifica **quando** ocorrem ações no modelo.
- Na representação baseada em atividades, o modelador identifica **por que** ações ocorrem no modelo.
- Na representação baseada em processos, o modelador identifica os componentes de um modelo e descreve a seqüência de ações de cada um.

3.4 Simulação e a Inteligência Artificial

A simulação, sendo uma tarefa que busca representar conceitualmente um sistema, precisa tratar com uma grande quantidade de informações. Em sistemas muito complexos, uma representação exata de todas as possibilidades de acontecimentos pode tornar-se inviável, principalmente em ambientes onde existe uma forte interação humana.

Cientistas e engenheiros trabalham rotineiramente com problemas que podem somente ser endereçados com a ajuda de modelos e simulação de computador. Com o advento de sistemas especialistas e tecnologia de IA (Inteligência Artificial), a necessidade de complementar essas técnicas com modelos e simulações (quantitativo e qualitativo) durante o processo de resolução do problema é largamente reconhecida. [MEN99]

A representação do comportamento de pessoas e grupos tornou-se essencial em algumas partes da comunidade de simulação. Técnicas desenvolvidas sobre a base da inteligência artificial, como a modelagem cognitiva podem resolver alguns dos problemas da área de simulação. Modelos de simulação tem incluído máquinas de estados finitos, sistemas especialistas, redes neurais, ambientes baseados em raciocínio e algoritmos genéticos, de forma a representar o comportamento humano presente nos sistemas com maior fidelidade e realismo.

Com isso, conceitos de inteligência artificial podem ser utilizados para tornar a representação das alternativas do sistema possível, através da representação de heurísticas observadas no ambiente real, possibilitando a obtenção de modelos próximos a realidade, onde informações relevantes são mantidas, fornecendo uma base para a representação dos componentes ativos do ambiente.

3.4.1 Sistemas Baseados em Conhecimento

Para [VIC91], o conceito de conhecimento é muito mais abrangente do que a noção de informação usada na programação convencional. O conhecimento explora as relações entre as porções de dados, atribuindo-lhes forte carga semântica. Pode aparecer sob a forma de objetos, asserções e definições, conceitos e relações, teoremas e regras, estratégias e táticas, e sob a forma de meta-conhecimento.

Os sistemas baseados em conhecimento são utilizados para implementar comportamentos inteligentes de especialistas humanos. O conhecimento do especialista deve ser coletado e interpretado pelo engenheiro do conhecimento, para facilitar o entendimento e garantir uma boa modelagem do sistema. Em uma simulação, o conhecimento pode ser usado para representar o comportamento dos componentes envolvidos no ambiente que está sendo simulado.

Para [MEN99], existem ao menos duas similaridades entre simulação e métodos de sistemas baseados em conhecimento. Primeiro, eles são baseados em uma representação modular de um sistema, com um mecanismo de inferência que dirige essa representação. Exemplos de módulos são: blocos e eventos na simulação e *frames* e regras em algum sistema baseado em conhecimento. Um mecanismo de inferência trabalha com qualquer número de módulos, os quais são apresentados em qualquer ordem e totalmente independentes de outros módulos. Em uma simulação baseada em eventos, por exemplo, o mecanismo de inferência é independente do número de eventos e sua ordem textual dentro da simulação. Isso é similar para um sistema baseado em conhecimento empregando regras de produção, que podem ser qualquer número de regras de produção, em qualquer ordem na base de conhecimento. Existe uma semelhança entre eventos condicionais na simulação e as regras de produção no sistema baseado em conhecimento baseado em regras.

3.4.2 Formas de Representação do Conhecimento

A base de conhecimento fornece o suporte para o funcionamento de um sistema baseado em conhecimento. Ela é utilizada para estabelecer conceitos e relacionar esses conceitos a fatos. É a representação simbólica do conhecimento a respeito de um domínio.

Os principais exemplos de forma de representação de uma base de conhecimento são: regras de produção, redes semânticas e *frames*.

A regra de produção é a representação de um domínio específico do conhecimento, na forma de um conjunto de regras (se...então...) baseadas na lógica. Essas regras formam a base do sistema, o mecanismo de inferência pode ser realizado por *forward chaining* (parte das evidências para chegar na conclusão, constituindo uma busca por dados) *e backward chaining* (não ocorre uma busca por dados, mas por objetivos e sub-objetivos). O objetivo é fazer com que a máquina seja capaz de realizar

deduções a partir das regras que lhe foram previamente fornecidas (é uma das formas mais conhecidas de representação do conhecimento para sistemas de pequeno porte).

Os *frames* são conjuntos de campos e de valores, cuja estrutura contém informações referentes a determinados objetos, os quais são armazenadas de forma hierárquica. Os *frames* podem ser criados a partir de uma coleção de *frames* (já existentes) conectados entre si. Isto se torna possível, uma vez que um atributo de um frame pode ser outro frame. De acordo com [RIC93], um frame é uma coleção de atributos, em geral chamados de *slots* e valores a eles associados (e possivelmente restrições a estes valores) que descrevem alguma entidade do mundo real. Um exemplo de uma base de conhecimento utilizando uma estrutura de frames é apresentado na figura 3.2.

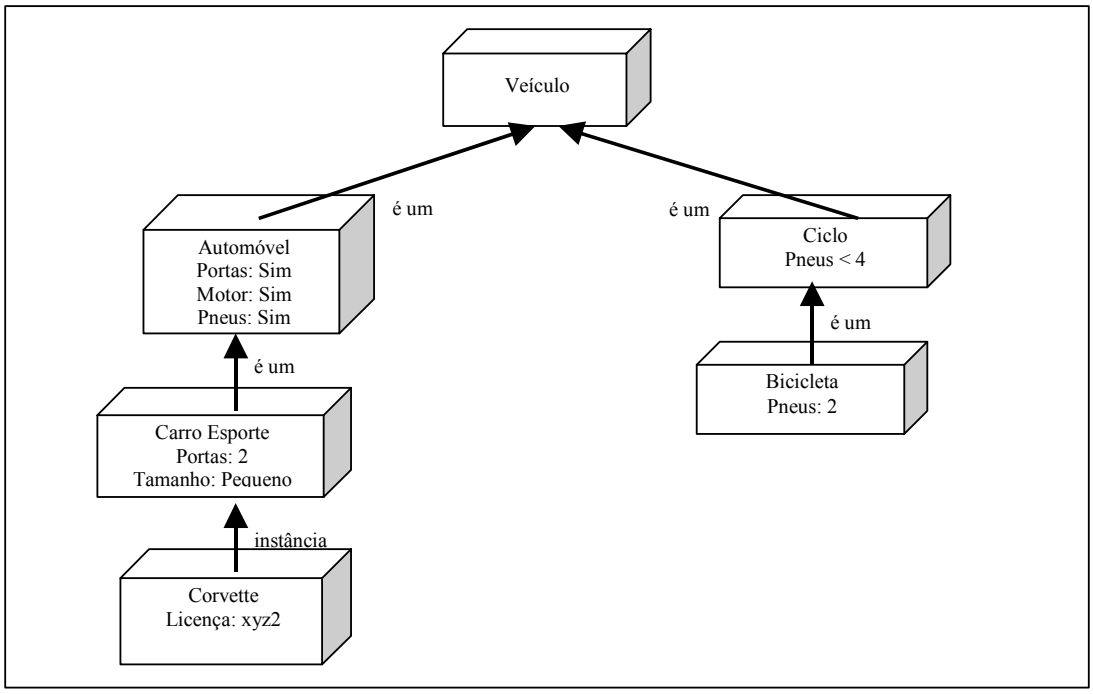

FIGURA 3.2 - Exemplo de Frame [BIG98]

As redes semânticas são representações gráficas do conhecimento. Como uma árvore de decisão, elas consistem em nós e arcos. Os nós contém informações, e os arcos mostram a relação entre os nós. A figura 3.3 mostra um exemplo de representação de conhecimento através das redes semânticas.

3.4.3 Sistemas Multiagentes

Os sistemas multiagentes são parte da Inteligência Artificial Distribuída (IAD), que se tornou nos últimos anos um domínio de pesquisa bastante promissor [ALV97]. Enquanto a IA clássica usa como modelo de inteligência o comportamento individual humano, o modelo de inteligência em IAD é baseado no comportamento social. Uma abordagem deste tipo torna-se desejável para resolver problemas grandes e complexos, que requeiram conhecimento de vários domínios de conhecimento distintos e que algumas vezes envolvam coleta de dados fisicamente distribuídos.

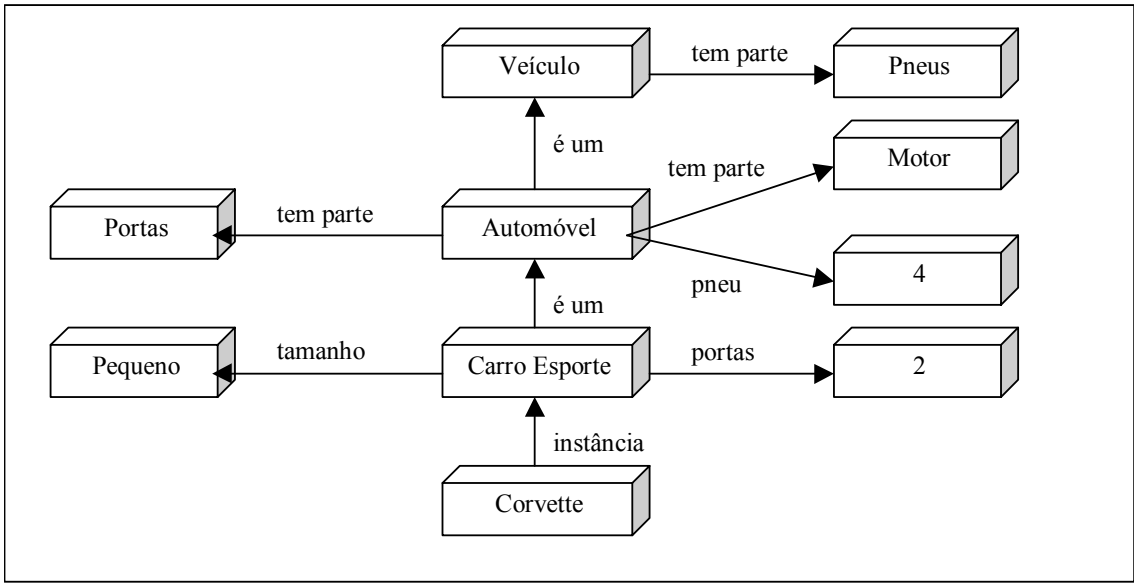

FIGURA 3.3 - Exemplo de redes semânticas [BIG98]

Sistemas multiagentes podem ser utilizados na área de simulação, na representação do comportamento autônomo dos componentes envolvidos em um ambiente, e é possível representar também a interação entre os componentes envolvidos.

Os **agentes** são as entidades ativas. O conjunto de agente forma uma **sociedade**. As entidades passivas são designadas pelo termo **ambiente**. Um agente raciocina sobre o ambiente, sobre os outros agentes e decide racionalmente qual objetivo deve perseguir, quais ações deve tomar, etc. Um agente é uma entidade real ou virtual; está em um ambiente; percebe o ambiente; é capaz de agir no ambiente; e pode se comunicar com outros agentes.

São características dos agentes [RET99]:

- Racionalidade - Os agentes são capazes de justificar suas decisões. Eles sabem escolher o melhor caminho para atingir seu objetivo.
- Intencionalidade e Iniciativa - Os agentes devem ser guiados por suas intenções na resolução de problemas. A intencionalidade é a expressão dos objetivos e dos meios para realiza-los.
- Engajamento - Os agentes devem ser compromissados, ou seja, engajados na realização dos objetivos.
- Autonomia - Os agentes tentam manter a autonomia na resolução de tarefas e não dependem (totalmente) uns dos outros.
- Adaptabilidade - Os agentes devem ser capazes de se adaptar aos conhecimentos novos, à forma de trabalho, às mudanças do meio e às interações estabelecidas entre os agentes.

Os sistemas multiagentes (parte da IAD) dividem-se em duas classes principais: Agentes **Cognitivos** e Agentes **Reativos**. A abordagem reativa [BRO86]

baseia-se na idéia de que agentes com ações elementares podem realizar trabalhos complexos. Um exemplo clássico é a colônia de formigas, cujo trabalho individual não é inteligente, mas o resultado é bastante complexo. Segundo [ALV97], nos sistemas multiagentes reativos não há representação explícita do conhecimento nem do ambiente; não há memória das ações; a organização é etológica, ou seja, similar a dos animais; e existe grande número de membros.

Os sistemas multiagentes cognitivos são baseados em modelos organizacionais humanos, como grupos, hierarquias e mercados. Nestes sistemas os agentes mantêm uma representação explícita de seu ambiente e dos outros agentes da sociedade; podem manter um histórico das interações e ações passadas; a comunicação entre os agentes é direta, através de mensagens; seu mecanismo de controle é deliberativo, ou seja, os agentes raciocinam e decidem seus objetivos, planos e ações; seu modelo de organização é sociológico e uma sociedade contém poucos agentes [ALV97].

Apesar da classificação, sistemas multiagentes podem não ser totalmente cognitivos ou reativos. Um sistema pode ser uma mistura dos dois para atender a solução de um problema. Os agentes inteligentes (como também são chamados na literatura) podem ser construídos e manipulados através das mesmas técnicas de representação do conhecimento, raciocínio e aprendizado da IA.

3.5 Simulação Baseada em Conhecimento

O desenvolvimento de sistemas de simulação baseados em conhecimento, resulta da combinação de simulação com tecnologias de sistemas baseados em conhecimento. O princípio é suprir o software com a maior quantidade possível de conhecimento e experiência sobre o domínio do problema [MEN99].

A simulação assume diversas formas, cada qual com características específicas de acordo com o tipo de problema que é tratado. No caso da simulação baseada em conhecimento, a intenção é identificar e utilizar padrões de comportamento de um sistema. Isso vai ajudar a simulação a representar o sistema mais fielmente.

Simulação baseada em conhecimento pode avaliar uma classe de modelos de processo ou um modelo de processo sem instâncias, e pode realizar "raciocínio" computacional na forma de inferência direcionada a padrões que podem ser implementados via regras de produção. Essas regras podem monitorar o valor dos objetos, atributos e relações armazenados numa base de conhecimento. [SCA98]

Quando uma ou mais regras estão habilitadas a disparar, o procedimento computacional definido na ação da regra é realizado e o estado do processo baseado em conhecimento é atualizado. O conhecimento representado pode ser inferido para definir ações em uma simulação baseada em conhecimento.

Em uma simulação baseada em conhecimento o estado dos processos pode ser explicitamente declarado ou pode ser representado implicitamente. Quando acontece representação implícita significa que o estado dos processos correspondem a valores que fazem parte de uma base de conhecimento que fornece suporte a simulação.

Para [MEN99] simulação baseada em conhecimento é uma possível forma de se obter muitas das características e muito da funcionalidade que cientistas e engenheiros buscam em simulação e ambientes de suporte a resolução de problemas. Cada simulação baseada em conhecimento é desenvolvida para satisfazer requisitos

específicos do usuário, em geral esses requisitos são colocados dentro de um ou mais dos apresentados a seguir:

- Facilita construção da simulação e facilita utilização: O objetivo de muitos ambientes de simulação baseada em conhecimento é ajudar o usuário no desenvolvimento, no uso e na simulação. Tais sistemas, tipicamente, ajudam o usuário a escolher ou construir uma simulação ou modelar para reunir critérios específicos; conduzir estudos que usam simulações ou modelos; e/ou interpretar os resultados de simulações ou modelos;
- Fornece aumento de capacidades de simulação: Experiências tem mostrado que nem sempre é possível desenvolver um modelo matemático (algoritmo), que seja aplicado em todas as situações ou que completamente descreva o estado ou comportamento do processo ou evento.

Nem sempre é possível utilizar um modelo matemático, os motivos são:

1. Limitações nos dados (incompleto, inconsistente) do qual a simulação está sendo obtida.
2. Ausência de completo entendimento do mecanismo, afetando um processo ou evento para ser simulado.
3. Inabilidade para simular um processo ou evento com os recursos computacionais disponíveis.

4 Modelos de Simulação de Processos de Software

De acordo com [LUD96], processo de software é visto como tecnologia chave não apenas para se obter projetos de software com melhores prazos e custos, mas também na busca de produtos de software com melhor qualidade. Técnicas tradicionais de modelagem de processos de software, como descrito em [CUR92] buscam capturar a melhor representação do processo com a ajuda de notações formais. Uma descrição completa do processo pode sempre ser analisada e melhorada. Após a implementação das melhoras o ciclo é reiniciado. Uma forma alternativa a essa técnica tradicional é a utilização de simulação de processos de software.

O princípio básico é criar um modelo de desenvolvimento de processo de software que pode ser interpretado por um simulador. O usuário da ferramenta pode usar o simulador para controlar interativamente o processo, buscando o caminho do sucesso para um projeto. Na proposta de [DRA98], um modelo pode ser complementado por um cenário de projeto inicial. Assim projetos de software com seus possíveis recursos ou restrições podem ser simulados.

Nesse capítulo serão revistos alguns ambientes de simulação utilizados na área de processos de software, através de suas características e aplicações. Os ambientes apresentados são representativos na tecnologia de desenvolvimento de software e serviram de base na definição do modelo de simulação apresentado nesse trabalho.

4.1 Articulator

O Articulator é um ambiente de simulação baseado em conhecimento para estudar processos de software, esse sistema é descrito em [MI90] e [SCA98]. A ferramenta é para ser usada no entendimento do funcionamento atual de um sistema, através da análise de processos "as-is" (modelos simulados de processos de software baseado em algum ambiente organizacional) e também no entendimento de processos "to-be" (modelos simulados de processos novos ou reprojetados que pretende-se utilizar em um ambiente organizacional).

São utilizadas tecnologias para entender, validar e refinar modelos de processo de software. O ambiente trabalha também com o intuito de fornecer suporte para modelar e simular os processos "here-to-there", que são passos necessários para levar um modelo de processos de software "to-be" a se tornar "as-is".

4.1.1 Ambiente do Articulator

No Articulator é realizado armazenamento persistente de eventos simulados, que permitem a realização da simulação *forward* (deriva novos fatos a partir de uma base) e *backward* (responde se uma afirmação é verdadeira ou falsa baseado nos fatos) para qualquer evento específico ou atualização na base de conhecimento. Persistência no Articulator foi implementada usando facilidades do gerenciamento de dados

orientado a objetos fornecido no ambiente de conhecimento de suporte.

O ambiente Articulator foi um dos primeiros que na modelagem e simulação de processos organizacionais utilizou agentes autônomos. De acordo com [SCA98] a tecnologia de agentes ainda estava em um estágio conceitual quando o Articulator foi inicialmente desenvolvido (1988-1990). Agentes modelados individualmente representam pessoas trabalhando em processos no mundo real, que são capazes de resolver problemas individualmente ou coletivamente. A utilização de uma perspectiva centrada em agentes ajudou no entendimento completo dos processos em estudo.

4.1.2 Arquitetura do Articulator

O Articulator possui um meta-modelo de processos de software, uma linguagem baseada em objetos para especificar modelos de processo de software e um mecanismo de simulação automatizado. A arquitetura do Articulator, representada na figura 4.1, consiste de cinco subsistemas: a base de conhecimento, o simulador comportamental, o mecanismo de *query*, o gerenciador de instanciação e o gerenciador de aquisição de conhecimento.

A base de conhecimento implementa o meta-modelo do Articulator através de uma abordagem baseada em objetos. O meta-modelo consiste de uma janela de recursos e uma de situações, que é a representação do modelo de desenvolvimento de software. A linguagem de especificação de processos de software habilita usuários a definir de forma customizada modelos de processo de software baseado no meta-modelo.

O gerenciador de instanciação gerencia os relacionamentos entre o meta-modelo, o modelo de processo de software customizado e suas instâncias. Ele mantém todos esses relacionamentos de acordo com o tempo de criação e linhas de herança, retornando a instância correta quando requisitado. Diferente de bancos de dados convencionais, não existe ligação explícita entre o meta-modelo, o modelo de processo de software e suas instâncias. Todos podem ser manipulados e modificados como uma instância associada a uma instância antiga sempre que necessário.

O simulador comportamental controla a simulação de um dado modelo de processo de software e cria uma trajetória do processo sobre um período de desenvolvimento. Simulação comportamental é uma representação simbólica de um modelo de processo de software descrito na linguagem de especificação de processo de software. Mecanismos para realizar atividades de processo de software são implementados dentro do simulador comportamental. Em termos de representações de processos de software existem três que estão disponíveis no Articulator: o modelo de processo prescritivo, a trajetória simulada da prescrição e o histórico do desenvolvimento descritivo armazenado. Cada um serve para diferentes propósitos no Articulator, mas todos seguem a mesma representação.

O mecanismo de *query* suporta regras lógicas para vários tipos de questões dedutivas. Ajuda os usuários a acessarem informações de forma eficiente. A fonte de informações é a base de conhecimento, isto é, o meta-modelo, o modelo de processo de software e suas instâncias. Usando técnicas de herança de objetos e mecanismo de inferência *backward*, o mecanismo de *query* "raciocina" sobre informações e conhecimentos para determinar respostas para vários tipos de questões.

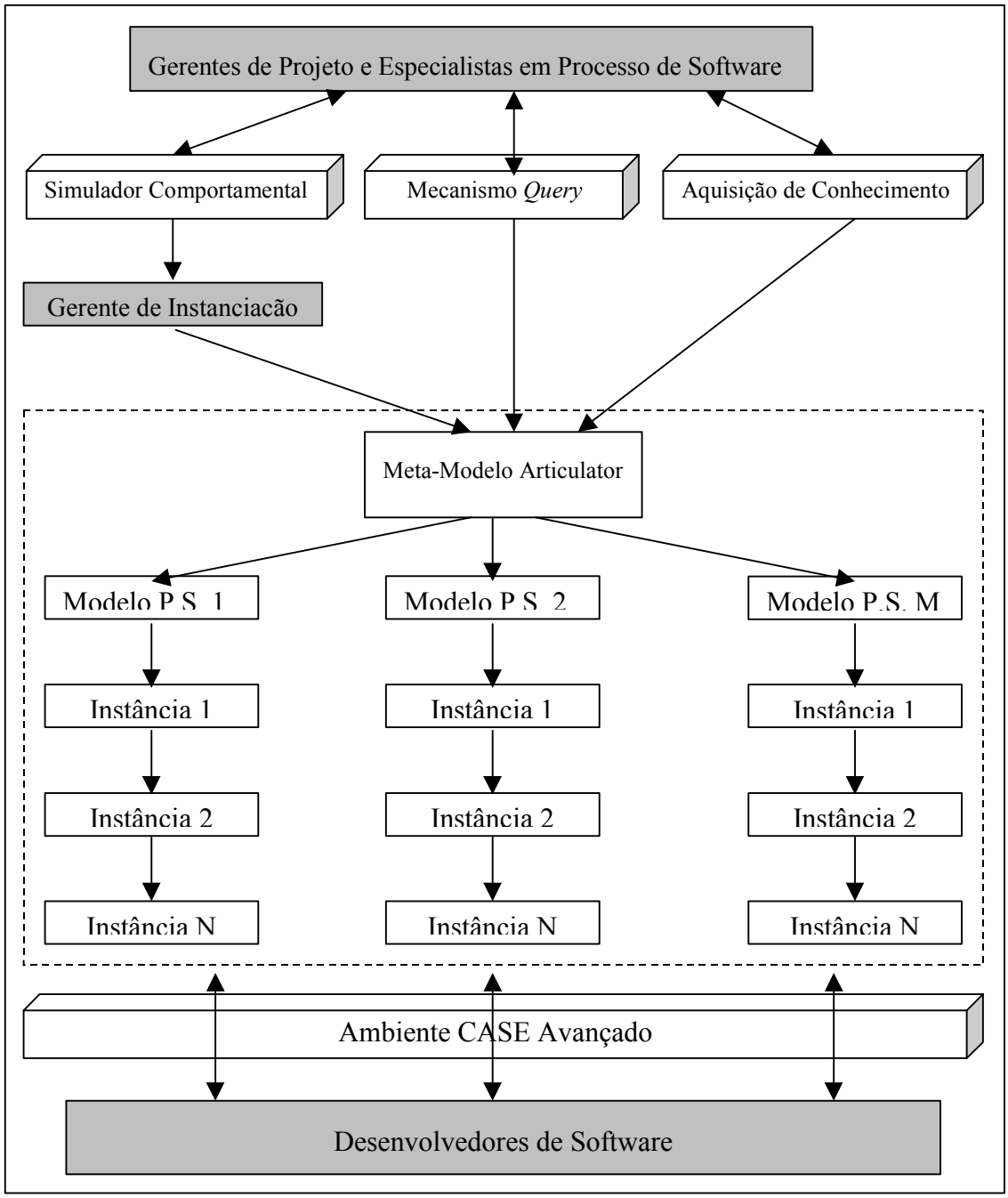

FIGURA 4.1 - Arquitetura do Articulator (adaptado de [MI90])

O gerenciador de aquisição de conhecimento é uma interface para o Articulator receber modelos de processo de software e dados associados. O gerenciamento de aquisição de conhecimento usa uma descrição associada de agentes, tarefas e recursos como entradas e transforma em um modelo de processo de software configurado. Ele também recebe informações de projetos de desenvolvimento de software controlado pelo Articulator e armazenados pelo usuário final.

4.1.3 Usuários do Articulator

Existe três categorias de usuários do Articulator: pesquisadores de processo, gerentes de projeto e desenvolvedores de software.

Os pesquisadores de processo de software estudam modelos de processo com o objetivo de identificar um que seja realmente eficiente, satisfaça diferentes requisitos de performance ou revele tópicos dinâmicos de processos de software, que necessitem estudos aprofundados. Esses usuários definem vários tipos de modelos de processo de software, eles testam usando simulação, comparam os resultados da simulação para observar históricos de desenvolvimento e refinam o modelo de acordo com certos critérios.

O gerente de projeto de software seleciona ou configura um modelo de processo existente, que representa suas necessidades de projeto e essencialmente fornece um guia para como realizar o projeto com sucesso. Gerentes usam o Articulator para ter acesso a modelos de processo de software, instanciá-los de acordo com suas situações de projeto local, simulá-los e refina-los com o objetivo de criar um plano para desenvolvimento e realiza-lo em seu próprio ambiente organizacional.

Desenvolvedores de software usam o Articulator através de um ambiente CASE. Dessa forma, o Articulator ajuda a coordenar suas atividades de desenvolvimento, de acordo com um modelo prescritivo e serve como um mecanismo de agenda ativa e um centro de troca de informações. Ao mesmo tempo, todas as atividades são armazenadas na base histórica do desenvolvimento, podem ser enviadas para gerentes permitindo o monitoramento do progresso do desenvolvimento.

4.2 SimObj - Ambiente de Simulação Orientado a Objetos

O trabalho [BOL93] apresenta um ambiente de simulação orientado a objetos. O ambiente fornece diferentes tipos de classes de objetos de simulação, incluindo simuladores contínuos, eventos-discreto e agentes baseados em conhecimento, cada qual sendo instanciado usando herança para tipos mais específicos de simulação.

Em uma linguagem de simulação orientada a objetos, os objetos de simulação representam entidades do mundo real, simuladas através do tempo. Esses objetos representam a unidade de simulação fundamental. A classe de simulação de mais alto-nível é denominada *SimObj*, os objetos tipicamente contém uma ou mais variáveis de estado, as quais são inicializadas para a simulação e então são atualizadas em resposta para um *Update*( ) no progresso da simulação. A classe *SimObj* fornece uma descrição abstrata, é responsável por manter dados, campos genéricos e trilhar resultados de simulação para todos os objetos de simulação derivados. *SimObj* fornece dados e comportamentos comum para todos os objetos de simulação e fornece as funcionalidades mais requeridas para interagir com o ambiente de simulação.

Classes de simulação definidas pelo usuário derivam da classe abstrata *SimObj*. O objeto de simulação possui dados e métodos como qualquer objeto, que são especificamente projetados para ajudar na simulação do objeto em si. Esses objetos de simulação, fornecem os dados e o comportamento específico para suas necessidades, enquanto a classe *SimObj* fornece dados e funcionalidade que é comum para todos objetos de simulação.

O ambiente permite avanço de tempo, tanto permitindo alterações contínuas na

variável tempo como baseada em eventos. Para realizar atualizações utiliza-se o método virtual *SimObj::State*( ), que empacota equações diferenciais necessárias para realizar a atualização e o método *SimObj::Update*( ), que é chamado se os objetos de simulação precisam ser atualizados. O método *SimObj::Update*( ) é utilizado nos dois tipos de simulação, o método *SimObj::State*( ) só é utilizado em simulações continuas.

A simulação inicia em algum ponto no tempo, normalmente ponto 0 (zero), inicializando o clock, inicializando a lista de estatísticas e inicializando a lista de eventos. O *clock* remove o primeiro item da lista de eventos, ajusta o *clock*, atualiza os eventos e repete o processo. Durante a atualização do sistema, novos eventos podem ser registrados. A simulação continuará até a lista de eventos tornar-se vazia ou alguma condição pré-especificada para parar a simulação ocorrer. A figura 4.2 representa esse esquema.

Para adicionar inteligência a um modelo de simulação é necessário apenas desenvolver objetos de simulação inteligentes que herdam capacidades da inteligência artificial. Esses objetos de simulação inteligentes (agentes), tipicamente sabem como responder questões e resolver problemas para outros objetos de simulação. Objetos de simulação podem interagir com agentes de quatro formas possíveis: mensagens em *broadcasting*, invocação direta de um método de objeto, envio para um quadro de avisos e observação do estado dos objetos pelos agentes.

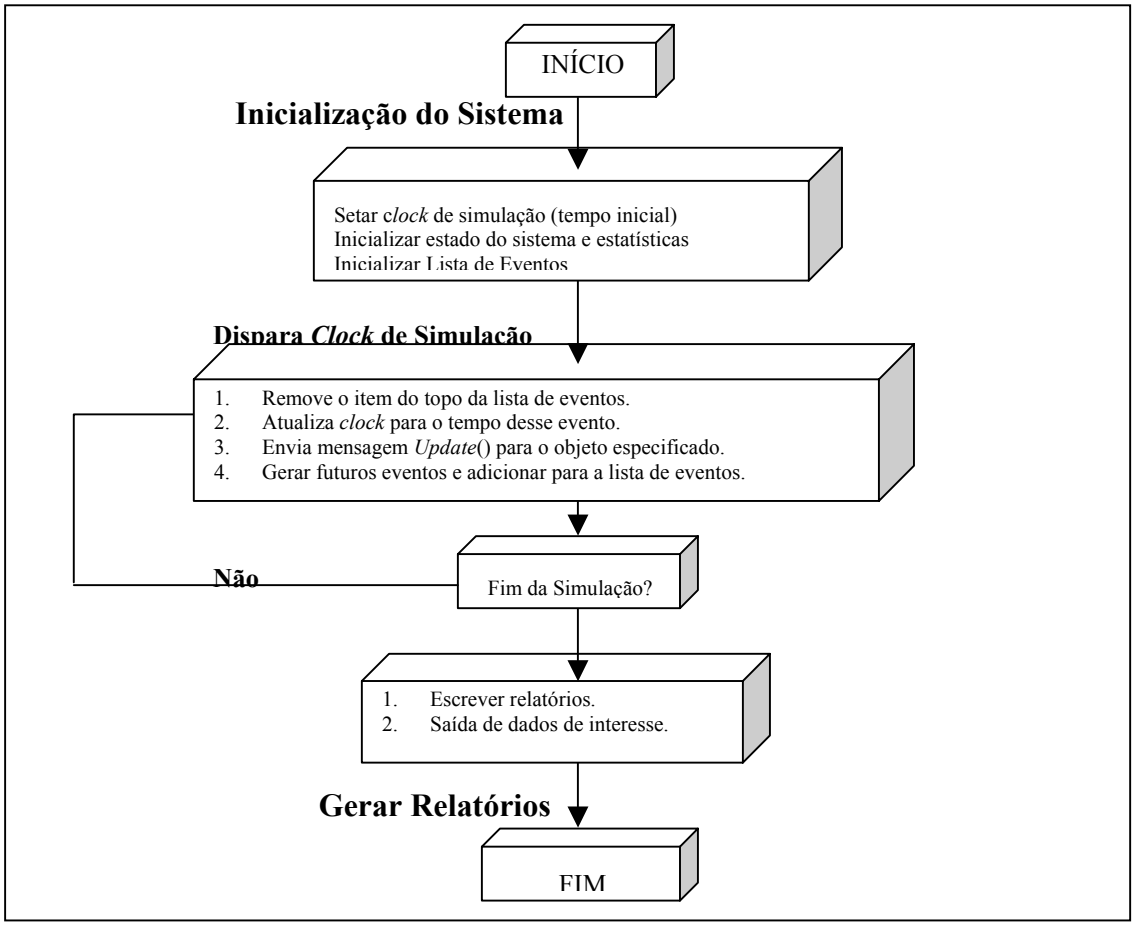

FIGURA 4.2 - Diagrama do fluxo de simulação de processo. [BOL93]

4.3 DSS - Sistema de Suporte a Decisão

O simulador de processos apresentado em [RUS98] é parte de um sistema de suporte a decisão para ajudar gerentes de projeto no planejamento ou aperfeiçoamento do processo de desenvolvimento de software, de uma forma dirigida a qualidade. O sistema de suporte a decisão, pode ser usado para planejamento de projeto, guia, monitoração e para predizer o efeito do gerenciamento e garantir decisões de engenharia. Pode também ser usado como ferramenta de treinamento para gerentes de projeto.

Para fins de simulação, um modelo precisa ser construído tal que ele reflita o processo como ele é e não uma visão idealizada do que o gerente gostaria que fosse. Assumindo que o modelo está corretamente construído para representar os processos reais, a simulação projetará o comportamento esperado do processo com o passar do tempo. No fim de um projeto, o modelo precisa representar o que aconteceu no processo, então será mais útil na próxima iteração do projeto ou na aplicação em um novo projeto na mesma organização.

Enquanto o modelo é útil para representar a estrutura do processo, a execução do modelo é útil no entendimento do comportamento do processo, permitindo, por exemplo, gerenciar uma ferramenta que tenha estado esperando por um longo tempo. As capacidades de simulação permitem a um gerente, tomar uma decisão e ter um alto grau de certeza no que resultará sua decisão.

O ambiente apresentado tem dois subsistemas: um sistema especialista para sugerir processos alternativos e um subsistema de simulação que permite ao usuário avaliar a alternativa selecionada, analisando as saídas do simulador.

O DSS (*Decision Support System*), mostrado na figura 4.3, é um conjunto de ferramentas baseada em computador, usadas por um gerente em conexão com sua resolução de problemas e tomada de decisão. O gerente toma as decisões e resolve problemas específicos.

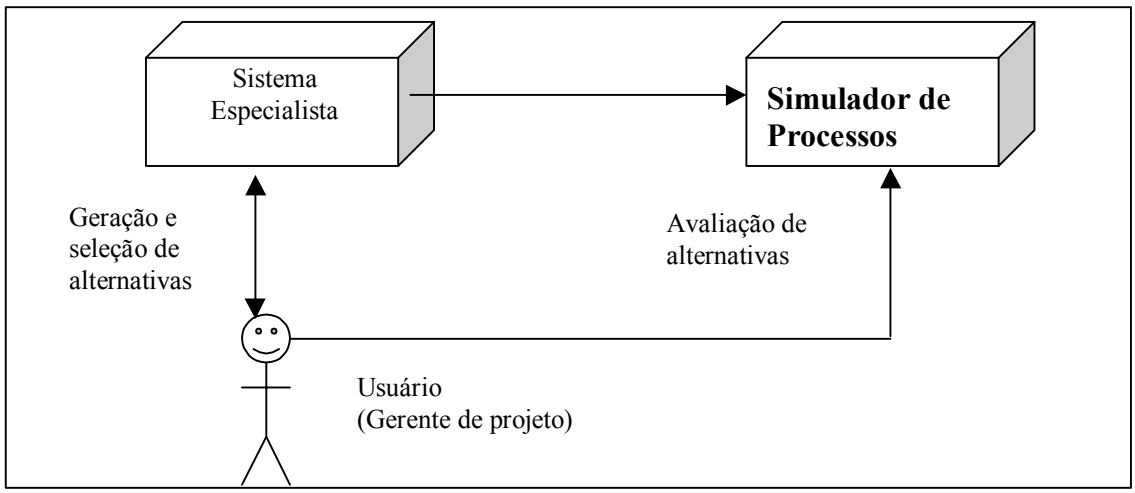

FIGURA 4.3 - Sistema de suporte a decisão. [RUS98]

No DSS, o modulo para gerar/sugerir alternativas, é um sistema especialista e o usuário fará a avaliação da decisão analisando a saída do simulador de processos. Selecionar uma estratégia alternativa e avaliá-la pode ser um processo iterativo, até os

usuários acreditarem terem encontrado a solução mais apropriada para seu projeto específico.

4.4 SESAM - Aprendizado colaborativo

O SESAM [LUD96] (Software Engineering by Simulation of Animated Models) foi desenvolvido na universidade de Stuttgart (Alemanha) como uma ferramenta para pesquisa de Engenharia de Software e educação, é um ambiente que suporta aprendizado colaborativo como um jogo entre o construtor do modelo e um jogador (gerente de sistemas, por exemplo). O SESAM serve como um canal através do qual um construtor do modelo e um jogador podem se comunicar ganhando entendimento e experiência sobre o modelo em estudo.

Usando o SESAM, o construtor do modelo e o jogador analisam a atividade de Engenharia de Software, construindo, validando e refinando um modelo de projeto de software através de um ciclo de 4 processos: 1) o construtor cria um modelo; 2) o jogador interage com a simulação baseada no modelo, quando ele experimenta dificuldades, realiza anotações e questionamentos; 3) o construtor analisa a execução da simulação armazenada; e 4) o construtor e o jogador examinam a execução conjuntamente.

Esse ambiente não assume que os modelos existentes sejam sempre apropriados, como acontece na maioria dos casos. O *feedback* fornecido pelo jogador é levado em consideração para realizar aperfeiçoamento no modelo.

O objetivo do jogador é produzir funcionalidade requerida e qualidade em um desenvolvimento no escalonamento das tarefas através da motivação dos membros do time. Essa motivação pode ser obtida assinalando os membros para tarefas adequadas e envolvendo o cliente na validação e evolução dos requisitos.

O sistema usa três tipos de representações para um modelo de simulação: o modelo de entidades e relacionamentos que define as entidades e relacionamentos do ambiente modelado; regras, que descrevem o comportamento dinâmico dos participantes do projeto simulado; e o modelo de situação, que instancia a simulação e representa o contexto para o jogador.

5 Modelo de Simulação *AgentProcess*

A tecnologia de simulação de processos de software pode ser de grande utilidade na validação de modelos de processo de software. A utilização de conceitos da inteligência artificial é útil na busca por modelos de simulação mais realísticos, que representem de forma aproximada o comportamento dos componentes de um ambiente de desenvolvimento.

As ferramentas tradicionais de simulação de processo de software descritas na literatura não tratam de forma efetiva o caráter cooperativo do processo de desenvolvimento, isto é, a possibilidade de existirem atividades que são realizadas por vários profissionais interagindo para desempenhar uma tarefa [MI90]. A partir disto, pode-se destacar nestas ferramentas algumas deficiências:

- A simulação não leva em consideração as aptidões e preferências dos profissionais existentes na organização de desenvolvimento de software;
- A granularidade dos eventos observados por estas ferramentas é grossa, ou seja, observa eventos que ocorrem somente no nível das atividades (com pouca ou nenhuma observação dos fenômenos que ocorrem no interior das atividades);
- Pouca assistência é fornecida ao gerente do desenvolvimento de software na seleção dos desenvolvedores mais indicados para ocupar cada uma das atividades do processo.

Nesse trabalho é definido um modelo de simulação de processo de software que procura levar em consideração aspectos como habilidade e afinidade entre desenvolvedores e utiliza uma base de conhecimento que influencia nos resultados obtidos com a simulação.

Esse modelo foi definido para ser inserido no ambiente PROSOFT. Para definir uma ferramenta do ambiente PROSOFT a ordem natural das ações seria defini-las através de ATOs algébricos, construções definidas em [NUN94], que serão apresentadas no capítulo 6, para depois, utilizando a ferramenta PROSOFT (a ser implementada em Java) gerar os ATOs PROSOFT, demonstrando os conceitos definidos nesse trabalho. No entanto, no momento ferramenta PROSOFT em Java ainda não esta disponível para uso. A alternativa encontrada foi a representação dos conceitos utilizando a UML (*Unified Model Language*) formando uma base de dados para a posterior implementação dos mesmos utilizando a linguagem Java.

O modelo de simulação apresentado nesse capítulo é denominado ***AgentProcess*** (Simulação de Processo de Software baseado em Agentes Cooperativos) [SIL99] e [SIL00]. Ele é baseado em conhecimento e utiliza agentes inteligentes para representar o comportamento dos desenvolvedores envolvidos em um projeto de desenvolvimento de software. O comportamento dos desenvolvedores é definido segundo um modelo de processos de software e as atividades são representadas através

da ocorrência de eventos. O algoritmo que representa o comportamento do simulador prevê consultas a uma base de conhecimento para tomada de decisão.

O modelo foi construído com o objetivo de contornar as deficiências apresentadas, com novas soluções para o problema da simulação de processo de software. Esse modelo foi desenvolvido a partir do modelo de processo de software apresentado em [LIM98] e [SIL99]. No modelo *AgentProcess* é definida a estrutura do modelo de processos e o relacionamento dos componentes do modelo (representados por classes) com o simulador de processos. O modelo representa componentes considerados importantes para a simulação, como as características de atividades, de habilidades e afinidades de desenvolvedores, que serão explicadas nesse capítulo, mais adiante.

A definição de um esquema de simulação para ser inserido em um ciclo de vida de um modelo de processo de software serve para a definição de uma ferramenta que seja utilizada para validação e refinamento do modelo de processo, que são atividades realizadas antes da execução do mesmo, podendo contribuir para o sucesso do PSEE em que estiver inserida. A simulação ajuda na detecção de inconsistências e falhas no comportamento do processo [MI90].

Nesse capítulo será apresentada a arquitetura do modelo e a descrição da definição UML do modelo através das classes definidas. Serão apresentados também os algoritmos considerados essenciais para o modelo.

5.1 Arquitetura do Modelo

O meta-processo apresentado na figura 1.1 apresenta a fase de simulação sendo realizada após a fase de instanciação de um modelo de processo, onde um modelo abstrato é instanciado de maneira que sejam definidos os componentes que atuarão no processo. A fase de simulação então recebe um modelo de processo instanciado e após a realização dessa simulação, caso o modelo seja considerado inconsistente, esse modelo pode ser reenviado para as fases de projeto ou de instanciação dependendo dos refinamentos obtidos pelo simulador.

Caso o modelo de processo instanciado remetido ao simulador seja considerado viável de ser executado ele então é submetido à fase de execução de processo com o *status* de modelo de processo validado.

A arquitetura levada em consideração para propor o modelo de simulação *AgentProcess* é apresentada na figura 5.1. Os componentes presentes na arquitetura são: o modelo de processo de software instanciado, a interface com a base de conhecimento, o mecanismo de simulação, a máquina de inferência, o módulo de representação de eventos e a base de conhecimento.

O **modelo de processo de software instanciado** é a entrada do simulador, sendo que as informações que estão definidas serão objeto da simulação. O projetista do sistema define as regras para representação do projeto que será simulado, sendo que em um modelo de processo são definidos os recursos, os desenvolvedores e o projeto, este último é composto de processos de software, métricas e estimativas. Para se modelar um processo são utilizadas atividades, ordem de dependência entre atividades, hierarquia de processos, desenvolvedores responsáveis por uma atividade e recursos necessários para a realização de uma atividade. Na definição de uma atividade, são utilizadas atividades padrão, adaptando-as as necessidades do projeto específico, ou podem ser

completamente definidas pelo usuário. A definição de projeto de software em [LIM98], contém regras para realização da modelagem de processo de software.

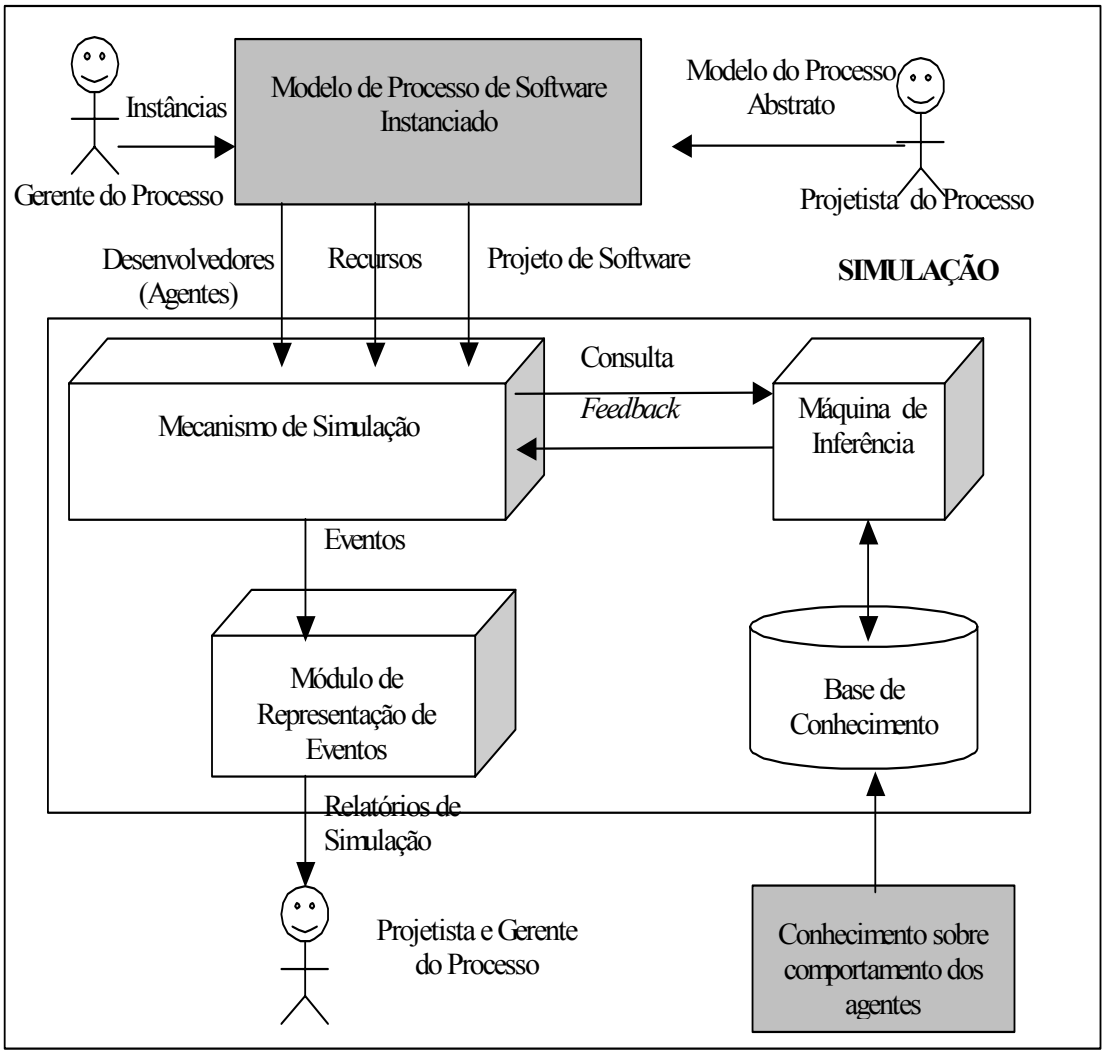

FIGURA 5.1 - Modelo de simulação de processos baseado em conhecimento

O **conhecimento sobre o comportamento dos agentes** permite que informações sobre o ambiente de desenvolvimento de software sejam armazenadas na base de conhecimento de forma estruturada. São utilizadas regras para a definição de comportamentos definidos como conhecimento. Esse conhecimento é útil na representação dos fatos que ocorrem no ambiente de desenvolvimento de software real em um ambiente simulado de forma mais aproximada. Alguns conhecimentos considerados importantes nesse ambiente de simulação são: grau de capacidade de um desenvolvedor em trabalhar com uma atividade que possua determinada característica; relação da produtividade de um desenvolvedor com o tamanho do grupo em que está inserido; relação do desempenho do desenvolvedor com a seqüência de atividades realizadas; grau de capacidade do desenvolvedor em exercer determinado cargo; dentre outras métricas possíveis.

O **mecanismo de simulação** é a parte da arquitetura onde a simulação propriamente dita ocorre. Recebe como entrada um projeto de software da interface de

definição de projeto de software, recursos e agentes definidos na interface de cadastro de entidades e utiliza a máquina de inferência quando informações armazenadas na base de conhecimento são importantes na tomada de alguma decisão. Para realizar a simulação são definidos ciclos para controle de tempo e de eventos que são obtidos do modelo de processo. Informações sobre a ocorrência dos eventos são enviadas para o módulo de representação de eventos.

A **máquina de inferência** recebe requisições do mecanismo de simulação e realiza buscas na base de conhecimento para obter uma resposta para a requisição recebida. A busca é realizada através do mecanismo denominado *forward chaining*, que parte de evidências (informações presentes na simulação) para chegar a uma conclusão, constituindo basicamente em uma busca por dados, que influenciarão o desenrolar da simulação.

O **módulo de representação de eventos** recebe do mecanismo de simulação os eventos gerados para representar a utilização do modelo de processo em um ambiente de desenvolvimento. Os eventos recebidos contêm as atividades realizadas, o nome dos desenvolvedores que as realizaram, os recursos usados ou consumidos e o tempo de ocorrência. Com essas informações disponíveis é possível representar a realização do processo e inferir as informações consideradas importantes na avaliação do modelo de processo de software e calcular o custo da realização do projeto. O custo calculado da realização do projeto pode ser comparado com o custo previsto, que é um parâmetro do projeto simulado, para verificar a viabilidade da aplicação do modelo. Para calcular os gastos com um projeto em uma simulação, são levados em consideração os gastos com a utilização de recursos e o custo por hora trabalhada de cada desenvolvedor envolvido no projeto. A validação dos prazos indicada nas atividades também é importante na avaliação do modelo. Os eventos de simulação representam todas as informações que podem ser obtidas da simulação. Essas informações são geradas pelo mecanismo de simulação e processados pelo módulo de representação de eventos. Isso é válido para o modelo, já que é necessário diferenciar as duas tarefas por questões de simplicidade, mas o mecanismo de simulação e a representação de eventos podem ser vistos como um módulo único devido ao grande fluxo de informações que existe entre eles.

Todos os desenvolvedores que participam da simulação e o próprio mecanismo de simulação são representados através de agentes inteligentes e cada agente inteligente presente no modelo possui uma estrutura de informações formando sua base de conhecimento própria. As bases de conhecimento são compostas de regras que podem ser compartilhados entre as bases existentes pertencente aos diferentes agentes.

5.2 Características do Modelo *AgentProcess*

O modelo de simulação de processos *AgentProcess* propõe uma forma de testar modelos de processos de software usando agentes inteligentes, que atuam no papel de desenvolvedores. Dado um modelo de processo instanciado (com atividades definidas, desenvolvedores e recursos alocados para as atividades, cronograma inicial, etc.), o modelo de simulação disponibiliza atividades para os agentes-desenvolvedores (através de uma agenda do agente), e estes utilizam seu conhecimento armazenado para executar as atividades nas quais eles estão envolvidos.

5.2.1 Projeto de Software

Existe uma grande quantidade de modelos que buscam representar os aspectos que envolvem a coordenação de um projeto de software. Para [PET99], planejamento de processo é mais complexo em domínios que requerem projetos de artefatos e planejamento de construção, especialmente quando os artefatos são grandes e muitas pessoas e ferramentas de software precisam ser coordenadas e gerenciadas. Para [HAD96] é necessário que as pessoas envolvidas e os agentes de software atuem sob um modelo de coordenação. É importante também que agentes, tanto humanos, como de software, compartilhem o entendimento sobre um protocolo comum de mensagens.

Projetos seguem planos sobre tarefas, suas durações e recursos, indica [PET99]. Um fator relevante no planejamento de processos é a definição do relacionamento de precedência entre as tarefas, que tradicionalmente são definidas no início do planejamento do projeto. Entretanto, normalmente o ajuste dinâmico de um plano influencia a alteração dessas precedências. A utilização de entradas e saídas de tarefas para definir os relacionamentos de dependência é sugerida por [PET99]. Nesse caso, se uma saída de uma tarefa é uma entrada de outra, existe ao menos uma relação de precedência entre as tarefas, em que a segunda tarefa não pode ser finalizada antes da primeira.

No *AgentProcess* existe uma classe que representa o projeto de software. A representação do projeto procura conter a maior quantidade possível de informações sobre o desenvolvimento de software que precisa ser realizado. No modelo, o projeto é composto basicamente por processos, métricas, estimativas e cargos. A figura 5.2 apresenta a parte do modelo *AgentProcess* que define projeto de software e seus relacionamentos.

O processo é composto por atividades, que por sua vez podem ser compostas por outras atividades, representadas pela classe AtividadeDecomposta, por atividades executáveis, representadas pela classe AtividadeFolha ou por atividades automáticas, que são realizadas sem intervenção humana e são representadas no modelo pela classe AtividadeAutomática. Mais informações sobre a representação de processos são encontrados na próxima seção.

A utilização de métricas e estimativas no modelo está diretamente relacionada com a avaliação da qualidade dos produtos e do processo, obtidos com a realização da simulação. Através dos dados obtidos será possível avaliar o impacto da simulação no processo. O simulador pode ser usado como ferramenta para estimar prazos e custos. Métrica de software refere-se à mensuração dos indicadores quantitativos do tamanho e complexidade de um sistema. Estes indicadores são, por sua vez, utilizados para relatar os desempenhos observados no passado a fim de derivar previsões de desempenho futuro. A métrica de software tem como princípios especificar as funções de coleta de dados de avaliação e desempenho, atribuir essas responsabilidades a toda equipe envolvida no projeto, reunir dados de desempenho pertencentes à complementação do software, analisar os históricos dos projetos anteriores, para determinar o efeito desses fatores e utilizar esses efeitos para pesar as previsões futuras. Estes princípios nos permitem prever o resto do processo, avaliar o progresso e reduzir a complexidade. As métricas podem ser basicamente orientadas a tamanho ou função.

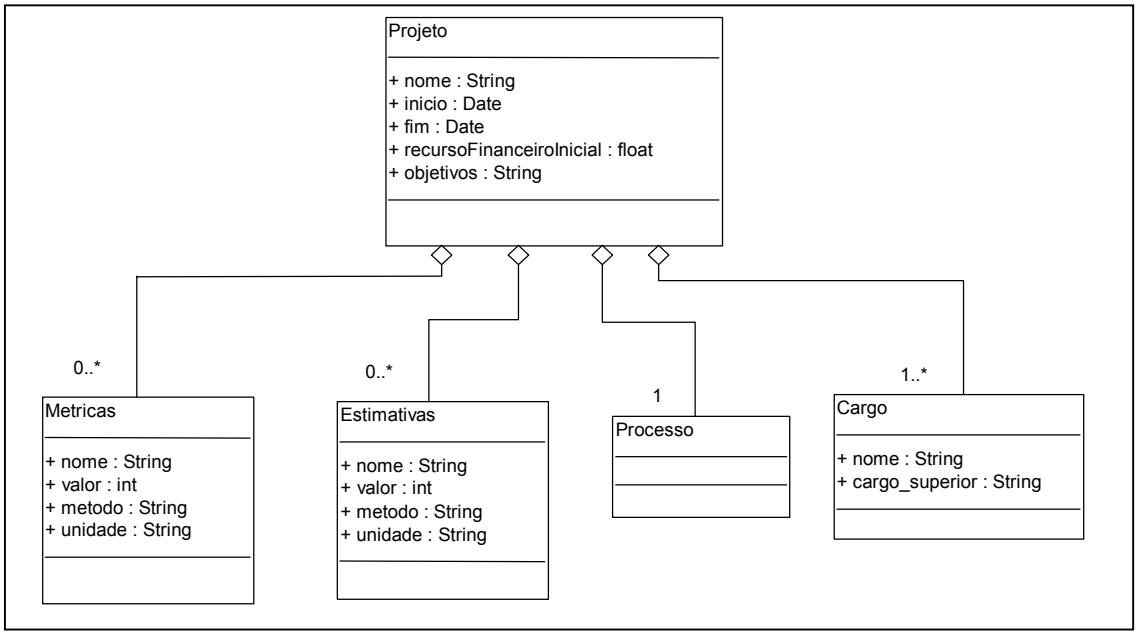

FIGURA 5.2 - *AgentProcess* – Projeto de software.

As estimativas normalmente são voltadas para prever quanto tempo uma equipe levará para concluir um projeto e quanto vai custar construí-lo. A estimativa de tempo normalmente leva em consideração a relação tempo/pessoas, observando que a relação nem sempre é direta, já que a duplicação de pessoas envolvidas não garante a redução do tempo de realização de um projeto à metade. O objetivo da estimativa de custo é calcular de maneira antecipada todo e qualquer custo que esteja associado ao sistema, tais como: construção, instalação, operação e manutenção.

A definição dos cargos do projeto permite a alocação de recursos humanos para trabalharem no projeto e define a hierarquia entre os envolvidos. Essas informações são utilizadas na simulação para prever que impacto tem a alocação de determinadas pessoas e o relacionamento entre essas pessoas no desenvolvimento do projeto.

5.2.2 Modelagem de Processos

Antes da simulação, o modelo de processo deve ser descrito. O modelo *AgentProcess* permite descrição de processo instanciado, ou seja, com desenvolvedores e recursos alocados bem como um cronograma previsto inicial. A descrição do processo é feita através da definição das atividades que compõem o mesmo. Um processo de software corresponde a um conjunto de atividades. Cada atividade é executada por um ou vários desenvolvedores (agentes), que lhe confere caráter de atividade cooperativa, necessita de recursos, possui cronograma previsto (estimativas) e cronograma real (dados simulados), possui um conjunto de atividades das quais ela depende, produz artefatos e possui um *script* onde é descrita a tarefa a ser realizada. O *script* pode ser um novo processo (hierarquia), uma atividade automática (executada por uma ferramenta sem intervenção de desenvolvedores) ou ainda uma descrição da tarefa a ser realizada. Toda atividade possui um estado que deve ser manipulado somente pelo simulador do processo. Os estados que uma atividade pode assumir são: **Esperando** (as anteriores

ainda não concluíram); **Pronta** (as anteriores já concluíram); **Ativa** (sendo realizada) e **Completa** (atividade concluída).

O simulador representa a realização de atividades calculando o tempo necessário para sua realização, baseado em dados históricos. Após a passagem de tempo necessária para a realização da atividade, ela é dada como encerrada. Caso mais de um desenvolvedor esteja trabalhando na atividade, a mesma só é dada como encerrada quando passar o tempo necessário para que todos os envolvidos completem suas tarefas. A dependência entre atividades é representada no modelo de processo através de setas. As atividades que não são destino de nenhuma seta estão prontas para serem executadas quando o processo é disparado. Aquelas que possuem atividades das quais são dependentes ficam esperando pela finalização destas para estarem prontas a iniciar.

A figura 5.3 representa a decomposição de uma atividade em novas atividades envolvidas num sub-processo, formando uma hierarquia de atividades. Quando isso ocorre, a realização da atividade é através da realização das suas atividades componentes, que seguem as mesmas regras de dependência das atividades de mais alto nível. No exemplo a atividade "a" é única que não é dependente (não é destino de nem uma seta), as atividades "b" e "c" são dependentes da atividade "a", enquanto a atividade "d" é dependente da atividade "b", as atividades "e" e "f" são dependentes de "c", "f" também depende de "d" e "g" é dependente de "f" e "e". A realização das atividades "a", "a.3" e "e" são realizadas através da realização dos processos de nível mais baixo (sub-processos).

As atividades utilizam objetos de software como entrada, esses objetos são consumidos pelas atividades e novos objetos podem ser produzidos durante a realização de uma atividade, os objetos produzidos na atividade são representados como parâmetros de saída da realização da atividade.

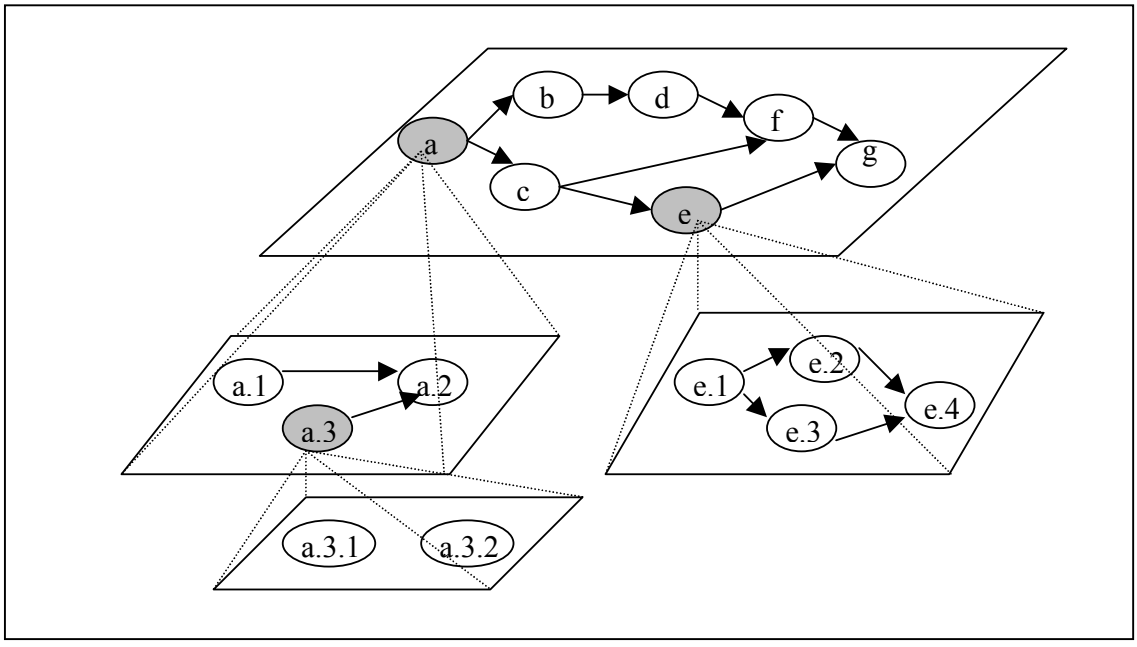

FIGURA 5.3 - Decomposição de Atividades de um processo de software [LIM98]

As classes são representadas na figura 5.4 e podem ser definidas assim:

- Atividade: A classe atividade representa as atividades que devem ser realizadas em um processo de desenvolvimento de software, possui como informações

nome, descrição e estado. Um auto-relacionamento indica as atividades anteriores, que são as atividades que devem ser executadas antes da atividade representada.

- Processo: É a classe que contém a lista de atividades que devem ser realizadas e é utilizada na decomposição de atividades compostas.
- Projeto: Representa os projetos que são simulados pelo simulador de processo.
- ObjetoSoftware: Possui como atributos o nome, o estado e a descrição dos objetos de software que são representados. Esses objetos são utilizados como produtos de entrada e produtos de saída.
- AtividadeAutomática: Representa atividades que são realizadas sem intervenção humana. Herda as características da classe Atividade.
- AtividadeDecomposta: São as atividades que são representadas por outras atividades (sub-processo).
- AtividadeFolha: Representa as atividades que são levadas em consideração na simulação. Herdam características da classe Atividade, consomem recursos e possuem atributos que identificam as datas de previsão realizada antes da simulação e as datas obtidas com a realização da simulação.
- Recurso: Representa os recursos (software ou hardware) necessários para a realização de uma atividade durante a simulação. Podem ser componentes de software, hardware, salas de reunião, aparelhos eletrônicos e tudo o mais que é consumido e pode ser representado na realização de um projeto.

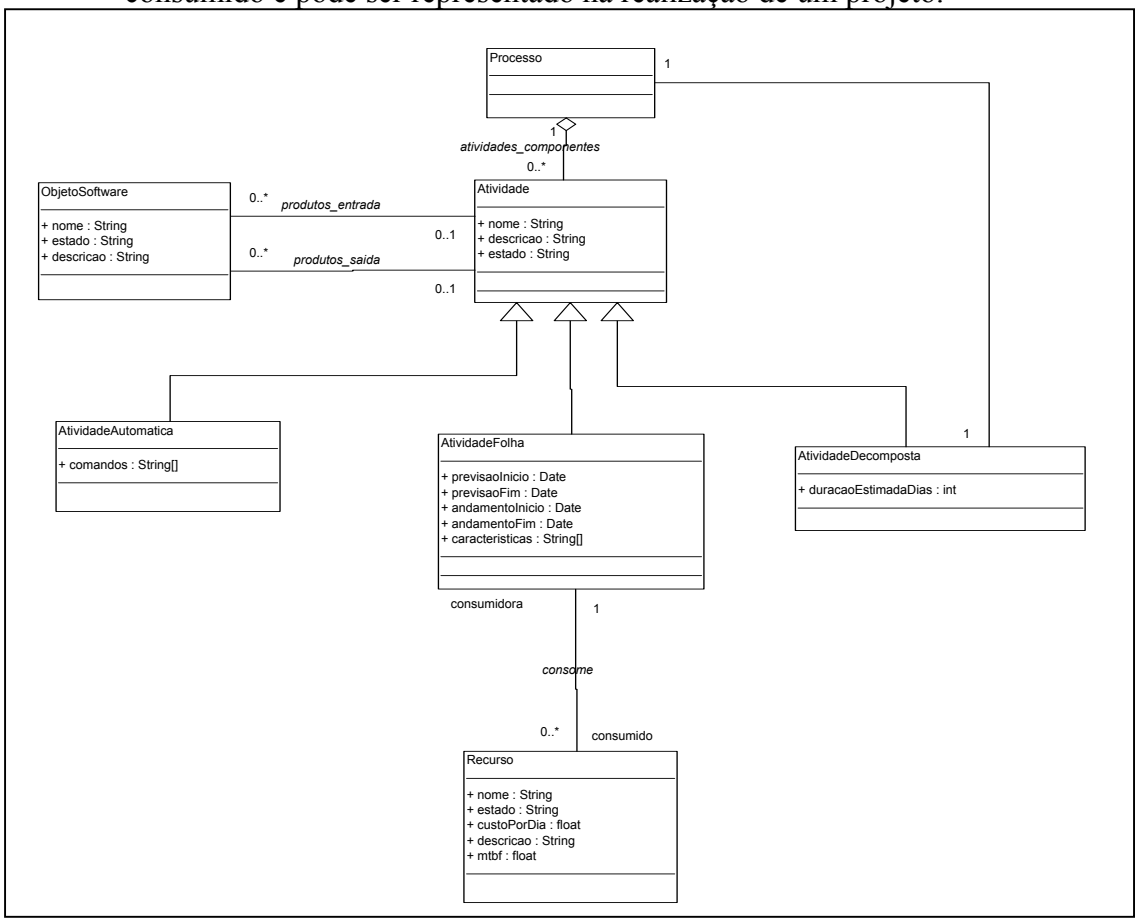

FIGURA 5.4 - *AgentProcess*: Modelo de Processo.

5.2.3 Agentes-Desenvolvedores

Os desenvolvedores envolvidos com as atividades são representados por estruturas denominados agentes-desenvolvedores, que são agentes inteligentes que simulam o comportamento de desenvolvedores reais em uma ambiente de desenvolvimento de software. Os agentes-desenvolvedores interagem com o ambiente e tomam decisões baseadas no estado do ambiente em que estão inseridos e se comunicam para realizar tarefas de forma cooperativa. Para implementar essas características, o modelo de simulação utiliza um modelo de sistema multiagentes cognitivos.

Os agentes-desenvolvedores possuem bases de conhecimento próprias, onde são representadas suas percepções sobre o ambiente. A utilização de bases de conhecimento diferentes para cada agente-desenvolvedor envolvido no projeto visa dar individualidade na representação dos mesmos. A estrutura da base de conhecimento é apresentada na seção 5.4.

Outra característica dos agentes-desenvolvedores é a representação de suas habilidades e afinidades que influenciam no cálculo do tempo que um agente vai necessitar para realizar determinada atividade. Esse cálculo é determinado de acordo com as características que uma atividade possui. Por exemplo, um agente que tem uma habilidade igual a 0,7 para a característica "análise orientada a objetos", de acordo com a tabela 5.1 (apresentada a seguir), vai realizar a atividade com uma economia de tempo de 10% em relação ao tempo previsto.

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 5.1 - Cálculo da variação do tempo em função da habilidade do agente

A definição das afinidades entre os agentes-desenvolvedores envolvidos na realização cooperativa de uma atividade procura representar o fato de que pessoas envolvidas no desenvolvimento de software não atuam isoladamente e a forma com que se relacionam com os demais componentes do ambiente de desenvolvimento influencia na sua produtividade. Assim como na habilidade cada agente-desenvolvedor possui um valor entre zero (0) e um (1) para definir sua afinidade com cada outro agente do modelo.

A modelagem dos agentes do simulador deve ser feita a partir de dados reais. Para obtenção de valores que representem o grau de habilidade e afinidade de um desenvolvedor em relação a alguma característica, podem ser utilizados dados históricos desse desenvolvedor em um ambiente de desenvolvimento. Quando não houver dados

históricos de um ambiente, os dados utilizados para enriquecer o modelo de simulação serão fornecidos pelo gerente do projeto. Os dados apresentados nas tabelas 5.1 e 5.2 servem apenas para ilustração e não foram obtidos de pesquisas em ambientes reais. Mais informações sobre o cálculo das habilidades e das afinidades de um agente está disponível em [SIL99].

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.

TABELA 5.2 - Cálculo da variação do tempo em função da afinidade do agente

Cada agente-desenvolvedor possui uma agenda com as atividades que ele deve seguir. O agente *Simulador de Processo* é quem adiciona nas agendas dos desenvolvedores as atividades que já podem ser realizadas. Um agente-desenvolvedor pode estar trabalhando em vários projetos ao mesmo tempo (vários processos podem estar sendo simulados) e existe uma agenda para cada projeto em que o agente participa. A agenda é o principal mecanismo de comunicação entre o agente e o simulador do processo. Para realizar uma atividade da sua agenda, o agente-desenvolvedor leva em consideração sua habilidade para realizar a tarefa e sua afinidade com os membros da equipe responsável pela atividade. Como a execução da atividade é simbólica, o agente apenas aguarda o período de tempo necessário para realizar a atividade e depois informa que a concluiu. Esse período de tempo nem sempre será o mesmo período previsto para a atividade, pois o agente pode atrasar ou mesmo adiantar a execução da sua tarefa. Em uma atividade cooperativa (com vários agentes) o tempo total da atividade será o tempo do agente mais lento.

Para simular a execução de uma atividade, é necessário realizar uma previsão de quanto tempo o desenvolvedor leva em média para realizar uma atividade. O comportamento básico do agente-desenvolvedor é mostrado na figura 5.5. Os acréscimos e decréscimos de tempo (*tempo_hab* e *tempo_afin*) da realização de uma atividade são obtidos das tabelas 5.1 e 5.2.

O agente-desenvolvedor escolhe uma atividade e requisita ao simulador o início da execução, se for dada autorização são calculadas suas afinidades (se a realização da atividade for cooperativa) e suas habilidades.

O agente-desenvolvedor interage com o ambiente através de sua base de conhecimento, verificando as características do ambiente naquele momento da execução da atividade. O tempo total então será o tempo previsto, os acréscimos e decréscimos obtidos do cálculo da habilidade e afinidade dos desenvolvedores e se alguma característica do ambiente interferir diretamente na realização a atividade, também será levada em consideração no cálculo do tempo.

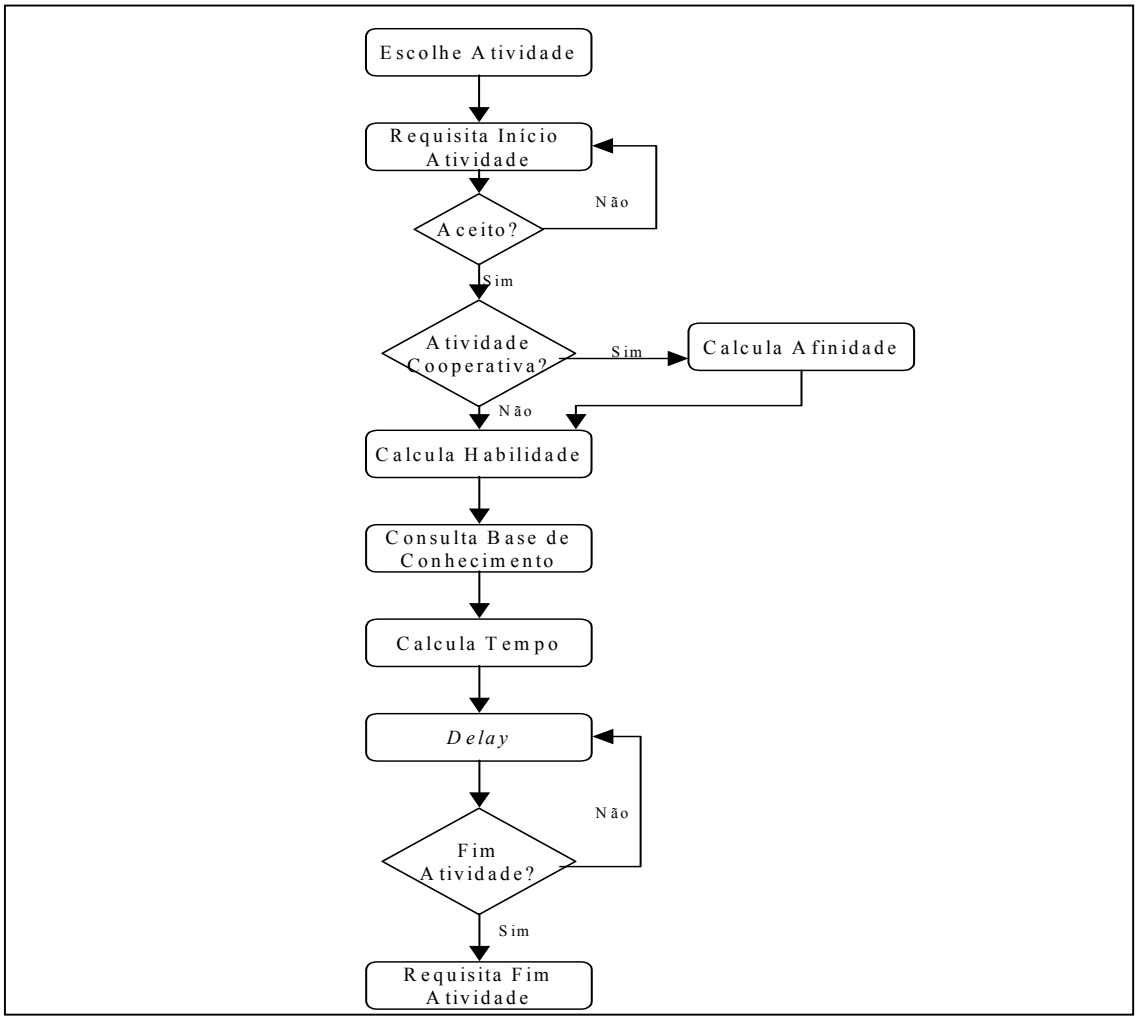

FIGURA 5.5 - Comportamento do Agente-Desenvolvedor

Para simular a execução da atividade, um *delay* com o tamanho do tempo calculado é realizado. Quando alcançar o fim da execução da atividade, o agente-desenvolvedor envia nova requisição ao simulador, dessa vez pedindo para finalizar a execução da atividade. As classes representadas no modelo importantes para a definição dos agentes-desenvolvedores são representadas na figura 5.6 e podem ser descritas assim:

- Agente-Desenvolvedor: Representa os desenvolvedores envolvidos com a realização das atividades na simulação. Possui uma agenda que deve indicar as atividades pelas quais o agente-desenvolvedor é responsável. Os objetos dessa classe realizam acesso a uma base de conhecimento para definir suas ações.
- Agenda: É composta de objetos representando agendas de projetos, já que um desenvolvedor (proprietário da agenda) pode participar de vários projetos ao mesmo tempo.
- ProjetoAgenda: Contém as atividades agendadas para serem realizadas por um desenvolvedor, dentro de um determinado projeto. Os objetos da classe ProjetoAgenda compõem a agenda do desenvolvedor.
- AtividadeAgenda: Representa objetos que são cópias dos objetos da classe AtividadeFolha, que fazem parte da classe ProjetoAgenda. Os objetos dessa

classe são utilizados pelos agentes-desenvolvedores para conter informações sobre o seu andamento na realização da atividade.

- Afinidade: Representa o grau de afinidade entre dois desenvolvedores. Essa informação é utilizada no cálculo do tempo de realização de uma atividade por um agente-desenvolvedor.
- Habilidade: Representa a habilidade de um agente-desenvolvedor em relação a determinada característica. As características são parte da definição das atividades simuladas. Essa informação também é útil no calculo do tempo de realização de uma atividade por um agente-desenvolvedor.
- Cargo: Cada projeto possui uma lista de cargos. Os cargos, por sua vez são ocupados por um ou mais desenvolvedores.

Na seção 5.4 será mostrado como é feita a consulta a base de conhecimento para definir o tempo de realização de uma atividade.

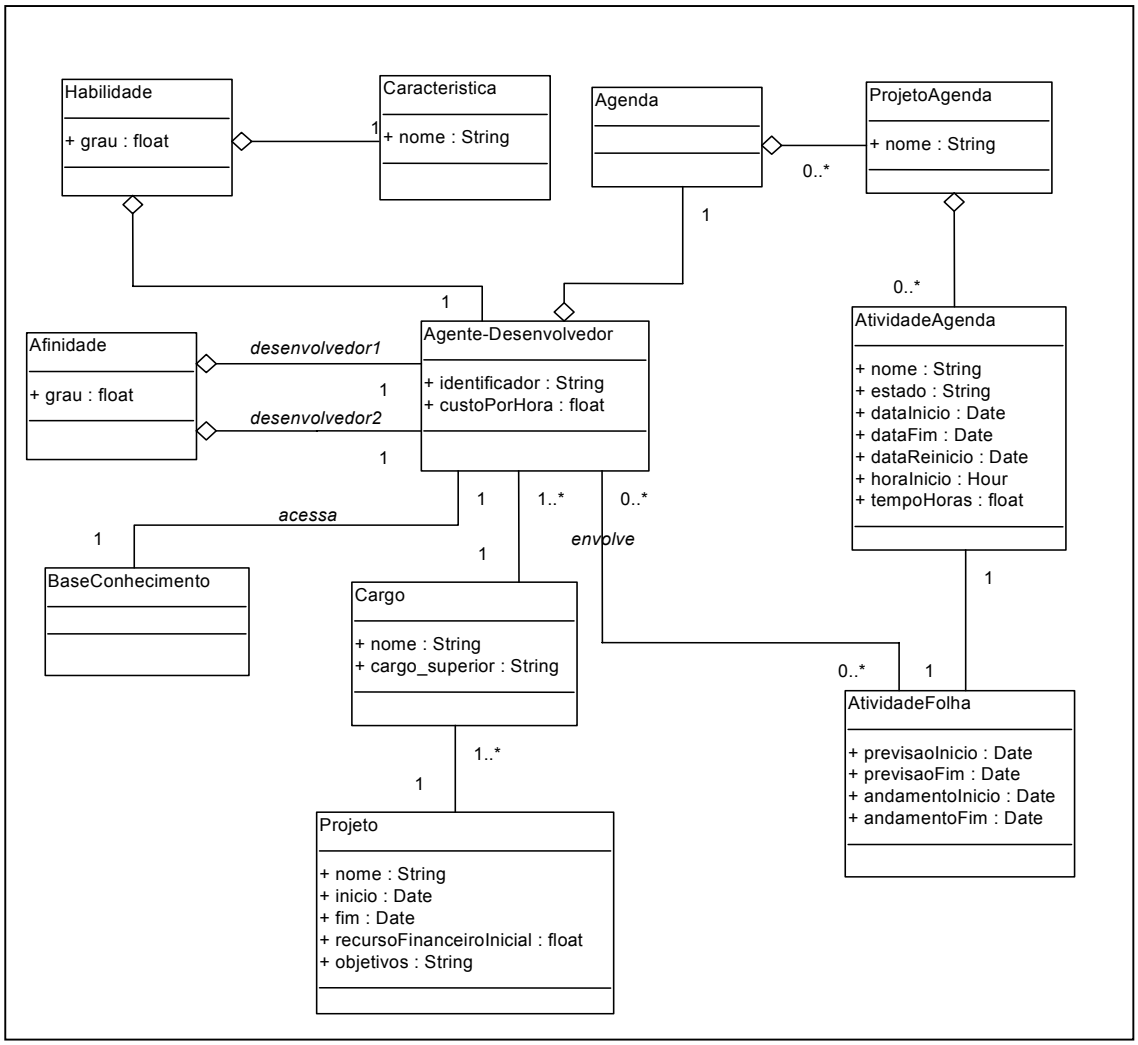

FIGURA 5.6 - *AgentProcess*:Desenvolvedores

5.2.4 Simulador do Processo

O agente inteligente encarregado da simulação do processo, coleta de métricas e coordenação de outros agentes é chamado *Simulador do Processo*. Este agente constitui a parte essencial do modelo. Suas características são as mesmas de uma máquina de execução de processos e foi baseado no modelo de execução do gerenciador de processos de software descrito em [LIM98].

Durante a simulação de um processo de software, o *Simulador do Processo* coleta métricas sobre o atraso das atividades, quantidade de agentes ociosos, agentes com carga de trabalho alta, utilização dos recursos, dentre outras avaliações. Os resultados obtidos são utilizados para gerar relatórios que serão avaliados pelo gerente da simulação.

São funções do simulador de processo:

- Controlar os ciclos da simulação;
- Iniciar e finalizar atividades;
- Verificar disponibilidade de recursos para realização de uma atividade;
- Alocar e liberar recursos para realização de uma atividade.

O Simulador de Processo é representado no modelo como o componente central, pois é responsável pelas principais atividades relacionadas com a simulação propriamente dita. Para controlar a simulação ele utiliza as informações modeladas, que indicam quais atividades devem ser realizadas, por quais desenvolvedores e consumindo quais recursos. Além disso, o simulador que é um agente inteligente, utiliza a base de conhecimento que possui informações sobre o ambiente, para controlar o andamento da simulação. O agente Simulador de Processo é o responsável por toda a execução do processo instanciado. O comportamento básico deste agente é descrito na figura 5.7.

Antes de realizar as atividades básicas o simulador identifica as atividades que estão prontas para executar e coloca-as nas agendas dos agentes-desenvolvedores responsáveis. As atividades básicas são para atender as requisições dos usuários que podem ser “iniciar atividade” ou “finalizar atividade”.

Quando um agente solicita o início de uma atividade o simulador faz uma consulta a base de conhecimento para verificar a possibilidade de realizar a operação, se for possível ele então verifica a disponibilidade de recursos necessários para a execução da atividade, se for possível os recursos são alocados, o estado da atividade é alterado para “ativa” e a agenda do agente é atualizada.

Para atender a uma solicitação de “finalizar atividade” a agenda do agente é alterada, ou seja, para o agente aquela atividade não será mais considerada. O simulador irá verificar o término real da atividade, isso significa verificar se todos os agentes que estavam responsáveis pela atividade já terminaram de realiza-la, caso sim os recursos são liberados para outras atividades e o estado da atividade é alterado para “completa”.

O simulador é definido através de recursos, agentes-desenvolvedores, projetos, relatório, base de conhecimento e atividades-padrão.

```
Repetir Ciclo:
    Para as atividades no estado "Esperando":
        Verificar a conclusão das atividades anteriores
    Para as atividades no estado "Pronto":
        Se a atividade é automática: Altera estado da atividade para "Completa"
        Se a atividades é folha:
            Verificar ambiente (consulta a base de conhecimento)
            Colocar as atividades nas agendas dos agentes responsáveis
        Aguardar solicitações dos agentes:
            Quando um agente solicitar "iniciar atividade":
                Verificar ambiente (consulta a base de conhecimento)
                Alocar recursos para a atividade
                Alterar estado da atividade para "ativa"
                Atualizar agenda do agente
            Quando um agente solicitar "termina atividade":
                Verificar ambiente (consulta a base de conhecimento)
                Alterar agenda do agente
                Verificar término real da atividade
                Se atividade do processo terminou:
                    Desalocar recursos
                    Alterar estado da atividade para "completa"
```

FIGURA 5.7 - Comportamento do Agente *Simulador do Processo*

A classe Recursos define os recursos disponíveis para o simulador. A classe Agentes-Desenvolvedores define os desenvolvedores no sistema. A classe Projeto representa os projetos a serem simulados. A classe Relatório define os eventos que ocorrem durante a simulação e a classe Atividade Padrão define componentes de processos pré-definidos e prontos para reutilização. Para facilitar a reutilização de uma atividade, utiliza-se um conjunto de características nas quais seus desenvolvedores devem ser hábeis.

A parte do modelo que representa o simulador do processo e seus relacionamentos são representados na figura 5.8. As classes envolvidas são:

- SimuladorProcesso: Representa o objeto que realiza a simulação do processo modelado. Esse objeto realiza acesso a uma base de conhecimento para definir algumas de suas ações. O simulador de processos é constituído de projeto, desenvolvedores, recursos, atividades padrão (disponíveis para reuso) e acessa uma base de conhecimento.
- Projeto: Define as métricas, estimativas, processos e cargos que serão manipulados pelo simulador para realização de suas tarefas.
- AtividadePadrão: Representa atividades que podem ser reutilizadas numa futura modelagem de processo.

- Recurso: Define os componentes que serão requisitados para a realização de alguma atividade por um agente-desenvolvedor.
- Agente-Desenvolvedor: Define os agentes-desenvolvedores disponíveis no sistema para realização das atividades definidas no projeto.
- Base de Conhecimento: Define as percepções do agente simulador a respeito do ambiente de desenvolvimento representado.
- Relatório: Define um depósito de eventos que ocorrem durante a simulação de um processo de desenvolvimento.

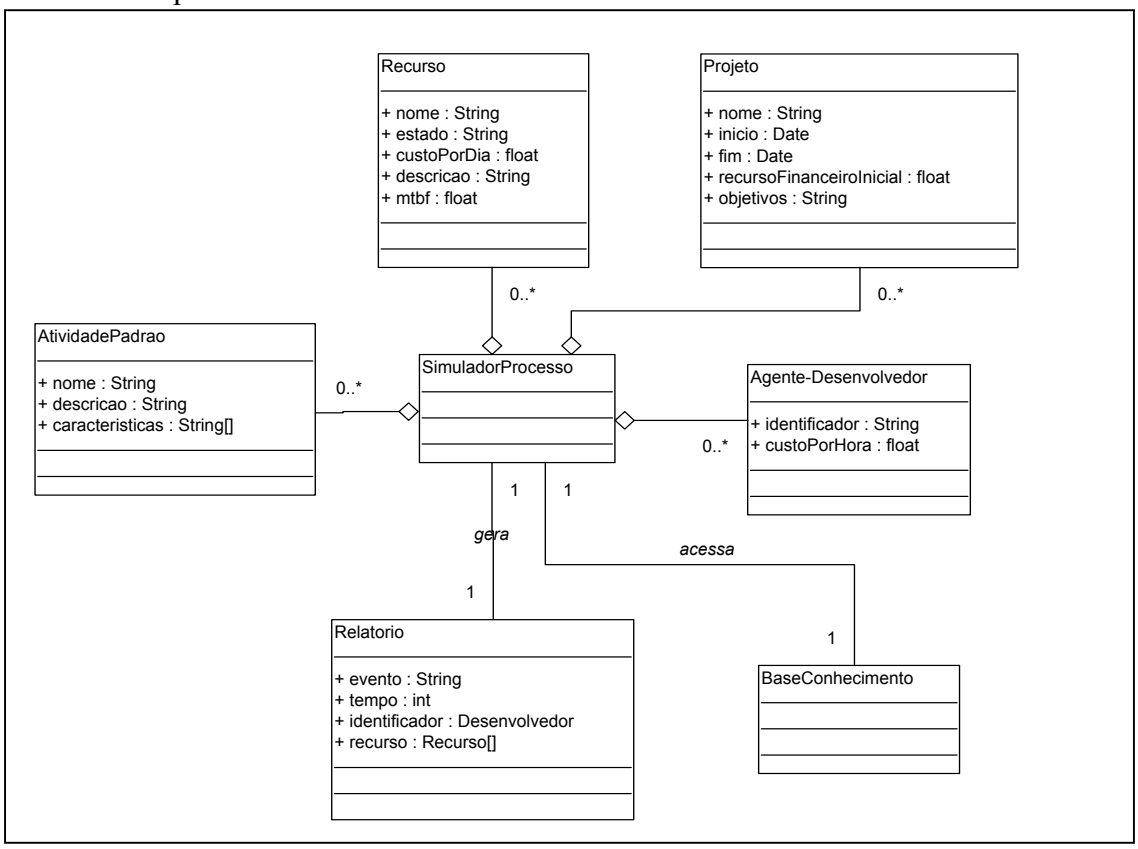

FIGURA 5.8 - *AgentProcess*: Simulador de Processo

5.3 Agenda

A agenda é o principal mecanismo de comunicação entre os agentes no ambiente de simulação. É através dela que os agentes podem cooperar na realização de atividades. O agente simulador, que é o principal agente do ambiente, aloca atividades nas agendas dos agentes-desenvolvedores. Um agente-desenvolvedor, por sua vez, pode enviar atividades para as agendas de outros agentes-desenvolvedores se ele não possuir competência para realiza-las sozinho.

Se o agente possuir mais de uma atividade na sua agenda, então ele deve optar por uma usando como critério o menor tempo previsto de conclusão da atividade, ou então o modelo pode ser configurado para seguir alguma heurística adicionada na base de conhecimento do agente.

O uso de uma heurística pode servir para evitar que uma atividade longa nunca seja iniciada, de forma que se defina que uma atividade pode ficar determinado tempo

(ciclos no caso da simulação) na agenda do agente-desenvolvedor antes da sua execução se tornar obrigatória

Quando um agente recebe uma atividade em sua agenda ele avalia as características daquela atividade, calculando o tempo que levaria para realiza-la e avalia o ambiente, procurando alternativas mais eficientes. Nesse caso, o agente deve seguir critérios definidos em sua base de conhecimento. Se a alternativa mais eficiente for a delegação da atividade a outro agente-desenvolvedor, a atividade é enviada para a agenda do escolhido, sendo que critérios de hierarquia devem ser respeitados. Se os dois agentes estiverem no mesmo nível hierárquico do projeto, o agente requisitado pode aceitar ou rejeitar a requisição de realização da atividade.

5.4 Base de Conhecimento

A base de conhecimento é utilizada para representar o comportamento intrinsecamente relacionado a cada um dos agentes do modelo. Por esse motivo, cada agente possui a sua base de conhecimento própria. O conhecimento representado é fruto da experiência de cada individuo em uma ambiente de desenvolvimento.

No caso do agente simulador de processo, ele realiza consulta a base quando necessita alocar atividades nas agendas dos agentes-desenvolvedores, quando recebe requisição de um agente-desenvolvedor para iniciar a execução de uma atividade e para finalizar a execução da atividade. A utilização da base de conhecimento na realização dessas operações permite ao simulador avaliar o impacto das ações no ambiente, antes de serem realizadas.

A estrutura da base de conhecimento é representada na figura 5.9. O modelo representa que cada agente-desenvolvedor e o simulador de processos possuem sua própria base de conhecimento. A base de conhecimento de cada agente é composta por um elemento denominado “Regras de Produção”. As regras de produção representam informações que um agente possui sobre o ambiente de desenvolvimento e a evolução do mesmo significa o acréscimo de novas regras a sua base. Para realizar essa representação são utilizadas as regras, que devem ser compostas de um antecedente e um conseqüente.

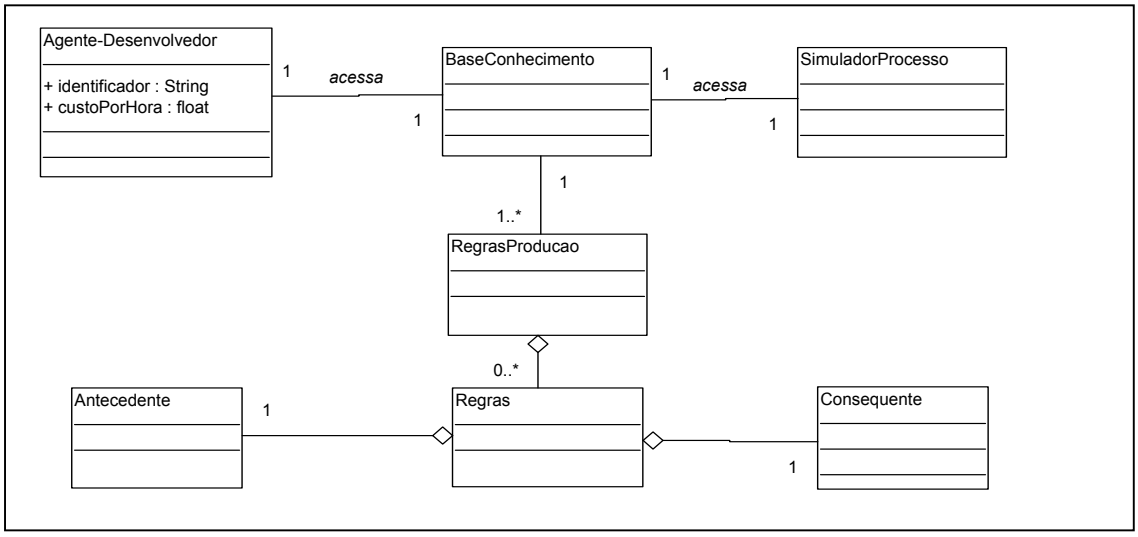

FIGURA 5.9 - Estrutura da representação de conhecimento

O antecedente indica a parte “se” de uma regra, deve ser composto essencialmente de atributos e premissas e deve representar as condições que serão observadas para se disparar uma regra. O conseqüente indica a parte “então” de uma regra, deve representar deduções baseadas nas premissas e podem ser representadas por ações ou pelo disparo de novas regras.

A base de conhecimento é alimentada por um mecanismo denominado “knowledge discovery”, através de informações obtidas em um ambiente de desenvolvimento. Este mecanismo é proposto no trabalho [LIM01]. O comportamento dos agentes então é influenciado pela percepção que o agente tem do meio em que está inserido através do que está representado na base de conhecimento. Alguns exemplos de situações que necessitam de consultas a base de conhecimento são apresentados na tabela 5.3.

5.5 Recursos

Para permitir que um agente-desenvolvedor realize alguma atividade, o simulador precisa verificar se os recursos requisitados para execução da atividade estão disponíveis. Se todos os recursos estiverem disponíveis, eles são alocados para a realização da atividade e então o simulador envia uma mensagem para o agente-desenvolvedor autorizando-o a executar a atividade.

Na representação de um recurso é definido o nome do mesmo, o estado, que pode ser disponível ou não disponível, o custo por hora de sua utilização, uma descrição e o mtbf (*mean time between failure*- tempo médio entre falhas), que representa a vida útil do recurso antes de apresentar problemas. Para efeito de simulação isso não está sendo levado em conta, pois é considerado que os recursos irão funcionar sempre de forma correta.

Situação	**Possíveis trilhas**
Simulador deseja alocar atividade em agenda de agente-desenvolvedor.	Verificar se agente-desenvolvedor está atrasado em relação a estimativas. Caso agente-desenvolvedor esteja atrasado, verificar se existe ação alternativa. Se não, alocar atividade. Se sim, pesquisar ação alternativa.
Simulador deseja iniciar atividade.	Verificar se atividade é cooperativa. Se sim, verificar se a inclusão de mais um agente-desenvolvedor vai melhorar a realização da atividade. Se não, autorizar agente-desenvolvedor a executar atividade com tempo 0 (zero).
Simulador deseja finalizar atividade	Avaliar desempenho do agente-desenvolvedor. Atualizar base.
Agente-desenvolvedor deseja calcular tempo de execução de atividade.	Avaliar prazo definido no projeto. Avaliar quantidade de envolvidos na atividade.

TABELA 5.3 - Exemplos de consultas a base de conhecimento

Existem dois tipos de alocação de recursos:

- Exclusiva: O recurso só pode estar sendo utilizado por uma atividade de cada vez.
- Não exclusiva: O recurso pode ser compartilhado por mais de uma atividade.

As regras descritas acima podem ser redefinidas na base de conhecimento do simulador, fazendo com que o mesmo possa trabalhar de uma forma mais eficiente na alocação dos recursos. Por exemplo, o simulador pode negar a alocação de um recurso disponível para uma atividade, se for detectada a possibilidade de ocorrência de um *deadlock*. Para isso é necessário que um algoritmo de detecção de *deadlock* esteja definido na base de conhecimento do agente.

5.6 Ferramenta *SimAgentProcess*

A *SimAgentProcess* é a ferramenta construída para implementar o modelo *AgentProcess*. Existem dois tipos de usuários ou papeis definidos para a *SimAgentProcess*, o projetista do processo e o gerente do projeto. O projetista do processo é responsável pelo modelo de processo de software, que será usado na simulação, tendo interesse nas avaliações do simulador em relação ao modelo, para definir a necessidade de modificações no mesmo. O gerente de projeto coordena a fase de simulação, como se tal fosse a realização propriamente dita de um desenvolvimento de software. Ele tem interesse na obtenção de um modelo de processo pronto para ser executado.

O gerente de um projeto define quando um modelo simulado está pronto para ser executado. A avaliação é realizada levando-se em consideração as informações obtidas do simulador, como o tamanho médio das filas para cada entidade envolvida, o tempo médio de espera de um desenvolvedor para poder realizar uma atividade, entre outras avaliações. Ao fim da simulação, pode-se atestar que o modelo de processos de software está pronto para ser utilizado em um ambiente de desenvolvimento ou então a necessidade de realização de uma nova análise de requisitos, uma nova modelagem ou um novo escalonamento de atividades por desenvolvedores, dependendo do nível das inconsistências encontradas.

A *SimAgentProcess* valida os algoritmos apresentados no modelo, para testar se os mesmos estão funcionando como planejado. Para isso é necessário que cada agente-desenvolvedor seja acionado em cada ciclo da simulação para realizar os eventos previstos.

Mais informações sobre a *SimAgentProcess* está disponível em [SIL00].

5.7 Considerações Finais

Apesar do modelo apresentado ter sido feito com vistas a ser integrado a um ambiente maior que ainda está em fase de definição, suas definições tem vida própria independente do contexto, podendo ser integrado a qualquer ambiente de desenvolvimento desde que este possua seus processos definidos.

A aplicação da simulação como uma ferramenta de apoio a validação de modelos de processo só faz sentido se aplicada a um ambiente com papeis bem definidos. Quanto maior for a fidelidade na representação dos componentes, melhores serão os resultados, principalmente no que diz respeito a aplicação das técnicas da inteligência artificial.

A definição do modelo procurou representar os aspectos considerados mais relevantes, no entanto a sua arquitetura permite sempre incorporações que o enriqueça em detalhes de maneira a melhorar o seu desempenho. Esses detalhes podem ser incorporados em uma redefinição das classes ou podem ser definidos na base de conhecimento.

No próximo capítulo será dada ênfase a definição do modelo de simulação incorporado ao ambiente PROSOFT. Sendo que o simulador será definido como uma ferramenta do ambiente, interagindo com as demais ferramentas utilizando os recursos disponíveis.

6 Modelo de Simulação para o ambiente PROSOFT

O modelo de simulação apresentado no capítulo 5 utilizando a notação UML será integrado ao ambiente de desenvolvimento de software orientado a processos do PROSOFT. O simulador é mais uma ferramenta de apoio ao desenvolvimento a ser disponibilizada pelo ambiente. Nesse capítulo será apresentada a especificação do modelo de simulação de acordo com o paradigma do ambiente PROSOFT.

A utilização do ambiente PROSOFT facilitou a definição do modelo de simulação devido as características pertencentes a ambientes de especificação formal, como a garantia de propriedades como a completeza, corretude e ausência de ambigüidade, tal como permitiu a reutilização de definições já realizadas no PROSOFT que são de grande utilidade para o modelo de simulação apresentado nesse trabalho, como as definições de um modelo de gerencia de processos de software e de um sistema especialista.

Para validar os conceitos de simulação especificados foi construída uma ferramenta utilizando a linguagem Java, pois como foi citado anteriormente não foi possível implementar a especificação utilizando o ambiente PROSOFT, já que o mesmo estava passando por um processo de reengenharia.

6.1 Ambiente PROSOFT

O PROSOFT é um ambiente de pesquisa que vem sendo desenvolvido em um projeto de pesquisa do CPGCC/UFRGS sob a coordenação do Prof. Dr. Daltro José Nunes. O objetivo é a definição e construção de um ambiente de desenvolvimento de software para apoiar o desenvolvimento formal de software, fornecendo integração de dados, controle e de apresentação entre suas ferramentas.

Trata-se de um ambiente composto por diversas ferramentas, que são utilizadas pelos desenvolvedores na realização de suas tarefas. As ferramentas definidas para o ambiente PROSOFT são utilizadas desde a especificação de requisitos até a implementação do sistema.

Atualmente, existe no ambiente PROSOFT ferramentas que auxiliam na construção de interfaces, de diagramas de fluxo de dados, de diagrama de Nassi-Schneidermann, entre outros. Recentemente foram propostas para o ambiente PROSOFT ferramentas para a definição de um ambiente especialista [MOR97], um ambiente cooperativo [REI98] e o ambiente de gerenciamento de processos de software [LIM98].

O paradigma de construção de ferramentas no PROSOFT foi influenciado pelos itens [NUN94]: estratégia *data-driven* [NUN92], conceito de modelos, cálculo lambda, conceito de tipo abstrato de dados e paradigma de orientação a objetos.

6.1.1 Arquitetura do PROSOFT

O ambiente PROSOFT tem evoluído para suportar desenvolvimento cooperativo de software [REI98], modelagem e execução de processos de software [LIM98] dentre outras características. Desta forma, sua arquitetura tem sido aperfeiçoada. Entretanto, basicamente a arquitetura é constituída de ATOs (Ambientes de Tratamento de Objetos) que se comunicam através da ICS (Interface de Comunicação do Sistema).

A arquitetura do PROSOFT, apresentada na figura 6.1, mostra que o gerenciamento de processo de software foi implementado sobre os serviços do PROSOFT Cooperativo (componente do ambiente responsável pelos serviços de cooperação e comunicação dos usuários).

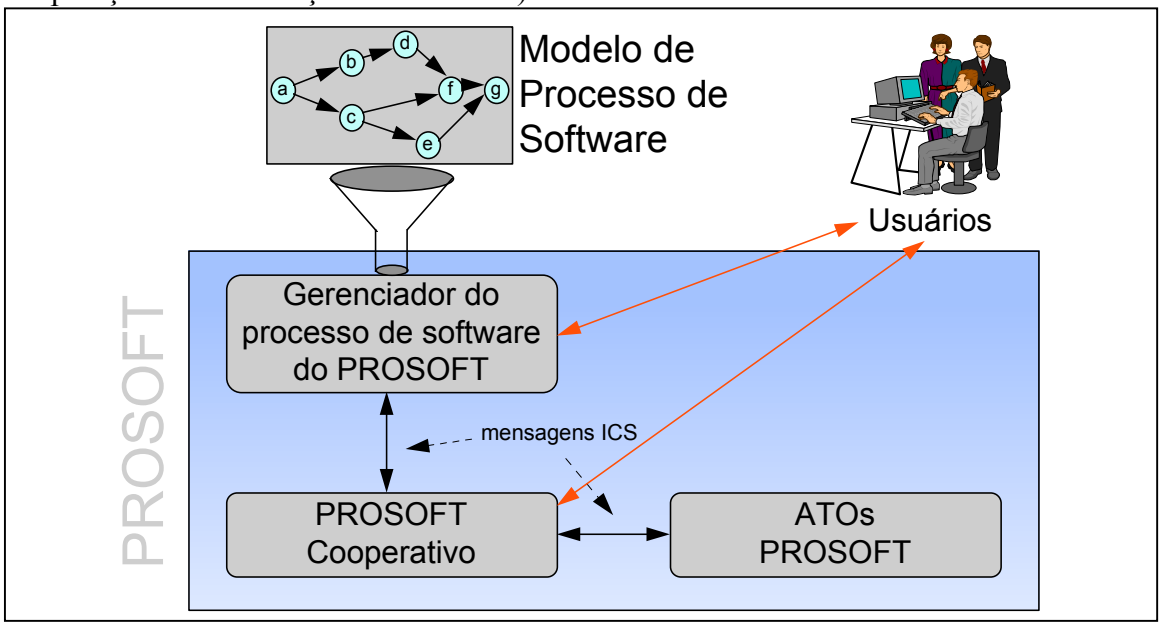

FIGURA 6.1 - Arquitetura do PROSOFT

Os ATOs PROSOFT representam as ferramentas do PROSOFT, enquanto a ICS é o mecanismo através do qual essas ferramentas interagem. A figura 6.2 representa vários ATOs interligados através da ICS.

Um ambiente de tratamento de objetos implementa um tipo abstrato de dados, sendo composto essencialmente de uma classe e de um conjunto de operações que atuam sobre os objetos dessa classe. A estrutura do ATO é definida em três partes:

- Classe: Representada graficamente e definida através da composição dos tipos de dados.
- Interface: Especifica formalmente as operações sobre a classe.
- Operações: Específica a semântica das operações sobre a classe, ou seja, os axiomas.

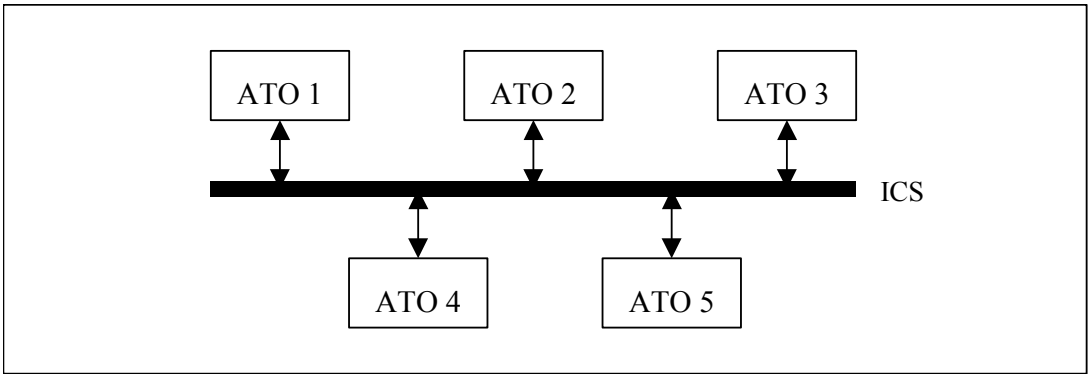

FIGURA 6.2 - Estrutura do PROSOFT [NUN94]

6.1.2 Tipos de Dados do PROSOFT

Os tipos de dados presentes no PROSOFT estão classificados em primitivos, compostos e definidos pelo usuário.

Os tipos primitivos são: Integer, String, Real, Char, Boolean, Date e Time. Eles são triviais, apresentando as características usuais das linguagens de programação do tipo PASCAL.

Os tipos de dados compostos representam os construtores dos tipos compostos definidos no método algébrico:

- Conjunto: Representa uma coleção de objetos obedecendo as características de conjuntos. São identificados através da letra "S";
- Lista: Representa uma seqüência (ordenada) de zero ou mais componentes. São identificados através de "*";
- Mapeamento: Expressa uma função estabelecendo uma correspondência entre objetos de um Tipo-Domínio com os objetos do Tipo-Imagem. São identificados através de uma seta do Tipo-Domínio ao Tipo-Imagem;
- Registro: Define tuplas de tipos heterogêneos e é usada para expressar o produto cartesiano de objetos. São identificados por linhas simples ligando o tipo resultante aos objetos do produto cartesiano.
- União disjunta: Define a união de tipos diferentes de elementos. A união disjunta é útil na definição de classes de dados que necessitem expressar valores alternativos. São identificados através de círculos.

Os tipos definidos pelo usuário são os tipos de dados representados pelos ATOs existentes no ambiente e que foram construídos pelo usuário.

6.2 Descrição do Simulador de Processos

O simulador é um conjunto de ATOs, cujo ATO principal é denominado ATO Simulador, tornando-se assim uma ferramenta do PROSOFT. O ATO Simulador define a funcionalidade do simulador de processos e utiliza operações definidas em ATOs já existentes, principalmente os apresentados em [LIM98], que interessam diretamente a esse trabalho, por definirem a modelagem de processos utilizada. Também são

utilizados os ATOs definidos em [MOR97] que são usados para definir a base de conhecimento utilizada pelo simulador.

Para que o ATO Simulador possa utilizar a funcionalidade de outros ATOs, são utilizadas as funcionalidades da ICS, que permite comunicação entre ATOs diferentes, tornando possível a utilização de operações pré-definidas. Para atender a simulação de processos novos ATOs foram definidos e outros ATOs definidos em trabalhos anteriores foram completamente aproveitados e outros ainda foram adaptados para atender as funcionalidades da simulação. Os conceitos ATO e ICS, assim como outros conceitos relacionados com o ambiente PROSOFT, são melhor explicados na próxima seção.

As principais ferramentas utilizadas pelo ambiente de simulação são:

- ATO Simulador (Simulador de Processos): É a principal ferramenta do ambiente. É composta por: um conjunto de recursos; agentes-desenvolvedores; projetos; uma base de conhecimento; um relatório e um conjunto de atividades padrão.

- ATO Recurso: É a ferramenta que permite a definição dos recursos que são requisitados pelo projeto e tem sua alocação simulada no ATO Simulador. Foi adaptado a partir do ATO definido em [LIM98].

- ATO Agentes-Desenv.: É a ferramenta que permite a representação das características dos desenvolvedores envolvidos em um projeto.

- ATO Projetos: É a ferramenta que permite definição e manipulação de projetos de desenvolvimento de software. Cada projeto tem como componente um processo de software, métricas, estimativas e cargos. Foi adaptado a partir do ATO definido em [LIM98].

- ATO BCSIM: É a ferramenta que permite a manipulação de informações que são usadas como conhecimento a respeito do ambiente de desenvolvimento pelo simulador e por componentes do ambiente (agentes-desenvolvedores). Esse ATO define a interface entre o ATO Simulador o ATO BCI, que permite a representação de conhecimento sobre o ambiente.

- ATO Relatório: É a ferramenta que controla o histórico de uma simulação armazenando os eventos ocorridos. Os eventos contêm as atividades realizadas, o nome dos agentes-desenvolvedores que as realizaram e o tempo de ocorrência.

- ATO Atividade Padrão: É a ferramenta que permite definição de atividades, que podem ser reusadas na inclusão de novas atividades pertencentes ao processo.

A arquitetura apresentada na figura 5.1 é redefinida na figura 6.3 com a indicação dos ATOs relacionados com cada componente definido.

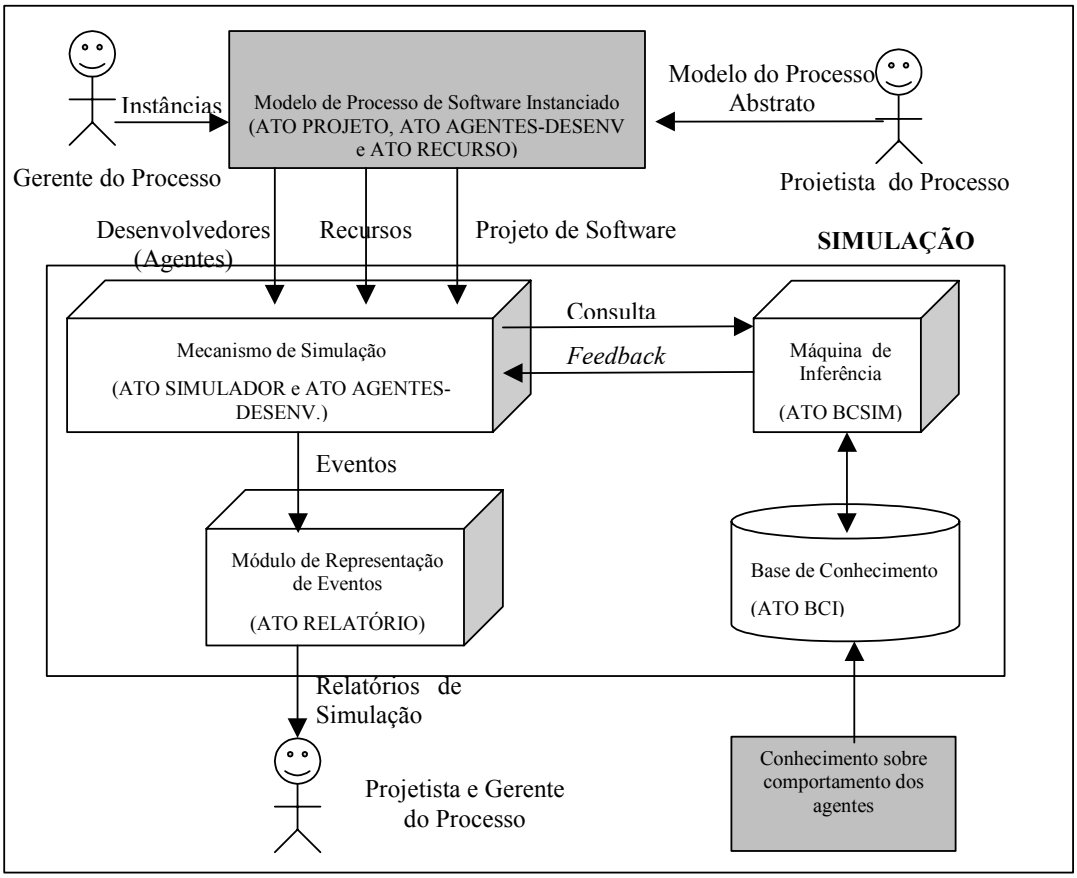

FIGURA 6.3 - Arquitetura do Simulador com ATOs

Essas ferramentas utilizam outras ferramentas, aqui descritas como ferramentas auxiliares:

- ATO Processo: É a ferramenta que define e manipula os processos de software, que estão associados a projetos; foi definido em [LIM98].
- ATO Atividade: É a ferramenta que define e manipula as atividades que compõe um processo de software. Foi definido em [LIM98].
- ATO Agenda: Utilizada para atribuir atividades aos agentes-desenvolvedores. Foi definido em [LIM98].
- ATO Projeto-Agenda: Relaciona as agendas dos desenvolvedores com os projetos em que eles estiverem envolvidos. Foi definido em [LIM98].
- ATO Métrica: Ferramenta que define e manipula um conjunto de métricas que podem ser coletadas automaticamente a partir do andamento de uma simulação de desenvolvimento de software. Foi definido em [LIM98].
- ATO Estimativa: É a ferramenta que define e manipula um conjunto de estimativas que podem ser definidas para permitir a gerência do desenvolvimento e comparação com o andamento real do processo. Foi definido em [LIM98].

- ATO Cargo: Relaciona agentes-desenvolvedores com projetos, através do cargo exercido.
- ATO BCI: Manipula a base de conhecimento utilizada pelo ATO BCSIM.

Para utilizar o simulador é necessário que o usuário defina os projetos que serão simulados. Com os projetos definidos através de modelos de processo instanciados, o simulador irá "carregar" as agendas dos agentes-desenvolvedores com atividades e responderá a requisições dos agentes-desenvolvedores para iniciar e finalizar a execução de atividades e também irá alocar ou desalocar recursos quando for necessário.

6.3 O Simulador

O ATO Simulador é o ATO principal do ambiente. É nesse ATO que estão definidas as operações disponíveis e necessárias para se realizar a simulação de um projeto de software, sendo que nele estão definidas também as chamadas de sistema, realizadas através de ICS, que acessam as operações necessárias disponíveis em outros ATOs.

O ATO Simulador é apresentada na figura 6.4, a interface e as operações são apresentadas no decorrer desta seção.

Para instanciar um simulador de processos, foram utilizados os tipos registro e conjunto do PROSOFT. Para tratar os componentes da definição da composição do simulador de processo, são utilizados outros ATOs (ATO Recurso, ATO Agentes-Desenv., ATO Projetos, ATO BCSIM, ATO Relatório e ATO Atividade-Padrão) que também serão descritos nesse capítulo.

Os ATOs apresentados aqui e as classes representadas na figura 5.8 e descritas na seção 5.2.4 são relacionados, ou seja, o que é definido na classe SimuladorProcesso é equivalente ao que é definido no ATO Simulador, valendo o mesmo para as classes Recursos, Agente-Desenvolvedor, Projeto, Base de Conhecimento, Relatório e AtividadePadrão em relação aos ATOS Recurso, Agentes-Desenv, Projeto, BCSIM, Relatório e Atividade-Padrão respectivamente.

O campo recursos no ATO Simulador estabelece o conjunto de recursos utilizados pelo ambiente de desenvolvimento de software simulado. O campo agentes-desenvolvedores contém a representação de todos os agentes-desenvolvedores envolvidos com um ambiente de desenvolvimento. O campo projetos representa os projetos que são simulados pelo simulador de processos. O campo BCSIM representa a base disponível para consulta por parte dos agentes do modelo. O campo relatório contém os eventos obtidos na simulação. O campo atividades padrão representa o conjunto de atividades disponíveis que podem ser utilizadas na definição de um processo.

Um objeto do tipo simulador de processos é criado através da operação *cria-simulador*. Na operação de criação é fornecido um parâmetro do tipo BCI, que é a base de conhecimento que vai ser utilizado na definição do comportamento dos agentes do simulador, pois ela contém informações sobre o ambiente de desenvolvimento da simulação. Para definir a base é necessário utilizar a ferramenta definida por [MOR97].

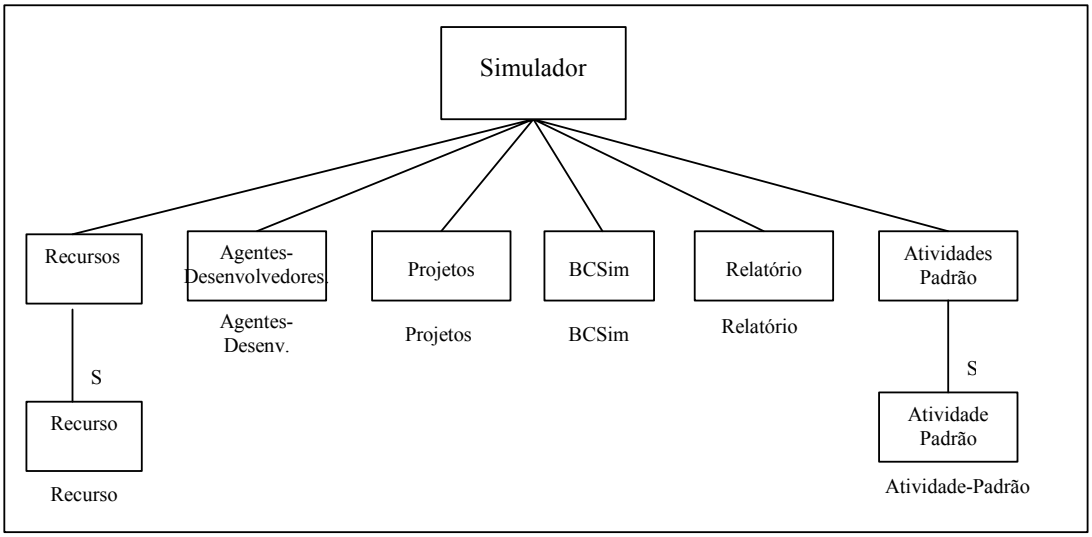

FIGURA 6.4 - Classe do ATO Simulador de Processos

Interface:

cria_simulador(_): -> SIMULADOR
cria_ag_desenv: -> AGENTES-DESENV
cria_projeto_vazio: -> PROJETOS
cria_relatório_novo: -> RELATÓRIO

Operações:

Variável Formal:

base : BCI

cria_simulador(base) = (Recursos empty-set, ntes-Desenvolvedores cria_ag_desenv(), etos cria_projetos_vazio, BCSIM *inclui_base_aux*(base), Relatório cria_relatório_novo, Atividades Padrão empty-set)

cria_ag_desenv = **ICS**(AGENTES-DESENV, cria_agente_des)

cria_projetos_vazio = **ICS**(PROJETOS, cria_projetos)

cria_relatório_novo = **ICS**(RELATÓRIO, cria_relatório)

Nas seções a seguir serão apresentadas as operações do ATO Simulador. A primeira operação é a de criação do simulador e só diz respeito ao ATO Simulador. As demais operações são definidas para atuarem sobre os componentes do ambiente de simulação e geralmente interagem com outros ATOs.

6.3.1 Operações de definição da simulação

Antes de realizar a simulação é necessário definir os componentes do ambiente. O simulador fornece ao usuário da ferramenta (gerente da simulação) operações que permitem a representação de recursos, agentes-desenvolvedores, projetos, base de conhecimento, relatório e atividades padrão. As operações serão apresentadas nessa seção e o anexo 1 apresenta mais detalhes sobre as suas especificações.

As operações que o gerente de simulação pode realizar sobre os recursos são:

Interface:

```
inclui_recurso(_,_,_,_,_): SIMULADOR, STRING, REAL, TEXTO, REAL   -> SIMULADOR
exclui_recurso(_,_): SIMULADOR, STRING                            -> SIMULADOR
altera_custo_recurso(_,_,_):SIMULADOR, STRING, REAL               -> SIMULADOR
altera_estado_recurso(_,_): SIMULADOR, STRING                     -> SIMULADOR
altera_descrição_recurso(_,_,_,_): SIMULADOR, STRING, TEXTO       -> SIMULADOR
a_mtbf_recurso(_,_,_): SIMULADOR, STRING, REAL                    -> SIMULADOR
```

Com a operação inclui_recurso é possível definir um recurso no ambiente informando seu nome, custo, descrição e mtbf, além de definir o estado inicial do recurso como "disponível" para uso. A operação exclui_recurso remove a definição do ambiente e as demais operações permitem ajustes nas definições dos recursos.

A operação altera_estado_recurso é utilizada pelo mecanismo de simulação para controlar a utilização do recurso, indicando se o mesmo está disponível ou não para ser utilizado na realização de alguma atividade.

As operações que o gerente da simulação pode realizar na definição dos agentes-desenvolvedores são:

Interface:

```
inclui_agente-des(_,_,_): SIMULADOR, STRING, REAL                          -> SIMULADOR
exclui_agente-des(_,_): SIMULADOR, STRING                                  -> SIMULADOR
altera_custo_agente-des(_,_,_): SIMULADOR, STRING, REAL                    -> SIMULADOR
inclui_habilidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
exclui_habilidade_agente-des(_,_,_): SIMULADOR, STRING, STRING             -> SIMULADOR
altera_habilidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
inclui_afinidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL  -> SIMULADOR
exclui_afinidade_agente-des(_,_,_): SIMULADOR, STRING, STRING              -> SIMULADOR
altera_afinidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL  -> SIMULADOR
```

Através dessas operações é possível definir e remover do ambiente de simulação definições de agentes-desenvolvedores. O agente-desenvolvedor possui um custo que pode ser modificado pelo gerente da simulação através da operação altera_custo_agente-des, sendo possível também a definição, alteração e exclusão de habilidades e afinidades, que foram explicadas no capítulo 5, relativas ao mesmo.

Os projetos são os principais objetos da simulação. Na definição dos projetos é necessária a definição de processos, métricas, estimativas e cargos. O modelo de processos de software utilizado pelo modelo de simulação é baseado no modelo definido em [LIM98]. A única diferença é que na definição de projetos utilizada pelo simulador é necessária a definição de cargos ocupados pelos agentes-desenvolvedores, para permitir o teste de escalonamentos diferentes.

As operações disponíveis para o gerente da simulação na definição de projetos são:

Interface:

```
define_novo_projeto(_,_,_,_,_,_): SIMULADOR, STRING, DATE, DATE, REAL,
                                  TEXTO                                    -> SIMULADOR
exclui_projeto(_,_): SIMULADOR, STRING                                     -> SIMULADOR
altera_dataini_projeto(_,_,_): SIMULADOR, STRING, DATE                     -> SIMULADOR
altera_datafim_projeto(_,_,_): SIMULADOR, STRING, DATE                     -> SIMULADOR
altera_recurso_financeiro_projeto(_,_,_): SIMULADOR, STRING, REAL          -> SIMULADOR
```

altera_objetivos_projeto(_,_,_): SIMULADOR, STRING, TEXTO -> SIMULADOR
inclui_produtof_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_produtof_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_processo_projeto(_,_,_): SIMULADOR, STRING, PROCESSO -> SIMULADOR
inclui_métrica_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_métrica_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_valor_métrica_projeto(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
inclui_estimativa_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_estimativa_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_valor_estimativa_projeto(_,_,_,_): SIMULADOR, STRING, STRING, REAL-> SIMULADOR
inclui_cargo_projeto(_,_,_,_,_): SIMULADOR, STRING, STRING, STRING,
STRING -> SIMULADOR
exclui_cargo_projeto(_,_,_):SIMULADOR, STRING, STRING -> SIMULADOR
altera_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_agente_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_agente_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING-> SIMULADOR

Ao definir um novo projeto o gerente da simulação informa data de início prevista, data de fim previsto, recursos financeiros disponíveis para o projeto e os objetivos do projeto. Todas essas informações podem ser modificadas através das operações de alteração disponíveis. Além das informações fornecidas na definição de um novo projeto é possível também definir o produto final (produto que deve ser obtido ao final da realização do projeto), métricas e estimativas para o projeto.

Através da operação de definição de cargos (inclui_cargo_projeto) o gerente da simulação irá definir os cargos existentes no projeto e os agentes-desenvolvedores que irão ocupa-los.

As demais operações do ATO Simulador relacionadas com projetos, dizem respeito a modelagem de processo de software. Um dos campos do projeto, permite a definição de um processo, que é um conjunto de atividades. As operações relativas a definição de atividades de um projeto são:

Interface:
inclui_atividade_projeto(_,_,_): SIMULADOR, STRING, ATIVIDADE -> SIMULADOR
exclui-atividade_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_previsão_início_ativ(_,_,_,_): SIMULADOR, STRING, STRING, DATE -> SIMULADOR
altera_previsão_fim_ativ(_,_,_,_): SIMULADOR, STRING, STRING, DATE -> SIMULADOR
inclui_prod_entrada_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING-> SIMULADOR
inclui_prod_saída_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_prod_entrada_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING-> SIMULADOR
exclui_prod_saída_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_ativ_anterior-ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui-ativ_anterior_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_recurso_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_recurso_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_agente-desenvs_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_agente-desenvs_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING-> SIMULADOR

Através dessas operações é possível definir e excluir atividades de um projeto. As operações de atividades permitem a manipulação de previsões de início e fim da atividade, produtos de entrada e saída, atividades anteriores (atividades que evem ser executadas e finalizadas antes que determinada atividade seja iniciada), recursos utilizados para realizar a atividade e os agentes-desenvolvedores responsáveis pela atividade.

Tendo uma base de conhecimento disponível, o simulador precisa aplicar uma máquina de inferência sobre as regras definidas para obter informações sobre o ambiente, para isso utiliza a operação realiza_inferência. Para disponibilizar a base de conhecimento no simulador utiliza-se a operação inclui_base_aux. Essas operações são apresentadas a seguir:

Interface:

realiza_inferência(_): SIMULADOR	-> SIMULADOR
inclui_base_aux(_): BCI	-> BCSIM

Operações

Variáveis Formais:

base_conhecimento : BCSIM

realiza_inferência((_,_,_,BCSIM base_conhecimento,_ ,_)
= (_,_,_,BCSIM **ICS**(BCSIM, gera_agenda, base_conhecimento),_,_)

inclui_base_aux(base) = **ICS**(BCSIM, inclui_base, base)

As ocorrências de uma simulação de projeto devem ser armazenadas para que no final da simulação sejam avaliados os processos definidos para os projetos simulados. As operações que geram ocorrências para o relatório são:

Interface:

gera_execução_simulador(_,_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
gera_execução_agente(_,_,_,_,_): SIMULADOR, STRING, AGENTE-DESENV, ATIVIDADE, STRING, REAL -> SIMULADOR

Operações

Variáveis Formais:

agentes_desenv	: AGENTES-DESENV
atividade	: ATIVIDADE
id_agente, nome_projeto, evento	: STRING
tempo_evento	: REAL

gera_execução_simulador((_,_,_,_,Relatório relatório,_), nome_projeto, evento, tempo_evento)
= (_,_,_,_,Relatório **ICS**(RELATÓRIO, *gera_ocorrência*, relatório, < "simulador", nome_projeto, "", evento, tempo_evento>),_)

gera_execução_agente((_,_,_,_,Relatório relatório,_), nome_projeto, agentes_desenv, atividade, evento, tempo_evento)
= (_,_,_,_,Relatório **ICS**(RELATÓRIO, *gera_ocorrência*, relatório, <**ICS**(AGENTES-DESENV, id_agente,agentes_desenv), nome_projeto, **ICS**(ATIVIDADE, nome_atividade, atividade), evento, tempo_evento>),_)

As atividades padrão são definições de atividades prontas para o reuso na definição de processos, que são disponibilizadas pelo ambiente de simulação. A definição dessas atividades é realizada através de um nome para a atividade, uma descrição e um conjunto de características. As operações do ATO Simulador sobre as atividades definidas são:

Interface:

cria_atividade_padrão(_,_,_): SIMULADOR, STRING, TEXTO	-> SIMULADOR
exclui_atividade_padrão(_,_): SIMULADOR, STRING	-> SIMULADOR
altera_descrição_ativ_padrão(_,_,_): SIMULADOR, STRING, TEXTO	-> SIMULADOR
inclui_característica_ativ_padrão(_,_,_): SIMULADOR, STRING, STRING	-> SIMULADOR
exclui_característica_ativ_padrão(_,_,_): SIMULADOR, STRING, STRING	-> SIMULADOR

6.3.2  erações da simulação de processos

Para representar a simulação de processos de software é necessário definir operações que atendam a requisição do usuário para definir os projetos que serão simulados, representar as ações dos agentes-desenvolvedores para iniciar e finalizar atividade, além das operações que dão suporte a realização das operações principais.

Para realizar a simulação, o gerente identifica quais projetos serão simulados, utilizando a operação **definir_simulação_projeto**. Com os projetos a serem simulados o gerente da simulação pode dar início a simulação utilizando a operação simular. Essa operação utiliza a operação **teste_realizar_ciclo** para realizar os ciclos necessários para simulação e isso servirá de parâmetro para a operação **emitir_relatório_aux** que através dos eventos adicionados ao campo relatório irá gerar uma lista de avaliações sobre a simulação.

As operações de simulação são definidas assim:

Interface:

definir_simulação_projeto(_,_): SIMULADOR, STRING	-> SIMULADOR
simular(_): SIMULADOR	-> LISTA_STRING
teste_realizar_ciclo(_): SIMULADOR	-> SIMULADOR
invocar_ciclo(_):SIMULADOR	-> SIMULADOR
atualiza_processo(_,_): SIMULADOR, STRING	-> SIMULADOR
atualiza_todos_processos(_,_): SIMULADOR	-> SIMULADOR
atualiza_agenda(_,_): SIMULADOR, STRING	-> SIMULADOR
atualiza_todas_agendas(_): SIMULADOR	-> SIMULADOR
emitir_relatório_aux(_): SIMULADOR	-> LISTA_STRING

Operações

Variáveis Formais:

simulador	: SIMULADOR
agentes_desenv	: CONJ_AGENTES-DESENV.
projetos	: PROJETOS
base	: BCSIM
relatório	: RELATÓRIO
recursos	: RECURSOS
nome_projeto	: STRING

 ir_simulação_projeto((_,_,Projetos projetos ,_, Relatório relatório,_),nome_projeto)
atualiza_agenda (*atualiza_processo*(*gera_execução_simulador*((_,_,Projetos projetos,_, Relatório relatório,_), nome-projeto, "inicio da simulação", **ICS**(RELATÓRIO, tempo_último_evento, relatório)), nome_projeto), nome_projeto)

simular(simulador) = *emitir_relatório_aux*(*teste_realizar_ciclo*(simulador))

teste_realizar_ciclo((_,_, Projetos projetos, BCSIM base,_,_))
= **if not** (*testa_processos_concluídos*(projetos) **and ICS**(BCSIM, consulta_base_conhecimento, base))
then *invocar_ciclo*((_,_,Projetos projetos,_,_,_)) **else** (_,_, Projetos projetos,_,_,_)

invocar_ciclo((_, Agentes-Desenv agentes_desenv,_,_, Relatório relatório,_))
= *teste_realizar_ciclo*(*atualiza_todas_agendas*(*atualiza_todos_processos*
(*gera_execução_simulador*(*executa_agentes_desenv*((_, Agentes-Desenv agentes_desenv,_,_,
Relatório relatório,_), agentes_desenv), "", "atualização do tempo de simulação",
ICS(RELATÓRIO, tempo_último_evento, relatório) + 1))))

atualiza_processo((_,_, Projetos projetos,_,_,_),nome_projeto)
= (_,_, Projetos **ICS**(PROJETOS, atualiza_processo, projetos, <nome_projeto>),_,_,_)

atualiza_todos_processos((_,_, Projetos projetos,_,_,_))
= (_,_, Projetos *atualiza_processos_aux*(projetos),_,_,_)

atualiza_agenda((_, Agentes-Desenv agentes_desenv, Projetos projetos,_,_,_),nome_projeto)
= (_, Agentes-Desenv *atualiza_agenda_agentes_des*(**ICS**(PROJETOS, processo, projetos,
<nome_projeto>), agentes_desenv, nome_projeto), Projetos projetos,_,_,_)

atualiza_todas_agendas((_,Agentes-Desenv agentes_desenv, Projetos projetos,_,_,_))
= (_,Agentes-Desenv *atualiza_agendas_aux*(agentes_desenv, projetos), Projetos projetos,_,_,_)

emitir_relatório_aux((_,_,_,_,Relatório relatório,_)) = **ICS**(RELATÓRIO, emitir_relatório, relatório)

Na definição das operações de simulação, são utilizadas as seguintes operações auxiliares:

Interface:

testa_processos_concluídos(_): PROJETOS -> BOOLEAN
atualiza_agenda_agentes_des(_,_,_): PROCESSO, AGENTES-DESENV, STRING -> AGENTES-DESENV
atualiza_processos_aux(_,_): PROJETOS -> PROJETOS
atualiza_agendas_aux(_,_): AGENTES-DESENV, PROJETOS ->AGENTES-DESENV

Operações

Variáveis Formais:

agentes_desenv : AGENTES-DESENV
projetos : PROJETOS
atividades : CONJ_ATIVIDADES
atividade : ATIVIDADE
nome_projeto, id_agente : STRING

testa_processos_concluídos(modify(nome_projeto,_, projetos)
= **if ICS** (PROCESSO, processo_concluído, **ICS**(PROJETO, processo, projetos, nome_projeto))
then *testa_processos_concluídos*(projetos)
else FALSE

testa_processos_concluídos(empty-mapping) = TRUE

atualiza_agenda_agentes_des(add(atividades, atividade), modify(id_agente,_,agentes_desenv), nome_projeto)
= *atualiza_agenda_agentes_des*(add(empty-set, atividade), modify(id_agente,_,empty-mapping), nome_projeto) ∪ *atualiza_agenda_agentes_des*(add(empty-set, atividade), agentes_desenv, nome_projeto) ∪ *atualiza_agenda_agentes_des*(atividades, modify(id_agente,_,empty-mapping), nome_projeto) ∪ *atualiza_agenda_agentes_des*(atividades, agentes_desenv, nome_projeto)

atualiza_agenda_agentes_des(add(empty-set, atividade), modify(id_agente,_, empty-mapping), nome_projeto)
= **if** agente-desenv ∈ (ATIVIDADE, conjunto_agentes, atividade) **and** (**ICS**(ATIVIDADE, estado_ativ, atividade) = "pronta" **or ICS**(ATIVIDADE, anteriores_de_ativ, atividade) = empty-set)
then ICS(AGENTES-DESENV, inclui_ativ_projeto, ICS(AGENTES-DESENV, inclui_projeto_agenda, modify(id_agente,_, empty-mapping), <id_agente, nome_projeto>), <id_agente, nome_projeto, **ICS**(ATIVIDADE, nome_atividade, atividade), **ICS**(ATIVIDADE, estado_atividade, atividade))
else modify(id_agente,_, empty-mapping)

atualiza_agenda_agentes_des(add(empty-set, atividade), modify(id_agente,_, agentes_desenv), nome_projeto)
= *atualiza_agenda_agentes_des*(add(empty-set, atividade), modify(id_agente,_, empty-mapping), nome_projeto) ∪ *atualiza_agenda_agentes_des*(add(empty-set, atividade), agentes_desenv, nome_projeto)

atualiza_agenda_agentes_des(add(atividades, atividade), modify(id_agente,_,empty-mapping),nome_projeto)
= *atualiza_agenda_agentes_des*(add(empty-set, atividade), modify(id_agente,_, empty-mapping), nome_projeto) ∪ *atualiza_agenda_agentes_des*(atividades, modify(id_agente,_, empty-mapping), nome_projeto)

atualiza_processos_aux(modify(nome_projeto,_, projetos))
= Merge(**ICS**(PROJETOS, atualiza_processo, modify(nome_projeto,_, empty-mapping), < nome_projeto>), *atualiza_processos_aux*(projetos))

atualiza_processos_aux(empty-mapping) = empty-mapping

atualiza_agendas_aux(agentes_desenv, modify(nome_projeto ,_, projetos)
= **if ICS**(PROJETO, projeto_iniciado, modify(nome_projeto, _, empty-mapping), <nome_projeto>)
then *atualiza_agendas_aux*(*atualiza_agenda_agentes_des*(**ICS**(PROETOS, processo, modify(nome_projeto,_,empty-mapping), nome_projeto), agentes_desenv, nome_projeto), projetos))
else *atualiza_agendas_aux*(agentes_desenv, projetos)

atualiza_agendas_aux(agentes_desenv, empty-mapping) = agentes_desenv

Quando os projetos são definidos para serem simulados seus processos são atualizados e as atividades que estiverem prontas para serem realizadas são colocadas nas agendas dos respectivos agentes-desenvolvedores responsáveis. Além disso é gerada uma ocorrência no relatório do simulador dando conta do início da simulação para o projeto.

6.3.3 Execução dos agentes-desenvolvedores

Quando o gerente da simulação utiliza a operação **realizar_simulação**, o simulador testa os processos de todos os projetos para verificar se existe projeto a ser simulado e também realiza uma consulta a sua base de conhecimento. Se os testes estiverem "ok", o simulador realiza a operação auxiliar **invocar_ciclo**. A realização alternada das operações **realizar_simulação** e **invocar_ciclo** controla os ciclos da simulação. Em cada ciclo de simulação o controle é passado para cada agente-desenvolvedor definido no ambiente. A operação responsável por passar o controle para

cada agente-desenvolvedor é a **executa_agentes_desenv** que utiliza a operação **executa_agente_desenv** para isso. Essas operações são definidas assim:

Interface:

executa_agentes_desenv(_): SIMULADOR, AGENTES-DESENV -> SIMULADOR
executa_agente_desenv(_): SIMULADOR, STRING -> SIMULADOR

Operações

Variáveis Formais:

agentes_desenv : CONJ_AGENTES-DESENV.
base : BCSIM
relatório : RELATÓRIO
id_agente : STRING

executa_agentes_desenv((_, Agentes-Desenv agentes_desenv,_,BCSIM base,_,_), modify(id-agente,_, agentes_desenv))
= **if ICS**(BCSIM, consulta_base_conhecimento, base, < "regra verifica ambiente">)
then *executa_agentes_desenv*(*executa_agente_desenv*(_, Agentes-Desenv agentes_desenv, _,BCSIM base,_,_), id_agente), agentes_desenv)
else (_, Agentes-Desenv agentes_desenv,_,BCSIM base,_,_)

executa_agentes_desenv((_, Agentes-Desenv agentes_desenv,_, _,_,_), empty-mapping)
= (_, Agentes-Desenv agentes_desenv,_,_,_,_)

executa_agente_desenv((_, Agentes-Desenv agentes_desenv,_, BCSIM base, Relatório relatório,_), id_agente)
= **if ICS**(AGENTES-DESENV, retorna_tpe, agentes_desenv, <id_agente>) = **ICS**(RELATÓRIO, tempo_último_evento, relatório) **and ICS**(BCSIM, consulta_base_conhecimento, base, < "regra executa agente">)
then if ICS(AGENTES-DESENV, retorna_pe, agentes_desenv, <id_agente>) = "iniciar"
then if ICS (AGENTES-DESENV, existe_atividade_pronta, agentes_desenv, <id_agente>)
then if ICS(AGENTES-DESENV, existem_atividades_prontas, agentes_desenv, <id_agente>)
then *executa_atividade*((_, Agentes-Desenv agentes_desenv,_,BCSIM base, Relatório relatório,_), id_agente, **ICS**(AGENTES-DESENV, escolher_atividade_projeto, agentes_desenv, <id_agente>))
else *executa_atividade*((_, Agentes-Desenv agentes_desenv,_,BCSIM base, Relatório relatório,_), id_agente, Union(**ICS**(AGENTES-DESENV, retorna_atividade_pronta, agentes_desenv, id_agente), **ICS**(AGENTES_DESENV, retorna_projeto_pronto, agentes_desenv, id_agente)))
else (_, Agentes-Desenv **ICS**(AGENTES-DESENV, próximo_ciclo, agentes_desenv, <id_agente>),_,BCSIM base,Relatório relatório,_)
else *finaliza_atividade*((_, Agentes-Desenv agentes_desenv,_,BCSIM base, Relatório relatório, _), id_agente, **ICS**(AGENTES-DESENV, retorna_atividade_ativa, agentes_desenv, id_agente), **ICS**(AGENTES-DESENV, retorna_projeto_ativo, agentes_desenv, id_agente))
else (_, Agentes-Desenv agentes_desenv,_,BCSIM base,Relatório relatório,_)

6.3.4 Operações de execução e finalização de atividade

As operações **executa_atividade** e **finaliza_atividade** são utilizadas para atender as requisições dos agentes-desenvolvedores para iniciar a execução e finalizar a execução de uma atividade. O simulador é responsável por verificar a disponibilidade de recursos quando é realizada uma requisição para iniciar a atividade e também é responsável por liberar os recursos quando uma atividade é finalizada. O simulador é

responsável também por atualizar o estado da atividade.

As operações **executa_atividade** e **finaliza_atividade** são definidas assim:

Interface:

executa_atividade(_,_): SIMULADOR, STRING, (STRING ∪ STRING) -> SIMULADOR
finaliza_atividade(_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR

Operações

Variáveis Formais:

agentes_desenv : CONJ_AGENTES-DESENV.
projetos : PROJETOS
recursos : RECURSOS
nome_ativiade, nome_projeto, id_agente : STRING

executa_atividade((Recursos recursos, Agentes-Desenv agentes_desenv, Projetos projetos,_,_,_), id_agente, Union(nome_atividade, nome_projeto))
= **if** *estado_ativ_folha_aux*(projetos, nome_projeto, nome_atividade) = "ativa"
then *executa_atividade_aux*((Recursos recursos, Agentes-Desenv **ICS**(AGENTES-DESENV, executa_atividade_agente, agentes_desenv, <projetos, id_agente, nome_projeto, nome_atividade>), Projetos projetos,_,_,_), id_agente, nome_atividade, nome_projeto)
else if *recursos_disponiveis_aux*(recursos, projetos, nome_projeto, nome_atividade)
then *executa_atividade_aux*((Recursos *altera_estado_recursos*(recursos, projetos, nome_projeto, nome_atividade, "bloqueado"), Agentes-Desenv **ICS**(AGENTES-DESENV, executa_atividade_agente, agentes_desenv, <projetos, id_agente, nome_projeto, nome_atividade>), Projetos altera_estado_ativ(projetos, nome_atividade, nome_projeto, "ativa"),_,_,_), id_agente, nome_atividade, nome_projeto)
else (Recursos recursos, Agentes-Desenv **ICS**(AGENTES-DESENV, próximo_ciclo, agentes_desenv, <id_agente>), Projetos projetos,_,_,_)

finaliza_atividade(_,Agentes-Desenv agentes_desenv,_,_,Relatório relatório,_), id_agente, nome_atividade, nome_projeto)
= *gera_execução_agente*(*finalizar_atividade_agenda_aux*((_, Agentes-Desenv **ICS**(AGENTES-DESENV, finaliza_atividade_agente, agentes_desenv, <id_agente>),_,_, Relatório relatório,_), id_agente, nome_projeto, nome_atividade), nome_projeto, id_agente, nome_atividade, "Agente informa fim da execução de atividade", **ICS**(RELATÓRIO, tempo_último_evento, relatório))

Na definição das operações para iniciar e finalizar atividades são utilizadas as seguintes operações auxiliares:

Interface:

executa_atividade_aux(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
executar_atividade_agenda_aux(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
finalizar_atividade_agenda_aux(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
estado_ativ_folha_aux(_,_,_):PROJETOS, STRING, STRING -> BOOLEAN
recursos_disponiveis_aux(_,_,_,_): CONJ_RECURSOS, PROJETOS, STRING, STRING -> BOOLEAN
recursos_disponíveis(_,_): CONJ_RECURSOS, CONJ_STRING -> BOOLEAN
recurso_disponível(_,_): CONJ_RECURSOS, STRING -> BOOLEAN
altera_estado_recursos(_,_,_,_,_): CONJ_RECURSOS, PROJETOS, STRING, STRING, STRING -> CONJ_RECURSOS
altera_estado_recurso(_,_,_): RECURSO, CONJ_STRING, STRING -> RECURSO
altera_estado_ativ(_,_,_,_): PROJETOS, STRING, STRING, STRING -> PROJETOS

verificar_termino_real(_,_,_,_,_): AGENTES-DESENV, STRING, STRING
STRING, CONJ_STRING -> BOOLEAN

verificar_termino_aux(_,_,_,_):AGENTES-DESENV, STRING, STRING
STRING -> BOOLEAN

Operações

Variáveis Formais:

agentes_desenv : AGENTES-DESENV
projetos : PROJETOS
relatório : RELATÓRIO
recursos : CONJ_RECURSOS
recurso : RECURSO
conj_nomes_rec, conj_agentes : CONJ_STRING
nome_projeto, nome_proj, id_agente, agente
nome_rec, nome_atividade, estado : STRING

executa_atividade_aux((_,_,_,_,Relatório relatório,_), id_agente, nome_atividade, nome_projeto)
= *gera_execução_agente*(*executar_atividade_agenda_aux*((_,_,_,_,Relatório relatório,_), id-agente, nome_projeto, nome_atividade), nome_projeto, id_agente, nome_atividade, "Agente requisita execução de atividade", **ICS**(RELATÓRIO, tempo_último_evento, relatório))

executar_atividade_agenda_aux((_,Agentes-Desenv agentes_desenv,_,_,Relatório relatório,_), id_agente, nome_projeto, nome_atividade)
= *gera_execução_simulador*((_, Agentes-Desenv **ICS**(AGENTES-DESENV, altera_estado_atividade, agentes_desenv,<id_agente, nome_projeto, nome_atividade, "ativa">),_,_,_Relatório relatório,_), nome_projeto, "Simulador executa atividade para agente", **ICS**(RELATÓRIO, tempo_último_evento, relatório))

finalizar_atividade_agenda_aux((Recurso recursos, Agentes-Desenv agentes_desenv, Projetos projetos, _,_,_), id_agente, nome_projeto, nome_atividade)
= **if** *verificar_termino_real*(agentes_desenv, id_agente, nome_projeto, nome_atividade, **ICS**(ATIVIDADE, conjunto_agentes, **ICS**(PROJETOS, acha_atividade, projetos, <nome_projeto, nome_atividade>)))
then *gera_execução_simulador*((Recursos *altera_estado_recursos*(recursos, projetos**,** nome_projeto, nome_atividade, "disponível"), Agentes-Desenv **ICS**(AGENTES-DESENV, altera_estado_atividade, agentes_desenv, <id_agente, nome_projeto, nome_atividade, "completa">), Projetos *altera_estado_ativ*(projetos, **ICS**(ATIVIDADE, nome_atividade, **ICS**(PROJETOS, acha_atividade, projetos, <nome_projeto, nome_atividade>)), nome_projeto, "completa"),_,_,_), nome_projeto, "Simulador finaliza execução de atividade", **ICS**(RELATÓRIO, tempo_último_evento, relatório))
else *gera_execução_simulador* ((Recurso recursos, Agentes-Desenv **ICS**(AGENTES-DESENV, altera_estado_atividade, agentes_desenv, <id_agente, nome_projeto, nome_atividade, "completa">), Projetos projetos,_,_,_) nome_projeto, "Simulador executa atividade para agente", **ICS**(RELATÓRIO, tempo_último_evento, relatório))

estado_ativ_folha_aux(projetos, nome_projeto, nome_atividade)
= **if ICS**(PROJETOS, estado_atividade, projetos, <nome_projeto, **ICS**(PROJETOS, acha_atividade, projetos, <nome_projeto, nome_atividade>)>) = "ativa"
then TRUE
else FALSE

recursos_disponiveis_aux(recursos, projetos, nome_projeto, nome_atividade)
= **if** recursos_disponíveis(recursos, **ICS**(ATIVIDADE, recursos_ativ, **ICS**(PROJETOS, acha_atividade, projetos, <nome_projeto, nome_atividade>)))
then TRUE
else FALSE

```
recursos_disponíveis(recursos, add(conj_nomes_rec, nome_rec))
=       recurso_disponível(recursos, nome_rec) and
        recursos_disponíveis(recursos, conj_nomes_rec)

recursos_disponíveis(recursos, empty_set) = TRUE

recurso_disponível(recursos, nome-rec)
=       if ICS(RECURSO, estado_recurso, retorna_recurso(recursos, nome-rec)) = "disponível"
        then TRUE
        else FALSE

altera_estado_recursos(add(recursos, recurso), projetos, nome_projeto,nome_atividade, estado)
=       add(altera_estado_recurso(recurso, ICS(ATIVIDADE, recursos_ativ, ICS(PROJETOS,
        acha_atividade, projetos, <nome_projeto, nome_atividade>)), estado),
        altera_estado_recursos(recursos, projetos, nome_projeto, nome_atividade, estado))

altera_estado_recursos(empty-set,_,_,_,_) = empty-set

altera_estado_recurso(recurso, add(conj_nomes_rec, nome_rec), estado)
=       if ICS(RECURSO, nome_recurso, recurso) = nome_rec
        then ICS(RECURSO, altera_estado_rec, recurso, <estado>)
        else altera_estado_recurso(recurso, conj_nomes_rec, estado)

altera_estado_recurso(recurso, empty_set, _) = recurso

altera_estado_ativ(modify(nome_proj,_,projetos), nome_atividade, nome_projeto, estado)
=       if nome_proj = nome_projeto
        then ICS(PROJETOS, altera_estado_ativ_projeto, nome_projeto, <nome_atividade, estado>)
        else modify(nome_proj,_, altera_estado_ativ(projetos, nome_atividade, nome_projeto, estado)

altera_estado_ativ(empty-mapping,_,_,_) = empty-mapping

verificar_termino_real(agentes_desenv, id_agente, nome_projeto, nome_atividade, add(agente,
conj_agentes))
=       if id_agente = agente
        then verificar_termino_real(agentes_desenv, id_agente, nome_projeto, nome_atividade, agentes)
        else verfificar_termino_aux(agentes_desenv, agente, nome_projeto, nome_atividade) and
             verificar_termino_real(agentes_desenv, id_agente, nome_projeto, nome_atividade, agentes)

verificar_termino_real(_,_,_,_,empty-set) = TRUE

verficar_termino_aux(agentes_desenv, id_agente, nome_projeto, nome_atividade)
=       if ICS(AGENTES-DESENV, estado_atividade_agente, agentes_desenv, <id_agente,
                nome_projeto, nome_atividade>) = “completa”
        then TRUE
        else FALSE
```

6.4 ATO Recurso

A classe recurso foi construída através das operações do tipo registro e união disjunta, como demonstra a figura 6.5. O ATO Recurso que manipula essa classe foi construído originalmente como ferramenta de apoio ao ATO GP (gerenciador de processos) [LIM98] e reutilizado pelo ATO Simulador. O campo MTBF (*mean time between failure*- tempo médio entre falhas) foi adicionado ao ATO Recurso.

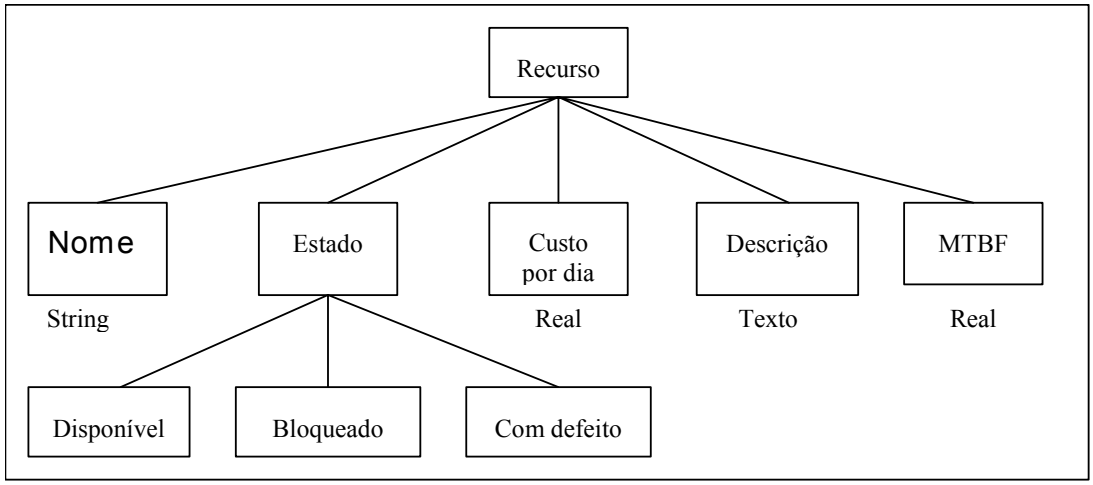

FIGURA 6.5 - Classe do ATO Recurso

A interface do ATO Recurso é apresentada a seguir. As operações e uma descrição mais detalhada está definida em [LIM98].

Interface:

cria_recurso(_,_,_,_): STRING, STRING, REAL, TEXTO	-> RECURSO
altera_estado_rec(_,_): RECURSO, STRING	-> RECURSO
altera_custo(_,_): RECURSO, REAL	-> RECURSO
altera_descrição(_,_): RECURSO, TEXTO	-> RECURSO
altera_mtbf(_,_): RECURSO, REAL	-> RECURSO
nome_recurso(_): RECURSO	-> STRING
estado_recurso(_): RECURSO	-> STRING
custo_por_dia(_): RECURSO	-> REAL
mtbf_recurso(_): RECURSO	-> REAL

6.5 ATO Agentes-Desenv

O ATO Agentes-Desenv foi criado para gerenciar as informações relativas aos agentes-desenvolvedores cadastrados no ambiente de simulação. Com esse ATO é posssível definir para cada agente-desenvolvedor o seu custo por hora e trabalho, assim como sua agenda, onde são colocadas as atividades em que ele precisa participar. São definidas também as habilidades e afinidades definidas, que são utilizadas no cálculo do tempo simulado para execução das atividades da agenda.

A classe agente-desenv foi construída a partir das operações mapeamento, lista e registro. Um agente-desenv é um mapeamento de uma string, que define o seu identificador para um registro chamado característica. O registro característica possui sete campos: agenda, custo_por_hora, habilidades, afinidades, bc-agente, tpe e pe. A agenda por sua vez é um mapeamento de uma string (nome_projeto) para uma lista (agenda_projeto) que é composta por registro de atividades. O campo custo_por_hora é do tipo real. O campo habilidade é um mapeamento de uma string (característica relacionada) para um real (grau de habilidade), o campo afinidade é um mapeamento de uma string (identificador do agente-desenvolvedor relacionado) para um real (grau de afinidade), o campo bc-agente é do tipo BCSIM e define a base de conhecimento do agente-desenvolvedor e os campos tpe e pe indicam o tempo do próximo evento e o próximo evento respectivamente, esses campos são utilizados no controle da simulação.

A classe agente-desenv é apresentada na figura 6.6 e as operações são apresentadas em seguida e suas definições estão no anexo.

Interface:

cria_agente_des: ->AGENTES-DESENV
inclui_agente_des(_,_,_):AGENTES-DESENV, STRING, REAL, BCI ->AGENTES-DESENV
exclui_agente_des(_,_):AGENTES-DESENV, STRING ->AGENTES-DESENV
inclui_projeto_agenda(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
exclui_projeto_agenda(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
inclui_ativ_projeto(_,_,_,_,_):AGENTES-DESENV, STRING, STRING, STRING, STRING ->AGENTES-DESENV
exclui_ativ_projeto(_,_,_,_):AGENTES-DESENV,STRING,STRING,STRING->AGENTES-DESENV
altera_estado_atividade(_,_,_,_,_):AGENTES-DESENV, STRING, STRING, STRING, STRING ->AGENTES-DESENV
altera_custo(_,_,_):AGENTES-DESENV, STRING, REAL ->AGENTES-DESENV
inclui_habilidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
exclui_habilidade(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
altera_habilidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
inclui_afinidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
exclui_afinidade(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
altera_afinidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV

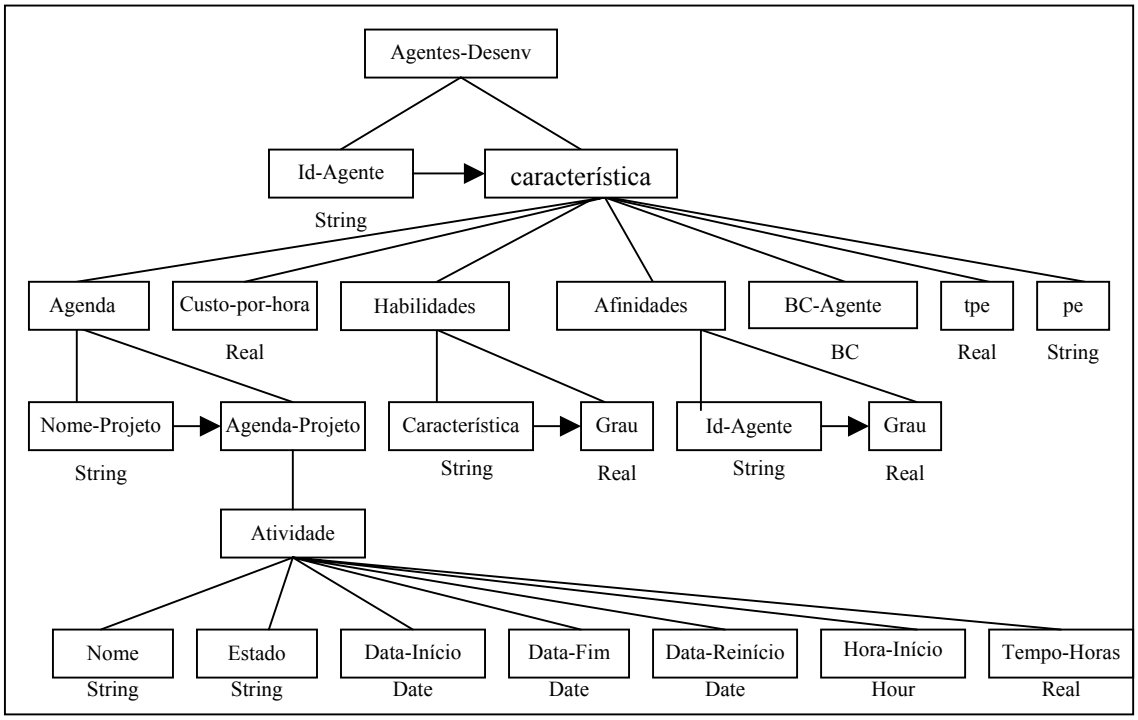

FIGURA 6.6 - Classe agente-desenv

O ATO AGENTES-DESENV é requisitado pelo ATO SIMULADOR para realizar operações da simulação do processo. A seguir são apresentadas operações do ATO AGENTES-DESENV que são requisitadas por operações de simulação.

Interface:

executa_atividade_agente(_,_,_,_): AGENTES-DESENV, PROJETOS, STRING, STRING, STRING -> AGENTES-DESENV
próximo_ciclo(_,_): AGENTES-DESENV, STRING -> AGENTES-DESENV
finaliza_atividade_agente(_,_): AGENTES-DESENV, STRING -> AGENTES-DESENV
calcula_tempo_atividade(_,_,_,_,_,_): PROJETOS, STRING, STRING, HABILIDADES, AFINIDADES, BCSIM -> REAL

tempo_previsto_atividade(_,_,_): PROJETOS, STRING, STRING -> REAL
habilidade_geral_agente(_,_,_,_): HABILIDADES, PROJETOS,STRING, STRING -> REAL
afinidade_geral_agente(_,_,_,_): AFINIDADES, PROJETOS, STRING, STRING -> REAL
calcula_habilidade_média(_,_): HABILIDADES, CONJ_STRING -> REAL
soma_habilidades(_,_): HABILIDADES, CONJ_STRING -> REAL
conta_habilidades(_,_): HABILIDADES, CONJ_STRING -> REAL
calcula_afinidade_média(_,_): AFINIDADES, CONJ_STRING -> REAL
soma_afinidades(_,_): AFINIDADES, CONJ_STRING -> REAL
conta_afinidades(_,_): AFINIDADES, CONJ_STRING -> REAL

Operações

Variáveis Formais:

agentes_desenv : AGENTES-DESENV
id-agente, agente, nome_projeto,
nome_atividade, pe, característica : STRING
características, conj_agentes : CONJ_STRING
tpe, grau : REAL
projetos : PROJETOS
habilidades : HABILIDADES
afinidades : AFINIDADES
base : BCSIM

executa_atividade_agente(modify(agente,(_,_,Habilidades habilidades,Afinidades afinidades, BC-Agente base,tpe tpe, pe pe), agentes_desenv), projetos, id_agente, nome_projeto, nome_atividade)
= **if** id_agente = agente
then modify(agente, (_,_,_,_,_,tpe tpe + *calcula_tempo_atividade*(projetos,nome_projeto, nome_atividade, habilidades, afinidades, base), pe "finalizar"), agentes_desenv)
else modify(agente, (_,_,_,_,_,tpe tpe, pe pe), *executa_atividade_agente*(agentes_desenv, id_agente, nome_projeto, nome_atividade))

executa_atividade_agente(empty-mapping,_,_,_,_) = empty-mapping

próximo_ciclo(modify(agente, (_,_,_,_,_,tpe tpe,_), agentes_desenv), id_agente)
= **if** id_agente = agente
then modify(agente, (_,_,_,_,_,tpe tpe + 1,_), agentes_desenv)
else modify(agente, (_,_,_,_,_,tpe tpe,_), *próximo_ciclo*(agentes_desenv, id_agente))

próximo_ciclo(empty-mapping,_) = empty-mapping

finaliza_atividade_agente(modify(agente, (_,_,_,_,_,tpe tpe,pe pe), agentes_desenv), id_agente)
= **if** id_agente = agente
then modify(agente, (_,_,_,_,_,tpe tpe + 1, pe "iniciar"), agentes-desenv)
else modify(agente, (_,_,_,_,_,tpe tpe,pe pe), *finaliza_atividade_agente*(agentes_desenv, id_agente))

finaliza_atividade_agente(empty-mapping,_) = empty-mapping

calcula_tempo_atividade(projetos, nome_projeto, nome_atividade, habilidades, afinidades, base)
= *tempo_previsto_atividade*(projetos, nome_projeto, nome_atividade) *
habilidade_geral_agente(habilidades, projetos, nome_projeto, nome_atividade) *
afinidade_geral_agente(afinidades, projetos, nome_projeto, nome_atividade) *
ICS(BCSIM, cálculo_base_conhecimento, base, "tempo atividade",
add(nome_atividade,add(nome_projeto, empty-set)))

```
tempo_previsto_atividade(projetos, nome_projeto, nome_atividade)
=       ICS(ATIVIDADE, previsão_fim_ativ, ICS(PROJETOS, acha_atividade, projetos,
        <nome_projeto, nome_atividade>)) - ICS(ATIVIDADE, previsão_inicio_ativ,
        ICS(PROJETOS, acha_atividade, projetos, <nome_projeto, nome_atividade>))

habilidade_geral_agente(habilidades, projetos, nome_projeto, nome_atividade)
=       calcula_habilidade_média(habilidades, ICS(ATIVIDADE,
        retorna_características_atividade, ICS(PROJETOS, acha_atividade, projetos, <nome_projeto,
        nome_atividade>)))

afinidade_geral_agente(afinidades, projetos, nome_projeto, nome_atividade)
=       calcula_afinidade_média(afinidades, ICS(ATIVIDADE, conjunto_agentes, ICS(PROJETOS,
        acha_atividade, projetos, <nome_projeto, nome_atividade>)))

calcula_habilidade_média(habilidades, características)
=       soma_habilidades(habilidades, características) / conta_habilidades(habilidades, características)

soma_habilidades(modify(característica, grau, habilidades), características)
=       if Is_in(características, característica)
        then grau + soma_habilidades(habilidades, características)
        else soma_habilidades(habilidades, características)

soma_habilidades(empty-mapping,_) = 0

conta_habilidades(modify(característica,_, habilidades), características)
=       if Is_in(características, característica)
        then 1 + conta_habilidades(habilidades, características)
        else conta_habilidades(habilidades, características)

conta_habilidades(empty-mapping,_) = 0

calcula_afinidade_média(afinidades, conj_agentes)
=       soma_afinidades(afinidades, conj_agentes) / conta_afinidades(afinidades, conj_agentes)

soma_afinidades(modify(id_agente, grau, afinidades), conj_agentes)
=       if Is_in(id_agente, conj_agentes)
        then grau + soma_afinidades(afinidades, conj_agentes)
        else soma_afinidades(afinidades, conj_agentes)

soma_afinidades(empty-mapping,_) = 0

conta_afinidades(modify(id_agente,_, afinidades), conj_agentes)
=       if Is_in(id_agente, conj_agentes)
        then 1 + conta_afinidades(afinidades, conj_agentes)
        else conta_afinidades(afinidades, conj_agentes)

conta_afinidades(empty-mapping,_) = 0
```

As operações mais importantes são **executa_atividade_agente** e **finaliza_atividade_agente**. Elas são responsáveis por representar as duas ações que um agente-desenvolvedor pode realizar em um ambiente de simulação: requisitar o início da execução de uma simulação e requisitar seu termino.

6.6 ATO Projetos

O ATO Projetos foi definido em [LIM98] para gerenciar todas as informações sobre os processos de desenvolvimento de software. Foi criado para dar apoio a uma ferramenta de gerência de processos. Através desse ATO são possíveis o armazenamento e gerenciamento das métricas, estimativas e processos definidos em projetos.

Para ser utilizado pelo ATO Simulador todas as definições originais do ATO Projetos foram mantidas, sendo incorporada a definição e gerenciamento dos cargos atribuídos aos agentes-desenvolvedores na simulação da execução de um processo de software. A definição de cargos diretamente relacionados ao projeto permite ao modelo testar diversas possibilidades de escalonamento de agentes por cargos exercidos em cada projeto permitindo avaliação do desempenho dos agentes-desenvolvedores através do cargo exercido.

A figura 6.7 apresenta a classe projeto estendida para utilização pelo ATO Simulador.

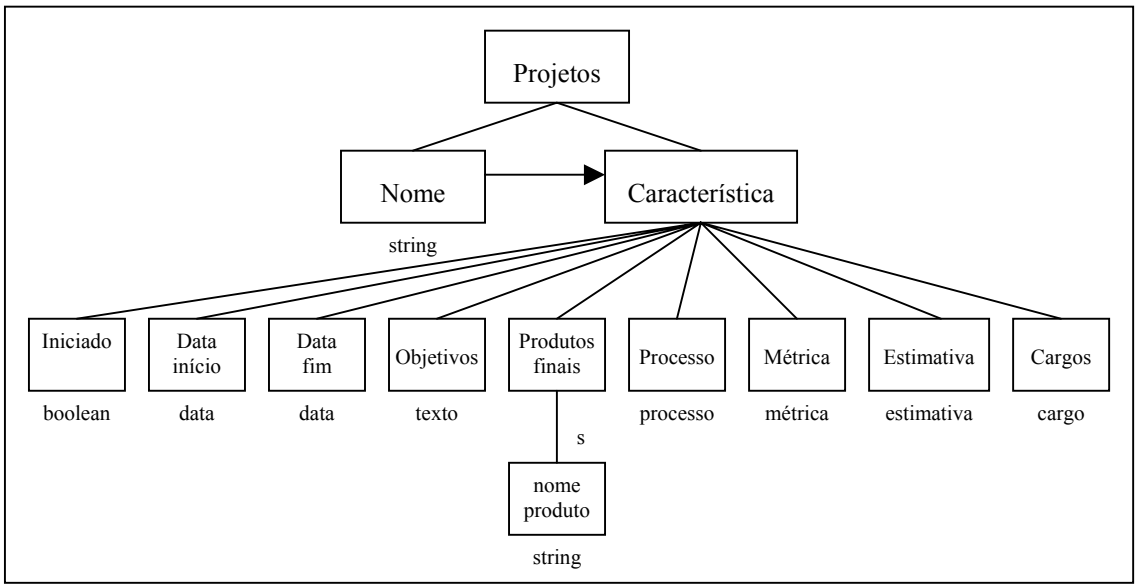

FIGURA 6.7 - Classe projeto

O ATO Projetos foi definido através das operações mapeamento e registros, além de utilizar operações dos ATOS Processo, Métrica e Estimativa que são definidos em [LIM98] e o ATO Cargo definido na seção 6.10. A seguir é apresentada a interface do ATO Projeto e as novas operações para atender o ATO Simulador são detalhadas. O detalhamento das demais operações e uma descrição mais completa são apresentados em [LIM98].

Interface:

cria_projetos:	-> PROJETOS
inclui_projeto(_,_,_,_,_): PROJETOS, STRING, DATE, DATE, TEXTO	-> PROJETOS
exclui_proeto(_,_): PROJETOS, STRING	-> PROJETOS
inclui_prodf(_,_,_):PROJETOS, STRING, STRING	-> PROJETOS
exclui_prodf(_,_,_):PROJETOS, STRING, STRING	-> PROJETOS
altera_nome(_,_,_):PROJETOS, STRING, STRING	-> PROJETOS
altera_data_ini(_,_,_):PROJETOS, STRING, DATE	-> PROJETOS
altera_data_fim(_,_,_):PROJETOS, STRING, DATE	-> PROJETOS

altera_objetivos(_,_,_):PROJETOS, STRING, TEXTO -> PROJETOS
altera_processo(_,_,_):PROJETOS, STRING, PROCESSO -> PROJETOS
altera_métricas(_,_,_):PROJETOS, STRING, MÉTRICA -> PROJETOS
altera_estimativas(_,_,_):PROJETOS, STRING, ESTIMATIVA -> PROJETOS
altera_cargos(_,_,_):PROJETOS, STRING, CARGO -> PROJETOS

A operação altera_cargos não faz parte a definição original. Ela é definida da seguinte forma:

Operação

Variáveis Formais:

projetos : PROJETOS
nome_proj, nomep : STRING
cargos, novos_cargos : CARGO

altera_cargos(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, novos_cargos)
= **if** nome_proj = nomep
then modify(nomep, , (_,_,_,_,_,_,_,_,_, Cargos novos_cargos)
else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *altera_cargos*(projetos, nome_proj, novos_cargos))

altera_cargos(empty-mapping,_,_) = empty-mapping

Fazem parte também da definição do ATO Projeto as operações sobre processos, métricas e estimativas definidas em [LIM98]. Além dessas, foram adicionadas as definições sobre os cargos, que são apresentadas abaixo.

Interface:

inclui_cargo(_,_,_,_,_):PROJETOS, STRING, STRING, STRING -> PROJETOS
exclui_cargo(_,_,_):PROJETOS, STRING, STRING -> PROJETOS
altera_cargo_sup(_,_,_,_):PROJETOS, STRING, STRING, STRING -> PROJETOS
inclui_agente_cargo(_,_,_,_): PROJETOS, STRING, STRING, STRING -> PROJETOS
exclui_agente_cargo(_,_,_,_):PROJETOS, STRING, STRING, STRING -> PROJETOS

Operações

Variáveis formais:

projetos : PROJETOS
cargos : CARGO
nomep, nome_proj, nome_cargo, cargo_superior, agente_desenv : STRING

inclui_cargo(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, nome_cargo, cargo_superior)
= **if** nome_proj = nomep
then modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos **ICS**(CARGO, inclui_cargo, cargos, <nome_cargo, cargo_superior>), projetos)
else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *inclui_cargo*(projetos, nome_proj, nome_cargo, cargo_superior))

inclui_cargo(empty-mapping,_,_,_,_) = empty-mapping

exclui_cargo(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, nome_cargo)
= **if** nome_proj = nomep
then modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos **ICS**(CARGO, exclui_cargo, cargos, <nome_cargo>), projetos)

else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *exclui_cargo*(projetos, nome_proj, nome_cargo))

exclui_cargo (empty-mapping,_,_) = empty-mapping

altera_cargo_superior(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, nome_cargo, cargo_superior)
= **if** nome_proj = nomep
then modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos **ICS**(CARGO, altera_cargo_superior, cargos, <nome_cargo, cargo_superior>), projetos)
else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *altera_cargo_superior*(projetos, nome_proj, nome_cargo, cargo_superior))

altera_cargo_superior(empty-mapping,_,_,_) = empty-mapping

inclui_agente_cargo(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, nome_cargo, agente_desenv)
= **if** nome_proj = nomep
then modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos **ICS**(CARGO, inclui_agente_cargo, cargos, <nome_cargo, agente_desenv>), projetos)
else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *inclui_agente_cargo*(projetos, nome_proj, nome_cargo, agente_desenv))

inclui_agente_cargo(empty-mapping,_,_,_) = empty-mapping

exclui_agente_cargo(modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), projetos), nome_proj, nome_cargo, agente_desenv)
= **if** nome_proj = nomep
then modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos **ICS**(CARGO, exclui_agente_cargo, cargos, <nome_cargo, agente_desenv>), projetos)
else modify(nomep, (_,_,_,_,_,_,_,_,_, Cargos cargos), *exclui_agente_cargo*(projetos, nome_proj, nome_cargo, agente_desenv))

exclui_agente_cargo(empty-mapping,_,_,_) = empty-mapping

6.7 ATO BCSIM

O simulador utiliza uma base de conhecimento gerada através do ATO BCI. Os dados presentes nessa base serão utilizados pelos agentes desenvolvedores na hora e tomar alguma decisão. Para que a base de conhecimento seja disponibilizada para os agentes dessa ferramenta é necessária a realização de uma operação de inferência sobre base que gera uma agenda que contém as regras realizadas. Os resultados das realizações dessas regras são utilizados pelos agentes.

O ATO BCSIM é composto de um registro com dois campos: base-conhecimento e agenda. O campo base de conhecimento é definido através do ATO BCI e o campo agenda é definido através do ATO AGENDA, ambos definidos em [MOR97], como mostra a figura 6.8.

O ATO BCI manipula a base de conhecimento com informações sobre o ambiente e o ATO AGENDA disponibiliza as informações obtidas após a aplicação da máquina de inferência sobre a base de conhecimento. Para o simulador ter acesso ao resultado da aplicação de uma regra definida na base de conhecimento, o simulador precisa consultar a agenda. As definições da interface e das operações do ATO BCI e do ATO AGENDA podem ser encontradas em [MOR97].

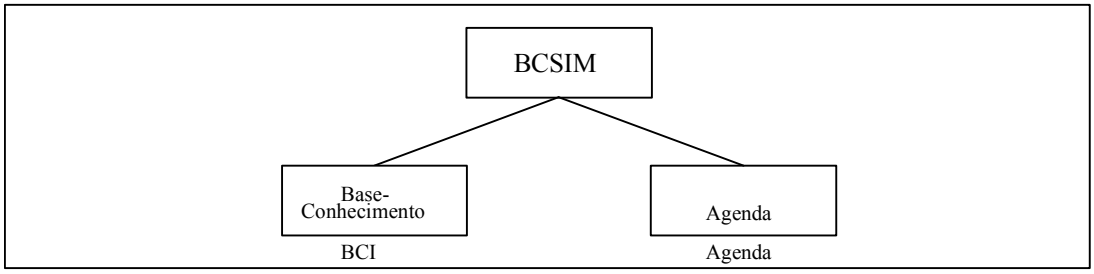

FIGURA 6.8 - Classe BCSIM

Interface:

inclui_base(_): BCI -> BCSIM
gera_agenda(_): BCSIM -> BCSIM
consulta_base_conhecimento(_,_): BCSIM, STRING -> BOOLEAN
solução_base_conhecimento(_,_,_): BCSIM, STRING, CONJ_STRING -> CONJ_STRING
calculo_base_conhecimento(_,_,_): BCSIM, STRING, CONJ_STRING -> REAL

Operação

Variáveis Formais:

base : BCI
parâmetros : CONJ_STRING

inclui_base(base) = (Base-Conhecimento base,_)

gera_agenda((Base-Conhecimento base, Agenda empty-register))
= (Base-Conhecimento base, Agenda **ICS**(BCI, mi, base))

consulta_base_conhecimento((Base-Conhecimento base, Agenda agenda), regra)
= **ICS**(AGENDA, retorna_conclusão_regra, regra)

solução_base_conhecimento((Base-Conhecimento base, Agenda agenda), regra, parâmetros)
= **ICS**(AGENDA, retorna_solução_regra, regra, <parâmetros>)

calculo_base_conhecimento((Base-Conhecimento base, Agenda agenda), regra, parâmetros)
= **ICS**(AGENDA, retorna_valor_regra, regra, <parâmetros>)

6.8 ATO Relatório

O ATO Relatório é uma ferramenta de apoio ao ATO Simulador. A função desse ATO é manter a estrutura de registro de todas as ocorrências de eventos que ocorram durante a simulação. A classe relatório é um conjunto de ocorrências e uma ocorrência é constituída de operações do tipo registro como representado na figura 6.9. O registro Ocorrência é constituído de cinco campos: Evento, Agente, Nome-Projeto, Nome-Atividade e Tempo-Evento. A interface e a operação do ATO Relatório é apresentada a seguir.

Interface:

cria_relatório: : -> RELATÓRIO
gera_ocorrência(_,_,_,_,_): RELATÓRIO, STRING, STRING,
STRING, STRING, REAL -> RELATÓRIO
tempo_último_evento(_): RELATÓRIO -> REAL

Operação

Variáveis Formais:

ocorrências	: RELATÓRIO
agente, projeto, atividade, evento	: STRING
tempo_evento	: REAL

cria_relatório = empty-list

gera_ocorrência(ocorrências, agente, projeto, atividade, evento, tempo_evento)
= Cons((Evento evento, Agente agente, Nome-Projeto projeto, Nome-Atividade atividade, Tempo-Evento tempo-evento), ocorrências)

tempo_último_evento(ocorrências)
= Select-Tempo-Evento(Last(ocorrências))

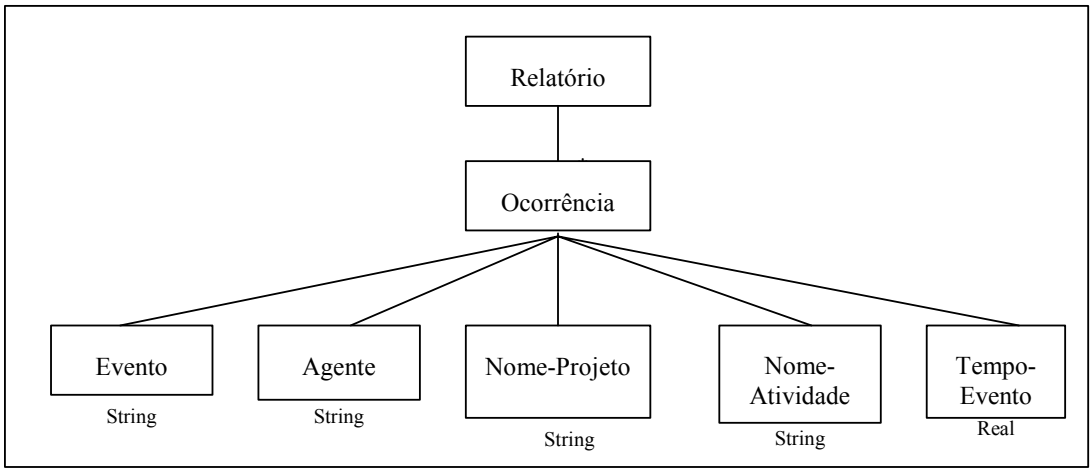

FIGURA 6.9 - Classe relatório

6.9 ATO Atividade-Padrão

Esse ATO trata as atividades que estão pré-definidas no ambiente e podem ser reutilizadas na definição de novas atividades. A classe atividade-padrão é formada pela operação registro com os campos nome, descrição e características que é um conjunto de características. O ATO Atividade-Padrão é composto da classe atividade-padrão, apresentada na figura 6.10 e pela interface e operações apresentadas a seguir.

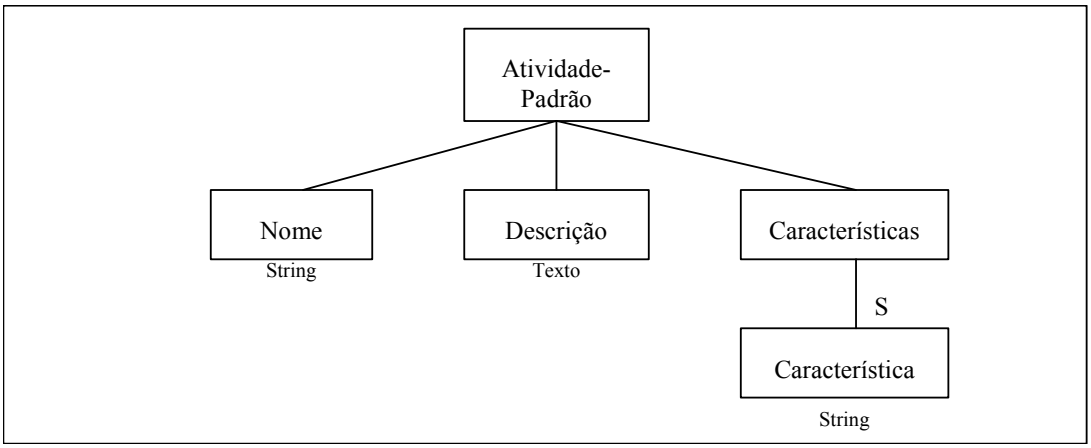

FIGURA 6.10 - Classe atividade-padrão

Interface:

cria_ativ_padrão:	-> ATIVIDADE-PADRÃO
cria_ativ_padrão(_,_): STRING, TEXTO	-> ATIVIDADE-PADRÃO
nome_ativ_padrão(_): ATIVIDADE-PADRÃO	-> STRING
altera_descrição(_,_): ATIVIDADE-PADRÃO, TEXTO	-> ATIVIDADE-PADRÃO
inclui_característica(_,_): ATIVIDADE-PADRÃO, STRING	-> ATIVIDADE-PADRÃO
exclui_característica(_,_):ATIVIDADE-PADRÃO, STRING	-> ATIVIDADE-PADRÃO

Operações

Variáveis Formais:

atividade-padrão	: ATIVIDADE-PADRÃO
características	: CONJ_STRING
característica, característica-alvo, nome_padrão, característica	: STRING
descrição, nova_descrição	: TEXTO

cria_ativ_padrão(nome_padrão, descrição) = (Nome nome_padrão, Descrição descrição,_)

nome_ativ_padrão(atividade-padrão) = select-Nome(atividade-padrão)

altera_descrição((_, Descrição descrição,_),nova-descrição) = (_, Descrição nova-descrição,_)

inclui_característica((_,_,Características características), característica)
= (_,_,Características add(características, característica))

exclui_característica((_,_,Características add(características, característica), característica-alvo)
= **if ICS**(CARACTERÍSTICA, *retorna_característica*, característica) = **ICS**(CARACTERÍSTICA, *retorna_característica*, característica-alvo)
then (_,_,Características características)
else add(exclui_característica((_,_, Características características), característica-alvo), característica)

exclui_característica((_,_,Características empty-set), _) = (_,_,Características empty-set)

6.10 ATO Cargo

O Ato Cargo é uma ferramenta de apoio a definição de projeto. Com ele é possível definir os cargos necessários em cada projeto e alocar responsáveis por esses cargos, que são agentes-desenvolvedores. É possível definir também uma hierarquia entre cargos, através do campo cargo-superior.

A classe cargo é apresentada na figura 6.11 e a definição das operações são apresentadas em seguida.

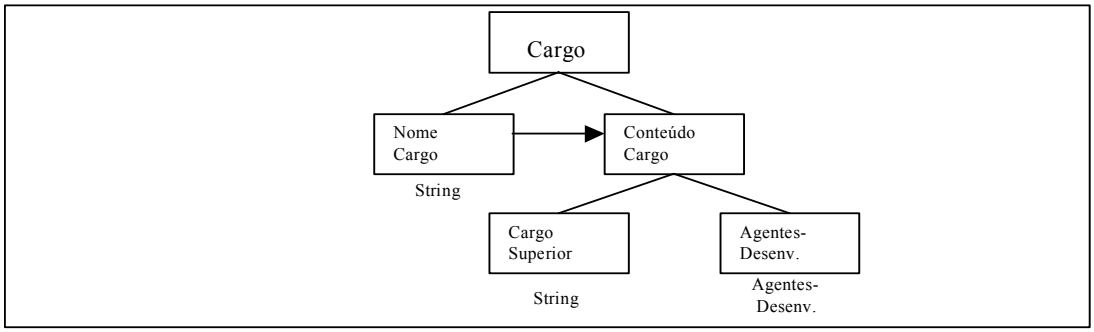

FIGURA 6.11 - Classe cargo

Interface:

cria_cargo:	-> CARGO
inclui_cargo(_,_,_): CARGO, STRING, STRING	-> CARGO
exclui_cargo(_,_): CARGO, STRING	-> CARGO
altera_cargo_superior(_,_,_): CARGO, STRING, STRING	-> CARGO
inclui_agente_cargo(_,_,_): CARGO, STRING, STRING	-> CARGO
exclui_agente_cargo(_,_,_):CARGO, STRING, STRING	-> CARGO

Operações

Variáveis Formais:

cargos	: CARGO
agentes-desenv	: CONJ_STRING
nome_cargo, cargo_superior, ncargo, novo_cargo_sup, agente-desenv	: STRING

cria_cargo = empty-mapping

inclui_cargo(cargos, nome_cargo, cargo_superior)
= modify(nome_cargo, (Cargo Superior cargo_superior,_), cargos)

exclui_cargo(cargos, nome_cargo) = Restrict_with(cargos, add(empty-set, nome_cargo))

altera_cargo_superior(modify(ncargo, (Cargo Superior cargo_superior,_), cargos), nome_cargo, novo_cargo_sup)
= **if** nome_cargo = ncargo
then modify(ncargo, (Cargo_Superior novo_cargo_sup,_), cargos)
else modify(ncargo, (Cargo Superior cargo_superior,_), *altera_cargo_superior*(cargos, nome_cargo, novo_cargo_sup))

altera_cargo_superior(empty-mapping,_,_) = empty-mapping

inclui_agente_cargo(modify(ncargo, (_,Agentes-Desenv. agentes-desenv), cargos), nome_cargo, agente-desenv)
= **if** nome_cargo = ncargo
then modify(ncargo, (_,Agentes-Desenv. add(agentes-desenv, agente-desenv), cargos)
else modify(ncargo, (_,Agentes-Desenv. agentes-desenv), *inclui_agente_cargo*(cargos, nome_cargo, agente-desenv))

inclui_agente_cargo (empty-mapping,_,_) = empty-mapping

exclui_agente_cargo(modify(ncargo, (_,Agentes-Desenv. agentes-desenv), cargos), nome_cargo, agente-desenv)
= **if** nome_cargo = ncargo
then modify(ncargo, (_,Agentes-Desenv. delete(agentes-desenv, agente-desenv), cargos)
else modify(ncargo, (_,Agentes-Desenv. agentes-desenv), *exclui_agente_cargo*(cargos, nome_cargo, agente-desenv))

exclui_agente_cargo (empty-mapping,_,_) = empty-mapping

7 Exemplo de Simulação do Projeto

Neste capítulo é mostrado um exemplo de simulação de um processo de software. O exemplo usado, foi adaptado do exemplo para validar o modelo de execução proposto por [LIM98] na definição do gerenciador de processos de software do PROSOFT, que por sua vez usou uma extensão do problema proposto em [KEL 90], usualmente conhecido como *ISPW-6 software process exampl*e, proposto no 6º *International Software Process Workshop* (ISPW-6). Consiste de vários componentes importantes dos processos de software do mundo real.

A descrição do problema é apresenta da em seguida, a qual foi modelada como atividades de um processo de software no gerenciador de processos, sendo mostrados os passos de sua simulação em seguida.

7.1 Descrição do Exemplo

O problema tem como escopo uma porção de um processo de manutenção de software, focalizando o projeto, codificação, teste de unidades e gerenciamento dessas atividades em um software. Esta modificação deve ocorrer por mudanças nos requisitos do software nas fases finais do ciclo de vida do software, tendo sido autorizada pela organização. Além disso, apenas um módulo deve ser afetado. O problema inicia com o gerente de projetos fazendo o escalonamento e atribuição de atividades aos desenvolvedores. O problema termina quando uma nova versão do código-fonte do software passou por novos testes de unidade. As atividades do problema são:

- **Escalonamento e atribuição de tarefas**: É o primeiro passo do processo exemplo. Envolve o escalonamento do trabalho necessário para a modificação do software e designação de tarefas individuais para membros específicos da equipe. Deve ser realizada pelo gerente de projetos;
- **Modificação do projeto**: Envolve a modificação do projeto para a unidade de código afetada pela mudança de requisitos. Deve ser realizada pelo projetista;
- **Revisão do projeto**: Envolve a revisão formal do projeto modificado, devendo ser realizada por uma equipe de revisão;
- **Modificação do código**: Envolve a modificação do código-fonte devido às mudanças ocorridas no projeto e sua posterior compilação. Esta atividade deve ser realizada pelo projetista;
- **Modificação dos planos de testes**: Modificação das funcionalidades e capacidades a serem testadas, feitas pelo engenheiro de qualidade de software;
- **Modificação do pacote de testes de unidade**: Envolve a modificação do atual pacote de testes para a unidade de código afetada, feita pelo engenheiro de qualidade;

- **Teste de unidade**: Envolve a aplicação do pacote de teste de unidade no código codificado, e análise dos resultados, feita pelo projetista e pelo engenheiro de qualidade;
- **Monitoração do progresso**: Envolve a monitoração do andamento dos trabalhos feita pelo gerente de projetos.

Uma descrição mais detalhada das atividades pode ser encontrada em [KEL90].

7.2 Representação no Simulador de Processos

A figura 7.1 a seguir ilustra a representação do processo usado pelo simulador. O projeto contém um processo de software onde as atividades principais são "1-Escalonar e atribuir tarefas", "2-Monitorar progresso" e "3-Realizar modificações". As atividades 1 e 2 têm como *script* uma descrição informal, enquanto a atividade 3 é decomposta em um novo processo de desenvolvimento chamado PROCESSO 3 representado na figura 7.2.

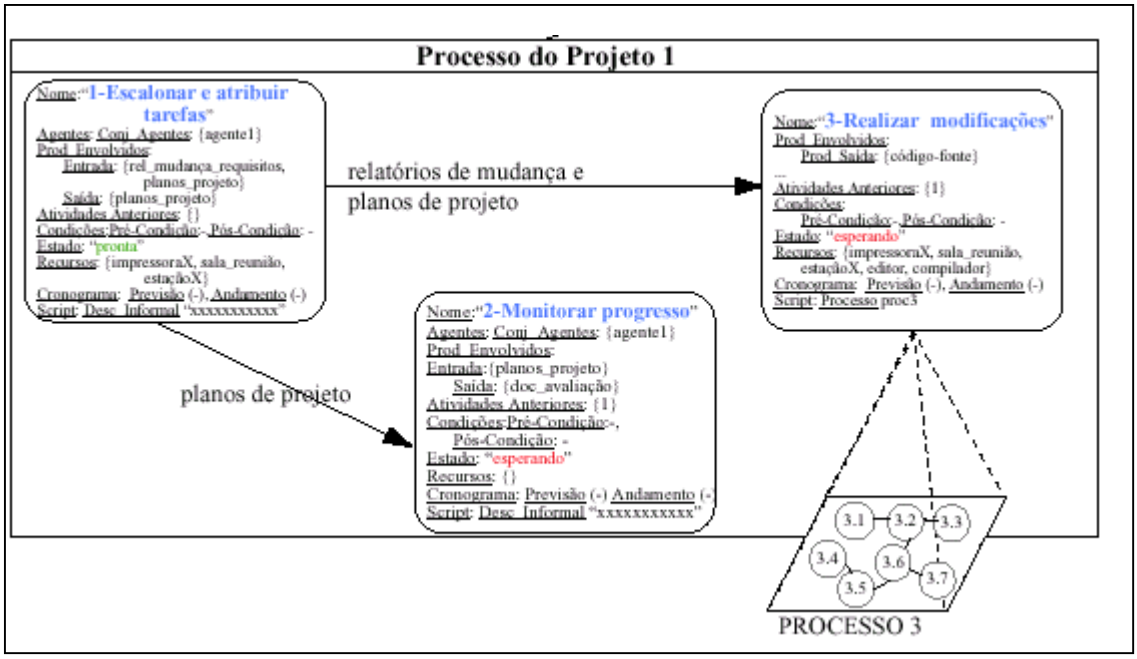

FIGURA 7.1 - Exemplo ISPW-6

7.3 Simulação do Processo

Para simular os processos representados acima são necessárias as estimativas de tempo para realização que são calculadas a partir das características das atividades relacionadas com as habilidades entre os agentes e das afinidades entre os desenvolvedores. Por questão de simplicidade esses cálculos serão abstraídos dessa demonstração. Os passos seguintes são realizados:

Passo 1: definir_simulação_projeto(simulador, "Projeto 1")

Com a execução dessa operação um evento é gerado no relatório da simulação com as informações: “simulador”, “projeto 1”, “inicio da simulação”, 0. Os processos são atualizados, ou seja, aqueles que não possuem atividades anteriores têm seus estados alterados para “pronto”. Por fim, as agendas são atualizadas, ou seja, as atividades no estado “pronto” são colocadas nas agendas dos seus respectivos agentes responsáveis. A única atividade pronta é a “atividade 1”.

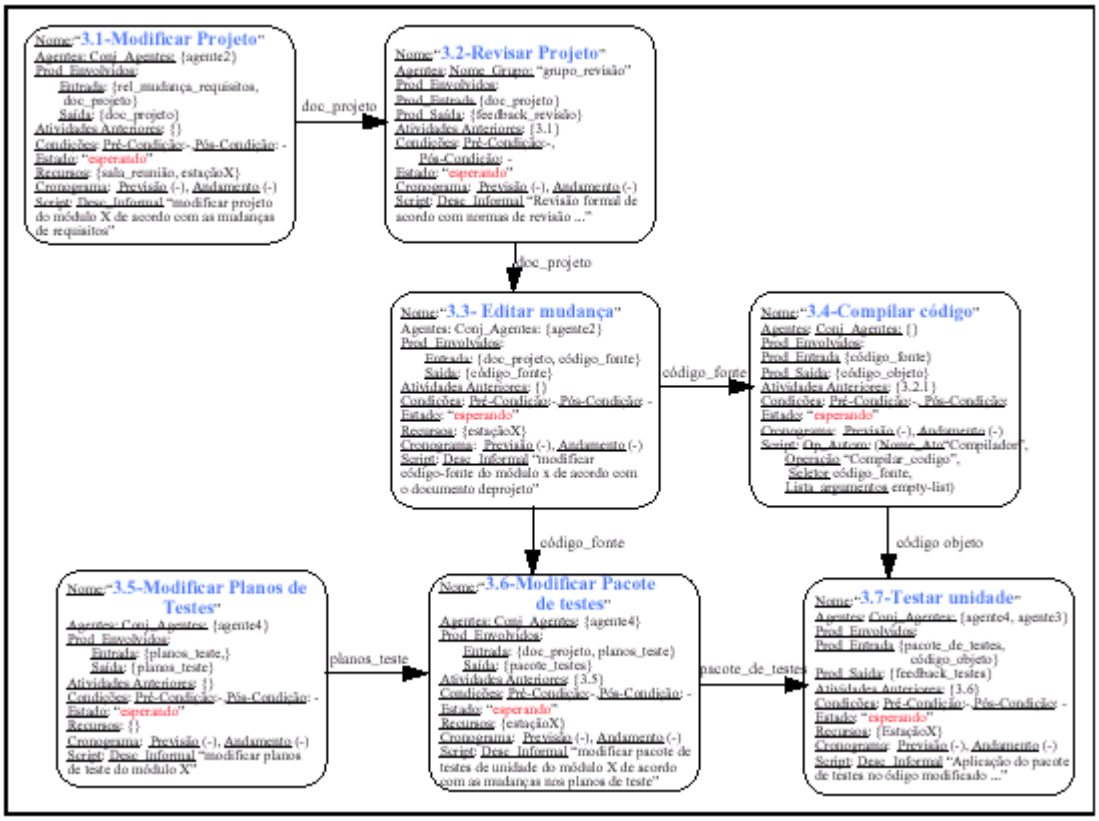

FIGURA 7.2 - Script da atividade 3

Passo 2: simular(simulador)

Está operação dispara a realização dos ciclos da simulação.

Passo 3: teste_realizar_ciclo(simulador)

Está operação é disparada pela execução do passo 2, testa se a simulação pode prosseguir. Para verifica se todos os processos foram concluídos e realiza consulta a base de conhecimento para testar se alguma heurística foi identificada que impressa o prosseguimento da simulação. No primeiro teste nem uma atividade apresenta o estado “concluída” e nenhum teste na base de conhecimento foi positivo, por isso a simulação prossegue.

Passo 4: invocar_ciclo(simulador)

Está operação realiza três operações em sequência:

- *executa_agentes_desenv*: Permite que cada agente representado verifique sua agenda para constatar se existe alguma atividade a ser realizada;
- *gera_execução_simulador*: Gera um evento “atualização do tempo de simulação” e adiciona 1 ao tempo, preparando o próximo ciclo;

- *teste_realizar_ciclo*: Retorna ao passo três para que um novo ciclo de simulação seja realizado se necessário.

As execuções alternadas dos passos 3 e 4 irão ocorrer até que todas atividades estejam completas ou alguma heurística seja identificada inviabilizando o prosseguimento da simulação.

A operação *executa_agentes_desenv* é responsável pela execução dos agentes-desenvolvedores. Através da operação *executa_agente_desenv*, a ação de cada agente-desenvolvedor é representada.

Passo 5: executa_agente_desenv(Simulador, "agente 1")

O agente verifica se a próxima ação a ser executada é para iniciar a execução de uma atividade ou para finalizar a atividade (as duas ações possíveis). Se for para iniciar que é o caso, então ele verifica a sua agenda, se tiver uma atividade pronta para ser executada, o agente executa a operação *executa_atividade*, que é uma requisição ao simulador para que a atividade seja executada. Se não houver atividade pronta, o agente-desenvolvedor é preparado para esperar um novo ciclo.

A próxima ação que agente pode ter que executar é a requisição do fim de execução de uma atividade, nesse caso ele chama a operação *finaliza_atividade* para indicar ao simulador o evento requisitado.

Passo 6: executa_atividade(Simulador, "Agente 1", ("Projeto 1" ∪ "Escalonamento e atribuição de tarefas"))

A realização do passo 6 é em decorrência da requisição realiza pelo agente 1. O simulador então verificará se a atividade está ativa (iniciada por outro agente), se sim ele autorizará a execução da mesma, pois os recursos já estão alocados a ela. Se a atividade ainda não estiver ativa o simulador verificará se os recursos podem ser alocados, se sim os a autorização é concedida e os recursos são alocados para a atividade, se não o agente terá que esperar um novo ciclo para requisitar a simulação.

Outros passos são apresentados na tabela 7.1.

Passos	**Operações**
7	*executa_agente_desenv*(Simulador, "agente 2"): Não possui atividades na agenda.
8	*executa_agente_desenv*(Simulador, "agente 3"): Não possui atividades na agenda.
9	*executa_agente_desenv*(Simulador, "agente 4"): Não possui atividades na agenda.
10	*executa_agente_desenv*(Simulador, "grupo de revisão"): Não possui atividades na agenda.
11	*gera_execução_simulador*(Simulador, "Projeto 1", "atualização do tempo de simulação", 1): Novo ciclo.
12	*teste_realizar_ciclo*(Simulador): Existe atividade diferente de "concluída".
13	*invocar_ciclo*(Simulador)
14	*executa_agente_desenv*(Simulador, "agente 1"): Próximo ação é para finalizar atividade.

15	*finaliza_atividade*(Simulador, “Agente 1”, “Escalonamento e atribuição de tarefas”, “Projeto 1”)
16	*executa_agente_desenv*(Simulador, “agente 2”): Não possui atividades na agenda.
17	*executa_agente_desenv*(Simulador, “agente 3”): Não possui atividades na agenda.
18	*executa_agente_desenv*(Simulador, “agente 4”): Não possui atividades na agenda.
19	*executa_agente_desenv*(Simulador, “grupo de revisão”): Não possui atividades na agenda.
20	*gera_execução_simulador*(Simulador, “Projeto 1”, “atualização do tempo de simulação”, 2): Novo ciclo.
21	*teste_realizar_ciclo*(Simulador): Existe atividade diferente de “concluída”.
22	*invocar_ciclo*(Simulador): Processo e agendas são atualizadas.
23	*executa_agente_desenv*(Simulador, “agente 1”): Executar atividade da agenda.
24	*executa_atividade*(Simulador, “Agente 1”, “Monitorar Progresso”, “Projeto 1”)
25	*executa_agente_desenv*(Simulador, “Agente 2”): Executa atividade da agenda.
26	*executa_atividade*(Simulador, “Agente 2”, (“Projeto 1” ∪ “Modificar Projeto”))
27	*executa_agente_desenv*(Simulador, “agente 3”): Não possui atividades na agenda.
28	*executa_agente_desenv*(Simulador, “agente 4”): Executa atividade na agenda.
29	*executa_atividade*(Simulador, “Agente 4”, (“Projeto 1” ∪ “Modificar planos de testes”))
30	*executa_agente_desenv*(Simulador, “grupo de revisão”): Não possui atividades na agenda.
31	*gera_execução_simulador*(Simulador, “Projeto 1”, “atualização do tempo de simulação”, 3): Novo ciclo.
32	*teste_realizar_ciclo*(Simulador): Existe atividade diferente de “concluída”.
33	*invocar_ciclo*(Simulador)
34	*executa_agente_desenv*(Simulador, “agente 1”): Continua executando atividade.
35	*executa_agente_desenv*(Simulador, “Agente 2”): Próxima ação é para finalizar atividade.
36	*finaliza_atividade*(Simulador, “Agente 2”, “Modificar Projeto”, “Projeto 1”)
37	*executa_agente_desenv*(Simulador, “agente 3”): Não possui atividades na agenda.

38	*executa_agente_desenv*(Simulador, “agente 4”): Próxima ação é para finalizar atividade.
39	*finaliza_atividade*(Simulador, “Agente 4”, “Modificar planos de testes”, “Projeto 1”)
40	*executa_agente_desenv*(Simulador, “grupo de revisão”): Não possui atividades na agenda.
41	*gera_execução_simulador*(Simulador, “Projeto 1”, “atualização do tempo de simulação”, 4): Novo ciclo.
42	*teste_realizar_ciclo*(Simulador): Existe atividade diferente de “concluída”.
43	*invocar_ciclo*(Simulador): Processo e agendas são atualizadas.
44	*executa_agente_desenv*(Simulador, “agente 1”): Continua executando atividade.
45	*executa_agente_desenv*(Simulador, “Agente 2”): Executa atividade da agenda.
46	*executa_atividade*(Simulador, “Agente 2”, (“Projeto 1” ∪ “Editar Mudança”))
47	*executa_agente_desenv*(Simulador, “agente 3”): Não possui atividades na agenda.
48	*executa_agente_desenv*(Simulador, “agente 4”): Não possui atividades na agenda.
49	*executa_agente_desenv*(Simulador, “grupo de revisão”): Não possui atividades na agenda.
50	*gera_execução_simulador*(Simulador, “Projeto 1”, “atualização do tempo de simulação”, 5): Novo ciclo.
51	*teste_realizar_ciclo*(Simulador): Existe atividade diferente de “concluída”.

TABELA 7.1 - Passos da simulação de processo

Obs. 1: Os ciclos se repetirão até que todas atividades sejam concluídas.

Obs. 2: Para facilitar a representação cada atividade é executada em um único ciclo nesse exemplo. Com exceção da atividade “Monitorar progresso” que é realizada em paralelo com as demais. Em paralelo significa no mesmo ciclo.

Obs. 3: O grupo de revisão é tratado como um único agente.

A representação dos passos permitiu a verificação da consistência do modelo. No entanto, faz-se ainda necessário estudos mais aprofundados do modelo para avaliar a sua aplicabilidade a um ambiente de desenvolvimento de software real.

8 Conclusão

Esse trabalho apresentou um modelo de simulação de processos de software baseado em conhecimento. O modelo apresentado é integrado a um meta-modelo de processo de software para auxiliar um gerente de projeto a validar modelos de processo instanciados antes de serem executados. Foram utilizados conceitos da inteligência artificial para representar o comportamento das pessoas envolvidas no processo. O simulador especificado formalmente e para valida-lo foi definido um modelo UML, apresentado no capítulo 5, para construir um protótipo.

Conceitos da tecnologia de processos de software foram de fundamental importância para este trabalho. Sendo que o trabalho foi definido para ser uma parte de um projeto maior que visa definir um modelo de ciclo de vida completo para integrar a tecnologia de processo de software a um ambiente de desenvolvimento de software. O objetivo é fornecer aos envolvidos um ambiente que os auxilia a obter produtos de software com qualidade.

A simulação de processo de software é uma importante área nova relacionada à automação do processo de software. Além de beneficiar sobremaneira o entendimento do processo de software, permite o aperfeiçoamento contínuo dos modelos construídos. Uma ferramenta que auxilie um gerente de desenvolvimento a validar um modelo antes de utilizá-lo permite a verificação de fatores como custo e tempo sem comprometer um orçamento disponível.

Agentes inteligentes foram utilizados com o objetivo de simular o comportamento dos desenvolvedores durante o ciclo de vida de software. Segundo [SCA98], os sistemas multiagentes podem revolucionar o desenvolvimento e utilização da simulação de processos de software baseada em conhecimento. Verifica-se que a simulação baseada em agentes incorpora os benefícios da utilização das ferramentas de simulação baseada em conhecimento e daquelas baseadas em eventos permitindo, por exemplo, observar os padrões de interação entre os desenvolvedores (recurso de sistemas baseados em conhecimento), tal como a identificação de gargalos e a encenação de processos (recursos típicos de sistemas baseados em simulação evento-discreta).

A definição de um modelo de simulação como uma ferramenta do ambiente PROSOFT incrementará a lista de aplicações disponíveis para auxiliar os usuários na realização de tarefas de desenvolvimento de software. A ferramenta de simulação definida nesse trabalho é fortemente relacionada com outros dois ATOs definidos para o PROSOFT, a ferramenta especialista e o gerenciador de processos de software. A utilização da especificação formal do PROSOFT permitiu a identificação de inconsistências na definição do simulador, principalmente na definição da comunicação entre os ATOs, tornando possível a reparação de erros antes que qualquer linha de código fosse escrita. Isso foi de fundamental importância no projeto do simulador devido a forte interação entre os ATOs utilizados.

A especificação do simulador utilizando UML foi uma alternativa encontrada para validação do trabalho. A utilização da mesma permitiu o uso de uma ferramenta automatizada para geração da estrutura de um programa utilizando a linguagem Java. Através da avaliação dos resultados do programa em Java foi possível constatar o

funcionamento do simulador. Houve a preocupação de mapear os conceitos da especificação em PROSOFT para a especificação UML, utilizando a relação de criação de uma classe para cada ATO definido.

Informações de um PSEE acerca do histórico real dos desenvolvedores em projetos que foram executados anteriormente (estabelecendo a base e conhecimento do simulador) será de grande contribuição para o simulador. Quanto maior a quantidade de aspectos de um ambiente for representado, mais aproximados serão os resultados. Atualmente, todas as informações relativas aos desenvolvedores devem ser fornecidas pelo usuário. Além disso a ferramenta pode ser adaptada para funcionar como um jogo educacional de Engenharia de Software – objetivando atuar como um mecanismo auxiliar ao professor no ensino de métodos e técnicas de Engenharia de Software e gerenciamento de projeto.

Um aspecto a ser ressaltado sobre o modelo de simulação proposto é a ênfase nos fatores tempo e custo do desenvolvimento. Investigação mais profunda precisa ser realizada para determinar mais fatores para o cálculo de atividade, além de habilidades e afinidades. Um fator muito importante para validar um processo de software é relacionado com a qualidade dos artefatos de software produzidos. Entretanto, este fator ainda não está sendo explorado satisfatoriamente nos ambientes de simulação atuais, pois envolve a análise de como os diversos aspectos tecnológicos e gerenciais afetam o processo e os produtos. Assim, investigações adicionais são necessárias para determinar a real implicação destes aspectos no processo de simulação de desenvolvimento de software.

Outros aspectos relacionados com o modelo de simulação foram identificados durante o desenvolvimento do trabalho, mas não foram tratados com a profundidade necessária. Esses aspectos apresentam-se como trabalhos futuros. Cita-se:

- Para “carregar” a base do simulador no que diz respeito a definição dos graus de habilidade e afinidade é necessária a utilização de dados históricos de um ambiente de desenvolvimento de software. Sugere-se a definição de uma ferramenta que realize avaliação do desempenho do desenvolvedor e lhe atribua um grau de maneira automática devido a grande quantidade de informação que precisa ser tratada nesses casos.
- Os aspectos psicológicos dos envolvidos com um ambiente de desenvolvimento influenciam os resultados obtidos. A representação de aspectos psicológicos nos agentes que representam desenvolvedores poderia tornar a representação mais realística.
- Armazenar um histórico sobre o ambiente de desenvolvimento. Esse histórico pode ser inferido e ações podem ser disparadas, baseadas em dados obtidos do histórico, realizando atualizações no modelo de simulação, principalmente no que diz respeito as informações sobre habilidades e afinidades dos desenvolvedores.
- Atualmente o modelo de simulação recebe como parâmetro de entrada um modelo de processo instanciado e produz um relatório identificando os eventos obtidos da simulação. Sugere-se a construção de uma ferramenta que permita política de escalonamento dinâmico (alocação de agentes-desenvolvedores em atividades baseado nas avaliações do simulador). Alterar o formalismo do modelo de simulação para receber como entrada um modelo de processo e retornar um modelo de processo ótimo, através da utilização de heurísticas e

técnicas de escalonamento dinâmico para redefinir a instanciação do modelo.

- Atualmente o simulador reage ao que está definido informando se as definições são adequados ou não de acordo com o que está definido. Modificar o modelo significa reiniciar uma simulação. Sugere-se o desenvolvimento e inclusão de técnicas para permitir o simulador se adaptar a modificações que possam ocorrer no ambiente de forma dinâmica.
- Identificação de características que comprometam a qualidade da simulação antes do seu início seria de grande proveito para o modelo, evitando que o simulador realize avaliações improváveis.

Os três primeiros tópicos citados já estão sendo tratados em [LIM01].

Anexo 1 Operações

Além das operações apresentadas no capítulo 6, o simulador é composto por outras operações que serão apresentadas nesse anexo. Além de complementar as definições de operações referenciadas anteriormente, serão apresentadas também operações auxiliares na definição do ambiente de simulação.

1.1 ATO Simulador

O ATO SIMULADOR é responsável pelo gerenciamento do ambiente de simulação e pela realização da simulação propriamente. No apoio ao gerenciamento são realizadas operações de alteração, inclusão, exclusão e observação, formando os quatro tipos que serão apresentadas a seguir.

1.1.1 Operações sobre Recursos

No ambiente de simulação os recursos podem ser adicionados, excluídos e atualizados. As operações utilizadas para definição de recursos no ambiente de simulação são:

Interface:

inclui_recurso(_,_,_,_,_): SIMULADOR, STRING, REAL, TEXTO, REAL	-> SIMULADOR
exclui_recurso(_,_): SIMULADOR, STRING	-> SIMULADOR
altera_custo_recurso(_,_,_):SIMULADOR, STRING, REAL	-> SIMULADOR
altera_estado_recurso(_,_): SIMULADOR, STRING	-> SIMULADOR
altera_descrição_recurso(_,_,_,_): SIMULADOR, STRING, TEXTO	-> SIMULADOR
altera_mtbf_recurso(_,_,_): SIMULADOR, STRING, REAL	-> SIMULADOR
altera_custo_recurso_aux(_,_,_):CONJ_RECURSOS, STRING, REAL	-> CONJ_RECURSOS
altera_estado_recurso_aux(_,_,_):CONJ_RECURSOS, STRING, STRING	-> CONJ_RECURSOS
altera_desc_recurso_aux(_,_,_):CONJ_RECURSOS, STRING, TEXTO	-> CONJ_RECURSOS
altera_mtbf_recurso_aux(_,_,_):CONJ_RECURSOS, STRING, REAL	-> CONJ_RECURSOS
exclui_recurso_aux(_,_): CONJ_RECURSOS, STRING	-> CONJ_RECURSOS

Operações

Variáveis Formais:

recursos	: CONJ_RECURSOS
recurso	: RECURSO
projetos	: CONJ_PROJETOS
nome_recurso, nome_rec	: STRING
desc_recurso, nova_descrição	: TEXTO
custo, novo_custo, mtbf, novo_mtbf	: REAL

inclui_recurso((Recursos recursos,_,_,_,_,_), nome_recurso, custo, desc_recurso, mtbf)
= **if not** (*existe_recurso*(recursos, nome_recurso))
then (Recursos add(recursos, **ICS**(RECURSO, *cria_recurso*, nome_recurso, <"disponível", custo, desc_recurso, mtbf>),_,_,_,_,_) **else** (Recursos recursos,_,_,_,_,_)

exclui_recurso((Recursos recursos,_,Projetos projetos,_,_,_), nome_recurso)
= **if** *existe_recurso*(recursos, nome_recurso)
  and ICS(PROJETOS, *recurso_pode_ser_excluído*, projetos, <nome_recurso>)
  then (Recursos *exclui_recurso_aux*(recursos, nome_recurso),_, Projetos projetos,_,_,_)
  else (Recursos recursos,_, Projetos projetos,_,_,_,_)

altera_custo_recurso((Recursos recursos,_,_,_,_,_), nome_recurso, novo_custo)
= **if** *existe_recurso*(recursos, nome_recurso)
  then (Recursos *altera_custo_recurso_aux*(recursos, nome_recurso, novo_custo),_,_,_,_,_)
  else (Recursos recursos,_,_,_,_,_)

altera_estado_recurso((Recursos recursos,_,_,_,_,_), nome_recurso)
= **if** *existe_recurso*(recursos, nome_recurso)
  then (Recursos *altera_estado_recurso_aux*(recursos, nome_recurso,"com defeito"),_,_,_,_,_)
  else (Recursos recursos,_,_,_,_,_)

altera_descrição_recurso((Recursos recursos,_,_,_,_,_), nome_recurso, nova_descrição)
= **if** *existe_recurso*(recursos, nome_recurso)
  then (Recursos *altera_desc_recurso_aux*(recursos, nome_recurso, nova_descrição),_,_,_,_,_)
  else (Recursos recursos,_,_,_,_,_)

altera_mtbf_recurso((Recursos recursos,_,_,_,_,_), nome_recurso, novo_mtbf)
= **if** *existe_recurso*(recursos, nome_recurso)
  then (Recursos *altera_mtbf_recurso_aux*(recursos, nome_recurso, novo_mtbf),_,_,_,_,_)
  else (Recursos recursos,_,_,_,_,_)

altera_custo_recurso_aux(add(recursos, recurso), nome_rec, novo_custo)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
  then add(recursos, **ICS**(RECURSO, altera_custo, recurso, <novo_custo>))
  else add(*altera_custo_recurso_aux*(recursos, nome_recurso, novo_custo), recurso)

altera_estado_recurso_aux(add(recursos, recurso), nome_rec, novo_estado)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
  then add(recursos, **ICS**(RECURSO, altera_estado, recurso, <novo_estado>))
  else add(*altera_estado_recurso_aux*(recursos, nome_recurso, novo_estado), recurso)

altera_desc_recurso_aux(add(recursos, recurso), nome_rec,nova_descrição)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
  then add(recursos, **ICS**(RECURSO, altera_descrição, recurso, <nova_descrição>))
  else add(*altera_desc_recurso_aux*(recursos, nome_rec, nova_descrição), recurso)

altera_mtbf_recurso_aux((add(recursos, recurso), nome_rec, novo_mtbf)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
  then add(recursos, **ICS**(RECURSO, altera_mtbf, recurso, <novo_mtbf>))
  else add(*altera_mtbf_recurso_aux*(recursos, nome_recurso, nova_mtbf), recurso)

exclui_recurso_aux(add(recursos, recurso), nome_rec)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
  then recursos
  else add(*exclui_recurso_aux*(recursos, nome_rec), recurso)
  exclui_recurso_aux(empty-set, _) = FALSE

1.1.2 Operações de Definição de Agentes-Desenvolvedores

No ambiente de simulação os agentes-desenvolvedores também podem ser adicionados, excluídos e atualizados. As operações para Definição de agentes-desenvolvedores no ambiente de simulação são:

Interface:

inclui_agente-des(_,_,_): SIMULADOR, STRING, REAL -> SIMULADOR
exclui_agente-des(_,_): SIMULADOR, STRING -> SIMULADOR
altera_custo_agente-des(_,_,_): SIMULADOR, STRING, REAL -> SIMULADOR
inclui_habilidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
exclui_habilidade_agente-des(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_habilidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
inclui_afinidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
exclui_afinidade_agente-des(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_afinidade_agente-des(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR

Operações

Variáveis Formais:

agente-desenvs	: AGENTES-DESENV.
projetos	: PROJETOS
características	: CONJ_STRING
identificador, identificador2, característica, projeto, cargo, cargo_superior	: STRING
custo_por_hora, novo_custo, grau_habilidade, grau_afinidade	: REAL

inclui_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, custo_por_hora)
= **if not (ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>))
then (_, Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *inclui_agente-des*,
agente-desenvs, <identificador,custo_por_hora>),_,_,_,_)
else (_,Agentes-desenvolvedores agentes-desenvs,_,_,_,_)

exclui_agente-des((_,Agentes-desenvolvedores agente-desenvs,Projetos projetos,_,_,_), identificador)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
and ICS(PROJETO, *agente-des_pode_ser_excluído*, projetos, <identificador>)
then (_, Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *exclui_agente-des*,
agente-desenvs, <identificador>),Projetos projetos,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,Projetos projetos,_,_,_)

altera_custo_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, novo_custo)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *altera_custo*,
agente-desenvs, <identificador, novo_custo>),_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_)

inclui_habilidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, característica, grau_habilidade)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, inclui_habilidade,
agente-desenvs, <identificador, característica,grau_habilidade >),_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_)

exclui_habilidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, característica)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *exclui_habilidade*, agente-desenvs, <identificador, característica>),_,_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs_,_,_,_,_)

altera_habilidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, característica, grau_habilidade)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, altera_habilidade, agente-desenvs, <identificador, característica,grau_habilidade >),_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_)

inclui_afinidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, identificador2, grau_afinidade)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *inclui_afinidade*, agente-desenvs, <identificador, identificador2,grau_afinidade >),_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_)

exclui_afinidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_,_), identificador, identificador2)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *exclui_afinidade*, agente-desenvs, <identificador, identificador2>),_,_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_,_)

altera_afinidade_agente-des((_,Agentes-desenvolvedores agente-desenvs,_,_,_,_), identificador, identificador2, grau_afinidade)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < identificador>)
then (_,Agentes-desenvolvedores **ICS**(AGENTES-DESENV, *altera_afinidade*, agente-desenvs, <identificador, identificador2,grau_afinidade >),_,_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs,_,_,_,_)

1.1.3 Operações de Definição de Projetos

As operações para definição de projetos no ambiente de simulação são:

Interface:

define_novo_projeto(_,_,_,_,_,_): SIMULADOR, STRING, DATE, DATE, REAL, TEXTO -> SIMULADOR
exclui_projeto(_,_): SIMULADOR, STRING -> SIMULADOR
altera_dataini_projeto(_,_,_): SIMULADOR, STRING, DATE -> SIMULADOR
altera_datafim_projeto(_,_,_): SIMULADOR, STRING, DATE -> SIMULADOR
altera_recurso_financeiro_projeto(_,_,_): SIMULADOR, STRING, REAL -> SIMULADOR
altera_objetivos_projeto(_,_,_): SIMULADOR, STRING, TEXTO -> SIMULADOR
inclui_produtof_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_produtof_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_processo_projeto(_,_,_): SIMULADOR, STRING, PROCESSO -> SIMULADOR
inclui_métrica_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_métrica_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_valor_métrica_projeto(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
inclui_estimativa_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
exclui_estimativa_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_valor_estimativa_projeto(_,_,_,_): SIMULADOR, STRING, STRING, REAL -> SIMULADOR
inclui_cargo_projeto(_,_,_,_,_): SIMULADOR, STRING, STRING, STRING, STRING -> SIMULADOR

exclui_cargo_projeto(_,_,_):SIMULADOR, STRING, STRING -> SIMULADOR
altera_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_agente_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_agente_cargo_projeto(_,_,_,_): SIMULADOR, STRING, STRING, STRING-> SIMULADOR

Operações

Variáveis Formais:

projetos	: PROJETOS
novo_processo	: PROCESSO
agentes-desenvs	: AGENTES_DESENV
nome_proj, nome_recurso, cargo, cargo_superior, id_agente, nome_métrica, nome_estimativa, nome_produto, unidade, método	: STRING
datai, dataf, nova_datai, nova_dataf	: DATE
rec_finan, novo_recursofinan, valor	: REAL
objetivos, novo_objetivo	: TEXTO

define_novo_projeto((_,_,Projetos projetos,_,_,_), nome_proj, datai, dataf, rec_finan, objetivos)
= **if not**(**ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>))
then (_,_, Projetos **ICS**(PROJETOS, *inclui_projeto*, projetos, <nome_proj, datai, dataf, rec_finan, objetivos>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

exclui_projeto((_,_,Projetos projetos,_,_,_), nome_proj)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *exclui_projeto*, projetos, <nome_proj>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

altera_data_ini_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nova_datai)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_data_ini*, projetos, <nome_proj, nova_datai>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

altera_data_fim_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nova_dataf)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_data_fim*, projetos, <nome_proj, nova_dataf>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

altera_recurso_financeiro_projeto((_,_,Projetos projetos,_,_,_), nome_prj, novo_recursofinan)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_recurso_financeiro*, projetos, <nome_proj, novo_recursofinan>),_,_,_) **else** (_,_,Projetos projetos,_,_,_)

altera_objetivos_projeto((_,_,Projetos projetos,_,_,_), nome_proj, novo_objetivo)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_objetivos*, projetos, <nome_proj, novo_objetivo>),_,_,_) **else** (_,_,Projetos projetos,_,_,_)

inclui_produtof_projeto((_,_, Projetos projetos,_,_,_), nome_proj, nome_produto)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *inclui_prodf*, projetos, <nome_proj, nome_produto>), _,_,_) **else** (_,_,Projetos projetos,_,_,_)

exclui_produtof_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nome_produto)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *exclui_prodf*, projetos, <nome_proj, nome_produto>), _,_,_) **else** (_,_,Projetos projetos,_,_,_)

altera_processo_projeto((_,_,Projetos projetos,_,_,_), nome_proj, novo_processo)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_processo*, projetos, <nome_proj,novo_processo>),
_,_,_) **else** (_,_,Projetos projetos,_,_,_)

inclui_métrica_projeto((_,_, Projetos projetos,_,_,_), nome_proj, nome_métrica, valor, unidade, método)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *inclui_métrica*, projetos,<nome_proj,nome_métrica,
valor, unidade, método>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

exclui_métrica_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nome_métrica)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *exclui_métrica*, projetos, <nome_proj, nome_métrica>),
_,_,_) **else** (_,_,Projetos projetos,_,_,_)

altera_valor_métrica_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nome_métrica, valor)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_valor_métrica*, projetos, <nome_proj,
nome_métrica, valor >),_,_,_) **else** (_,_,Projetos projetos,_,_,_)

inclui_estimativa_projeto((_,_, Projetos projetos,_,_,_), nome_proj, nome_estimativa)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *inclui_estimativa*, projetos, <nome_proj,
nome_estimativa, valor, unidade>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

exclui_estimativa_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nome_estimativa)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *exclui_estimativa*, projetos, <nome_proj,
nome_estimativa>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

altera_valor_estimativa_projeto((_,_,Projetos projetos,_,_,_), nome_proj, nome_estimativa, valor)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
then (_,_, Projetos **ICS**(PROJETOS, *altera_valor_estimativa*, projetos, <nome_proj,
nome_estimativa, valor>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

inclui_cargo_projeto((_,Agentes-desenvolvedores agente-desenvs, Projetos projetos,_,_,_), nome_proj,
id_agente, cargo, cargo_superior)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
and ICS(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < id_agente>)
then (_,Agentes-desenvolvedores agente-desenvs, Projetos **ICS**(PROJETOS, *inclui_cargo*,
projetos, <nome_proj, id_agente, cargo, cargo_superior),_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs, Projetos projetos,_,_,_)

exclui_cargo_projeto((_,_,Projetos projetos,_,_,_), nome_proj, cargo)
= **if not**(**ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>))
then (_,_, Projetos **ICS**(PROJETOS, *exclui_cargo*, projetos, <nome_proj,cargo>),_,_,_)
else (_,_,Projetos projetos,_,_,_)

altera_cargo_projeto((_,Agentes-desenvolvedores agente-desenvs, Projetos projetos,_,_,_), nome_proj,
id_agente, cargo)
= **if ICS**(PROJETOS,*existe_projeto*, projetos,<nome_proj>)
and ICS(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, < id_agente>)
and ICS(PROJETOS, *existe_cargo*, projetos,<nome_proj, cargo>)
then (_,Agentes-desenvolvedores agente-desenvs, Projetos **ICS**(PROJETOS, *altera_cargo*,
projetos, <nome_proj, id_agente, cargo),_,_,_)
else (_,Agentes-desenvolvedores agente-desenvs, Projetos projetos,_,_,_)

Além das operações apresentadas, existem também as operações para definição de atividades em um projeto. As operações para definição de atividades são:

Interface:

inclui_atividade_projeto(_,_,_): SIMULADOR, STRING, ATIVIDADE -> SIMULADOR
exclui-atividade_projeto(_,_,_): SIMULADOR, STRING, STRING -> SIMULADOR
altera_previsão_início_ativ(_,_,_,_): SIMULADOR, STRING, STRING, DATE -> SIMULADOR
altera_previsão_fim_ativ(_,_,_,_): SIMULADOR, STRING, STRING, DATE -> SIMULADOR
inclui_prod_entrada_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_prod_saída_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_prod_entrada_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_prod_saída_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_ativ_anterior-ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui-ativ_anterior_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_recurso_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_recurso_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
inclui_agente-desenvs_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR
exclui_agente-desenvs_ativ(_,_,_,_): SIMULADOR, STRING, STRING, STRING -> SIMULADOR

Operações

Variáveis Formais:

projetos : PROJETOS
atividade : ATIVIDADE
recursos : CONJUNTO_RECURSOS
agentes-desenvs : AGENTES_DESENV.
nome_proj, nome_ativ, nome_produto, nome_ativ_anterior, nome_recurso, id_agente : STRING
nova_previsão_início, nova_previsão_fim : DATA

inclui_atividade_projeto((_,Agentes-Desenvolvedores agentes-desenvs, Projetos projetos,_,_,_), nome_proj, atividade)
= (_,Agentes-Desenvolvedores *atualiza_agenda_agentes_des* (add(empty-set, atividade), agentes-desenvs, nome_proj), Projetos **ICS**(PROJETOS, *atualiza_processo*, **ICS**(PROJETOS, *inclui_atividade*, projetos, <nome_proj, atividade>)),_,_,_)

exclui_atividade_processo((_,Agentes-Desenvolvedores agentes-desenvs,Projetos projetos,_,_,_), nome_proj, nome_ativ)
= **if ICS**(PROJETOS, *projeto_iniciado*, projetos, <nome_proj>)
then (_,Agentes-Desenvolvedores *exclui_ativ_agendas*(agentes-desenvs, nome_proj, nome_ativ), Projetos **ICS**(PROJETOS, *exclui_atividade*, projetos, <nome_proj,nome_ativ>),_,_,_)
else (_,Agentes-Desenvolvedores agentes-desenvs,Projetos **ICS**(PROJETOS, *exclui_atividade*, projetos, <nome_proj,nome_ativ>),_,_,_)

altera_previsão_início_atividade((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nova_previsão_início)
= (_,_, Projetos **ICS**(PROJETOS, *altera_dataini_prev_ativ_proj*, projetos, <nome_proj, nome_ativ, nova_previsão_início>),_,_,_)

altera_previsão_fim_atividade((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nova_previsão_fim)
= (_,_, Projetos **ICS**(PROJETOS, *altera_datafim_prev_ativ_proj*, projetos, <nome_proj, nome_ativ, nova_previsão_fim>),_,_,_)
inclui_prod_entrada_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ,nome_produto)
= (_,_, Projetos **ICS**(PROJETOS, *inclui_prod_ent_ativ_proj*, projetos, <nome_proj, nome_ativ, nome_produto>),_,_,_)

inclui_prod_saída_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nome_produto)
= (_,_, Projetos **ICS**(PROJETOS, *inclui_prod_sai_ativ_proj*, projetos, <nome_proj, nome_ativ, nome_produto>),_,_,_)

exclui_prod_entrada_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nome_produto)
= (_,_, Projetos **ICS**(PROJETOS, ex*clui_prod_ent_ativ_proj*, projetos, <nome_proj, nome_ativ, nome_produto>),_,_,_)

exclui_prod_saída_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nome_produto)
= (_,_, Projetos **ICS**(PROJETOS, *exclui_prod_sai_ativ_proj*, projetos, <nome_proj, nome_ativ, nome_produto>),_,_,_)

inclui_ativ_anterior_ativ((_,_,Projetos projetos,_,_,_), nome_prj, nome_ativ, nome_ativ_anterior)
= (_,_, Projetos **ICS**(PROJETOS, *inclui_ativ_ant_proj*, projetos, <nome_proj, nome_ativ, nome_ativ_anterior>),_,_,_)

exclui_ativ_anterior_ativ((_,_,Projetos projetos,_,_,_,_),nome_proj,nome_ativ, nome_ativ_anterior)
= (_,_, Projetos **ICS**(PROJETOS, ex*clui_ativ_ant_proj*, projetos, <nome_proj, nome_ativ, nome_ativ_anterior>),_,_,_,_)

inclui_recurso_ativ((Recursos recursos,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nome_recurso)
= **if** *existe_recurso*(recursos, nome_recurso)
then (Recursos recursos,_, Projetos **ICS**(PROJETOS, *inclui_recurso_ativ_proj*, projetos, <nome_proj,nome_ativ, nome_recurso>),_,_,_)
else (Recursos recursos,_,Projetos projetos,_,_,_)

exclui_recurso_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, nome_recurso)
= (_,_, Projetos **ICS**(PROJETOS, *exclui_recurso_ativ_proj*, projetos, <nome_proj, nome_ativ, nome_recurso>),_,_,_)

inclui_agente_des_ativ((_,Agentes-desenvolvedores agentes-desenvs, Projetos projetos,_,_,_), nome_proj, nome_ativ, id_agente)
= **if ICS**(AGENTES-DESENV, *existe_agente_des*, agente-desenvs, <id_agente>)
then (_,Agentes-desenvolvedores agentes-desenvs, Projetos **ICS**(PROJETOS, *inclui_agente_ativ_proj*, projetos, <nome_proj,nome_ativ, id_agente>),_,_,_)
else (_,Agentes-desenvolvedores agentes-desenvs, Projetos projetos,_,_,_)

exclui_agente-des_ativ((_,_,Projetos projetos,_,_,_), nome_proj, nome_ativ, id_agente)
= (_,_, Projetos **ICS**(PROJETOS, *exclui_agente_ativ_proj*, projetos, <nome_proj,nome_ativ, id_agente>),_,_,_)

1.1.4 Operações de Definição de Atividades Padrão

As operações para definição de atividades padrão no ambiente de simulação são:

Interface:

cria_atividade_padrão(_,_,_): SIMULADOR, STRING, TEXTO -> SIMULADOR
exclui_atividade_padrão(_,_): SIMULADOR, STRING -> SIMULADOR
altera_descrição_ativ_padrão(_,_,_): SIMULADOR, STRING, TEXTO -> SIMULADOR
inclui_característica_ativ_padrão(_,_,_): SIMULADOR, STRING, CARACTERÍSTICA -> SIMULADOR
exclui_característica_ativ_padrão(_,_,_): SIMULADOR, STRING, CARACTERÍSTICA -> SIMULADOR
altera_desc_ativ_padrão(_,_,_): CONJ_ATIV_PADRÃO, STRING, STRING-> CONJ_ATIV_PADRÃO
exclui_ativ_padrão_aux(_,_): CONJ_ATIV_PADRÃO, STRING -> CONJ_ATIV_PADRÃO
inclui_caract_ativ_padrão(_,_,_): CONJ_ATIV_PADRÃO, STRING, CARACTERÍSTICA -> CONJ_ATIV_PADRÃO

exclui_caract_ativ_padrão(_,_,_): CONJ_ATIV_PADRÃO, STRING,
CARACTERÍSTICA -> CONJ_ATIV_PADRÃO

Operações

Variáveis Formais:

atividades_padrão : CONJ_ATIVIDADES_PADRÃO
atividade_padrão : ATIVIDADE_PADRÃO
características : CONJ_CARACTERÍSTICAS
nome_padrão, característica : STRING
descrição, nova_descrição : TEXTO

cria_atividade_padrão((_,_,_,_,_,Atividades Padrão atividades_padrão), nome_padrão, descrição)
= **if not** (*existe_atividade_padrão*(atividades_padrão, nome_padrão))
then (_,_,_,_,_, Atividades Padrão add(atividades_padrão,**ICS**(ATIVIDADE_PADRÃO,
cria_ativ_padrão, nome_padrão, <descrição>)))
else (_,_,_,_,_,Atividades_Padrão atividades_padrão)

exclui_atividade_padrão((_,_,_,_,_,Atividade_Padrão atividades_padrão), nome_padrão)
= **if** *existe_atividade_padrão*(atividades_padrão, nome_padrão)
then (_,_,_,_,_,Atividades Padrão *exclui_ativ_padrão_aux*(atividades_padrão,
nome_padrão))
else (_,_,_,_,_,Atividades_Padrão atividades_padrão)

altera_descrição_ativ_padrão((_,_,_,_,_,Atividades_Padrão atividades_padrão), nome_padrão,
nova_descrição)
= **if** *existe_atividade_padrão*(atividades_padrão,nome_padrão)
then (_,_,_,_,_,Atividades Padrão *altera_desc_ativ_padrão*(atividades_padrão,
nome_padrão, nova_descrição))
else (_,_,_,_,_,Atividades_Padrão atividades_padrão)

inclui_característica_ativ_padrão((_,_,_,_,_,Atividades_Padrão atividades_padrão), nome_padrão,
característica)
= **if** *existe_atividade_padrão*(atividades_padrão,nome_padrão)
then (_,_,_,_,_,Atividades Padrão *inclui_caract_ativ_padrão*(atividades_padrão,
nome_padrão, característica))
else (_,_,_,_,_,Atividades_Padrão atividades_padrão)

exclui_característica_ativ_padrão((_,_,_,_,_,Atividades_Padrão atividades_padrão), nome_padrão,
característica)
= **if** *existe_atividade_padrão*(atividades_padrão,nome_padrão)
then (_,_,_,_,_,Atividades Padrão *exclui_caract_ativ_padrão*(atividades_padrão,
nome_padrão, característica))
else (_,_,_,_,_,Atividades_Padrão atividades_padrão)

altera_desc_ativ_padrão(add(atividades_padrão, atividade_parão),nome_padrão, nova_descrição)
= **if ICS**(ATIVIDADE_PADRÃO, nome_ativ_padrão, atividade_padrão) = nome_padrão
then add(atividades_padrão, **ICS**(ATIVIDADE_PADRÃO, altera_descrição,
atividade_padrão, <nova_descrição>))
else add(*altera_desc_ativ_padrão*(atividades_padrão, nome_padrão, nova_descrição),
atividade_padrão)

exclui_ativ_padrão_aux(add(atividades_padrão, atividade_padrão), nome_padrão)
= **if ICS**(ATIVIDADE_PADRÃO, nome_ativ_padrão, atividade_padrão) = nome_padrão
then atividades_padrão
else add(*exclui_ativ_padrão_aux*(atividades_padrão, nome_padrão), atividade_padrão)

exclui_ativ_padrão_aux(empty-set, _) = FALSE

inclui_caract_ativ_padrão(add(atividades_padrão, atividade_padrão),nome_padrão, característica)
= **if ICS**(ATIVIDADE_PADRÃO, nome_ativ_padrão, atividade_padrão) = nome_padrão
then add(atividades_padrão, **ICS**(ATIVIDADE_PADRÃO, inclui_característica,
atividade_padrão, <característica>))
else add(*inclui_caract_ativ_padrão*(atividades_padrão, nome_padrão, característica),
atividade_padrão)

inclui_caract_ativ_padrão(empty-set,_,_) = empty-set

exclui_caract_ativ_padrão(add(atividades_padrão, atividade_padrão), nome_padrão, característica)
= **if ICS**(ATIVIDADE_PADRÃO, nome_ativ_padrão, atividade) = nome_padrão
then add(atividades_padrão, **ICS**(ATIVIDADE_PADRÃO, exclui_característica,
atividade_padrão, <característica>))
else add(*exclui_caract_ativ_padrão*(atividades_padrão, nome_padrão, característica),
atividade_padrão)

exclui_caract_ativ_padrão(empty-set,_,_) = empty-set

1.1.5 Operações Observadoras

A seguir são apresentadas operações internas do ATO Simulador relativas a observações das informações constantes em seus componentes.

Interface:

existe_recurso(_,_):CONJ_RECURSOS, STRING -> BOOLEAN
existe_atividade_padrão(_,_): CONJ_ATIVIDADES_PADRÃO, STRING -> BOOLEAN

Operações

Variáveis Formais:

recursos	: CONJ_RECURSOS
recurso	: RECURSO
atividades_padrão	: CONJ_ATIVIDADES_PADRÃO
atividade_padrão	: ATIVIDADE_PADRÃO
nome_rec, nome_padrão	: STRING

existe_recurso((Recursos add(recursos, recurso),_,_,_,_,_), nome_rec)
= **if ICS**(RECURSO, nome_recurso, recurso) = nome_rec
then TRUE
else *existe_recurso*((Recursos recursos,_,_,_,_,_), nome_rec)

existe_recurso((Recursos empty-set,_,_,_,_,_),_) = FALSE

existe_atividade_padrão(add(atividades_padrão, atividade_padrão), nome_padrão)
= **if ICS**(ATIVIDADE_PADRÃO, nome_ativ_padrão, atividade_padrão) = nome_padrão
then TRUE
else *existe_atividade_padrão*(atividades_padrão, nome_padrão)

existe_atividade_padrão(empty-set,_) = FALSE

1.2 ATO AGENTES-DESENV

O ATO AGENTES-DESENV é responsável pelo gerenciamento dos agentes-desenvolvedores que representam no modelo o comportamento dos desenvolvedores envolvidos em um ambiente de desenvolvimento de software. A seguir são apresentadas as operações de definição dos agentes-desenvolvedores realizadas no ATO AGENTES-DESENV.

1.2.1 Operações de Gerenciamento

A seguir são apresentadas operações do ATO AGENTES-DESENV para realizar funções de gerenciamento do ATO, incluindo, excluindo e alterando informações.

Interface:

cria_agente_des: ->AGENTES-DESENV
inclui_agente_des(_,_,_):AGENTES-DESENV, STRING, REAL, BCI ->AGENTES-DESENV
exclui_agente_des(_,_):AGENTES-DESENV, STRING ->AGENTES-DESENV
inclui_projeto_agenda(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
exclui_projeto_agenda(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
inclui_ativ_projeto(_,_,_,_,_):AGENTES-DESENV, STRING, STRING,
STRING, STRING ->AGENTES-DESENV
exclui_ativ_projeto(_,_,_,_):AGENTES-DESENV,STRING,STRING,STRING->AGENTES-DESENV
altera_estado_atividade(_,_,_,_,_):AGENTES-DESENV, STRING, STRING,
STRING, STRING ->AGENTES-DESENV
altera_custo(_,_,_):AGENTES-DESENV, STRING, REAL ->AGENTES-DESENV
inclui_habilidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
exclui_habilidade(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
altera_habilidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
inclui_afinidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV
exclui_afinidade(_,_,_):AGENTES-DESENV, STRING, STRING ->AGENTES-DESENV
altera_afinidade(_,_,_,_):AGENTES-DESENV, STRING, STRING, REAL ->AGENTES-DESENV

Operação

Variáveis Formais:

agentes_desenv : AGENTES-DESENV
id_agente, identificador, identificador2, nome_proj,
característica, nome_ativ, estado_ativ, novo_estado, n_ato,
n_oper, contexto, critério, item, id_regra : STRING
custo_por_hora, novo_custo, grau_habilidade, grau_afinidade
novo_grau_habilidade, novo_grau_afinidade : REAL
agenda : AGENDA
habilidades : HABILIDADES
afinidades : AFINIDADES
base : BCI

cria_agente_des = empty-mapping

inclui_agente_des(agentes_desenv,identificador,custo_por_hora, base)
= modify(identificador, (Agenda empty-mapping, Custo-por-hora custo_por_hora,
Habilidades empty-mapping, Afinidades empty-mapping, BC-Agente *inclui_base_aux*(base),
tpe 0, pe “iniciar”), agentes_desenv)

exclui_agente_des(agentes_desenv,identificador)
= Restrict-with(agentes_desenv, add(empty-set, identificador))

inclui_projeto_agenda(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), identificador, nome_proj)
= **if** id_agente = identificador
then modify(id_agente, (Agenda *inclui_proj_agenda_aux*(agenda, nome_proj),_,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (Agenda agenda,_,_,_,_,_,_), *inclui_projeto_agenda*(agentes_desenv, identificador, nome_proj))

inclui_projeto_agenda(empty-mapping,_,_) = empty-mapping

exclui_projeto_agenda(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), identificador, nome_proj)
= **if** id_agente = identificador
then modify(id_agente, (Agenda *exclui_proj_agenda_aux*(agenda, nome_proj),_,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (Agenda agenda,_,_,_,_,_,_), *exclui_projeto_agenda*(agentes_desenv, identificador, nome_proj))

exclui_projeto_agenda(empty-mapping,_,_) = empty-mapping

inclui_ativ_projeto(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), identificador, nome_proj, nome_ativ, estado_ativ)
= **if** id_agente = identificador
then modify(id_agente, (Agenda *inclui_ativ_agenda*(agenda, nome_proj, nome_ativ, estado_ativ) ,_,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (Agenda agenda,_,_,_,_,_,_), *inclui_ativ_projeto*(agentes_desenv, identificador, nome_proj, nome_ativ, estado_ativ))

inclui_ativ_projeto (empty-mapping,_,_,_,_) = empty-mapping

exclui_ativ_projeto(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), identificador, nome_proj, nome_ativ)
= **if** id_agente = identificador
then modify(id_agente, (Agenda *exclui_ativ_agenda*(agenda, nome_proj, nome_ativ),_,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (Agenda agenda,_,_,_,_,_,_), *exclui_ativ_projeto*(agentes_desenv, identificador, nome_proj, nome_ativ))

exclui_ativ_projeto (empty-mapping,_,_,_,_) = empty-mapping

altera_estado_atividade(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), identificador, nome_proj, nome_ativ, novo_estado)
= **if** id_agente = identificador
then modify(id_agente, (Agenda *altera_estado_ativ_agenda*(agenda, nome_proj, nome_ativ, novo_estado),_,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (Agenda agenda,_,_,_,_,_,_), *altera_estado_atividade*(agentes_desenv, identificador, nome_proj, nome_ativ, novo_estado))

altera_estado_atividade(empty-mapping,_,_,_,_) = empty-mapping

altera_custo(modify(id_agente, (_,Custo-por-hora custo_por_hora,_,_,_,_,_), agentes_desenv), identificador, novo_custo)
= **if** id_agente = identificador
then modify(id_agente, (_,Custo-por-hora novo_custo,_,_,_,_,_), agentes_desenv)
else modify(id_agente, (_,Custo-por-hora custo_por_hora,_,_,_,_,_), *altera-custo*(agentes_desenv, identificador, novo_custo))

altera_custo(empty-mapping,_,_) = empty-mapping

inclui_habilidade(modify(id_agente,(_,_,Habilidades habilidades,_,_,_,_), agentes_desenv), identificador, característica, grau_habilidade)
= **if** id_agente = identificador
then modify(id_agente, (_,_,Habilidades modify(característica,grau_habilidade,habilidades)
,_,_,_,_), agentes_desenv)
else modify(id_agente, (_,_,Habilidades habilidades,_,_,_,_), *inclui_habilidade*(agentes_desenv, identificador, característica,grau_habilidade))

inclui_habilidade(empty-mapping,_,_,_) = empty-mapping

exclui_habilidade(modify(id_agente, (_,_,Habilidades habilidades,_,_,_,_), agentes_desenv), identificador, característica)
= **if** id_agente = identificador
then modify(id_agente, (_,_,Habilidades Restrict_with(habilidades,
add(empty-set,característica)), _,_,_,_), agents_desenv)
else modify(id_agente, (_,_,Habilidades habilidades,_,_,_,_), *exclui_habilidade*(agentes_desenv, identificador, característica))

exclui_habilidade(empty-mapping,_,_) = empty-mapping

altera_habilidade(modify(id_agente, (_,_,Habilidades habilidades,_,_,_,_), agentes_desenv), identificador, característica, novo_grau_habilidade)
= **if** id_agente = identificador
then modify(id_agente, (_,_,Habilidades *altera_hab_aux*(habilidades, característica,
novo_grau_habilidade,),_,_,_,_), agentes_desenv)
else modify(id_agente, (_,_,Habilidades habilidades,_,_,_,_), *altera_habilidade*(agentes_desenv, identificador, característica, novo_grau_habilidade))

altera_habilidade(empty-mapping,_,_,_) = empty-mapping

inclui_afinidade(modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), agentes_desenv), identificador, identificador2, grau_afinidade)
= **if** id_agente = identificador
then modify(id_agente, (_,_,_,Afinidades modify(identificador2, grau_afinidade,
afinidades),_,_,_), agentes_desenv)
else modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), *inclui_afinidade*(agentes_desenv, identificador, identificador2,grau_afinidade))

inclui_afinidade(empty-mapping, _, _, _) = empty-mapping

exclui_afinidade(modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), agentes_desenv), identificador, identificador2)
= **if** id_agente = identificador
then modify(id_agente, (_,_,_,Afinidades Restrict_with(afinidades, add(empty-set,
identificador2)),_,_,_), agentes_desenv)
else modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), *exclui_afinidade*(agentes_desenv, identificador, identificador2))

exclui_afinidade(empty-mapping, _, _) = empty-mapping

altera_afinidade(modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), agentes_desenv), identificador, identificador2, novo_grau_afinidade)
= **if** id_agente = identificador
then modify(id_agente, (_,_,_,Afinidades *altera_afin_aux*(afinidades, identificador2,
novo_grau_afinidade),_,_,_), agentes_desenv)
else modify(id_agente, (_,_,_,Afinidades afinidades,_,_,_), *altera_afinidade*(agentes_desenv, identificador, identificador2, novo_grau_afinidade))

altera_afinidade(empty-mapping,_,_,_) = empty-mapping

1.2.2 Operações de apoio ao gerenciamento

A seguir são apresentadas operações internas do ATO AGENTES-DESENV que são requisitadas por operações que realizam funções de gerenciamento do ATO.

Interface:

inclui_proj_agenda_aux(_,_): AGENDA, STRING	-> AGENDA
exclui_proj_agenda_aux(_,_): AGENDA, STRING	-> AGENDA
inclui_ativ_agenda(_,_,_,_): AGENDA, STRING, STRING, STRING	-> AGENDA
exclui_ativ_agenda(_,_,_,_): AGENDA, STRING, STRING	-> AGENDA
exclui_ativ_list(_,_,_,_): LISTA_ATIVIDADES, STRING	-> LISTA_ATIVIDADES
altera_estado_ativ_agenda(_,_,_,_): AGENDA, STRING, STRING, STRING	-> AGENDA
altera_estado_list_ativ(_,_,_): LISTA_ATIVIDADES, STRING, STRING	-> LISTA_ATIVIDADES
altera_hab_aux(_,_,_): HABILIDADE, STRING, REAL	-> HABILIDADE
altera_afin_aux(_,_,_):AFINIDADE, STRING, REAL	-> AFINIDADE
inclui_base_aux(_): BCI	-> BCSIM

Operação

Variáveis Formais:

agentes_desenv	:AGENTES_DESENV
identificador, identificador2, característica, carac, nome_proj, nome_projeto, nome_ativ, estado_ativ, nome_at, novo_estado	: STRING
grau_habilidade, grau_afinidade novo_grau_habilidade, novo_grau_afinidade	: REAL
agenda	: AGENDA
lista-ativs	: LISTA_ATIVIDADES
habilidades	: HABILIDADES
afinidades	: AFINIDADES

inclui_proj_agenda_aux((nome_proj,_,agenda), nome_projeto)
= **if** nome_proj = nome_projeto
then modify(nome_proj,_,agenda)
else modify(nome_proj,_, *inclui_proj_agenda_aux*(agenda, nome_projeto)

inclui_proj_agenda_aux(empty-mapping, nome_projeto)
= modify(nome_projeto, empty_list, empty-mapping)

exclui_proj_agenda_aux((nome_proj,_,agenda), nome_projeto)
= Restrict-with(agenda, add(empty-set, nome_projeto))

inclui_ativ_agenda(modify(nome_proj, lista_ativs, agenda), nome_projeto, nome_ativ, estado_ativ)
= **if** nome_proj = nome_projeto
then modify(nome_proj, cons((nome_ativ,estado_ativ, “”, “”, “”, “00:00”, 0.0), lista_ativs), agenda)
else modify(nome_projeto, lista_ativs, *inclui_ativ_agenda*(agenda, nome_projeto, nome_ativ, estado_ativ))

inclui_ativ_agenda(empty-mapping,_,_,_) = empty-mapping

exclui_ativ_agenda(modify(nome_proj, lista_ativs, agenda), nome_projeto, nome_ativ)
= **if** nome_proj = nome_projeto
then modify(nome_proj, *exclui_ativ_list*(lista_ativs, nome_ativ), agenda)
else modify(nome_proj, lista_ativs, *exclui_ativ_agenda*(agenda, nome_projeto, nome_ativ))

exclui_ativ_agenda(empty-mapping,_,_) = empty-mapping

exclui_ativ_list(cons((Nome nome_at,_,_,_,_,_,_), lista_ativs), nome_ativ)
= **if** nome_at = nome_ativ
then lista_ativs
else cons((Nome nome_at,_,_,_,_,_,_), *exclui_ativ_list*(lista_ativs, nome_ativ))

exclui_ativ_list(empty-list,_) = empty-list

altera_estado_ativ_agenda(modify(nome_proj, lista_ativs,agenda), nome_projeto, nome_ativ, novo_estado)
= **if** nome_proj = nome_projeto
then modify(nome_proj, *altera_estado_list_ativ*(lista_ativs, nome_ativ, novo_estado), agenda)
else modify(nome_proj, lista_ativs, *altera_estado_ativ_agenda*(agenda, nome_projeto, nome_ativ, novo_estado))

altera_estado_ativ_agenda(empty-mapping,_,_,_) = empty-mapping

altera_estado_list_ativ(cons((Nome nome_at,Estado estado_ativ,_,_,_,_,_), lista_ativs), nome_ativ, novo_estado)
= **if** nome_at = nome_ativ
then cons((Nome nome_at,Estado novo_estado,_,_,_,_,_), lista_ativs)
else cons((Nome nome_at,_,_,_,_,_,_), *altera_estado_list_ativ*(lista_ativs, nome_ativ, novo_estado))

altera_estado_list_ativ(empty-list,_,_) = empty-list

altera_hab_aux(modify(característica, grau_habilidade, habilidades), carac, novo_grau_habilidade)
= **if** característica = carac
then modify(característica, novo_grau_habilidade, habilidades)
else modify(característica, grau-habilidade, *altera_hab_aux*(habilidades, carac, novo_grau_habilidade))

altera_hab_aux(empty-mapping,_,_) = empty-mapping

altera_afin_aux(modify(identificador, grau_afinidade, afinidades), identificador2, novo_grau_afinidade)
= **if** identificador = identificador2
then modify(identificador, novo_grau_afinidade, afinidades)
else modify(identificador, grau_afinidade, altera_afin_aux(afinidades, identificador2, novo_grau_afinidade))

altera_afin_aux(empty-mapping,_,_) = empty-mapping

inclui_base_aux(base) = **ICS**(BCSIM, inclui_base, base)

1.2.3 Operações de observação e consulta

A seguir são apresentadas operações do ATO AGENTES-DESENV que são utilizadas para informar se algum componente do ATO possui determinada características ou para consultar alguma característica pertencente aos componentes.

Interface:

existe_agente_des(_,_): AGENTES-DESENV, STRING -> BOOLEAN
existe_atividade_pronta(_,_): AGENTES-DESENV, STRING -> BOOLEAN
existe_atividade_pronta_aux(_): AGENDA -> BOOLEAN
procura_atividade_estado (_): LISTA-ATIVIDADES, STRING -> BOOLEAN
existem_atividades_prontas(_,_):AGENTES-DESENV, STRING -> BOOLEAN
existem_atividades_prontas_aux(_): AGENDA -> BOOLEAN
existem_atividades_prontas_projeto(_): LISTA-ATIVIDADES -> BOOLEAN
retorna_atividade_pronta(_,_):AGENTES-DESENV, STRING -> STRING
retorna_atividade_pronta_aux(_): AGENDA -> STRING

procura_atividade_projeto(_): LISTA-ATIVIDADES, STRING -> STRING
retorna_projeto_pronto(_,_):AGENTES-DESENV, STRING -> STRING
retorna_projeto_pronto_aux(_): AGENDA -> STRING
retorna_atividade_ativa(_,_):AGENTES-DESENV, STRING -> STRING
retorna_atividade_ativa_aux(_): AGENDA -> STRING
retorna_projeto_ativo(_,_):AGENTES-DESENV, STRING -> STRING
retorna_projeto_ativo_aux(_): AGENDA -> STRING
estado_atividade_agente(_,_,_,_): AGENTES-DESENV, STRING, STRING, STRING -> STRING
estado_atividade_agenda(_,_,_,_): AGENDA, STRING, STRING -> STRING
estado_atividade_agenda_projeto(_,_,_,_): LISTA-ATIVIDADES, STRING -> STRING
retorna_pe(_,_):AGENTES-DESENV, STRING -> STRING
retorna_tpe(_,_):AGENTES-DESENV, STRING -> REAL
escolher_atividade_projeto(_,_): AGENTES-DESENV, STRING -> CONJ-STRING
retorna_atividades_prontas(_): AGENDA -> CONJ-STRING
retorna_atividades_agenda_projeto(_,_): STRING, LISTA-ATIVIDADES -> CONJ-STRING

Operações

Variáveis Formais:

agentes_desenv : AGENTES-DESENV
id-agente, agente, nome_projeto, estado,
nome_proj, nome_atividade, pe : STRING
tpe : REAL
agenda : AGENDA
agenda_projeto : LISTA-ATIVIDADES
atividade : ATIVIDADE
base : BCSIM

existe_agente_des(agentes_desenv,id_agente) = id_agente ∈ domain(agentes_desenv)

existe_atividade_pronta(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
= **if** id_agente = agente
then *existe_atividade_pronta_aux*(agenda)
else *existe_atividade_pronta*(agentes_desenv, agente)

existe_atividade_pronta(empty-mapping,_) = FALSE

existe_atividade_pronta_aux(modify(nome_projeto, agenda_projeto, agenda))
= **if** *procura_ atividade_estado* (agenda_projeto, "Pronta")
then TRUE
else *existe_atividade_pronta_aux*(agenda)

existe_atividade_pronta_aux(empty-mapping) = FALSE

procura_atividade_estado (Cons(atividade, agenda_projeto), estado)
= **if** select-estado(atividade) = estado
then TRUE
else *procura_atividade_estado* (agenda_projeto, estado)

procura_atividade_estado (empty-list,_) = FALSE

existem_atividades_prontas(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
= **if** id_agente = agente
then *existem_atividades_prontas_aux*(agenda)
else *existem_atividades_prontas*(agentes_desenv, agente)

existem_atividades_prontas(empty-mapping,_) = FALSE

```
existem_atividades_prontas_aux(modify(nome_projeto, agenda_projeto, agenda))
=       if existem_ atividades_prontas_projeto(agenda_projeto)
        then TRUE
        else if procura_estado_aividade(agenda_projeto, "Pronta")
             then  if existe_atividade_pronta(agenda)
                   then TRUE
                   else FALSE
             else existem_atividades_prontas_aux(agenda)

existem_atividades_prontas_aux(empty-mapping) = FALSE

existem_atividades_prontas_projeto(Cons(atividade, agenda_projeto))
=       if select-estado(atividade) = "Pronta"
        then if existe_atividade_pronta_projeto(agenda_projeto)
             then TRUE
             else FALSE
        else existem_atividades_prontas_projeto(agenda_projeto)

existem_atividades_prontas_projeto(empty-list) = FALSE

retorna_atividade_pronta(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
=       if id_agente = agente
        then retorna_atividade_pronta_aux(agenda)
        else retorna_atividade_pronta(agentes_desenv, agente)

retorna_atividade_pronta_aux(modify(nome_projeto, agenda_projeto, agenda))
=       if procura_atividade_estado (agenda_projeto, "Pronta")
        then procura_atividade_ projeto(agenda_projeto, "Pronta")
        else retorna_atividade_pronta_aux(agenda)

procura_atividade_projeto(Cons(atividade, agenda_projeto), estado)
=       if select-estado(atividade) = estado
        then select-nome(atividade)
        else procura_atividade_ projeto(agenda_projeto)

retorna_projeto_pronto(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
=       if id_agente = agente
        then retorna_projeto_pronto_aux(agenda)
        else retorna_projeto_pronto(agentes_desenv, agente)

retorna_projeto_pronto_aux(modify(nome_projeto, agenda_projeto, agenda))
=       if procura_atividade_estado (agenda_projeto, "Pronta")
        then nome_projeto
        else retorna_projeto_pronto_aux(agenda)

retorna_atividade_ativa(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
=       if id_agente = agente
        then retorna_atividade_ativa_aux(agenda)
        else retorna_atividade_ativa(agentes_desenv, agente)

retorna_atividade_ativa_aux(modify(nome_projeto, agenda_projeto, agenda))
=       if procura_estado_atividade(agenda_projeto, "Ativa")
        then procura_atividade_ projeto(agenda_projeto, "Ativa")
        else retorna_atividade_ativa_aux(agenda)

retorna_projeto_ativo(modify(id_agente, (Agenda agenda,_,_,_,_,_,_), agentes_desenv), agente)
=       if id_agente = agente
        then retorna_projeto_ativo_aux(agenda)
        else retorna_projeto_ativo(agentes_desenv, agente)
```

```
retorna_projeto_ativo_aux(modify(nome_projeto, agenda_projeto, agenda))
=       if procura_atividade_estado (agenda_projeto, "Ativa")
        then nome_projeto
        else retorna_projeto_ativo_aux(agenda)


estado_atividade_agente(modify(agente, (Agenda agenda,_,_,_,_,_,_),agentes_desenv), id_agente,
nome_projeto, nome_atividade)
=       if agente = id-agente
        then estado_atividade_agenda(agenda, nome_projeto, nome_atividade)
        else estado_atividade_agente(agentes-desenv, id-agente, nome_projeto, nome_atividade)

estado_atividade_agenda(modify(nome_proj, agenda_projeto, agenda), nome_projeto, nome_atividade)
=       if  nome_proj = nome_projeto
        then estado_atividade_agenda_projeto(agenda_projeto, nome_atividade)
        else estado_atividade_agenda(agenda, nome_projeto, nome_atividade)

estado_atividade_agenda_projeto(Cons(atividade, agenda_projeto), nome_atividade)
=       if nome_ativ = nome_atividade
        then select-estado(atividade)
        else estado_atividade_agenda_projeto(agenda_projeto,nome_atividade)

retorna_pe(modify(id_agente, (_,_,_,_,_,_,pe pe), agentes_desenv), agente)
=       if id_agente = agente
        then pe  else retorna_pe(agentes_desenv, agente)

retorna_tpe(modify(id_agente, (_,_,_,_,_,tpe tpe,_), agentes_desenv), agente)
=       if id_agente = agente
        then tpe
        else retorna_tpe(agentes_desenv, agente)

escolher_atividade_projeto(modify(id_agente, (Agenda agenda,_,_,_,BC-Agente base,_,_),
agentes_desenv), agente)
=       if id_agente = agente
        then ICS(BCSIM, solução_base_conhecimento, base, <"escolher atividade projeto",
                retorna_atividades_prontas(agenda)>)
        else escolher_atividade_projeto(agentes_desenv, agente)

retorna_atividades_prontas(modify(nome_projeto, agenda_projeto, agenda))
=       Union(retorna_atividades_agenda_projeto(nome_projeto,agenda_projeto),
retorna_atividades_prontas(agenda))

retorna_atividades_prontas(empty-mapping) = empty-set

retorna_atividades_agenda_projeto(nome_projeto, Cons(atividade, agenda_projeto))
=       if select-estado(atividade) = "pronta"
        then add(add(select-nome(atividade), add(nome_projeto, empty-set)),
         retorna_atividades_agenda_projeto(nome_projeto, agenda_projeto))
        else retorna_atividades_agenda_projeto(nome_projeto, agenda_projeto))

retorna_atividades_agenda_projeto(_,empty-list) = empty-set
```

Bibliografia

[ALO97] ALONSO, G. et al. Distributed Processing over Stand-Alone Systems and Applications. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, VLDB'97, 23., 1997, Athens, Greece. **Proceedings**... Athens: Morgan Kaufmann editors, 1997.

[ALO99] ALONSO, Gustavo. OPERA: A Design and Programming Paradigm for Heterogeneous Distributed Applications. In: INTERNATIONAL PROCESS TECHNOLOGY WORKSHOP, IPTW, 1999, Grenoble, France. **Proceedings...** Disponível em: <http://www-adele.imag.fr/IPTW/Program.htm>. Acesso em: 25. Abr.2001

[ALV97] ALVARES, L.; SICHMAN, J. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 16., 1997, Brasília. **Anais...** Brasília:Unb, 1997. P. 1-37.

[AMB90] AMBRIOLA, V.; CIANCARINI, P.; MONTANGERO, C. Software Process Enactment in Oikos. **Software Engineering Notes**, New York, v. 15, n.6, Dec. 1990. Trabalho apresentado no ACM SIGSOFT Symposium on Software Development Environments, 4., 1990, Irvine, US.

[ARM92] ARMENISE, P. et al. Software Processes Representation Languages: Survey and Assessment. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 4., 1992, Capri, Italy. **Proceedings...** [S.l.]: IEEE Press, June 1992.

[BAN92] BANDINELLI, S et al. The Architecture of SPADE-1 Process-Centered. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS TECHNOLOGY, 2., 1992, Trondheim, Norway. **Proceedings...** Disponível em:

<http://www.elet.polimi.it/section/compeng/se/swproc/public.html>. Acesso em: 25. Abr.2001

[BEC99] BECATTINI, Fabrizio et al. Exploiting MOOs to provide multiple views for Software Process Support. In: INTERNATIONAL PROCESS TECHNOLOGY WORKSHOP, IPTW, 1999, Grenoble, France. **Proceedings...** Disponível em: <http://www-adele.imag.fr/IPTW/Program.htm >. Acesso em: 25. Abr.2001

[BEL91] BELKHATIR, N.; ESTUBLIER, J.; MELO, W. Adele 2 An Approach to software Development Coordination. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS MODELING, 1., 1991, Milan, Italy. **Proceedings...** [S.l.]: AICA Press, 1991.

[BIG98] BIGUS, Joseph P.; BIGUS, Jennifer. **Constructing Intelligent agents with Java**. New York: John Wiley, c1998. 379p.

[BOL93] BOLTE, J.P.; FISHER, J.A.; ERNST, D.H. An object-oriented, message-based environment for integrating continuous, event-driven and knowledge-based simulation. In: APPLICATION OF ADVANCED INFORMATION TECHNOLOGIES: EFFECTIVE MANAGEMENT OF NATURAL RESOURCES. June 18-19, **Proceedings**... Spokane: ASAE, 1993.

[BRO86] BROOKS, R. A Robust Layered Control System for a Mobile Robot. **IEEE Journal of Robotics and Automation**, New York, v. 2, n. 1, p. 14-23, march 1986.

[CHR 95] CHRISTIE, Alan. **Software Process Automation**: The Technology and its adoption. Berlin: Springer Verlag, 1995.

[COL98] COLLOFELLO, James; RUS, I.; RAMKRISHNAN; DOOLEY, K. Using Software Simulation to Assist Software Development Organizations in Making "Good Enough Quality" Decisions. SUMMER COMPUTER SIMULATION CONFERENCE, SCSC98, 1998, Nevada, US. **Proceedings...** [S.1]: SCS Publications, 1998.

[CON 91] CONRADI, R. et al. Initial software process management in EPOS. **Software Engineering Journal**, [S.l.], v. 6, n. 5, Sept. 1991.

[CON 94] CONRADI, R. et al. EPOS: Object-Oriented Cooperative Process Modelling. In: FINKELSTEIN, A. et al. (Ed.). **Software Process Modelling and Technolog**y. Taunton: Research Studies Press, 1994. p. 33-70.

[CUR92] CURTIS, B et al. Process Modeling. **Cummunications of the ACM,** New York, v. 35, n. 9, Sept. 1992.

[DAV79] DAVIES, N.R. **Interactive Simulation Program Generation. Methodology in Systems Modeling and Simulation**. North-Kolland Publishing Company, 1979.

[DOW94] DOWSON, M.; FERNSTRÖN, C. Towards Requirements for Enactment Mechanisms. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS TECHNOLOGY, 3., 1994. Villard de Lans, France. **Proceedings...** Berlin: Springer-Verlag, 1994.

[DRA98] DRAPPA, A.; LUDEWIG, J. Quantitative Modeling for the Interactive Simulation of Software Projects. In: INTERNATIONAL WORKSHOP ON SOFTWARE PROCESS SIMULATION MODELING, ProSim, 1998, Silver Falls, US. **Proceedings…** Disponível em: <http:www.informatik.uni-stuttgart.de/ifi/se/publications/index.html>. Acesso: 07 maio 2001.

[EMM91] EMMERICH, W.; JUNKERMANN, G.; SCHAFER, W. Merlin: Knowledge-based Process Modeling. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS MODELING, 1., 1991, Milan, Italy. **Proceedings...** [S.l.]: AICA Press, 1991.

[FER93] FERNSTRÖN, Christer. PROCESS WEAVER: Adding ProcessSupport to UNIX. In: INTERNATIONAL CONFERENCE ON THE SOFTWARE PROCESS, 2., 1993, Berlin. **Proceedings...** Berlin: IEEE Computer Society Press, 1993.

[FRA99] FRANCH, Xavier; RIBÓ, Josep M. Some Reflexion in the Modeling of Software Process. In: INTERNATIONAL PROCESS TECHNOLOGY WORKSHOP, IPTW, Grenoble, France, September 1999. **Proceedings...** Disponível em: <http://www-adele.imag.fr/IPTW/Program.htm >. Acesso em: 25. Abr.2001

[FRO94] FROELICH, Garry. **Process Modeling Support in Metaview.** Canada: University of Sakatchewan, 1994. Master of Science. Thesis.

[GAR96] GARG, Pankaj K.; JAZAYERI, Mehdi. **Process-Centered Software Engineering Environments.** Los Amitos, California: IEEE Computer Society Press, 1996.

[GIM94] GIMENES, I.M.S. **Uma Introdução ao Processo de Engenharia de Software**: Ambientes e Formalismos. Caxambu, MG: SBC, 1994. 42f. Trabalho apresentado na Jornada de Atualização em Informática, 13., 1994.

[HAD96] HADDADI, A. **Communication and Cooperation in Agent-Systems:** A Pragmatic Theory. Berlin:Springer Verlag, 1996. (Lecture Notes in Computer Science, v. 1056)

[HUM99] HUMPHREY, Watts S. **Managing the Software Process**. New York: Addison-Wesley, 1999.

[KEL 90] KELLNER, M. et al. ISPW-6 Software Process Example. In:INTERNATIONAL SOFTWARE PROCESS WORKSHOP, 6., 1990, Kioto, Japan. **Proceedings...** New York: IEEE Computer Society Press, Oct. 1990.

[LAW84] LAWRENCE, S.R. **An experimental Investigation of Heuristic Scheduling Technics - GSIA.** Pittsburgh:Carnegie-Mellon University, 1984.

[LIM98] LIMA, Carla A. G. **Um gerenciador de Processos de Software para o Ambiente PROSOFT**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[LIM01] LIMA REIS, Carla A. **Instanciação, Validação e Execução de processos de software baseados em conhecimento.** Porto Alegre: PPGC-UFRGS, 2001 (a ser publicado). Proposta de Tese.

[LUD96] LUDEWIG, J.; DEININGER, M. Teaching software project management by simulation: The SESAM project. In: European Conference on Software Quality, 5., 417 – 426. 1996, Dublin. **Proceedings...** Dublin:Irish Quality Association, 1996.

[MAD99] MADACHY, Raymond J.; BOEHM, Barry W. **Software Process Dynamics**. Early draft version 4/99. Los Angeles, California: IEEE Computer Society Press, 1999. Disponível em:<http://sunset.usc.edu/people/ray/spd/>. Acesso em 07.mai.2001

[MEN99] MENG, Teo Y. **Advanced Modelling & Simulation Techniques.** Term Paper Application of AI Technics in Simulation. Submitted to NUS School of Computing. Master of Technology 1998/1999, semester 1. Disponível em: <http://www.comp.nus.edu.sg/~teoym/ic52z1/ai.html>. Acesso em 25.abr.2001

[MI90] MI, P.; SCACCHI,W. A Knowledge-based Environment for Modeling and Simulation Software Engineering Processes. **IEEE Trans. Knowledge and Data Engeneering**, New York, v.2, n.3, p. 283-294, 1990.

[MOR97] MORAES, Sílvia M. W. **Um Ambiente EXPERT para apoio ao Desenvolvimento de Software**. Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de Mestrado.

[NAN81] NANCE, R.E. The Time and State Relationships in Simulation Modeling. **Communications of the ACM**, New York, v.24, n. 4, p. 173-179, April, 1981.

[NUN 92] NUNES, D. J. **Estratégia Data-Driven no Desenvolvimento de Software**. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 6.,1992, Gramado. **Anais...** Porto Alegre: Intituto de Informática/UFRGS,1992. v. 1, p. 81-95.

[NUN94] NUNES, Daltro J. **PROSOFT**. Porto Alegre. CPGCC da UFRGS, 1994. Relatório de pesquisa interno.

[PAG94] PAGE Jr., Ernest **Simulation Modeling Methodolgy:** Principles and Etiology of Decision Support. Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Pholosophy in Computer Science. September, 1994. Disponível em: <http://ms.ie.org/page/papers/unref/dissert/dw.html>. Acesso em:07 maio 2001.

[PAU95] PAULK, Mark et al. **Capability Maturity Model for Software, Version 1.1**. Reading: Addison-Wesley Publishing Company, 1995.

[PET99] PETRIE, Charles; GOLDMANN, Sigrid; RAQUET, Andreas. **Agent-Based Project Management**. Berlin: Springer-Verlag, 1999. (Lecture Notes in Artificial Inteligence, v. 1600)

[REI98] REIS, Rodrigo Q. **Uma Proposta de Suporte ao Desenvolvimento Cooperativo de Software no Ambiente**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[RET99] RETICULAR Systems. **AgentBuilder –** An Integrated Toolkit for Constructing Intelligent Software Agents. San Diego: Reticular Systems, 1999.

[RIC93] RICH, Elaine; KNIGHT, Kevin. **Inteligência Artificial**. São Paulo: Makron Books, 1993.

[RUS98] RUS, Ioana; COLLOFELLO, James; LAKEY, P Software Process Simulation for Reliability Strategy Assessment, In: INTERNATIONAL WORKSHOP ON SOFTWARE PROCESS SIMULATION MODELING, ProSim, 1998, Silver Falls, US. **Proceedings...** Disponível em: <http://enuxsa.eas.asu.edu/~rus/ >. Acesso em 07.mai.2001

[SCA98] SCACCHI, Walt. **Experience with Software Process Simulation and Modeling.** In: INTERNATIONAL WORKSHOP ON SOFTWARE PROCESS SIMULATION MODELING, ProSim, 1998, Silver Falls, US. **Proceedings...** Disponível em: <http://cwis.usc.edu/dept/ATRIUM/index.html>. Acesso: 07 mai. 2001

[SCH96] SCHRIBER, Thomas J.; BRUNNER, Daniel T. Inside Simulation Software: How it works and why it matters. WINTER SIMULATION CONFERENCE, vol. 26, n. 4, 1996. **Proceedings**... [S1]: ORMS, 1996.

[SHI75] SHIMIZU, Tamio. **Simulação em Computador Digital**. São Paulo: Edgard Blucher, 1975.

[SIL99] SILVA, Fábio; REIS, Rodrigo Quites; REIS, Carla Alessandra Lima; NUNES, Daltro. Um Modelo de Simulação de Processo de Software baseado em Agentes Cooperativos. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, SBES, 13., 1999. **Proceedings...** Florianópolis: SBC/UFSC, Oct., 1999.

[SIL00] SILVA, Fábio; REIS, Carla Alessandra;REIS, Rodrigo Quites; NUNES, Daltro José. SimAgentProcess: Uma Ferramenta para Simulação de Processos de Software Baseada em Conhecimento. In: IDEAS, 2000. **Proceedings...** [S1: s.n.], 2000.

[STR84] STRACK, Jair. **GPSS - Modelagem e Simulação de Sistemas**. Rio de Janeiro: LTC, 1984.

[TAY96] TAYLOR, Richard N.; BOLCER, Gregory A. Endeavors: A Process System Integration Infrastructure. In: INTERNATIONAL CONFERENCE ON SOFTWARE PROCESS, ICSP, 4., 1996, Brighton, UK. **Proceedings...** [S.1]:IEEE Press, 1996.

[TIA99] TIAKO, Pierre F.; DERNIAME, Jean-Claude. Modelling Trusted Process Components for Distributed Software Development. In: INTERNATIONAL PROCESS TECHNOLOGY WORKSHOP, IPTW, 1999, Grenoble, France. **Proceedings...** Disponível em: <http://www-adele.imag.fr/IPTW/Program.htm >. Acesso em: 25. Abr.2001

[VIC91] VICCARI, R.M. Sistemas Especialistas. In.: Escola Regional de Informática 2, 1991, Curitiba, PR. **Anais**... Curitiba: [s.n.], 1991.

[YOU94] YOUNG, Patrick S. **Customizable Process Specification and Enactment for Technical and Non-Technical Users**. University of California Irvine, USA, 1994. Ph.D. Dissertation.