THIAGO HENRIQUE CORTEZ

PROPOSTA DE IMPLEMENTAÇÃO ORIENTADA A OBJETOS PARA O SIMULADOR DO DESEMPENHO HUMANO S. PERERE

Assis
2008

PROPOSTA DE IMPLEMENTAÇÃO ORIENTADA A OBJETOS PARA O SIMULADOR DO DESEMPENHO HUMANO S. PERERE

THIAGO HENRIQUE CORTEZ

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis (IMESA), como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Aluno: _____

Orientador: _____

Analisador (1): _____

Analisador (2): _____

Assis
2008

THIAGO HENRIQUE CORTEZ

PROPOSTA DE IMPLEMENTAÇÃO ORIENTADA A OBJETOS PARA O SIMULADOR DO DESEMPENHO HUMANO S. PERERE

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis (IMESA), como requisito do Curso de Graduação.

Orientador: Prof. Dr. Luiz Carlos Begosso

Área de Concentração: Ciência da Computação

Assis
2008

DEDICATÓRIA

Dedico este trabalho à minha família que esteve sempre ao meu lado, e à minha querida avó Josefa, que partiu deste mundo deixando saudades.

AGRADECIMENTOS

A Deus, por me ajudar a cumprir mais esta importante etapa da minha vida e por encorajar-me a enfrentar os desafios encontrados ao longo do caminho.

Ao Prof. Dr. Luiz Carlos Begosso pela constante orientação neste trabalho, mormente pela amizade, paciência e pela confiança em mim depositada.

Aos meus pais, Antonio e Maria pela motivação e incentivo aos estudos, e aos meus irmãos Robson e Lucas. Agradeço a minha família pelo amor, apoio e compreensão nos momentos difíceis.

Aos professores que tive em toda minha vida, pois sem dúvida foram os construtores da base de conhecimento existente em mim, sem esta certamente não seria possível a realização deste trabalho.

Aos meus caros amigos e colegas da Classe, pela motivação e pelos cinco anos que convivemos juntos. Sempre será minha segunda família.

Aos meus amigos e ex-companheiros de trabalho do CEPEIN e a todos os funcionários da FEMA, pela boa convivência e amizade ao longo desses cinco anos.

Também a todos os meus amigos e irmãos na fé, pelo apoio e motivação.

RESUMO

Motivado pela crença de construir uma ferramenta para auxiliar no processo de projeto e melhoria de sistemas interativos, Begosso (2005) propôs o *Simulator of Performance in Error* - S. PERERE, um simulador da ação humana que leva em consideração o erro humano na interação homem-computador. Esta ferramenta é apoiada por uma arquitetura cognitiva denominada ACT-R (The Atomic Components of Thought – Rational), e tem como objetivo a simulação do comportamento humano associado ao conceito de erro. Este trabalho propõe para o S. PERERE uma implementação orientada a objetos com base na sua arquitetura, focando no desenvolvimento dos módulos responsáveis pela entrada de dados no simulador. Neste trabalho faz-se uso das mais modernas técnicas de modelagem e desenvolvimento de software, dando ao S. PERERE características de qualidade no processo de desenvolvimento.

ABSTRACT

Motivated by the belief of building a tool to assist in process design and improvement of interactive systems, Begosso (2005) proposed of the Simulator of Performance in Error - S. PERERE, a simulator of human action that takes into account human error in the interaction man - computer. This tool is supported by a cognitive architecture called ACT-R (The Atomic Components of Thought - Rational), and has the objective of the simulation of human behavior associated with the concept error. This paper proposes for S. PERERE an implementation directed to objects based on its architecture, focusing on development of the modules responsible for data entry in simulator. This work is made more use of modern techniques of modeling and software development, giving the S. PERERE characteristics of quality in the development process.

LISTA DE FIGURAS

Figura 1. Projetista de Sistema Crítico e o S. PERERE **18**
Figura 2. Arquitetura Cognitiva (extraído de Begosso, 2005. p 18). **29**
Figura 3. Organização da informação no ACT-R (Extraído de Begosso (2005), p 75) **31**
Figura 4. Arquitetura do S. PERERE **33**
Figura 5. Packages do Projeto **42**
Figura 6. Packages dos módulos **43**
Figura 7. Packages de builds **43**
Figura 8. Relação entre os packages dos grupos 'modules' e 'builds' **44**
Figura 9. Packages dos comportamentos **45**
Figura 10. Packages dos tipos de erros **45**
Figura 11. Diagrama de classes principais do projeto **46**
Figura 12. Diagrama de classes do padrão de acessos e manipulação de arquivos **48**
Figura 13. Processo de descrição de Interface **50**
Figura 14. Diagrama de classes do Editor de Interface **51**
Figura 15. Recursividade na inserção de objetos **52**
Figura 16. Configuração de objetos de interface **53**
Figura 17. Visualização das propriedades do objeto **54**
Figura 18. Diagrama de classes do compilador de interface **56**
Figura 19. Estrutura dos objetos para a compilação de interface **57**
Figura 20. Processo de descrição de Tarefas **58**
Figura 21. Diagrama de classes do Editor de Tarefas **59**
Figura 22. Objetos da árvore de tarefas **60**
Figura 23. Inserção de tarefas – Elementary Behaviour **61**
Figura 24. Inserção de tarefas – General **62**
Figura 25. Comentário de uma tarefa **63**
Figura 26. Configuração de operador temporal **64**
Figura 27. Símbolos dos operadores temporais **65**
Figura 28. Diagrama de classes do Simulador de Tarefas **66**
Figura 29. Simulador de Tarefas **67**
Figura 30. Tarefa descrita no Editor de Tarefas **69**
Figura 31. Tarefa iniciada no Simulador de Tarefas **69**
Figura 32. Primeira execução da terafa **69**
Figura 33. Traços da simulação de tarefas **70**
Figura 34. Diagrama de classes da compilação do módulo Editor de Tarefas **71**
Figura 35. Diagrama de classes do Editor de Conhecimento Semântico **72**
Figura 36. Configuração de objeto semântico **73**
Figura 37. Visualização das propriedades do objeto semântico **74**
Figura 38. Exemplo de tarefa Calcular **75**
Figura 39. Mapa conceitual do comportamento Calcular **75**
Figura 40. Diagrama de classes da compilação do módulo Descritor de Conhecimento Semântico **76**
Figura 41. IDE do Editor de Interface **77**
Figura 42. IDE do Editor de Tarefas **78**
Figura 43. IDE do Editor de Conhecimento Semântico **79**
Figura 44. Interface do experimento **81**
Figura 45. Tarefas do experimento **81**
Figura 46. Conhecimento Semântico do experimento **82**
Figura 47. Geração de produções ACT-R **83**
Figura 48. Seleção de erros **84**

LISTA DE TABELAS

Tabela 1. Comportamentos do processo cognitivo (extraído de Filgueiras; Vitti (2006), p. 23, adaptado de Begosso (2005) p. 85-109) 20
Tabela 2: Comportamentos do processo perceptivo (extraído de Filgueiras; Vitti (2006), p. 24, adaptado de Begosso (2005) p. 85-109) 20
Tabela 3: Comportamentos do processo motor (extraído de Filgueiras; Vitti (2006), p. 24, adaptado de Begosso (2005) p. 85-109) 21
Tabela 4. Manifestações de Deslizes (adaptado de Filgueiras, 1996, p.31) 24
Tabela 5. Erro em relação à intenção e ao estágio cognitivo (Extraído de Filgueiras, 1996, p.32) 25
Tabela 6. Modos de falha relacionados ao nível de desempenho, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67) 26
Tabela 7. Modos de falha relacionados ao nível de regra, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67) 27
Tabela 8. Modos de falha relacionados ao nível de conhecimento, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67) 27
Tabela 9. Extensões de arquivos do S. PERERE 49
Tabela 10. Ferramentas do Editor de Interface 52
Tabela 11. Lista de eventos dos objetos de interface 55
Tabela 12. Ferramentas do Editor de Tarefas 60
Tabela 13. Estrutura dos objetos do mapa conceitual 73
Tabela 14. Valores compilados das tarefas do experimento 82
Tabela 15. Classificação do comportamento Comparar (Begosso, 2005) 111
Tabela 16. Classificação do comportamento Lembrar (Begosso, 2005) 111
Tabela 17. Classificação do comportamento Calcular (Begosso, 2005) 112
Tabela 18. Classificação do comportamento Decidir (Begosso, 2005) 112
Tabela 19. Classificação do comportamento Escolher (Begosso, 2005) 113
Tabela 20. Classificação do comportamento Verificar (Begosso, 2005) 113
Tabela 21. Classificação do comportamento Interpolar (Begosso, 2005) 114
Tabela 22. Classificação do comportamento Monitorar (Begosso, 2005) 114
Tabela 23. Classificação do comportamento Observar (Begosso, 2005) 115
Tabela 24. Classificação do comportamento Localizar (Begosso, 2005) 115
Tabela 25. Classificação do comportamento Explorar (Begosso, 2005) 116
Tabela 26. Classificação do comportamento Ler (Begosso, 2005) 116
Tabela 27. Classificação do comportamento Teclar (Begosso, 2005) 117
Tabela 28. Classificação do comportamento Clicar (Begosso, 2005) 117
Tabela 29. Classificação do comportamento Posicionar (Begosso, 2005) 118
Tabela 30. Classificação do comportamento Mover (Begosso, 2005) 118
Tabela 31. Classificação do comportamento Ajustar (Begosso, 2005) 119
Tabela 32. Classificação do comportamento Instalar (Begosso, 2005) 119

LISTA DE ABREVIATURAS E SIGLAS

IHC	Interação Homem-computador
S. PERERE	Simulator of Performance in Error
GEMS	Generic Error Modeling System
ACT-R	The Atomic Components of Thought – Rational
CTT	Concurrent Task Tree
CTTE	CTT Environment
POO	Programação Orientada a Objetos
IOO	Implementação Orientada a Objetos
MOO	Modelagem Orientada a Objetos
UML	Unified Modeling Language
XML	eXtensible Markup Language
W3C	World Wide Web Consortium
IDE	Integrated Development Environment

SUMÁRIO

1. INTRODUÇÃO 13

1.1. OBJETIVOS *14*

1.2. JUSTIFICATIVAS *14*

1.3. MOTIVAÇÃO *16*

1.4. ESTRUTURA DO TRABALHO *16*

2. FUNDAMENTOS DO S. PERERE 17

2.1. OBJETIVOS DO S. PERERE *17*

2.2 TAREFAS *18*

2.2.1. Taxonomia dos comportamentos elementares 19

2.3 O ERRO HUMANO *22*

2.3.1. Modelo de Rasmussen 23

2.3.2. Classificação do erro humano 24

2.3.3. Taxonomia do erro humano 26

2.4 CLASSIFICAÇÃO DOS COMPORTAMENTOS ELEMENTARES *28*

2.5 ARQUITETURA COGNITIVA *28*

2.5.1. Arquitetura ACT-R 30

3. ARQUITETURA DO S. PERERE 33

3.1. ESTRUTURA GERAL *33*

3.2. MEMÓRIA DECLARATIVA *34*

3.3. MEMÓRIA PROCEDURAL *34*

3.4. MÓDULO MOTOR E MÓDULO PERCEPTIVO *35*

3.5. DESCRITOR DE INTERFACE *35*

3.6. DESCRITOR DE TAREFAS *36*

3.7. DESCRITOR DE CONHECIMENTO SEMÂNTICO *36*

3.8. MÓDULO PRÉ-PROCESSADOR *37*

3.9. MÓDULO DISPARADOR *37*

3.10. MÓDULO PERTURBADOR *38*

4. IMPLEMENTAÇÃO E MODELAGEM DO S. PERERE 39

4.1. ASPECTOS GERAIS DA IMPLEMENTAÇÃO *39*

4.1.1. Linguagem JAVA 40

4.1.2. Arquivos no S. PERERE 41

4.2. ESTRUTURA DO PROJETO *42*

4.2.1. Principais classes 46

4.2.2. Padrão de acessos e manipulação de arquivos 47

4.3. EDITOR DE INTERFACE *50*

4.3.1. Eventos dos objetos de interface 55

4.3.2. Compilação do modelo de interface 56

4.4. EDITOR DE TAREFAS *58*

4.4.1. Tarefas 61
4.4.2. Configuração dos operadores temporais 64
4.4.3. Simulador de tarefas 66
4.4.3.1. Relações de Causalidade 68
4.4.4. Compilação de tarefas 71
4.5. EDITOR DE CONHECIMENTO SEMÂNTICO *71*
4.5.1. Comportamento Calcular 74
4.5.2. Compilação do conhecimento semântico 76
4.6. AMBIENTE DE DESENVOLVIMENTO INTEGRADO DO S. PERERE *76*
4.6.1. Compilação de Projetos SPR 79
4.7. GERAÇÃO DE CÓDIGO ACT-R *80*
5. EXPERIMENTO REALIZADO 81
6. CONCLUSÕES 85
APÊNDICE 1 – CODIGO ACT-R DO EXPERIMENTO – PRODUÇÃO CORRETA 86
APÊNDICE 2 – CODIGO ACT-R DO EXPERIMENTO – PRODUÇÃO PERTURBADA 97
APÊNDICE 3 – TRAÇOS DO ACT-R – REFERENTE À PRODUÇÃO DO APÊNDICE 1 . 108
APÊNDICE 4 – TRAÇOS DO ACT-R – REFERENTE À PRODUÇÃO DO APÊNDICE 2 . 110
ANEXO 1 – CLASSIFICAÇÃO DOS COMPORTAMENTOS ELEMENTARES 111
7. REFERÊNCIAS BIBLIOGRÁFICAS 120

1. INTRODUÇÃO

A área de IHC (Interação Homem-computador) é conhecida no Brasil desde meados dos anos noventa. Um dos maiores esforços dos cientistas nessa área é a investigação e produção de alternativas que garanta segurança e confiabilidade na interação do ser humano com sistemas computacionais críticos.

Sistemas computacionais críticos são sistemas complexos e intolerantes a erros e falhas, Estes sistemas estão presentes em todos os seguimentos da sociedade e podem ser facilmente observados nos hospitais, nas aeronaves, nos automóveis, nas indústrias, etc.

O nível de automação faz aumentar o grau de complexidade dos sistemas (Reason; Maddox, 2003), que quando operados por seres humanos, estão constantemente vulneráveis a diversos tipos de erros, até porque o erro é uma condição normal do ser humano. O erro humano pode ter um papel positivo no aprendizado e na experiência do operador com determinada tarefa (Masson; Koning, 2001), mas a maioria de suas ocorrências tem causado inúmeros transtornos.

A participação humana em acidentes e incidentes com sistemas críticos aumenta à medida que esses sistemas adquirem maior confiabilidade. Isto ocorre pelo aumento da complexidade desses sistemas que nem sempre são compreensíveis para os seus operadores. Somente um projeto de interação homem-computador eficiente é capaz de garantir um alto grau de confiança na interatividade do usuário com os sistemas computacionais.

Motivado pela crença de construir uma ferramenta para auxiliar no processo de projeto e melhoria de sistemas interativos, Begosso (2005) propôs o *Simulator of Performance in Error* - S. PERERE, um simulador da ação humana que leva em consideração o erro humano na interação homem-computador. Esta ferramenta é apoiada por uma arquitetura cognitiva denominada ACT-R, e tem como objetivo a simulação do comportamento humano associado ao conceito de erro. Em outras palavras pode-se dizer que o S.PERERE desenvolve uma relação do homem, ainda que simulado e o computador.

De acordo com Begosso; Filgueiras (2006) a proposta inicial da construção do simulador apresentava algumas limitações e necessitava de melhorias. Nesse caso destaca-se que a forma de entrada de informações no S. PERERE requer um grande esforço e atenção do projetista para alimentar o conhecimento necessário pelo sistema. Partindo deste princípio Cortez (2007) propôs algumas melhorias para o S. PERERE desenvolvendo novas formas de entrada de dados no simulador.

A partir da necessidade de construir uma nova versão do simulador S. PERERE, para acoplar os módulos propostos por Cortez (2007) aos demais módulos que compõe a ferramenta e também à arquitetura cognitiva ACT-R, propõe-se, neste trabalho, uma implementação orientada a objetos para o simulador em questão.

Desta forma, torna-se necessária a escolha de ferramentas compatíveis e adequadas para atender as metas deste trabalho. No tocante à implementação do simulador, optou-se por utilizar a tecnologia Java e para representar seu modelo serão utilizados alguns recursos da notação *Unified Modeling Language (UML).*

1.1. OBJETIVOS

O objetivo deste trabalho é modelar e implementar o Simulador do Desempenho Humano S. PERERE, a partir da especificação de Begosso (2005), a fim de proporcionar documentação e características de qualidade no processo de desenvolvimento do simulador.

A arquitetura do S. PERERE é dividida em módulos, cada um com suas particularidades. Com relação a esses módulos, é importante ressaltar que o foco principal deste trabalho está nos módulos responsáveis pela entrada de dados no simulador, dando continuidade assim nas propostas de Cortez (2007).

1.2. JUSTIFICATIVAS

A forma mais clara de representar a implementação de um software é através da modelagem, que é a parte integrante do seu desenvolvimento. “Um modelo é uma

abstração de alguma coisa, cujo propósito é permitir que se conheça essa coisa antes de construí-la" (Rumbaugh; Blaha; Premerlani, 1991). No paradigma de Modelagem Orientada a Objetos (MOO) é procurada uma representação clara e eficiente do mundo real, afim de, facilitar o desenvolvimento, implementação e manutenção de um software.

Para representar os aspectos da implementação proposta deste trabalho, faz-se uso de alguns recursos da notação gráfica Unified Modeling Language (UML). Uma das grandes vantagens da UML é o fato dela ser totalmente extensível e adaptável. De acordo com a visão de Booch; Rumbaugh; Jacobson (2000), não se adapta a modelagem à UML, mas selecionam-se os elementos da UML que melhor expressarão a modelagem pretendida.

Diversos estudos entre eles os de Rumbaugh; Blaha; Premerlani (1991) demonstram que uma modelagem UML eficiente pode facilitar o entendimento e a implementação de softwares. Compartilha-se desta mesma visão para a modelagem do simulador aqui proposta, destacando o fato de ser compatível com a linguagem de programação selecionada para a implementação, a Linguagem Java.

A escolha da tecnologia Java se dá por diversas razões, além de possuir uma linguagem totalmente orientada a objetos ela pertence à família das ferramentas Open Source, é distribuída, interpretada, robusta, segura, portável, multi-thread e dinâmica. Em virtude das vantagens significativas da plataforma Java sobre outras linguagens, e do grande uso que a comunidade científica faz desta tecnologia é que se fez a opção por seu uso neste projeto.

Na área científica ressalta-se o exemplo da linguagem Java na construção do software CTTE (Concur Task Trees Environment), interface gráfica *opensource* que permite a descrição de tarefas através ícones (que representam as tarefas) e ligações entre eles (relação temporal). Esta ferramenta computacional foi desenvolvida e está baseada no modelo de tarefas, CTT (Concurs Task Trees) proposto por Mori; Paterno; Santoro (2002).

1.3. MOTIVAÇÃO

Segundo Begosso (2005), apesar dos avanços tecnológicos, nota-se que apenas recentemente surgiram preocupações com o elemento humano na interação com sistemas computacionais.

Sabendo que aumentando a qualidade de projeto e melhoria de sistemas interativos, aumentará também a segurança e confiabilidade na Interação Homem-computador (IHC), diminuindo os riscos de acidentes e incidentes causados por falhas neste setor, este trabalho propõe uma contribuição significativa para a concepção do simulador S. PERERE.

1.4. ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em seis capítulos, sendo o primeiro esta introdução, que demonstra a área para qual este trabalho contribui, assim como a importância de estudar a automação dos Sistemas Críticos. No capítulo 2, apresenta as teorias e conceitos que serviram de base para a construção do simulador S. PERERE. O capítulo 3 destaca a estrutura arquitetônica do simulador e os objetivos de cada módulo. São apresentados no capítulo 4, todos os aspectos inerentes à implementação orientada a objetos do simulador e a sua integração com a arquitetura cognitiva ACT-R. O capítulo 5 apresenta como experimento uma tarefa sendo modelada no S. PERERE e executada pela arquitetura cognitiva. Finalmente, o capítulo 6 discute os aspectos importantes obtidos a partir da elaboração deste trabalho. São discutidas também as perspectivas de futuros trabalhos.

2. FUNDAMENTOS DO S. PERERE

Para compreender melhor os objetivos do trabalho aqui proposto, torna-se necessário uma visão geral da base teórica e conceitual na qual o simulador S. PERERE se fundamenta.

Neste capítulo é apresentada, resumidamente, a base teórica e conceitual na qual o S. PERERE está apoiado, iniciando pelos objetivos do simulador, em seguida demonstrando o conceito de tarefas e suas relações com as unidades elementares de comportamento humano, que representam a interação homem-computador. Também são demonstradas as tarefas como principal produto de entrada de informações no simulador, assim como um breve estudo do conceito de Erro Humano, e suas classificações, que são muito importantes, pois conduzem a construção do simulador no tocante aos tipos de erros que ele produzirá. Finalmente é apresentado o conceito de arquiteturas cognitivas e a arquitetura selecionada para a construção do simulador.

2.1. OBJETIVOS DO S. PERERE

O avanço tecnológico obriga as pessoas a se adequarem cada vez mais ao novo paradigma de vida, forçando-as a conviverem com dispositivos automatizados e ainda interagir com eles. Isso é intensificado à medida que os sistemas computacionais ganham mais poder e funcionalidades.

O ser humano vive em um mundo rodeado de sistemas complexos e intolerantes a erros e falhas. Estes sistemas são conhecidos como sistemas críticos e estão presentes em todos os seguimentos da sociedade, podem ser facilmente observados nos hospitais, nas aeronaves, nos automóveis, nas indústrias, etc.

Entretanto esses sistemas, quando operados por seres humanos, estão constantemente sujeitos aos erros. Porém é impossível dispensar o elemento humano desta interação.

Com o objetivo de minimizar os problemas relacionados ao erro humano na interação homem-computador, Begosso (2005) especificou e desenvolveu a

arquitetura do S. PERERE – Simulator of Performance in Error. Esta ferramenta está apoiada por uma arquitetura cognitiva denominada ACT-R (v. seção 2.5.1), que tem o objetivo de simular um operador humano interagindo com alguma interface computacional. Segundo o autor, este simulador pode ser utilizado pelos projetistas em projetos de interfaces para sistemas críticos, visando contribuir com estudos do erro humano através da observação do resultado do simulador, listando todas as possibilidades de erros associados a uma determinada tarefa. Uma boa prática de uso pode ser observada na Figura 1 onde as tarefas são identificadas e modeladas pelo projetista tanto na fase inicial do projeto como nas fases de testes e também nas fases intermediárias.

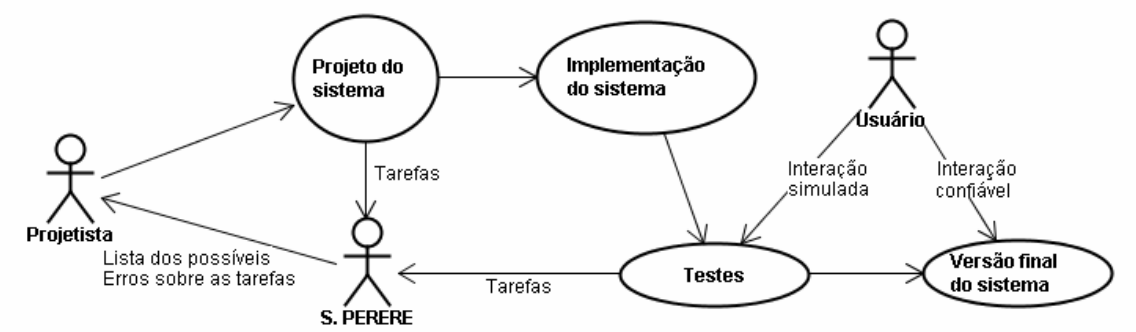

Figura 1. Projetista de Sistema Crítico e o S. PERERE

2.2 TAREFAS

Uma tarefa pode ser entendida como obra ou porção de trabalho que tem de ser concluído num determinado prazo. No contexto deste trabalho as tarefas são as ações do operador humano na interação com alguma interface computacional.

As tarefas humanas, independentemente dos recursos utilizados para cumprí-las, devem ser representadas no simulador através de modelos.

Segundo Begosso (2005) um cientista é levado a construir modelos do mundo real na tentativa de explicar ou compreender aspectos dos fenômenos naturais, de maneira que se possa ter controle sobre certas situações.

A construção de modelos baseados no conhecimento atual da cognição humana é uma tentativa de explicar alguns comportamentos humanos. O Simulador S. PERERE está fundamentado neste conceito, e a forma de entrada das informações

no simulador foi projetada a fim de facilitar a construção de modelos respeitando estes princípios.

O simulador possui um módulo proposto por Cortez (2007), responsável pela descrição de tarefas, que são os elementos principais de entradas de informações no simulador. A construção deste módulo baseou-se no trabalho de Filgueiras; Vitti (2006). Estes autores propuseram uma nova forma de descrever tarefas no simulador com base no modelo CTT proposto por Mori; Paterno; Santoro (2002).

A escolha do modelo CTT como uma forma de descrever tarefas se justifica, segundo Filgueiras; Vitti (2006), pela liberdade de vocabulário que permite tanto o uso de termos próximos aos procedimentos operacionais por parte do usuário do simulador como aplicações de especificações (regras e restrições) para a adequação à sintaxe ACT-R (necessárias no processo de compilação, v seção 2.5.1).

As tarefas são descritas no simulador através de representações de comportamentos elementares na operação de sistemas de controle de processos, que representam o conhecimento sobre como o operador humano executa determinada operação de forma correta.

2.2.1. Taxonomia dos comportamentos elementares

Os comportamentos elementares estão baseados na taxonomia proposta por Berliner; Angell; Shearer (1964), que segundo Begosso (2005) é uma das principais contribuições para a construção do simulador. Esta taxonomia é composta por verbos que traduzem unidades de comportamentos cognitivos, perceptivos, motores e comunicativos e foi desenvolvida justamente para auxiliar na descrição de procedimentos operacionais. A escolha dos verbos segundo os autores se deu pelo freqüente uso em sistemas industriais.

De acordo com Filgueiras; Vitti (2006), o aperfeiçoamento na forma de descrever tarefas no simulador, exigiu algumas modificações nos comportamentos humanos propostos por Begosso (2005) eliminando algumas redundâncias. Nas tabelas

abaixo estão listados os comportamentos elementares da taxonomia de Berliner; Angell; Shearer (1964) utilizados no processo de descrição de tarefas no simulador.

Processo Cognitivo – Comportamentos Simples
Comparar: verifica a igualdade de dois atributos. Se a igualdade for verificada, é realizada uma seqüência de operações. Caso contrário, outra seqüência de operações é tomada.
Lembrar: recupera informação da memória (comportamento intrínseco na arquitetura cognitiva)
Calcular: realiza cálculos binários simples da aritmética (soma, subtração, multiplicação e divisão)
Processo Cognitivo – Comportamentos Complexos
Decidir: opção entre realizar um processo antes do outro, ou escolher entre dois ou mais processos para se atingir a mesma meta baseada em algumas verificações do sistema.
Escolher: como o comportamento Decidir, mas o processo de escolha não depende de verificações ou processos de escolha pré-definidos, ele é resultado de fatores cognitivos intrínsecos do operador.
Verificar: observa um atributo de algum objeto visual e o compara com um valor esperado.
Interpolar: ato de ponderar valores numéricos.

Tabela 1. Comportamentos do processo cognitivo (extraído de Filgueiras; Vitti (2006), p. 23, adaptado de Begosso (2005) p. 85-109)

Processo perceptivo – comportamentos simples
Monitorar: Manter a atenção em determinado objeto do campo visual até alguma característica ser observada.
Observar: toma algum valor de um objeto no monitor. A informação tomada é trazida para a memória.
Localizar: procura por um objeto no monitor.
Processo perceptivo – comportamentos complexos
Explorar: rápido exame dos componentes do campo visual do operador. Leva a sua memória declarativa informações sobre localização e aspecto geral do ambiente.
Ler: aplicação do comportamento observar a objetos tipo texto.

Tabela 2: Comportamentos do processo perceptivo (extraído de Filgueiras; Vitti (2006), p. 24, adaptado de Begosso (2005) p. 85-109)

Processo motor – comportamentos simples
Teclar: diz respeito ao pressionamento de uma tecla do teclado.
Clicar *: diz respeito ao pressionamento do mouse sobre algum objeto na interface.
Posicionar: manipular um controle de estados discretos.
Mover: ato de deslocar um objeto até determinada posição.
Processo motor – comportamentos complexos
Ajustar: Variação do comportamento posicionar. Refere-se ao ajuste de um controle contínuo a determinada posição.
Instalar: Ato de colocar um objeto em posição pré-determinada.

* novo comportamento elementar adicionado por Filgueiras; Vitti (2006).

Tabela 3: Comportamentos do processo motor (extraído de Filgueiras; Vitti (2006), p. 24, adaptado de Begosso (2005) p. 85-109)

Segundo Begosso (2005), esta taxonomia é importante pelos motivos descritos a seguir:

- Pela forma lingüística de verbos representativos do comportamento humano, que facilita a descrição de tarefas no simulador.
- Por estabelecer certa granularidade na especificação das tarefas em um nível genérico, permitindo a representação do comportamento humano em qualquer domínio.
- Por se julgar que os comportamentos humanos à serem afetados de erros pelo simulador podem ser descrito através desta taxonomia.

Esta taxonomia possui representações dos processos do comportamento humano de cognição, percepção, motor e comunicação, mas Begosso (2005) destaca que o processo de comunicação não faz parte do escopo do S. PERERE.

Nesta seção demonstrou-se as relações entre as tarefas que um indivíduo realiza no mundo real com as tarefas elementares representadas através de modelos, que são os maiores veículos condutores ao entendimento do comportamento humano na interação homem-computador. Os detalhes de implementação relacionados ao módulo de descrição das tarefas, assim como os demais módulos do simulador, serão discutidos nos próximos capítulos.

A taxonomia dos comportamentos elementares está relacionada com a saída promovida pelo simulador. È gerada, a partir da entrada, o comportamento do operador humano afetado de erros. Em uma mesma tarefa podem ocorrer inúmeras possibilidades de ações errôneas, ou seja, um operador executando determinada tarefa pode cometer vários tipos de erros.

2.3 O ERRO HUMANO

O erro humano é considerado o principal responsável por uma série de incidentes e acidentes industriais, de acordo com Reason (1990): Bhopal em 1984, Chernobyl e Challenger em 1986 são exemplos. Em ambientes onde o grau de criticidade é baixo, o erro humano ainda pode ser um grande contribuinte para a degradação do desempenho do sistema.

De acordo com Begosso (2005): o termo ‘erro’ pode ser estudado com perspectivas diferentes (engenheiros, psicólogos, sociólogos, etc.), porém é natural existir várias concepções à respeito, mas conforme encontrado na literatura, podemos acreditar que o erro é uma ação ou efeito de errar, desacerto, praticado por desconhecimento, inaptidão ou ignorância. No contexto deste trabalho, o erro está apoiado nas discussões da confiabilidade, que abriga aspectos da confiabilidade humana[1].

As pessoas sempre cometem erros, já que este é uma condição normal do comportamento humano. Nem sempre o erro assume um papel negativo em determinadas situações. De acordo com Masson; Koning (2001): os erros podem ajudar as pessoas na adaptação com as particularidades situacionais de uma tarefa e no aprendizado sobre seu desempenho em relação à mesma. Os problemas acontecem quando estes erros não são detectados e passam por todas as barreiras construídas pelos projetistas e ao se aliarem ao ambiente ou ao processo crítico contribuem para eventos indesejáveis.

[1] **Confiabilidade Humana:** É a probabilidade de um ser humano desempenhar uma função, sob condições específicas, de forma adequada, como previsto no projeto. Garantia de execução de funcionalidades sistêmicas para atender requisitos não-funcionais.

Existem na história da ciência, várias tentativas de reduzir esse erro, algumas com sucesso, outras sem sucesso, mas o certo, é que tem sido um fator contribuinte para a melhoria do projeto da interação homem-computador.

Na próxima seção será apresentado resumidamente o modelo de Rasmussen (1983) que representa características importantes de comportamento humano na realização de alguma tarefa, tal modelo é a base fundamental para o entendimento das classificações dos erros, na qual conduz a construção do simulador S. PERERE no tocante aos erros que ele produzirá sobre as unidades de comportamentos elementares apresentadas na seção anterior.

2.3.1. Modelo de Rasmussen

Para entender psicologicamente o comportamento humano na resolução problemas, Rasmussen (1983) propôs um modelo de processamento humano de informações. Este modelo é capaz de representar as fases existentes no tratamento das informações pelo operador humano, classificando nessas fases três tipos de comportamentos:

- Comportamentos baseados em habilidade (skill-based): relacionado a tarefas que necessitam de habilidades manuais, que normalmente são práticas rotineiras de alguma atividade. Este é o modo em que as pessoas costumam trabalhar na maior parte do tempo.
- Comportamentos baseados em regras (rule-based): relacionado a tarefas controladas por situações pré-existentes. O indivíduo faz uso de regras existentes na base de conhecimento para a execução da ação;
- Comportamentos baseados em conhecimento (knowledge-based): relacionado a tarefas mais complexas, é um nível em que as pessoas entram relutantemente, só em último caso, em novas situações, nas quais não se aplicam nem rotina e nem regra.

Os comportamentos podem se desenrolar paralelamente com outras atividades, podendo originar situações errôneas.

O uso deste modelo por Begosso (2005) se justifica pela facilidade de classificar o erro humano na execução de tarefas. Esta classificação será apresentada a seguir.

2.3.2. Classificação do erro humano

Quase todos os erros que as pessoas cometem no dia a dia são deslizes de atenção ou lapsos (erro, engano, etc.) de memória. Tais erros sempre ocorrem quando se pretende fazer algo e logo o indivíduo defronta-se fazendo outra coisa.

Encontra-se na literatura várias definições para deslizes de atenção ou lapsos, destacamos as seguintes definições, propostas por Reason (1990):

- Deslizes: falhas de atenção e percepção em ações observáveis.
- Lapsos: eventos internos geralmente relacionado a falhas de memória.

Os deslizes ou lapsos são geralmente decorrentes da falta de atenção do indivíduo, caracterizados pela execução de várias tarefas ao mesmo tempo, já que o ser humano é capaz de manter a atenção em uma coisa ou ação a cada momento, mas na realidade, sempre as pessoas se deparam realizando várias coisas ao mesmo tempo.

De acordo com Reason (1990), os deslizes (slips) podem se manifestar como explicado na Tabela 4:

Manifestações	Características
Omissão	Quando um passo do plano deixa de ser executado.
Seleção indesejada	Objeto selecionado evidentemente, por conta da semelhança com o objeto desejado, durante a execução de um passo da tarefa.
Repetição	Repetição de um passo da tarefa já realizado.
Inversão seqüencial	Execução, dos passos da tarefa, fora da ordem prevista.

Tabela 4. Manifestações de Deslizes (adaptado de Filgueiras, 1996, p.31)

De acordo com Begosso (2005), lapso ou engano pode ser classificado em duas classes, de acordo com o nível de atividades em que eles ocorrem.

- Engano no nível de regras: envolve falha na escolha de regras para resolução de problemas. A regra pode ter sido aplicada de forma errada, ou a regra correta foi aplicada em circunstâncias inadequadas, desencadeando situações inapropriadas.
- Engano no nível de conhecimento: se caracteriza pela necessidade da resolução de novos problemas, para os quais o indivíduo não possua regras correspondentes. Nesse caso a solução deve ser apresentada a partir dos seus conhecimentos prévios.

Em resumo, os tipos de erros, de acordo com o modelo de Rasmussen (v. seção 2.3.1), podem ser classificados conforme mostra a Tabela 5:

Comportamento	Estágio Cognitivo	Tipo de erro	Nível de desempenho
Ação não acontece conforme o plano	Execução	deslize	Nível de habilidade (skill-based)
Ação não acontece conforme o plano	Memória	lapso	Nível de habilidade (skill-based)
Plano selecionado não obtém resultado pretendido	Planejamento	engano	Nível de regras (rule-based)
Plano criado não obtém resultado pretendido	Planejamento	engano	Nível de conhecimento (knowledge-based)

Tabela 5. Erro em relação à intenção e ao estágio cognitivo (Extraído de Filgueiras, 1996, p.32)

Deslizes ou lapsos estão ligados ao comportamento especializado, raramente ocorrem durante o aprendizado, quando os atos são conscientes e ainda não automatizados.

Em outras palavras pode-se dizer que o erro humano corresponde ao resultado de processos conscientes que levam as pessoas a tomarem decisões incorretas, pode ser definido como a falha das ações planejadas sem a intervenção de algum evento inesperado.

Na próxima seção será apresentada uma taxonomia do erro humano, que detalha as discussões desta seção em uma linguagem mais próxima da arquitetura do S. PERERE.

2.3.3. Taxonomia do erro humano

Para compreender a natureza dos erros, Reason (1990) desenvolveu um modelo denominado GEMS (Generic Error Modeling System) que objetiva a classificação dos erros. O modelo está organizado de acordo com o modelo de processamento humano de informações proposto por Rasmussen (1983) (v. Seção 2.3.1) e é aceito como a taxonomia mais completa do erro humano. O referido modelo está estruturado de acordo com os três níveis de desempenho: o nível de habilidade (skill-based), o nível de regras (rule-based) e o nível de conhecimento (knowledge-based).

As tabelas abaixo apresentam as relações dos modos de falhas especificados por Begosso (2005) para o S. PERERE. Tais relações correspondem à visão de Reason (1990) sobre os comportamentos errôneos humanos.

Erros humanos relacionados ao desempenho no nível de habilidade
Redução de intencionalidade: o indivíduo possui uma meta para atingir, porém, por algum motivo, ele se esquece do objetivo proposto.
Confusão perceptiva: normalmente no dia a dia, as pessoas realizam tarefas sem prestar atenção naquilo que estão fazendo. Exemplo: abrir a porta do carro com a chave da casa.
Sobrecarga motora*: quando o indivíduo, realizando uma tarefa motora, inicia quase simultaneamente outra tarefa da mesma categoria. O resultado é sobrepor uma delas.
Percepção falsa*: o indivíduo percebe a informação, num painel por exemplo, mas por alguma causa ele pensa não ter visto tal informação.
Omissão: durante a realização de alguma tarefa, o indivíduo omite algum passo da mesma, ou toda a realização da tarefa a partir de certo ponto.
Repetição: o indivíduo repete um ou mais passos da tarefa. Isto normalmente ocorre por achar que uma tarefa não é tão longa quanto ela realmente é.
Inversão: ocorre quando a seqüência original da tarefa é invertida. Por exemplo: um operador precisa pressionar os botões 1, 2 e 3. Uma verificação não adequada dos botões pode mudar toda a ordem de pressionamento.

* novos mecanismos de erros adicionados por Begosso (2005)

Tabela 6. Modos de falha relacionados ao nível de desempenho, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67)

Erros humanos relacionados ao desempenho no nível de regra
Força de regra: em situações onde não exista uma combinação perfeita de regras, provavelmente uma regra parecida ou com a mais alta força[2] seja disparada.
Redundância: refere-se à informação desnecessária para a combinação de uma regra. Uma vez ausente a informação perfeita, a ocorrência da informação redundante pode levar ao disparo de uma regra errada.
Deficiências na codificação: quando as condições de uma regra possuem deficiências na sua codificação, ou foram mal representadas.
Regra errada: a ação realizada está completamente errada.
Sobrecarga: quando o indivíduo, realizando uma tarefa motora, inicia quase simultaneamente outra tarefa da mesma categoria. O resultado é sobrepor uma delas.

Tabela 7. Modos de falha relacionados ao nível de regra, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67)

Erros humanos relacionados ao desempenho no nível de conhecimento
Seletividade: durante a realização de alguma tarefa, o indivíduo focaliza sua atenção em informações não relevantes ao invés de prestar atenção nas características corretas.
Limitação da memória de trabalho: quando o problema excede os recursos da memória de trabalho do indivíduo.
Aprendizado deficiente*: ao realizar uma tarefa que necessite tomada de decisão, o indivíduo apresenta comportamento deficiente não sabendo qual atitude tomar no momento da decisão.

* novos mecanismos de erros adicionados por Begosso (2005)

Tabela 8. Modos de falha relacionados ao nível de conhecimento, especificados para o S. PERERE a partir de Reason (1990) (adapt. de Begosso, 2005. p. 67)

Cada mecanismo de erro apresentado corresponde a um tipo de perturbação nos comportamentos elementares de Berliner; Angell; Shearer (1964). Esta relação pode ser observadas nas tabelas da seção seguinte.

[2] Quanto mais vezes uma regra for disparada, mais forte ela será considerada.

2.4 CLASSIFICAÇÃO DOS COMPORTAMENTOS ELEMENTARES

Esta seção tem por objetivo explicar as relações entre os comportamentos elementares de Berliner; Angell; Shearer (1964) apresentados na seção 2.2.1 com a taxonomia do erro humano de Reason (1990) apresentada na seção anterior.

O Anexo 1 apresenta estas relações, onde cada comportamento elementar está representado em uma tabela, a partir de visão de Begosso (2005), que demonstra a classificação dos mecanismos de acordo com o processo envolvido. Em suma, destaca-se a importância do Anexo 1 uma vez que é apresentado o nível de desempenho e a lista dos erros correspondentes.

As tabelas de classificação dos comportamentos elementares (Anexo 1) provam que, nem todo o tipo de erro pode ocorrer em todos os comportamentos e circunstâncias.

É importante ressaltar que cada mecanismo de erro especificado para o S. PERERE está inteiramente ligado à arquitetura cognitiva na qual o simulador se apóia.

A próxima seção apresenta o conceito de arquitetura cognitiva e as suas relações com o simulador S. PERERE.

2.5 ARQUITETURA COGNITIVA

O Simulador S. PERERE depende de uma arquitetura cognitiva para o seu funcionamento, em outras palavras pode-se dizer que a arquitetura cognitiva é o cérebro do S. PERERE, sendo assim é importante conhecer basicamente o funcionamento de uma arquitetura cognitiva.

De acordo com Begosso (2005), as arquiteturas cognitivas são implementações computacionais de aspectos inerentes aos sistemas cognitivos, perceptivos e motores do ser humano.

Ritter et al. (2002) descrevem que a expressão arquitetura cognitiva possui dois significados:

1. Especificação dos principais módulos e mecanismos inerentes à

cognição humana;

2. Implementação de programas computacionais dessas especificações. Estes significados são separados e distintos, mas são usados indistintamente.

Neste trabalho adota-se o significado empregado em 2. Assim, as arquiteturas cognitivas oferecem uma plataforma para desenvolver modelos cognitivos e manter uma coerência teórica entre estes modelos.

Na forma de implementações computacionais as arquiteturas cognitivas são desenvolvidas com o intuito de expressar o desempenho humano. Elas apresentam um padrão genérico que foi representado por Wickens (1992), este padrão está ilustrado na Figura 2.

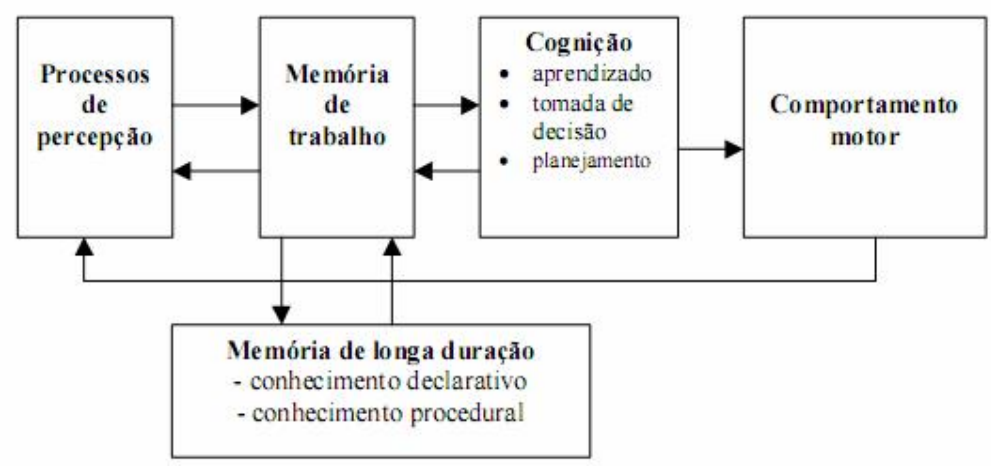

Figura 2. Arquitetura Cognitiva (extraído de Begosso, 2005. p 18).

O processo cognitivo processa as informações vindas do processo de percepção.

As informações são armazenadas no módulo de memória, e estão classificadas em dois tipos:

- Memória de trabalho: armazena informação temporariamente para o processo cognitivo efetuar transformações;
- Memória de longa duração: armazena grandes quantidades de informação que representam o conhecimento do sistema.

O processo de cognição processa as informações do módulo de memória e transforma-as em estímulos característicos aos da cognição humana.

O comportamento motor simula o comportamento desempenhado pelo sistema neuromuscular para realizar ações solicitadas pelos processos cognitivos.

A arquitetura cognitiva selecionada por Begosso (2005) na construção do S. PERERE é a arquitetura ACT-R (The Atomic Components of Thought – Rational) apresentada à seguir.

2.5.1. Arquitetura ACT-R

O ACT-R é uma arquitetura cognitiva de código aberto, que pretende fornecer uma descrição integrada de alguns aspectos da cognição humana. Esta arquitetura foi desenvolvida por Anderson (1993) e Anderson; Lebiere (1998).

De acordo com Leiden et al. (2001), os mecanismos aplicados no ACT-R estão baseados nas considerações de que a cognição humana deveria ser simulada em termos de computação neural tais quais os processos que ocorrem no cérebro humano.

Alguns autores descrevem a arquitetura ACT-R como um conjunto de módulos, em que cada um é responsável por um tipo diferente de processamento de informação.

O ACT-R, de acordo com Budiu (2004), possui três tipos de módulos:

- Módulo visual: responsável por identificar objetos e o módulo motor que controla as atividades motoras no ambiente externo que se relaciona com o ACT-R. Estes dois módulos, chamados de ACTR/ PM, desempenham o papel de interface do sistema com o mundo real.
- Módulos de memória: que distinguem dois tipos de conhecimento de longa duração: conhecimento declarativo (memória declarativa) e conhecimento procedural (memória procedural).
- Módulo de metas: que tem por objetivo sustentar um comportamento cognitivo (goal buffer), mantendo o estado interno (retrieval buffer) do que fora planejado sem que um estímulo externo prejudique este comportamento. Em outras palavras, pode-se dizer que este módulo é

responsável por manter o estado atual durante a resolução de um problema.

A coordenação que estrutura estes módulos é realizada por um sistema central de produção (Figura 3). Este sistema não é sensível a todas as atividades dos módulos descritos acima. Ele responde a um limitado conjunto de informações que são depositados nos buffers destes módulos. Uma importante função do sistema de produção de regras é a atualização dos buffers.

Existem dois tipos de representação de conhecimento no ACT-R: o conhecimento declarativo e o conhecimento procedural.

O conhecimento declarativo descreve os precisando seus componentes elementares e a natureza das relações existentes entre eles.

O conhecimento procedural descreve a organização das ações que permite atingir um objetivo dado.

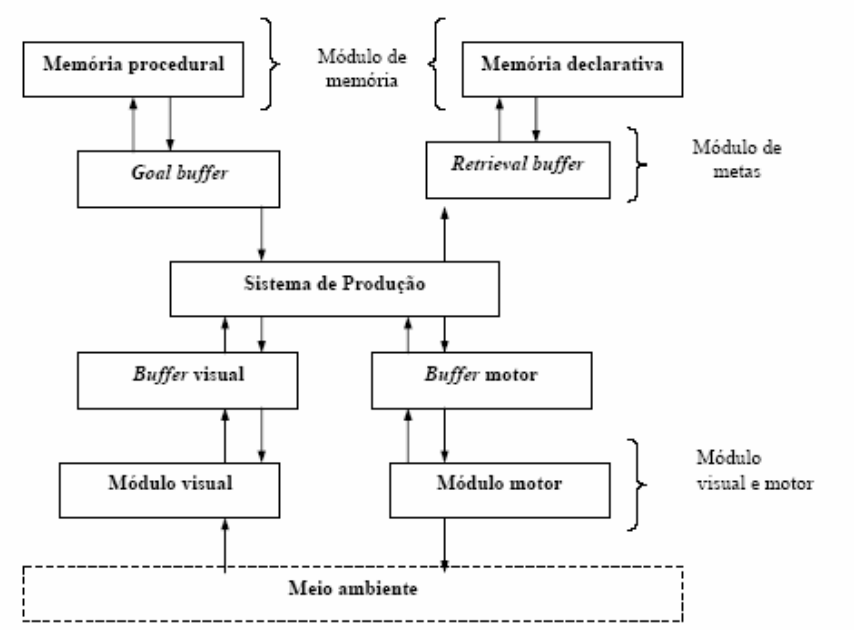

Figura 3. Organização da informação no ACT-R (Extraído de Begosso (2005), p 75)

Dentre várias arquiteturas cognitivas existentes, Begosso (2005) optou por utilizar a ACT-R na construção do S. PERERE por ser a mais completa e a que melhor atende aos propósitos do Simulador. A relação do S. PERERE com a arquitetura cognitiva ACT-R será apresentada nos próximos capítulos.

Neste capítulo apresentou-se o conceito de tarefas e as unidades elementares de comportamento humano, o conceito de erro humano e a sua taxonomia, que é a base fundamental para a construção dos módulos do S. PERERE responsáveis pela perturbação das tarefas. Também apresentou-se uma descrição resumida das características principais de arquitetura cognitiva, e a arquitetura ACT-R. No próximo capítulo, será apresentada a estrutura arquitetônica do simulador S. PERERE, destacando cada módulo componente da ferramenta.

3. ARQUITETURA DO S. PERERE

Pretende-se apresentar neste capítulo uma breve descrição de todos os módulos que compõe o simulador S. PERERE, demonstrando suas características e importância.

O entendimento da estrutura arquitetônica do S. PERERE conduz à modelagem e implementação proposta neste trabalho.

3.1. ESTRUTURA GERAL

A arquitetura do simulador S. PERERE pode ser observada na Figura 4.

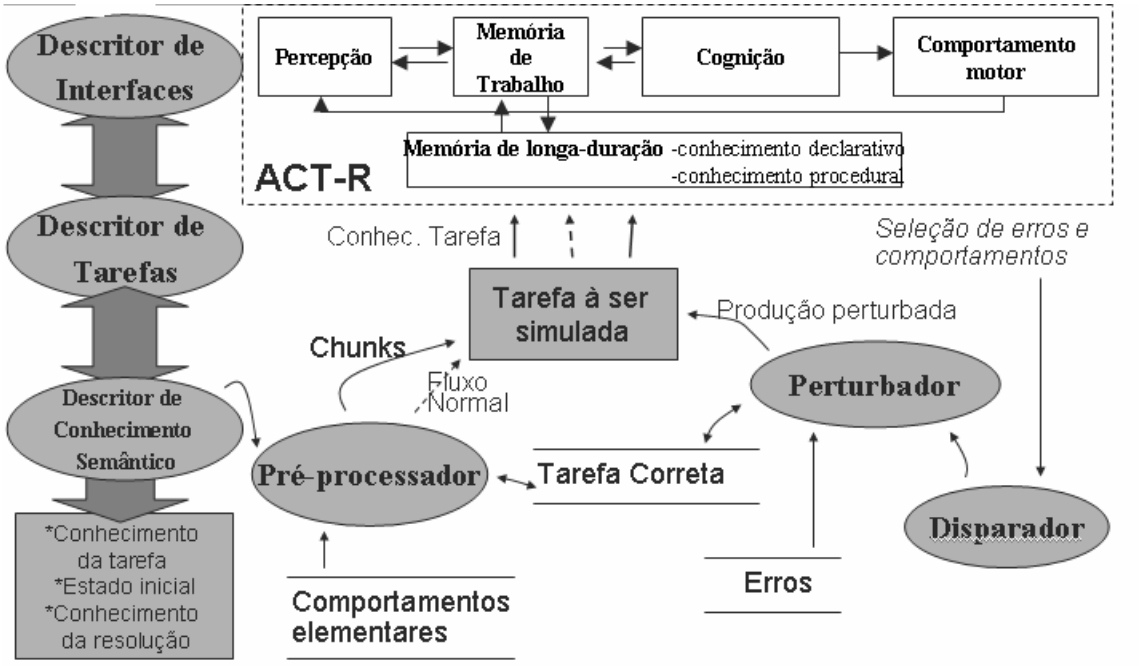

Figura 4. Arquitetura do S. PERERE

Os módulos responsáveis pela entrada de informações no S. PERERE (Descritor de Interfaces, Descritor de Tarefas, Descritor de Conhecimento Semântico) não fazem parte da arquitetura original do simulador, esses módulos foram propostos por Cortez (2007), apresentados e discutidos por Cortez e Begosso (2008) e acoplados

à arquitetura original proposta por Begosso (2005). Nas próximas seções apresenta-se cada um desses módulos.

3.2. MEMÓRIA DECLARATIVA

Este módulo pertence à arquitetura cognitiva ACR-T na qual o simulador está acoplado, porém é fundamental ressaltar sua importância, pois o mesmo é responsável pelo armazenamento das informações que representam as tarefas à serem simuladas.

A memória declarativa faz parte da memória de longa duração do ACT-R, nesta memória são armazenados tanto o conhecimento que o operador possuí para a realização das tarefas (conhecimento declarativo) como o conhecimento a respeito do mundo (conhecimento semântico). Este conhecimento é alimentado pelo usuário do simulador (que é um especialista da tarefa a ser simulada) através dos módulos descritos nas seções 3.5, 3.6 e 3.7.

3.3. MEMÓRIA PROCEDURAL

Esta memória é responsável pelo armazenamento de regras necessárias para a resolução de uma dada tarefa e também faz parte da memória de longa duração da arquitetura ACT-R.

O conhecimento procedural é representado através de regras de produção do tipo condição-ação.

De acordo com Begosso (2005) esta memória estabelece que o sistema faça uso do conhecimento adquirido pela memória declarativa e especifica a forma que o S. PERERE usará este conhecimento para atender as metas propostas.

3.4. MÓDULO MOTOR E MÓDULO PERCEPTIVO

Estes módulos também fazem parte da arquitetura cognitiva ACT-R. Segundo Begosso (2005), estes dois módulos são canais a serem sincronizados com o meio ambiente. No caso do S. PERERE, o ambiente é a interface homem-computador de um sistema de interativo.

O módulo perceptivo verifica no ambiente, informações particulares de uma tarefa e envia-as à memória declarativa.

O módulo motor desempenha comportamentos motores discretos ou contínuos relacionados a uma tarefa descrita na memória procedural.

3.5. DESCRITOR DE INTERFACE

O Descritor de Interface é um dos módulos desenvolvidos para o S. PERERE, com a finalidade de fazer parte de um conjunto de módulos responsáveis pela entrada de dados no Simulador.

Segundo Cortez (2007), o objetivo deste módulo é fornecer ao usuário do S. PERERE recursos para descrever graficamente, de forma simples, qualquer interface computacional, com propósito de simular a interface real de um sistema interativo.

Uma interface possui um conjunto de características com os quais os usuários interagem com as máquinas, dispositivos, programas de computador ou alguma outra ferramenta complexa. Ela fornece métodos para:

- Entrada, permitindo ao utilizador manipular o sistema;
- Saída, permitindo ao sistema produzir os efeitos (as respostas) das ações do usuário.

“O termo interface do usuário, entretanto é usado mais frequentemente no contexto de computadores e outros dispositivos eletrônicos. Para máquinas industriais ou veículos geralmente é usado o termo interface homem-máquina” (Cortez, 2007, p 14).

A representação da interface de um dado sistema como parte do modelo de entrada de dados no S. PERERE, se justifica pela crença de que, a interface de um sistema está associada à funcionalidade do mesmo, servindo de base para a identificação das tarefas humanas.

3.6. DESCRITOR DE TAREFAS

O Descritor de Tarefas é o outro módulo desenvolvido por Cortez (2007) a partir da especificação de Filgueiras; Vitti (2006), que segundo o autor é de fundamental importância para a entrada de dados no simulador, já que esses dados elementos componentes das tarefas.

O Descritor de Tarefas permite que o usuário descreva uma tarefa de forma hierárquica, em forma de árvores de tarefas relacionadas por meio de operadores temporais. A forma de descrever tarefas neste módulo se baseia no modelo de tarefas (CTT) proposto por Mori; Paterno; Santoro (2002).

As tarefas descritas neste módulo devem ser sincronizadas com a interface representada no Descritor de Interface descrita na seção anterior.

3.7. DESCRITOR DE CONHECIMENTO SEMÂNTICO

O módulo Descritor de Conhecimento semântico é um novo módulo que será acoplado aos outros dois módulos apresentados (Descritor de Interfaces e Descritor de Tarefas). Este módulo é responsável por representar o mapa do conhecimento humano de como resolver determinada tarefa de forma correta.

Segundo Turban (1992) o conhecimento semântico refere-se às estruturas cognitivas dos objetos e à forma de armazená-los em memória. Inclui informação sobre: palavras e outros símbolos; significado dos símbolos e regras associadas; relacionamentos entre símbolos como sinônimos e antônimos, e formas de manipulação dos símbolos e conceitos. No caso do S. PERERE, os objetos são as tarefas, que representam as ações do operador humano interagindo-se com alguma

interface computacional. O objetivo da descrição do conhecimento semântico é a definição do conhecimento de resolução dessas tarefas para alimentar a memória declarativa da arquitetura ACT-R.

3.8. MÓDULO PRÉ-PROCESSADOR

Este módulo tem a função de sincronizar as informações obtidas através dos módulos descritores de tarefa (v. seção 3.5, 3.6 e 3.7) com a memória declarativa e procedural da arquitetura cognitiva ACT-R. Esta relação permite ao pré-processador ler cada porção de conhecimento armazenado na memória declarativa e interpretá-los como unidades de comportamento elementar, segundo a taxonomia de Berliner; Angel; Shearer, 1964 (v. seção 2.2.1). Também permite ao pré-processador ler as regras de produção para gerar a tarefa a ser simulada com sintaxe padrão para a execução na arquitetura ACT-R.

O módulo pré-processador também gera a tarefa a ser simulada em sua forma correta, ou seja, sem perturbações.

3.9. MÓDULO DISPARADOR

O módulo disparador representa a mecânica do disparo dos erros, levando em consideração que as perturbações não podem ser totalmente aleatórias porque nem todo o tipo de erro ocorrerá em qualquer momento ou situação. Por exemplo, não se pode disparar um erro relacionado ao comportamento cognitivo enquanto o sistema desempenha uma função do comportamento motor.

De acordo com Begosso (2005) o disparador realiza o sorteio de uma perturbação possível para determinada situação e informa ao perturbador sobre a perturbação escolhida.

3.10. MÓDULO PERTURBADOR

Segundo Begosso (2005) o módulo perturbador é o núcleo do S. PERERE. É importante ressaltar que esta funcionalidade de perturbação, não é encontrada nas arquiteturas cognitivas.

Este módulo é responsável por simular e afetar de erros os comportamentos elementares dos módulos motor e perceptivo da arquitetura ACT-R.

A partir da perturbação escolhida pelo módulo disparador, o perturbador gera situações errôneas que afetam os comportamentos de uma determinada tarefa e envia-as ao pré-processador que gera como saída a tarefa perturbada.

As especificações das perturbações utilizadas neste módulo se baseiam na taxonomia do erro humano de Reason (1990) (v. seção 2.3.3).

“Para cada erro gerado, o S. PERERE cria um arquivo texto, sintaticamente correto do ponto de vista da linguagem ACT-R, para ser executado no ambiente da arquitetura cognitiva, que são as saídas do S. PERERE: tarefa correta, tarefa perturbada e relação de perturbações” (Begosso, 2005, p. 65).

Neste capítulo foram apresentados os objetivos cada módulo que compõe o simulador de acordo com a arquitetura do S. PERERE. No próximo capítulo será apresentada a proposta de implementação orientada a objetos com base nos módulos discutidos neste capítulo, bem como a integração do simulador com a arquitetura cognitiva ACT-R.

4. IMPLEMENTAÇÃO E MODELAGEM DO S. PERERE

Neste capítulo são apresentados todos os aspectos inerentes à implementação do simulador S. PERERE, com base na sua estrutura arquitetônica, apresentada no capítulo anterior.

O conteúdo deste capítulo reflete o produto proposto neste trabalho, que teve o foco no desenvolvimento dos módulos responsáveis pela entrada de informações no S. PERERE e na implementação orientada a objetos (IOO) da sua arquitetura.

Como já mencionado no capítulo anterior, a entrada de dados no S. PERERE é composta por três módulos: *Descritor de Interface, Descritor de Tarefas e Descritor de Conhecimento Semântico*, que serão apresentados nas próximas seções. Esses três módulos em conjunto, devem ser capazes de descrever as ações de um operador humano interagindo com alguma interface de sistema interativo. O conteúdo descrito pelo usuário do S. PERERE nesses módulos serão tratados à partir deste momento como *Projeto SPR*.

Com relação às vantagens de utilizar a IOO, destaca-se a facilidade de dividir o projeto em partes, isto reflete a organização deste capítulo. Cada seção apresentará a implementação de um módulo do S. PERERE com base na sua arquitetura. Finalmente serão apresentados os resultados obtidos através deste trabalho.

4.1. ASPECTOS GERAIS DA IMPLEMENTAÇÃO

Para o desenvolvimento do S. PERERE optou-se por utilizar o padrão de implementação orientado a objetos (IOO), por se tratar de um padrão elaborado para facilitar a vida do programador, no sentido de que existem partes do código que assumem responsabilidades por sobre seus componentes, dados e métodos, mesmo em tempo de execução. Para isto existe um formalismo que deve ser seguido na elaboração da estrutura dos dados e uma necessidade de maior abstração da modelagem do problema.

Fazendo-se uma análise na arquitetura do S. PERERE (Figura 4), observa-se que a aplicação da IOO dá-se de forma natural, com benefícios em muitos aspectos. O S. PERERE tem uma estrutura física bastante adaptável e a forma em que seus módulos se conectam possibilita a representação através de classes de objetos.

RUMBAUGH et al. (1994) afirmam que "... uma operação que tem características próprias deve ser modelada como classe...". Assim, quando uma determinada operação é detectada como pertencente a alguma funcionalidade da arquitetura do S. PERERE, esta assume por sua vez uma importância relevante no projeto, adquirindo uma identidade, e deve ser representada como uma classe.

O ponto de partida para a IOO de um software é a definição da sua estrutura através de classes de objetos. A modelagem dos objetos deve representar estaticamente os elementos do mundo real e suas relações devem servir de base para as próximas etapas do projeto (RUMBAUGH et al., 1994).

De maneira geral um bom projeto orientado a objetos não é uma tarefa fácil (ZHU, 1999; GAMMA et al., 2000). Dentre muitas características que esta metodologia deve oferecer ao projeto, destacam-se a flexibilidade para manutenções e atualizações, estrutura de dados robusta, facilidade de desenvolvimento em equipes, etc. Além disto, um bom software deve ser elegante no sentido de utilizar técnicas modernas e claras de programação e também deve ter um bom desempenho computacional.

Para implementar o S. PERERE fazendo uso destas vantagens, foi necessária a escolha de uma linguagem de programação compatível com este paradigma, assim, escolheu-se a linguagem Java.

4.1.1. Linguagem JAVA

Java vai além de uma simples linguagem de programação, ela é classificada como uma Plataforma de desenvolvimento. A plataforma Java da empresa Sun Microsystem tornou-se uma das mais difundidas ferramentas de desenvolvimento de softwares do mundo.

Uma das características mais importantes desta plataforma é a sua máquina virtual (*Java Virtual Machine - JVM*), que abstrai o equipamento no qual o software é executado, permitindo portabilidade real para vários dispositivos sem a necessidade de sequer re-compilar o código fonte.

A linguagem de desenvolvimento da plataforma Java, possui varias extensões, e cada uma delas possuem várias API's e frameworks, ela também possui vários ambientes de desenvolvimento, não é o objetivo deste trabalho descrevê-los, mas é importante ressaltar os que dão suporte para o desenvolvimento do mesmo.

Utilizou-se na implementação do S. PERERE recursos da extensão JSE (*Java Standard Edition*), que contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a máquina virtual (JVM), o compilador Java e suas API's.

Neste trabalho faz-se uso da API *Java Swing* que dá suporte para a construção de interfaces gráficas. Esta API é de alto nível, ou seja, possuí maior abstração e menos aproximação das API's do sistema operacional, dando aos seus softwares uma aparência independente do sistema operacional utilizado.

O ambiente utilizado para o desenvolvimento do S. PERERE é o *NetBeans.* Criado pela Sun Microsystems, o NetBeans é um ambiente de desenvolvimento integrado gratuito e de código aberto, que suporta todos os recursos da plataforma Java.

A maturidade da plataforma Java atribui ao S. PERERE muitas vantagens significativas, entre elas destacam-se as duas mais observáveis: facilidade de implementação do projeto em partes, descentralizando assim seu processo de desenvolvimento; portabilidade em termos de sistemas operacionais.

4.1.2. Arquivos no S. PERERE

As informações do *Projeto SPR* são salvas e recuperadas através de arquivos. Para estruturar as informações nestes arquivos é utilizada a linguagem XML (eXtensible Markup Language), uma especificação da W3C (World Wide Web Consortium) definida como o formato universal para criar dados estruturados.

A linguagem XML possui uma notável compatibilidade com a linguagem Java. O recurso utilizado neste trabalho para facilitar a manipulação de códigos XML é a biblioteca *JDOM.*

JDOM é um componente que fornece soluções para manipular entrada e saída de dados XML à partir do código Java, pode-se dizer que este componente permite a utilização da linguagem XML através da linguagem Java.

A partir da seção 4.2 uma nova versão do simulador S. PERERE será apresentada, através dos recursos discutidos.

4.2. ESTRUTURA DO PROJETO

Esta seção apresenta a forma em que o projeto foi estruturado a partir da arquitetura do S. PERERE. Esta representação será feita em um nível menor de abstração, ou seja, os principais módulos foram identificados para criar uma estrutura geral do projeto.

Todas as classes e interfaces relacionadas ao projeto estão agrupadas em *Packages (Pacotes).*

Os Packages são mecanismos para organização de elementos em grupos, assim, pode-se dizer que um package é um conjunto de classes e interfaces relacionadas.

Existem algumas vantagens em utilizar packages, como por exemplo: facilidade de encontrar e utilizar as classes; ausência de conflitos no que fiz respeito aos nomes; controle de acessos, etc.

As figuras à seguir ilustram os packages criados na implementação do S. PERERE.

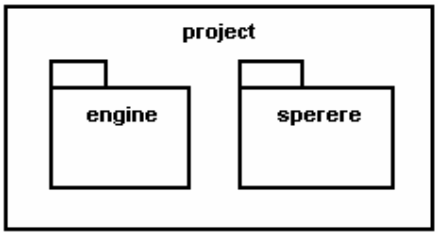

Figura 5. Packages do Projeto

A Figura 5 ilustra os packages que agrupam as classes de características gerais do projeto. O package *engine* é o agrupamento das classes genéricas com métodos acessados pela maioria das classes de outros packages, e o package *sperere* agrupa as classes que representam as principais interfaces gráficas do S. PERERE, são classes estendidas da classe JFrame da biblioteca Swing.

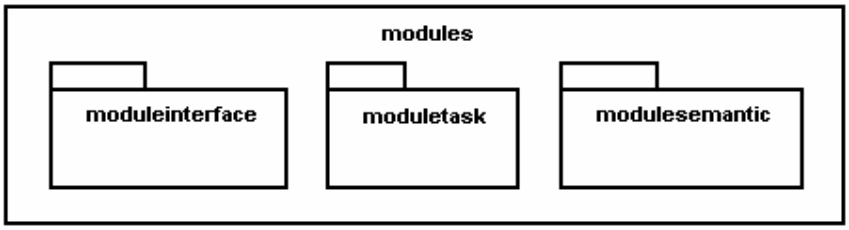

Figura 6. Packages dos módulos

A Figura 6 apresenta os packages que representam os módulos de entradas de dados no simulador. O package *moduleinterface* agrupa as classes específicas do módulo *Descritor de Interface*, o package *moduletask* agrupa as classes pertencentes ao módulo *Descritor de Tarefas* e o package *modulesemantic* agrupa as classes do módulo *Descritor de Conhecimento Semântico.* As classes agrupadas nesses packages são classes distintas que possuem métodos especialistas para manter o funcionamento de seus respectivos módulos.

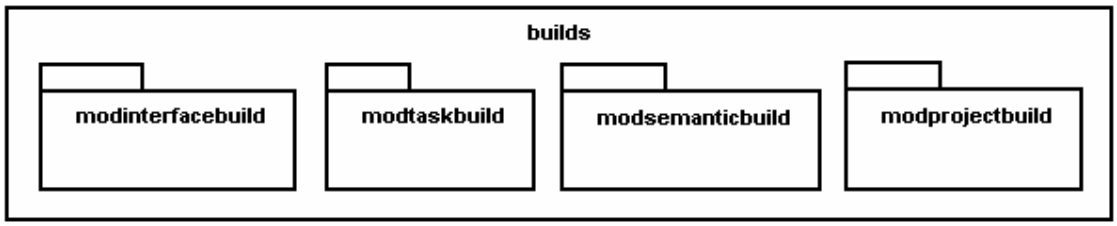

Figura 7. Packages de builds

A Figura 7 ilustra os packages que agrupam classes responsáveis pela validação e compilação do *Projeto SPR,* ou seja, agrupam classes especializadas em validar e compilar o modelo descrito pelo usuário em cada módulo responsável pela entrada de dados no S. PERERE.

Os packages *modinterfacebuild, modtaskbuild* e *modsemanticbuild* agrupam nesta seqüência, classes especialistas na validação dos modelos descritos nos módulos:

Descritor de Interfaces, Descritor de Tarefas e *Descritor de Conhecimento Semântico.* O modprojectbuild será explicado mais adiante, ainda nesta seção.

Esta relação, descrita acima, pode ser observada através da Figura 8.

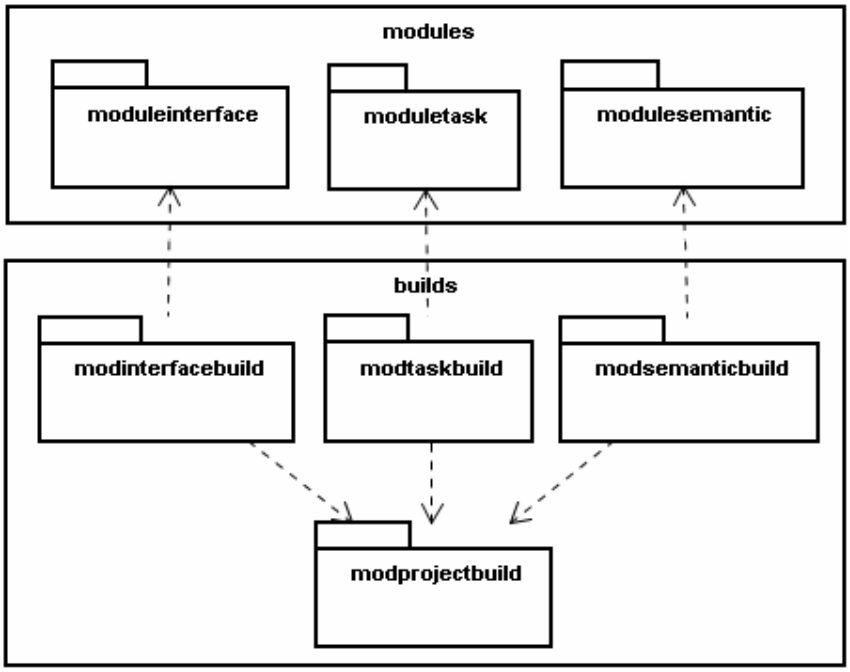

Figura 8. Relação entre os packages dos grupos 'modules' e 'builds'

O funcionamento das classes de cada package *builds*, ocorre através do acesso às classes correspondentes agrupadas no package *modules***.** O package *modprojectbuild* possui uma classe que estabelece conexão com os demais packages do seu grupo, em outras palavras pode-se dizer que, esta classe aciona simultaneamente os métodos de compilação e validação dos três módulos responsáveis pela entrada de dados no simulador, e extrai os traços de compilação reunindo-os em uma única classe.

Essas características dão ao S. PERERE recursos para compilar o modelo em cada módulo separadamente e também para compilar todo o *Projeto SPR* de uma só vez.

Outra característica importante do package *modprojectbuild* é o agrupamento das classes que representam os módulos: *Pré-Processador, Perturbador e Disparador.*

A justificativa para o agrupamento desses módulos no package *modprojectbuild,* é o fato de que tais módulos, não necessitam de packages específicos porque

diferentemente dos módulos de entrada, que necessitam de várias classes para o seu funcionamento, esses módulos podem ser construídos em uma única classe.

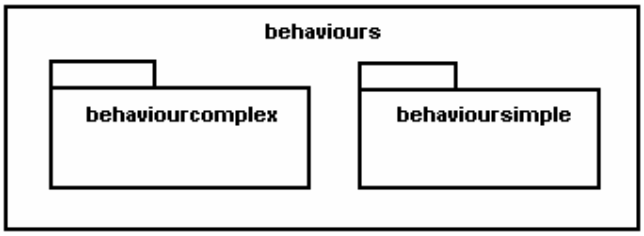

Figura 9. Packages dos comportamentos

A Figura 9 ilustra os packages que agrupam as classes referentes aos comportamentos elementares de Berliner; Angell; Shearer (1964) descritos na seção 2.2.1. O package *behaviourcomplex* agrupa as classes dos comportamentos complexos e o package *behaviioursimple* agrupa as classes que representam os comportamentos simples. Estes comportamentos estão representados na Tabela 1, Tabela 2 e Tabela 3, da seção 2.2.1.

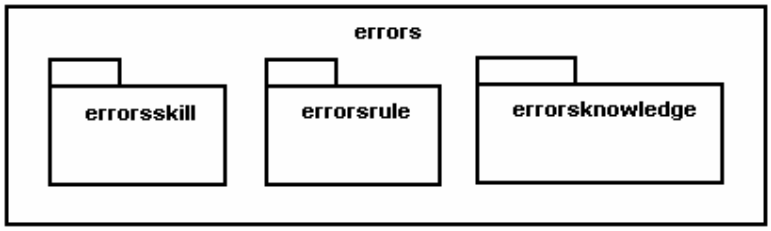

Figura 10. Packages dos tipos de erros

A Figura 10 ilustra os packages que agrupam as classes correspondentes aos tipos de erros, de acordo com os níveis de desempenho humano do modelo de Rasmussen, descrito na seção 2.3.1. O package *errorsskill* corresponde ao agrupamento das classes referentes aos erros do nível de habilidade (Tabela 6), o package *errorsrule* agrupa as classes referentes aos erros do nível de regra (Tabela 7) e o package *errorsknowledge* realiza o agrupamento das classes relacionadas aos erros do nível de conhecimento (Tabela 8).

4.2.1. Principais classes

No contexto deste trabalho considera-se como principais classes, aquelas que dão suporte para execução de todas as demais classes do projeto, pode-se dizer que estas classes são responsáveis pelo ponto de partida da execução do S. PERERE.

Estas classes podem ser observadas através do diagrama de classes da Figura 11.

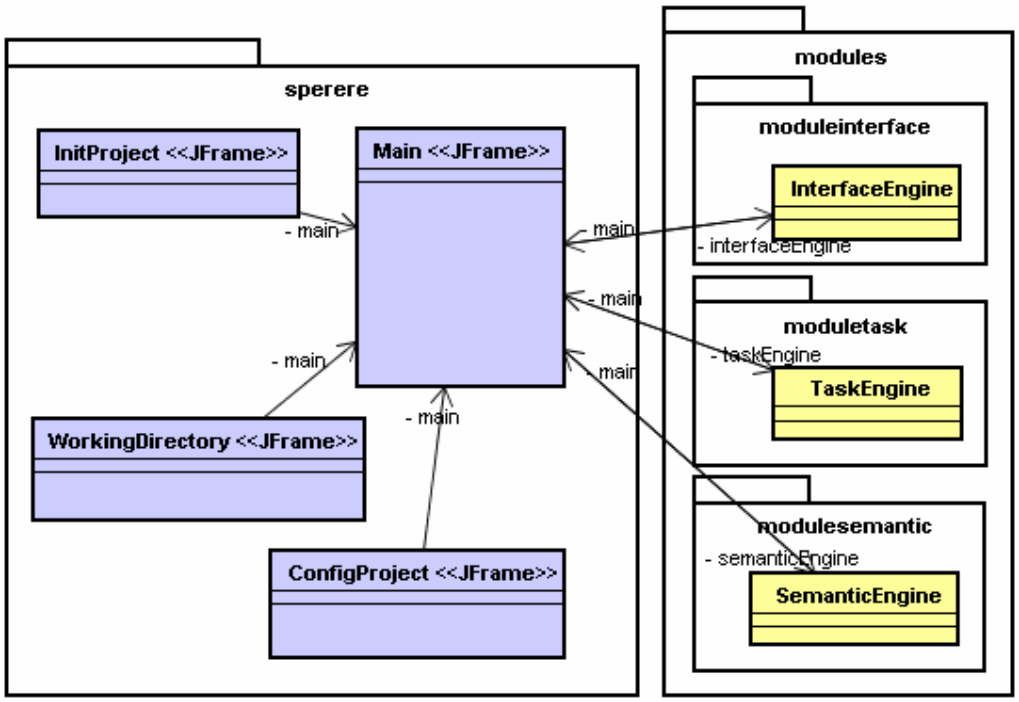

Figura 11. Diagrama de classes principais do projeto

As classes representadas na cor 'azul' são extensões da classe *JFrame*, que pertence ao pacote Swing, responsável por desenhar telas e tratar eventos sobre elas.

A classe *InitProject* é a primeira classe a ser instanciada quando o sistema entra em modo de execução, ela fornece opções para abrir um *Projeto SPR* existente ou para criar um novo.

A classe *WorkingDirectory* é responsável pela configuração dos principais diretórios utilizados pelo S. PERERE, esses diretórios correspondem aos locais padrões para gravação e recuperação de arquivos de acordo com suas respectivas extensões. As informações referentes a esses diretórios são salvos em um arquivo XML, que está alocado no diretório padrão do sistema. Toda vez que o sistema realizar uma ação

de gravar ou recuperar um arquivo, as configurações *WorkingDirectory* serão levadas em consideração.

A classe *ConfigProject* permite a configuração de alguns aspectos do *Projeto SPR* atual, tais como, título, diretório, etc.

A classe *Main* é a principal classe do projeto, pois ela permanece instanciada desde o início da execução do sistema até o final. Ela dá suporte aos ambientes gráficos dos módulos *Editor de Interfaces, Editor de Tarefas* e *Editor de Conhecimento Semântico,* assim como a possibilidade de acesso aos demais recursos do S. PERERE.

As relações da classe *Main* com os módulos de entrada de dados no simulador S. PERERE podem ser observadas na Figura 11. Nota-se que cada um desses módulos possui uma classe em comum (*Engine)*, que representa a mecânica dos seus funcionamentos. Cada uma delas é instanciada logo após o início da classe *Main*. Assim como a classe *Main,* essas três classes também permanecem em memória enquanto o sistema estiver em execução. A próxima seção apresenta um padrão estabelecido para organização dessas classes.

4.2.2. Padrão de acessos e manipulação de arquivos

Estabeleceu-se um padrão entre os módulos de entrada de dados no S. PERERE e a classe *Main,* conforme ilustra a Figura 11. Esse padrão se aplica também aos outros recursos que estes módulos possuem em comum, tais como, acessos aos recursos dos módulos e manipulação de arquivos.

A Figura 12 apresenta o diagrama de classes que representa o padrão das relações entre os módulos *Editor de Interface, Editor de Tarefas* e *Editor de Conhecimento Semântico* e a classe principal do sistema (*Main*). Esta figura representa uma abstração das classes *Engine,* ilustradas na Figura 11. Também são representadas as classe que possuem recursos para gravar e recuperar os arquivos referentes aos projetos dos módulos, de acordo com as suas extensões (Tabela 9).

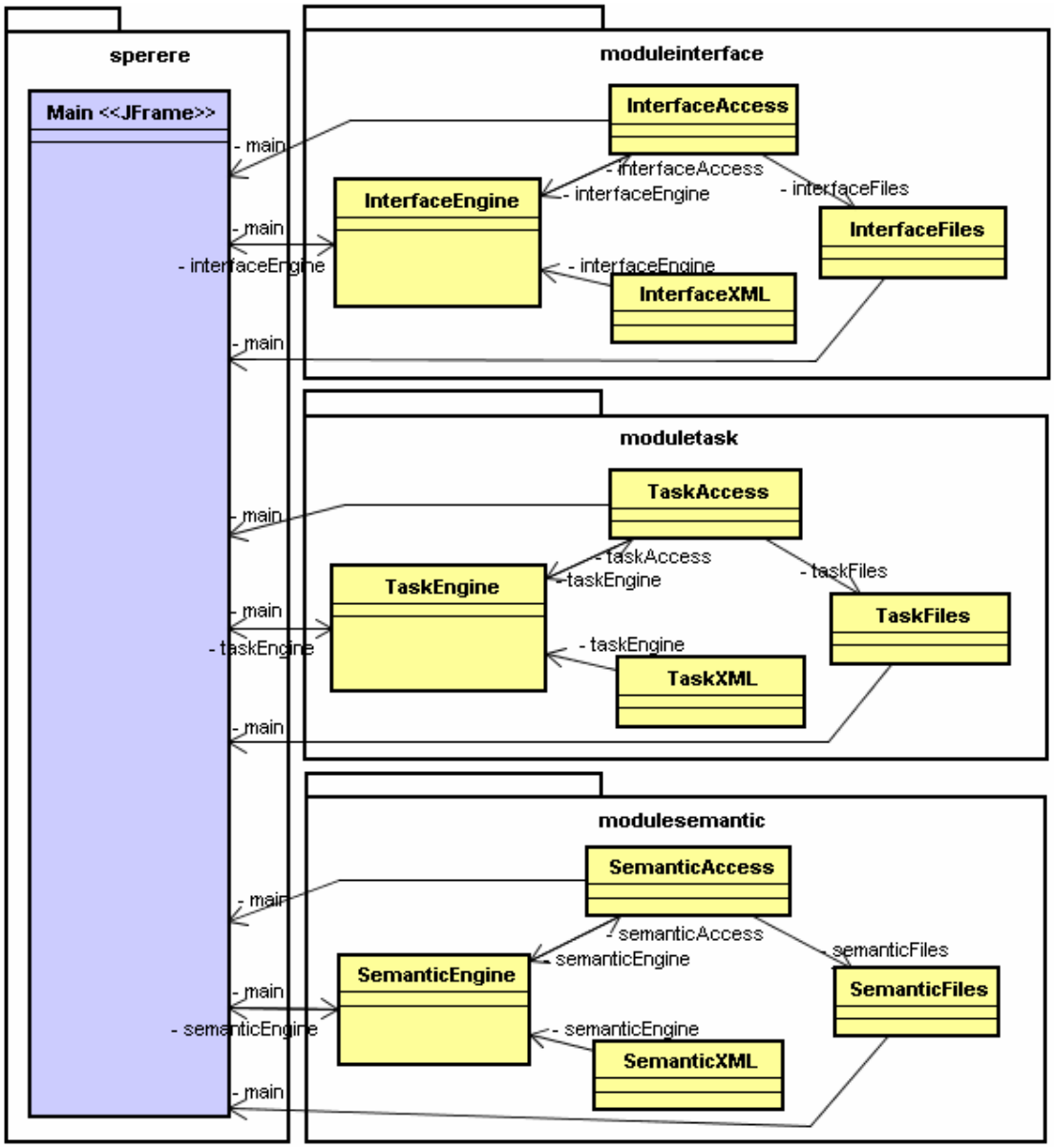

Figura 12. Diagrama de classes do padrão de acessos e manipulação de arquivos

As classes *Engine* representam o motor dos seus respectivos módulos, elas são instanciadas logo após o inicio da classe *Main,* e são responsáveis pelo controle de toda a mecânica do módulo.

O acesso aos recursos destes módulos é feito através das classes *Access,* que são instanciadas a partir das classes *Engine.* Suas relações com a classe *Main* justificam-se pelo fato de que, qualquer recurso do módulo ou de qualquer outra

parte do sistema são disponíveis para o usuário através interface gráfica principal do sistema *(*Main*).*

As classes *XML* possuem métodos para gravar e recuperar arquivos no formato XML. O processo de gravação de um arquivo é realizado da seguinte forma:

- A classe *XML* gera uma *String* referente ao conteúdo criado pelo usuário e retorna-a para a classe *Engine* no formato XML;
- A classe *Engine* acessa através da classe *Access* a instância da classe *Files.*
- A classe *Engine* envia a *String XML* ao método da classe *Files* responsável por salvar determinado arquivo no diretório selecionado.

Finalmente, o processo de recuperação de um arquivo é realizado da seguinte forma:

- A classe *Engine* acessa através da classe *Access* a instância da classe *Files.*
- A classe *Files* retorna à classe *Engine* o arquivo selecionado;
- A classe *Engine* envia o arquivo recuperado para a classe *XML*, que por sua vez converte-o em atributos de objetos;
- A classe *XML* retorna estes atributos à classe *Engine,* que por sua vez aplica-os aos objetos do módulo.

Cada módulo possui recursos para Importar e Exportar seus respectivos arquivos. O sistema faz uso destes recursos para gravar e recuperar informações referentes ao projeto todo (Projeto SPR) em um só arquivo. Estes processos são idênticos aos descritos acima, porém, os arquivos são manipulados pelo S. PERERE de acordo com suas extensões, conforme mostra a Tabela 9.

Módulos	**Extensão**
Editor de Interfaces	*.spi
Editor de Tarefas	*.spt
Editor de Conhecimento Semântico	*.sps
Projeto SPR	**.spr*

Tabela 9. Extensões de arquivos do S. PERERE

A próxima seção apresenta a implementação do mecanismo de funcionamento do módulo Editor de Interface.

4.3. EDITOR DE INTERFACE

O Editor de Interface deve oferecer ao usuário do S. PERERE recursos para descrever graficamente qualquer interface computacional, no domínio da simulação de uma tarefa de interação homem-computador. Este módulo faz parte do conjunto de módulos responsáveis pela entrada de dados no simulador.

Na seqüência de atividades que levam o usuário do S. PERERE a modelar as ações de um operador humano interagindo com alguma interface computacional, a modelagem da interface do sistema é a primeira etapa a ser cumprida. Esta relação pode ser observada na Figura 1Figura 13, onde o usuário do S. PERERE identifica no sistema interativo, a interface sobre a qual as tarefas humanas serão representadas e modela-a através do Editor de Interface.

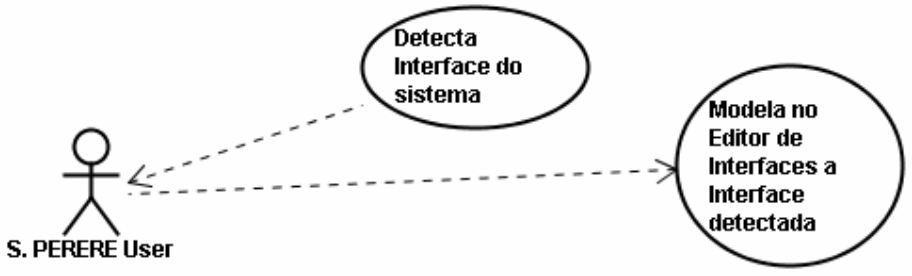

Figura 13. Processo de descrição de Interface

A detecção da interface de um sistema consiste em identificar objetos interativos na realização de alguma tarefa por parte do usuário. Esses objetos normalmente são displays (caixa de texto), botões, *leds* entre outros.

Este módulo deve oferecer recursos para descrever graficamente estes objetos. A Figura 14 apresenta o diagrama de classes que representa a infra-estrutura do módulo Editor de Interface, é uma abstração do package *moduleinterface* apresentado na Figura 6.

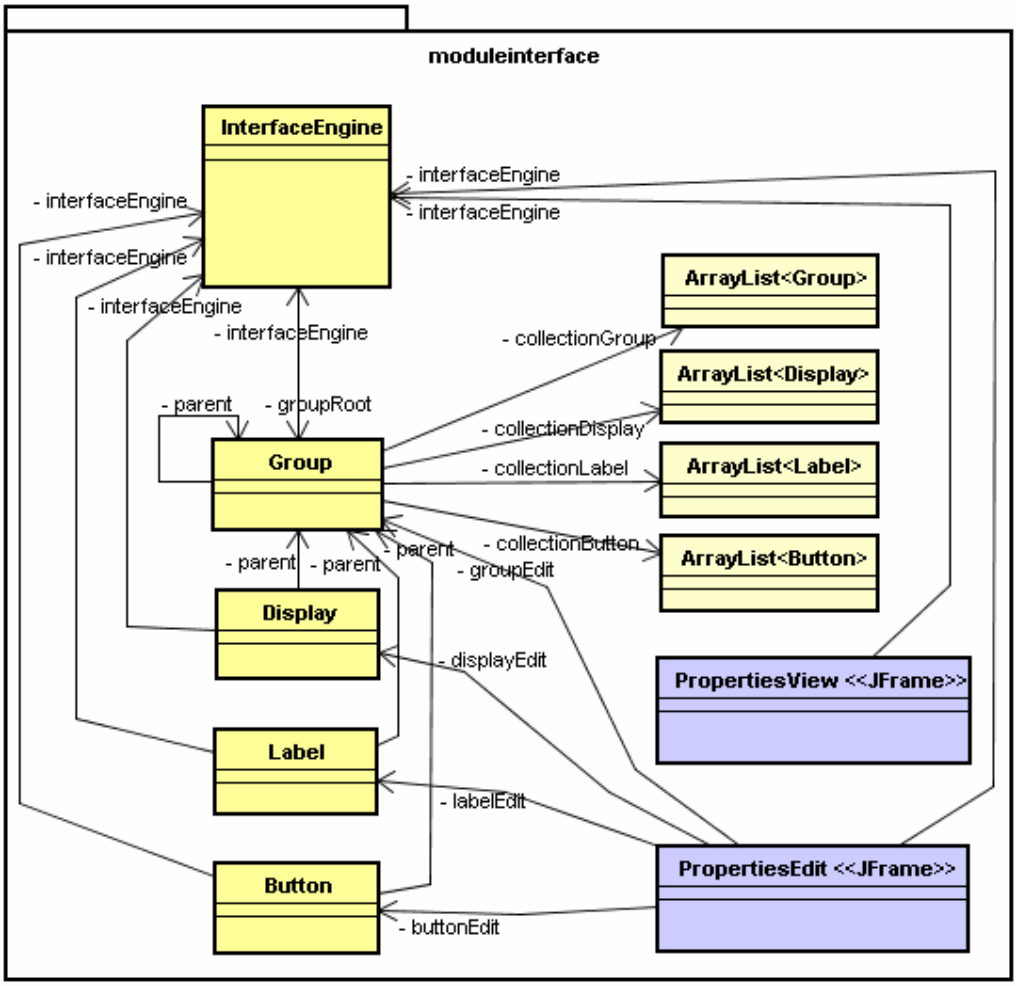

Figura 14. Diagrama de classes do Editor de Interface

Como já mencionado nas seções anteriores, a classe *InterfaceEngine* representa a mecânica do módulo, através desta as outras classes são instanciadas e seus respectivos atributos e métodos são acessados.

A descrição do modelo de interface neste módulo é realizada através de algumas ferramentas, as quais estão organizadas em dois grupos: *ferramentas de modelagem* e *ferramentas de manutenção*. As ferramentas de modelagem são: *Group, Display, Label* e *Button,* responsáveis pela criação e configuração inicial dos objetos. As ferramentas de configuração são: *Edit, Delete* e *View,* responsáveis pela edição, exclusão e visualização dos objetos de modelagem.

O texto foca apenas as ferramentas de modelagem, que doravante será denominada *ferramenta.*

A cada classe de ferramenta, é associado um objeto correspondente da biblioteca *Swing*, que a representará na interface gráfica do módulo. A Tabela 10 descreve esta afirmação demonstrando a classificação das ferramentas.

Ferramenta	Classificação	Nome da Classe	Objeto Swing
Group	modelagem	Group	JPanel
Display	modelagem	Display	JText
Label	modelagem	Label	JLabel
Button	modelagem	Button	JButton
Edit	**manutenção**	--	--
Delete	**manutenção**	--	--
View	**manutenção**	--	--

Tabela 10. Ferramentas do Editor de Interface

A classe *InterfaceEngine* possuí um atributo chamado *GroupRoot,* que é um objeto do tipo *Group,* este atributo é instanciado automaticamente quando o módulo é iniciado e representa a raiz dos objetos descritos pelo usuário.

O objeto *Group* é a base para a inserção de outros objetos, em um *Group* podem ser inseridos diversos Objetos, que por sua vez podem ser um *Group.* Isto prova que os objetos são inseridos no Editor de Interface de forma recursiva, conforme mostra a Figura 15. Recursividade na inserção de objetosFigura 15.

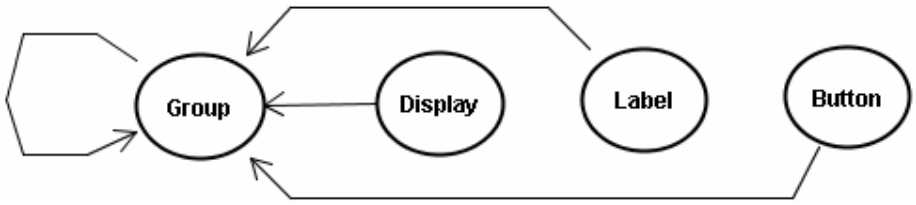

Figura 15. Recursividade na inserção de objetos

Os objetos inseridos em um *Group* são armazenados em coleções de objetos (*ArrayList),* de acordo com seu tipo, conforme mostra a Figura 14. As estruturas destas coleções estão implementadas dentro da classe *Group.*

Cada classe de objetos de interface possui uma instância da classe *InterfaceEngine,* possibilitando o acesso aos seus métodos e atributos e também à classe *Main.*

A classe *PropertiesEdit* é uma extensão da classe *JFrame* do *Swing,* ela foi projetada para que o usuário possa ter controle sobre as configurações dos objetos.

A interface gráfica da classe *PropertiesEdit* pode ser observada na Figura 16, que exemplifica a configuração das propriedades de um objeto *Group.*

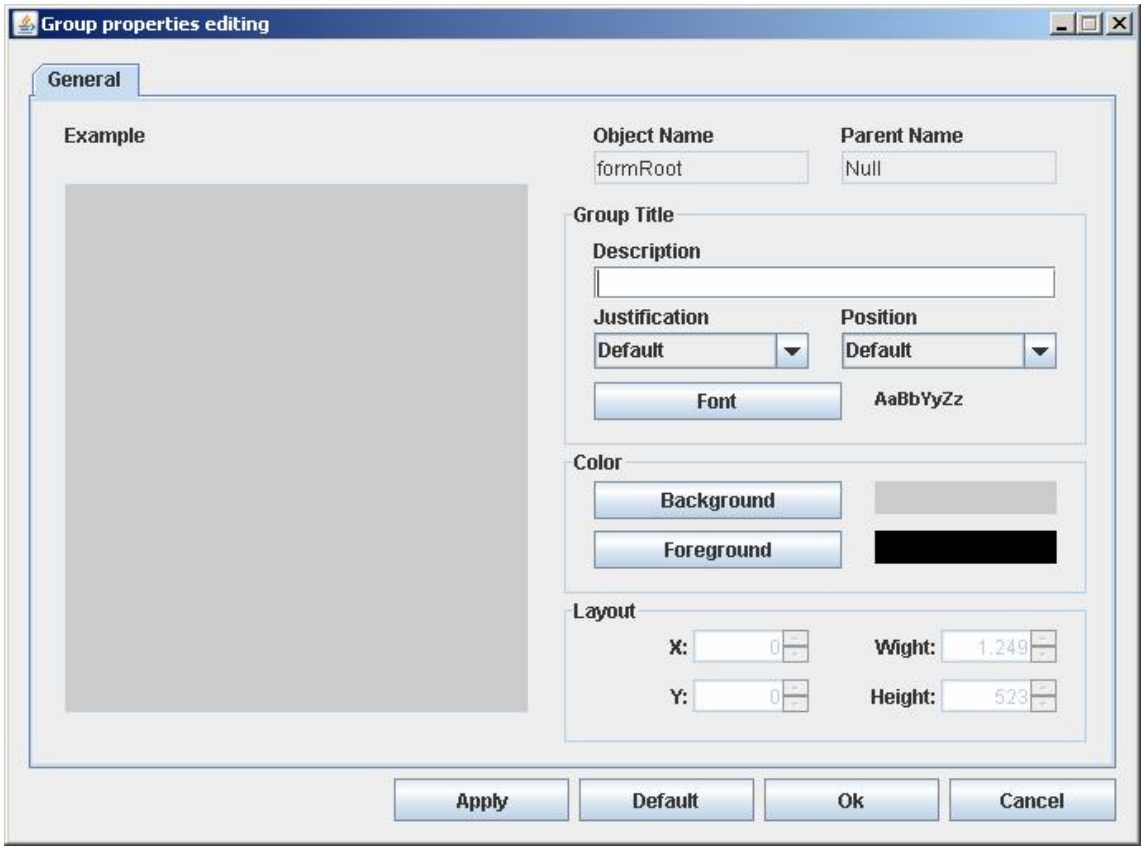

Figura 16. Configuração de objetos de interface

A propriedade **Object Name** mostra o nome do objeto que está sendo configurado e a propriedade **Parent Name** representa o nome do objeto pai, ou seja, do *Group* em que ele está inserido, esta propriedade terá o valor **null** apenas para o objeto *GroupRoot*. As duas propriedades não podem ser alteradas.

A propriedade **Description,** no caso do objeto *Group* representa o título na tela, no caso do *Display* representa o texto do respectivo *JText,* no caso do *Label* representa o seu conteúdo padrão e no caso do *Button,* representa a sua identificação na tela.

As propriedades **Justification** e **Position** referem-se ao posicionamento do título ou identificação dos objetos na tela.

A propriedade **Background** possibilita a alteração da cor do fundo de um objeto e a propriedade **Foreground** permite a alteração da cor do seu texto.

As propriedades do grupo **Layout** definem o tamanho do objeto e o seu posicionamento em relação ao seu Parent**.**

O objeto em alteração é clonado na propriedade **Example.** As alterações realizadas são imediatamente aplicadas ao objeto clone, antes de serem aplicadas ao original.

Para uma simples visualização das propriedades dos objetos, é utilizada a interface gráfica da classe *PropertiesView* conforme a Figura 17.

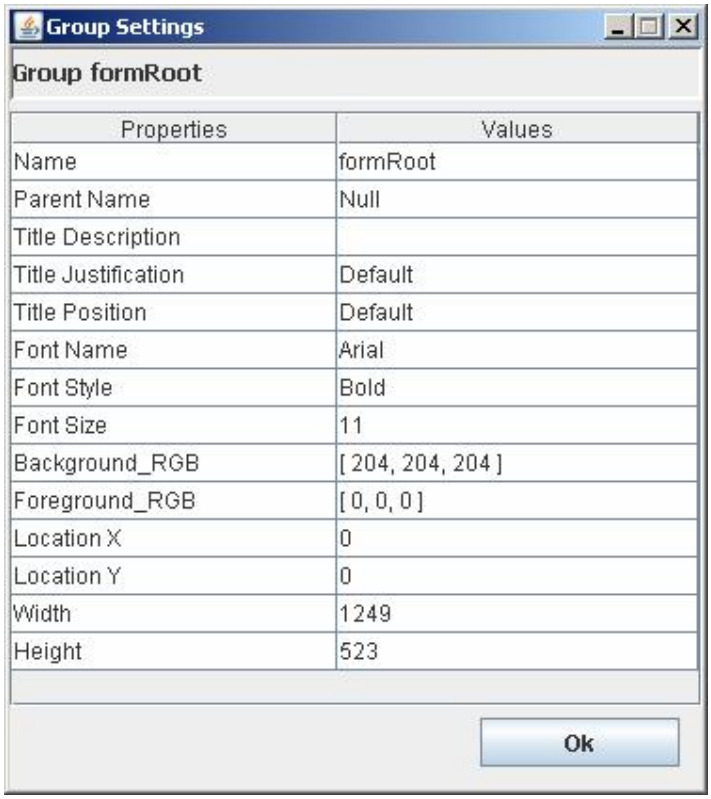

Figura 17. Visualização das propriedades do objeto

As propriedades da tela de visualização podem variar de acordo com o tipo de objeto, mas no geral, segue-se um mesmo padrão para todos os objetos.

A seguir, será apresentado resumidamente o tratamento de eventos dos objetos.

4.3.1. Eventos dos objetos de interface

Tratar eventos dos objetos em tempo de execução é um trabalho nada fácil, mas é uma das funcionalidades mais importantes na descrição de interface. Isto permite ao usuário do S. PERERE a criação e manipulação dos objetos na tela através do cursor do mouse. A

Evento	Descrição
mouseDraggedGroup	Arrastando o mouse pressionado.
mouseMovedGroup	Movimentando o cursor sobre o objeto.
mouseClickedGroup	Clique do mouse.
mouseEnteredGroup	Entrada do cursor no objeto.
mouseExitedGroup	Saída do cursor no objeto.
mousePressedGroup	Pressionamento do mouse.
mouseReleasedGroup	Soltar pressionamento do mouse.

Tabela 11 apresenta a lista dos eventos tratados na implementação e julgados como comuns para todos os tipos de objetos.

Evento	Descrição
mouseDraggedGroup	Arrastando o mouse pressionado.
mouseMovedGroup	Movimentando o cursor sobre o objeto.
mouseClickedGroup	Clique do mouse.
mouseEnteredGroup	Entrada do cursor no objeto.
mouseExitedGroup	Saída do cursor no objeto.
mousePressedGroup	Pressionamento do mouse.
mouseReleasedGroup	Soltar pressionamento do mouse.

Tabela 11. Lista de eventos dos objetos de interface

Estes eventos estão relacionados aos objetos *Swing* apresentados na Tabela 10, e correspondem as ações humanas sobre estes objetos, levando em consideração o cursor do mouse.

Eles são criados pelo S. PERERE em tempo de execução toda vez que um novo objeto é inserido no ambiente do *Editor de Interface* da classe *Main.* Os métodos e

atributos responsáveis por essa ação estão implementados nas classes referentes a cada objeto.

O tratamento de eventos implementado neste módulo fornece ao usuário a possibilidade para arrastar os objetos na tela, alterar seu tamanho e muitos outros recursos que certamente facilita a descrição da interface.

Outra característica importante deste módulo é o poder de validar o modelo descrito pelo usuário, que será apresentado adiante.

4.3.2. Compilação do modelo de interface

A compilação do modelo de interface é o ato de validar os objetos descritos pelo usuário, garantindo assim uma modelagem correta. A estrutura implementada para este recurso pode ser observada na Figura 18.

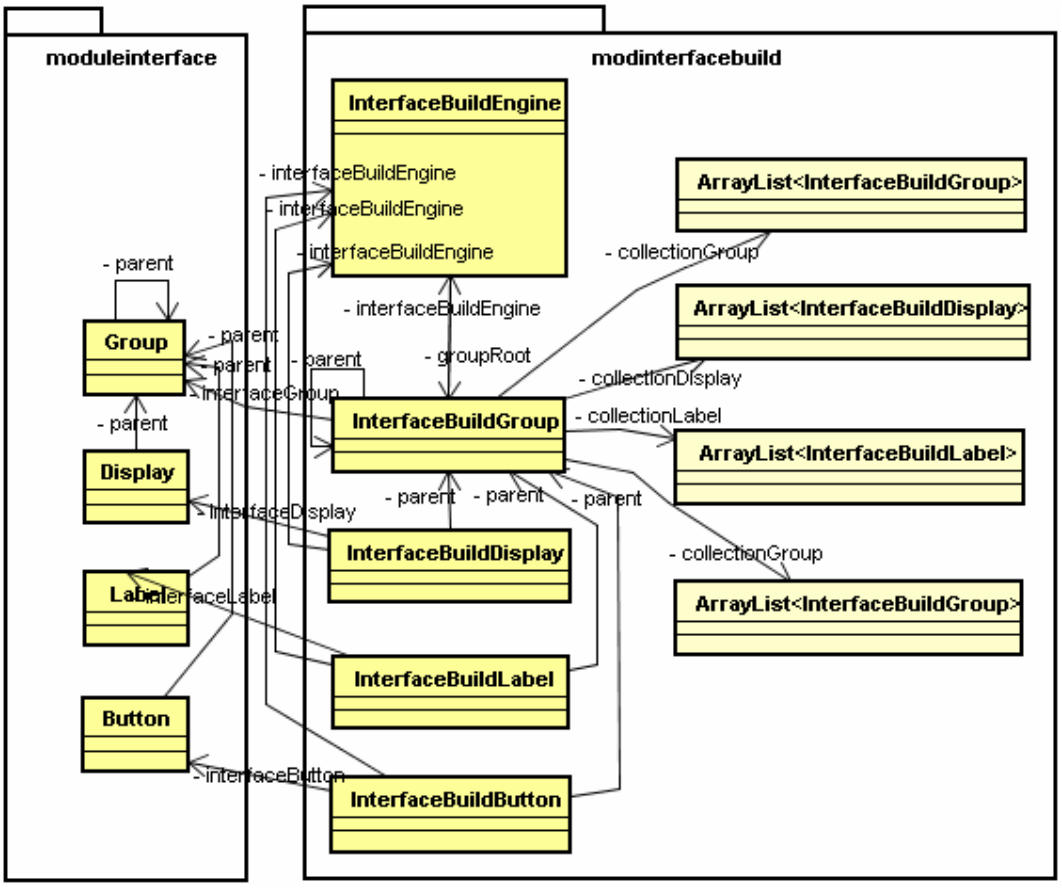

Figura 18. Diagrama de classes do compilador de interface

No tocante às relações entre as classes de objetos e a classe *Engine,* nota-se que o diagrama da Figura 18 possui algo em comum com o diagrama apresentado na Figura 14, bem como a sua estrutura recursiva ilustrada na Figura 15, e o esquema de armazenamento dos objetos em coleções (*ArrayList).* Nota-se que até a nomenclatura das classes manteve um padrão, exceto pelo acréscimo do *'Build',* provando que são classes diferentes, agrupadas em pacotes diferentes, porém, se relacionam diretamente com as classes dos objetos (*Group, Display, Label* e *Button).*

Esta relação implica o encapsulamento dos objetos de interface às classes correspondentes do módulo *Build,* facilitando o processo de compilação. Para entender melhor este processo, observa-se a Figura 19, que apresenta a estrutura dos objetos de interface criada para o processo de compilação.

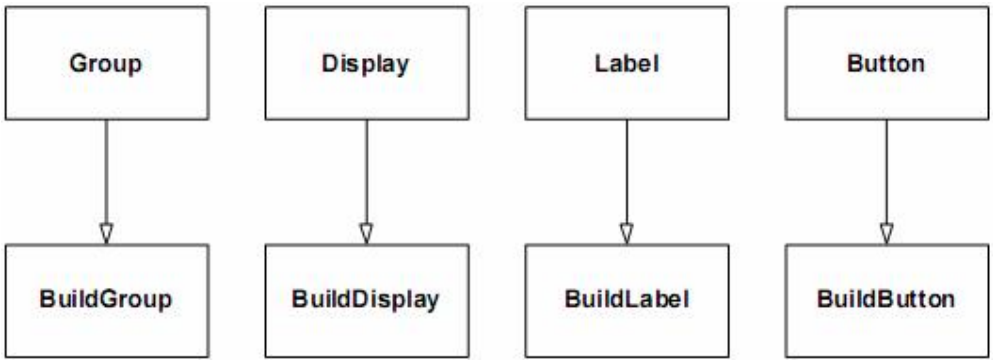

Figura 19. Estrutura dos objetos para a compilação de interface

As classes *Build* possuem uma instância do objeto correspondente. Por exemplo, *BuildGroup* possui uma instância da classe *Group* e assim sucessivamente.

Isto se justifica pelo fato de que, as classes *Build* precisam acessar as propriedades do objeto criado na interface, além de possuir seus próprios métodos e atributos necessários ao processo de compilação.

A mecânica do processo de compilação deste módulo está na classe *InterfaceBuildEngine,* que objetiva executar as seguintes ações:

- Verificar consistência dos objetos (*Displays* e *Buttons)* com os elementos da árvore de tarefas do módulo Editor de Tarefas (v. seção 4.4);

- Verificar a existência de objetos não utilizados na árvore de tarefas.

O produto do compilador de interface é a geração de possíveis erros e advertências (*warnings*) existentes entre os objetos de interface.

Na próxima seção será apresentado o módulo *Editor de Tarefas,* que justificará a proposta do *Editor de Interface*.

4.4. EDITOR DE TAREFAS

Esta seção apresenta a implementação do módulo *Editor de Tarefas* e as suas relações com os demais módulos de entrada de dados no S. PERERE. A construção deste módulo está baseada no modelo CTT (Concurrent Task Tree) proposto por Mori; Paterno; Santoro (2002). Este modelo possibilita a representação de tarefas hierarquicamente, em forma de árvore, com os nodos relacionados entre si por meio de operadores temporais.

Para provar o modelo CTT os autores propuseram o software CTTE (CTT Environment), que se tornou um dos ambientes mais difundidos para a modelagem de tarefas na atualidade. Este software também manipula seus modelos em arquivos XML, possibilitando a ligação com outros softwares.

A construção do módulo *Editor de Tarefas* se justifica por algumas particularidades inexistentes no CTTE, como por exemplo, a relação entre os nodos da árvore de tarefas e os comportamentos elementares de Berliner; Angell; Shearer (1964), e a comunicação entre os módulos: *Descritor de Interface* e *Descritor de Conhecimento Semântico.*

O *Editor de Tarefas* também elimina algumas funcionalidades encontradas no software CTTE que são desnecessárias no contexto deste trabalho, simplificando assim a usabilidade na descrição de tarefas.

O Processo de descrição de tarefas pelo usuário do S. PERERE pode ser observado na Figura 21.

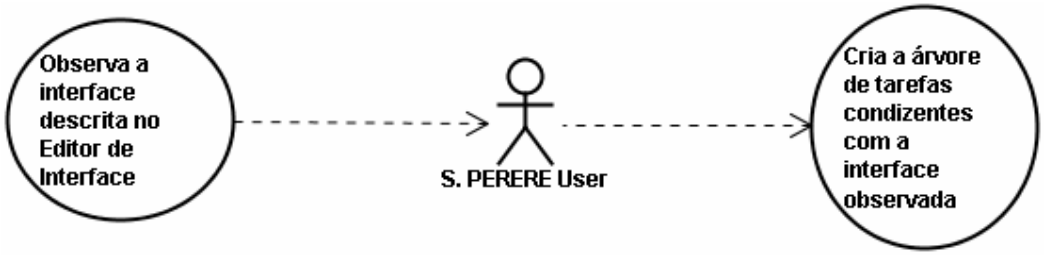

Figura 20. Processo de descrição de Tarefas

O usuário descreve a tarefa condizente com a interface descrita no modulo *Editor de Interface*, estabelecendo uma relação entre os objetos.

A estruturação das classes responsáveis pela construção deste módulo segue o mesmo padrão organizacional apresentado na seção anterior, esta estrutura pode ser observada no diagrama de classes da Figura 21.

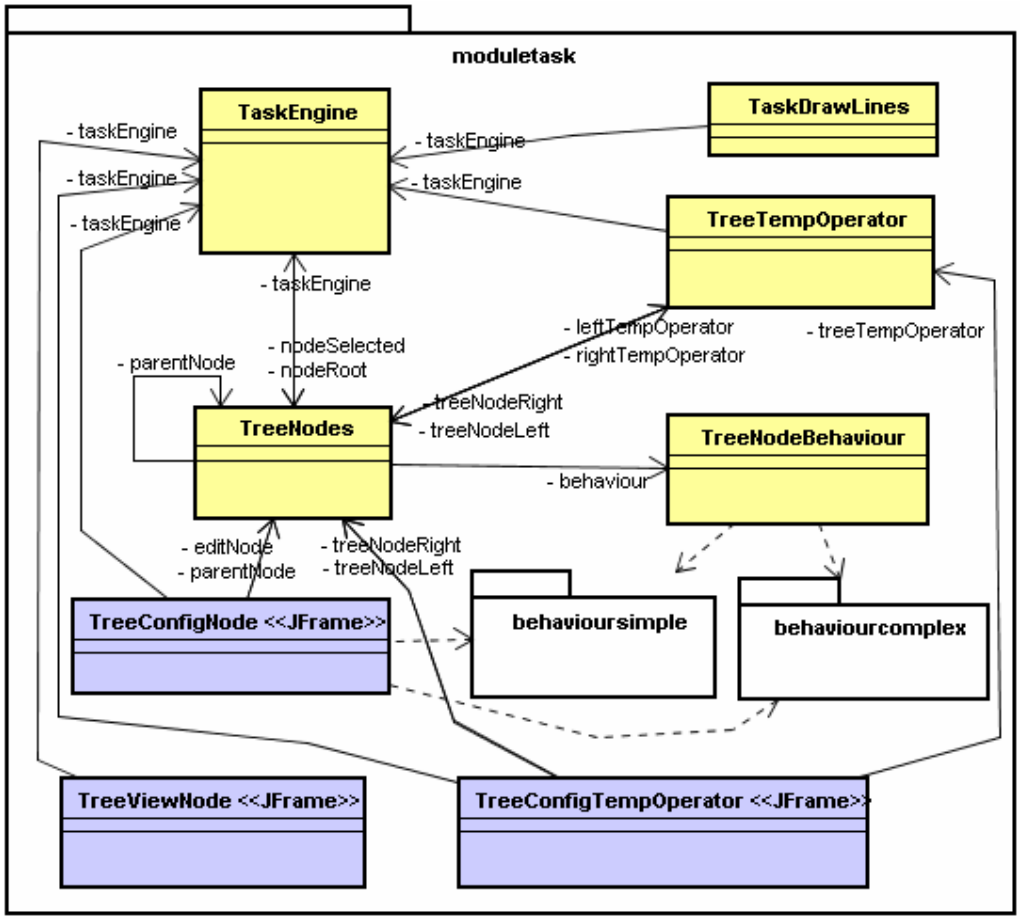

Figura 21. Diagrama de classes do Editor de Tarefas

Novamente a classe *TaskEngine* representa a mecânica do módulo, dando suporte ao seu funcionamento como um todo e estabelecendo relações com os demais módulos do sistema.

As ferramentas que compõe o processo de criação da árvore de tarefas, assim como as ferramentas do Editor de Interface, estão organizadas em dois grupos: *ferramentas de modelagem* e *ferramentas de manutenção*. As ferramentas de modelagem são: *Group Node*, *Task Node* e *Temporal Operator.*

A ferramenta *Group Node* é derivada da classe *TreeNodes,* responsável por representar os nodos que agrupam outros nodos. A ferramenta *Task Node* também é derivada da classe *TreeNodes,* porém, os nodos da árvore com características *Task Node* são folhas, ou seja, não possuem nenhum filho. A ferramenta *Temporal Operator* é responsável por criar operadores temporais que ligam dois nodos da mesma altura.

A Figura 22 exemplifica uma árvore de tarefas criada no *Editor de Tarefas* com os objetos originados das ferramentas descritas.

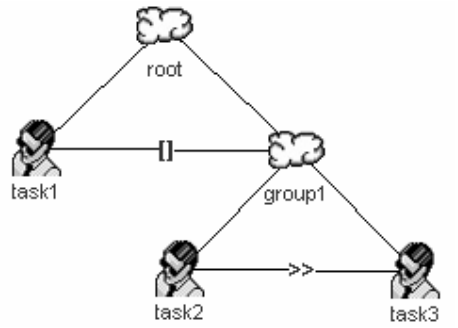

Figura 22. Objetos da árvore de tarefas

Os nodos *root* e *group1* são derivados da ferramenta *Group Node,* os nodos *task1, task2* e *task3* são derivados da ferramenta *Task Node* e os operadores temporais originados da ferramenta *Temporal Operator* são representados pelos símbolos [ ] e >>. Para cada Operador Temporal existe um símbolo correspondente.

As classes de interface que representam os objetos da árvore podem ser observados na

Edit	**manutenção**	--	--
Delete	**manutenção**	--	--
View	**manutenção**	--	--

Tabela 12. Ferramentas do Editor de Tarefas.

Ferramenta	Classificação	Nome da Classe	Objeto Swing
Group Node	modelagem	TreeNodes	JLabel
Task Node	modelagem	TreeNodes	JLabel
Temporal Operator	modelagem	TreeTempOperator	JLabel
Edit	**manutenção**	--	--
Delete	**manutenção**	--	--
View	**manutenção**	--	--

Tabela 12. Ferramentas do Editor de Tarefas

O *Editor de Tarefas* possui um ambiente de desenvolvimento acoplado à classe *Main,* responsável pela descrição de tarefas na forma de árvores. Os nodos desta árvore são representados pela classe *TreeNode, que* simboliza as tarefas desde a raiz da árvore, até as folhas.

A classe *TreeNode* possui um atributo denominado *parentNode,* este atributo é uma instância da própria classe *TreeNode,* representando o seu objeto pai.

O nodo raiz da árvore, representado na classe *TaskEngine* pelo atributo *nodeRoot,* mantém o seu atributo *parentNode* sempre com o valor *null,* auxiliando no processo de identificação.

4.4.1. Tarefas

O módulo Editor de Tarefas disponibiliza para o usuário uma interface para a descrição de tarefas em forma de árvore. Uma tarefa é um nodo da árvore, derivado da classe *TreeNodes,* podendo ser um grupo de tarefas (*Group Node*) ou uma tarefa (*Task Node*).

O processo de inserção de uma nova tarefa na árvore é feito através da interface gráfica da classe *TreeConfigNode,* ilustrada na Figura 23.

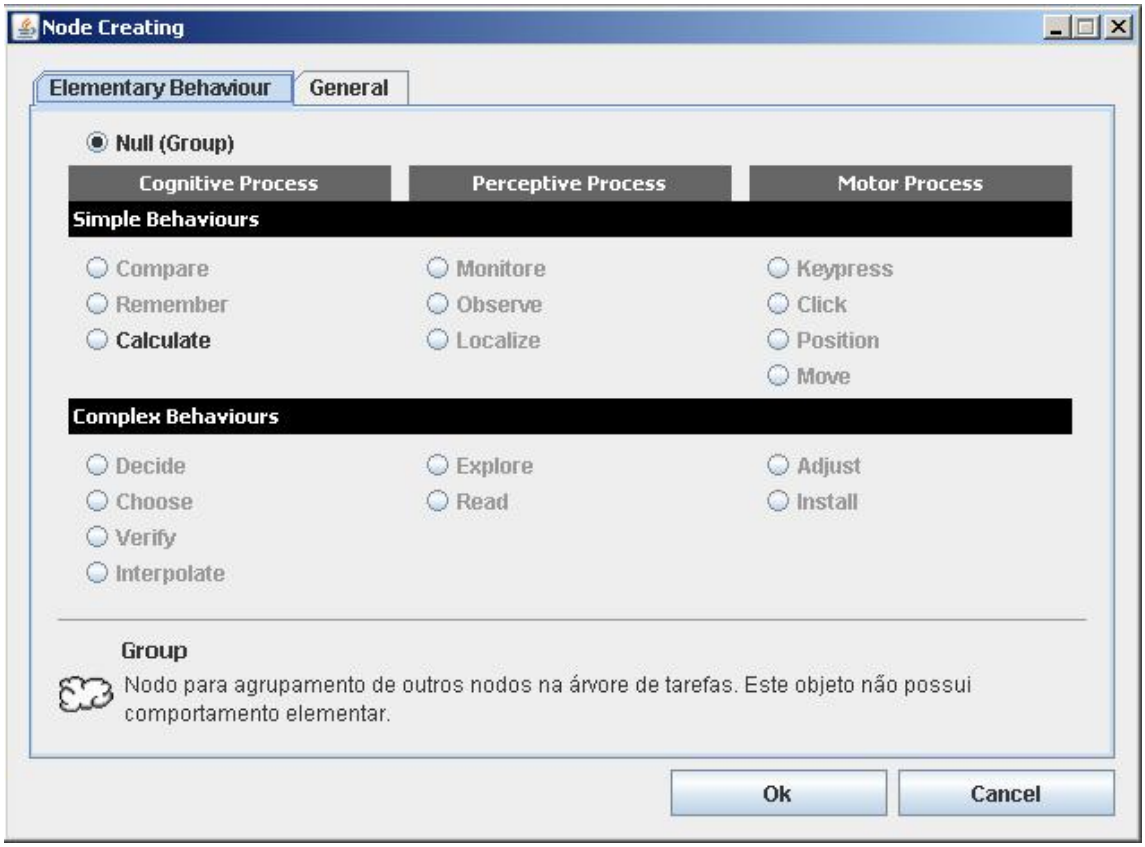

Figura 23. Inserção de tarefas – Elementary Behaviour

A interface gráfica da classe *TreeConfigNode* é dividida em duas abas: **Elementary Behaviour** e **General** (Figura 24).

A aba **Elementary Behaviour** permite a seleção do comportamento elementar da tarefa, de acordo com a taxonomia de Berliner; Angell; Shearer (1964) (v seção 2.2.1).

Conforme ilustra o diagrama de classes da Figura 21, o comportamento elementar de uma tarefa é definido através da classe *TreeNodeBehaviour,* instanciada na classe *TreeNodes*. A classe *TreeNodeBehaviour* por sua vez, de acordo com o comportamento escolhido na classe *TreeConfigNode*, cria uma instância da classe correspondente. Cada comportamento elementar possui uma classe correspondente, essas classes estão agrupadas no pacote *behaviourssimple* e *behaviourcomplex.*

O processo de inserção de uma tarefa ainda depende de algumas configurações agrupadas na aba **General** da interface gráfica da classe *TreeConfigNode,* conforme mostra a Figura 24.

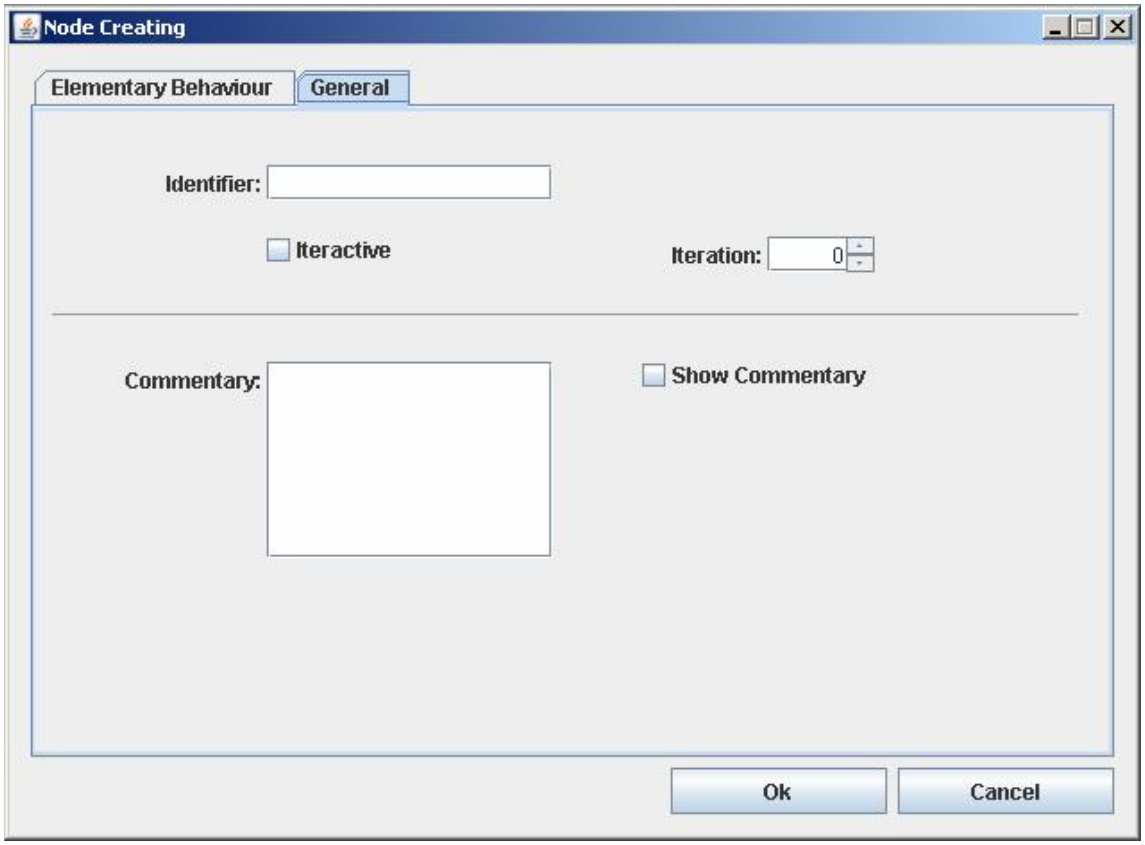

Figura 24. Inserção de tarefas – General

Na aba **General** contém recursos para configurar algumas propriedades importantes de um nodo de tarefa.

A propriedade **Identifier** representa a identificação do objeto, ou seja, o título do nodo da árvore.

A opção **Iteractive** permite selecionar a tarefa como iterativa ou não, e a propriedade **iteraction** possibilita a configuração da quantidade de iteração.

A iteratividade de uma tarefa na árvore, de acordo com o modelo CTT, consiste na repetição do seu processo de execução.

Na opção **commentary** o usuário pode escrever comentários sobre a tarefa, e a opção **show commentary** permite que o usuário decida se o comentário estará ou não disponível. Se o comentário estiver disponível, um ícone ira aparecer automaticamente no canto superior esquerdo do ícone principal da tarefa. O comentário é exibido toda vez que o cursor é posicionado sobre esse ícone. A Figura 25 demonstra esse exemplo.

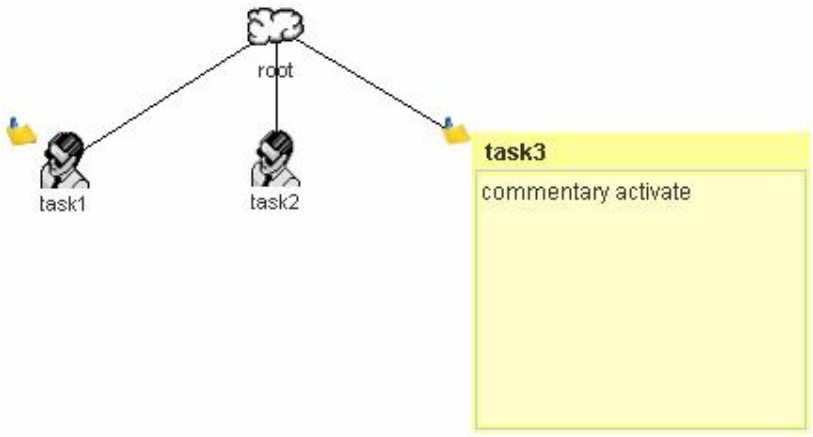

Figura 25. Comentário de uma tarefa

O exemplo ilustra uma árvore de tarefas descrita no Editor de Tarefas, que contém três nodos. A tarefa do nodo *task1* possuí um comentário disponível, assim como a tarefa do nodo *task3,* porém, o cursor esta posicionado na tarefa *task3.*

4.4.2. Configuração dos operadores temporais

Um objeto *TreeNode* também encapsula informações referentes aos relacionamentos entre os operadores temporais. Essas informações são tratadas pelos atributos *leftTempOperator* e *rightTempOperator,* que são instâncias da classe *TreeTempOperator.*

Estes relacionamentos, de acordo com o modelo CTT, são realizados apenas entre os nodos visinhos (esquerdo e direito), e devem se localizar na mesma altura, também devem ser filhos do mesmo pai (possuir o mesmo atributo *parentNode*).

As referências sobre possíveis relacionamentos de um nodo são armazenadas nos atributos *treeNodeLeft* e *treeNodeRight*, que guardam instâncias dos nodos à esquerda e à direita, se não houver relação o atributo assume o valor *null.*

A configuração dos operadores temporais de um nodo é realizada sempre da esquerda para direita, obedecendo aos parâmetros necessários para tal funcionalidade.

A Figura 26 ilustra a interface gráfica da classe *TreeConfigTempOperator,* que representa a configuração de um operador temporal entre as tarefas **task1** *e* **task2***.*

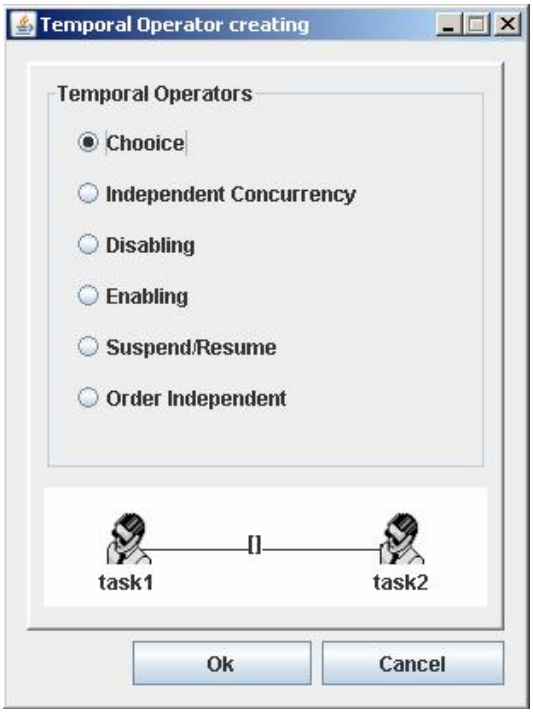

Figura 26. Configuração de operador temporal

De acordo com o modelo CTT foram incorporados no Editor de Tarefas do S. PERERE seis operadores temporais. São eles:

- *Chooice:* especificação de que duas tarefas estão habilitadas, porém quando uma é iniciada a outra é desabilitada;
- *Independent Concurrent:* especificação de que as tarefas podem ser executadas em qualquer ordem ou simultaneamente, podendo uma delas ser iniciada sem que a outra tenha sido finalizada;
- *Disabling:* especificação de que a primeira tarefa é completamente interrompida pela execução da segunda;
- *Enabling:* especificação de que a segunda tarefa (à direita do operador) só poderá ser iniciada após o término da primeira (à esquerda do operador);

- *Suspend/Resume:* especificação de que a primeira tarefa poderá ser interrompida pela segunda e reativada do ponto em que foi suspensa quando a segunda tarefa for finalizada*;*
- *Order Independent:* especificação de que duas tarefas devem ser executadas, porém quando uma delas é iniciada seu término deve ocorrer antes do início da outra;

A Figura 27 apresenta uma árvore de tarefas com a finalidade de ilustrar os símbolos de cada operador temporal, na mesma seqüência apresentada na interface gráfica da Figura 26.

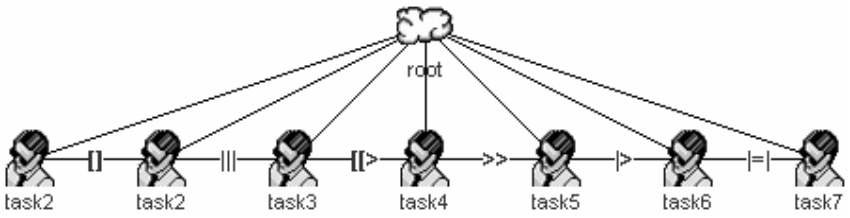

Figura 27. Símbolos dos operadores temporais

Uma vez entendido o significado dos operadores temporais e as suas relações com o modelo CTT, apresenta-se a seguir a aplicação desses conceitos na construção do Simulador de Tarefas.

4.4.3. Simulador de tarefas

O simulador de tarefas é um sub-módulo do módulo Editor de Tarefas, seus objetivos são apresentados como segue:

- Validar a árvore de tarefa;
- Gerar traços da execução das tarefas de acordo com os operadores temporais.

O diagrama de classes da Figura 28 demonstra a sua estrutura.

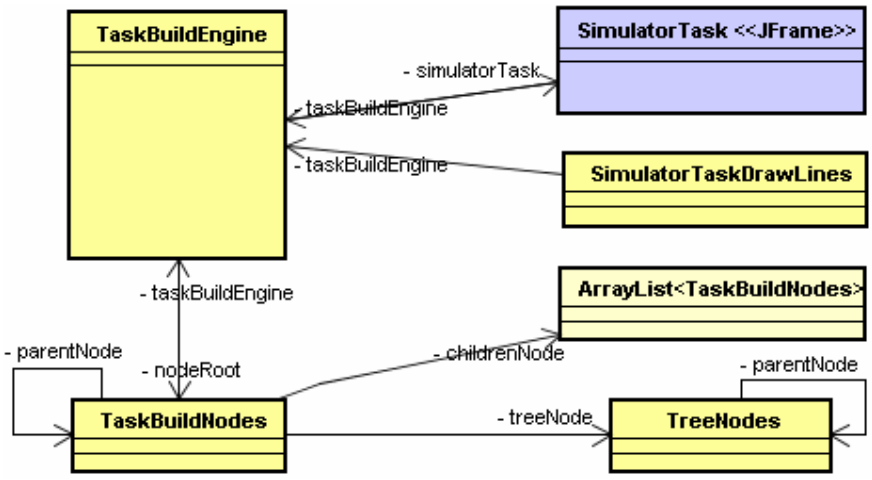

Figura 28. Diagrama de classes do Simulador de Tarefas

A classes *TaskBuildEngine* é a classe principal deste sub-módulo, nela esta encapsulado todos os atributos e métodos responsáveis pela compilação da árvore de tarefas e pela simulação da sua execução.

Os nodos da árvore são encapsulados dentro da classe *TaskBuildNodes,* que por sua vez coleciona os objetos através de um *ArrayList,* instanciado pelo atributo *childenNode.*

Quando o Simulador de Tarefas é iniciado, automaticamente o primeiro processo executado é a compilação da árvore. A execução das tarefas é habilitada somente se não houver nenhum erro na descrição das tarefas.

A interface gráfica do Simulador de Tarefas é representada pela classe *SimuladorTask* ilustrada na Figura 29*,* esta classe é uma extenção da classe *JFrame.*

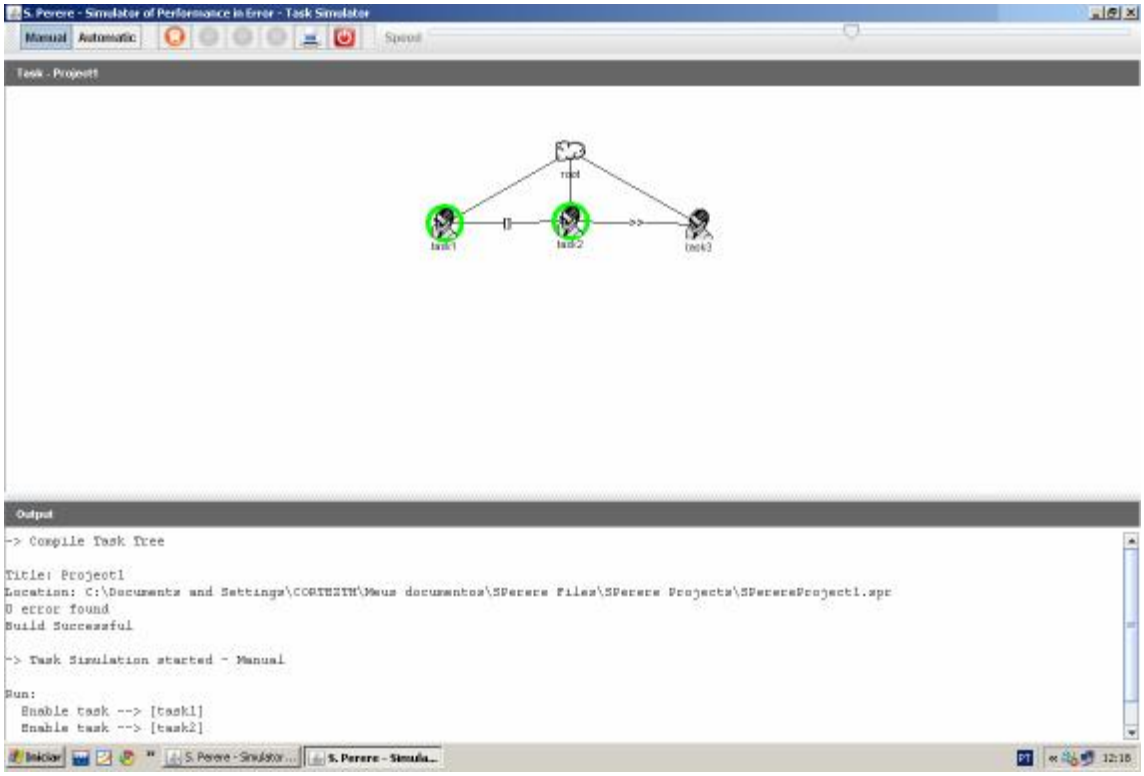

Figura 29. Simulador de Tarefas

Inicialmente as tarefas são compiladas, não havendo nenhum erro são habilitadas de acordo com seus operadores temporais.

As tarefas habilitadas são identificadas pelo círculo verde que contorna o ícone da tarefa. A execução de uma tarefa no modo manual consiste no clique do mouse sobre a tarefa habilitada e no modo automático existem mecanismos de tomada de decisão para executar as tarefas de acordo com seu nível de prioridade, definido pelo usuário na criação da tarefa.

A cada execução de uma tarefa, novamente o simulador aciona os mecanismos responsáveis pela habilitação de novas tarefas, de acordo com a tarefa executada. Este mecanismo esta baseado na teoria das Relações de Causalidade, descrita como segue.

4.4.3.1. Relações de Causalidade

Na filosofia, causalidade é conceituada como o conjunto de todas as relações de causa e efeito. Exemplo: Se uma bola está parada no chão e alguém lhe dá um chute, ela é atirada ao longe. Então dizemos que a causa do seu movimento foi a força muscular aplicada à bola através do chute.

Na física, a causalidade é a detecção da origem do fenômeno físico, na maioria das vezes pela aplicação da terceira das leis de Newton segundo a qual a toda ação, corresponde uma reação de igual intensidade e em sentido contrário.

No direito, a causalidade é a relação factual entre o agente (ou sujeito ativo, ou autor) e o resultado danoso (infração de direito, causação de prejuízo), também chamado de nexo causal.

Essa teoria possui aplicações em diversas áreas, mas no contexto deste trabalho pode-se utilizar as seguintes definições:

- Permite a estruturação de um comportamento complexo em termos de sub-comportamentos menos complexos e seus relacionamentos;
- Permite que uma ação e a condição para sua ocorrência sejam definidas em sub-comportamentos distintos.

Alguns mecanismos padrão são fundamentais nesta teoria:

- Pontos de entrada: pontos no comportamento no qual suas ações podem ser habilitadas por ações de outros comportamentos;
- Pontos de saída: condições de causalidade em um comportamento que podem ser utilizados para habilitar ações em outros comportamentos.

A teoria das Ralações de Causalidade possui um embasamento matemático bastante complexo, não é o objetivo deste trabalho descreve-los, porém, apresentar-se-á a sua aplicação na construção do Simulador de Tarefas para o S. PERERE.

A escolha desta teoria para a construção do Simulador de Tarefas se justifica pela existência nativa dos operadores temporais, idênticos aos utilizados pela construção da árvore de tarefas.

Para fins de demonstrações será utilizada a tarefa apresentada dentro da interface gráfica da Figura 29.

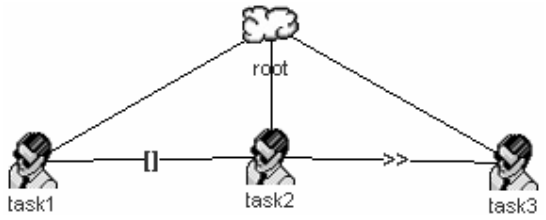

Figura 30. Tarefa descrita no Editor de Tarefas

A Figura 30 apresenta uma tarefa descrita no Editor de Tarefas, antes de ser construída no Simulador.

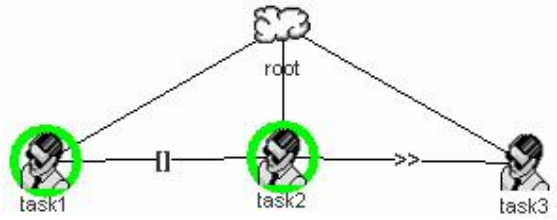

Figura 31. Tarefa iniciada no Simulador de Tarefas

A Figura 31 apresenta a mesma tarefa da Figura 30, porém, executada no Simulador de Tarefas, percebe-se que somente as folhas são habilitadas e executadas, em conformidade com o modelo CTT.

No exemplo, as tarefas *Task1* e *Task2* foram habilitadas porque o operador temporal existente entre elas representa uma escolha (choose), assim, e execução de qualquer uma das tarefas habilita a tarefa *task3* conforme mostra a Figura 32.

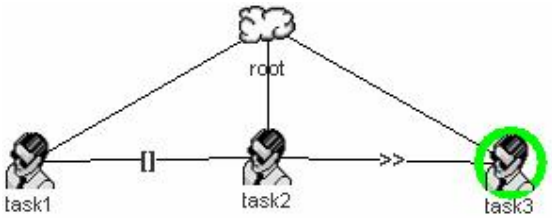

Figura 32. Primeira execução da terafa

A tarefa *Task3* foi habilitada logo após a execução das tarefas à esquerda, porque seu operador temporal à esquerda é uma habilitação (Enable).

Após a execução da tarefa *task3* a simulação é finalizada porque não existe nenhuma tarefa à direita.

A Figura 33 apresenta os traços da compilação e execução da tarefa no simulador, esses traços correspondem ao exemplo apresentado nas figuras 30, 31 e 32.

```
Output
-> Compile Task Tree

Title: Project1
Location: C:\Documents and Settings\CORTEZTH\Meus documentos\SPerere Files\S
0 error found
Build Successful

-> Task Simulation started - Manual

Run:
  Enable task --> [task1]
  Enable task --> [task2]
**Execute task --> [task1]
  Disable task --> [task1]
  Disable task --> [task2]
  Enable task --> [task3]
  Enable task --> [task3]
**Execute task --> [task3]
  Disable task --> [task3]

2 tasks executed
Terminated simulation
```

Figura 33. Traços da simulação de tarefas

Cada operador temporal exerce uma função no Simulador de Tarefas, de acordo com a teoria das Relações de Causalidade. Não há necessidade de exemplificar todos porque suas definições já foram apresentadas na seção 4.4.2.

À seguir será apresentada a compilação das tarefas, em conformidade com o padrão apresentado na seção 4.3.2.

4.4.4. Compilação de tarefas

A compilação de tarefas esta baseado no mecanismo idêntico ao utilizado na seção 4.3.2 para compilação de interface, podem ser apresentados conforme o diagrama de classe da Figura 34.

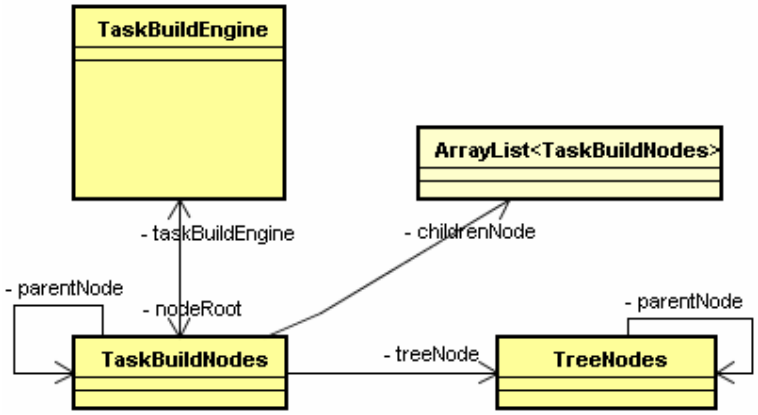

Figura 34. Diagrama de classes da compilação do módulo Editor de Tarefas

A estruturação das classes deste módulo é semelhante ao representado na Figura 28, porém sem fazer uso dos recursos de simulação da tarefa.

4.5. EDITOR DE CONHECIMENTO SEMÂNTICO

Nesta seção será apresentada a modelagem do módulo Editor de Conhecimento Semântico e as suas relações com os módulo Editor de Interface e Editor de Tarefas, apresentados nas seções anteriores.

Na seqüência da descrição da entrada de dados no simulador, este módulo é o último passo a ser seguido, nele descreve-se o conhecimento semântico da execução de uma determinada tarefa.

As classes construídas para permitir o funcionamento deste módulo podem ser observadas no diagrama de classes da Figura 35.

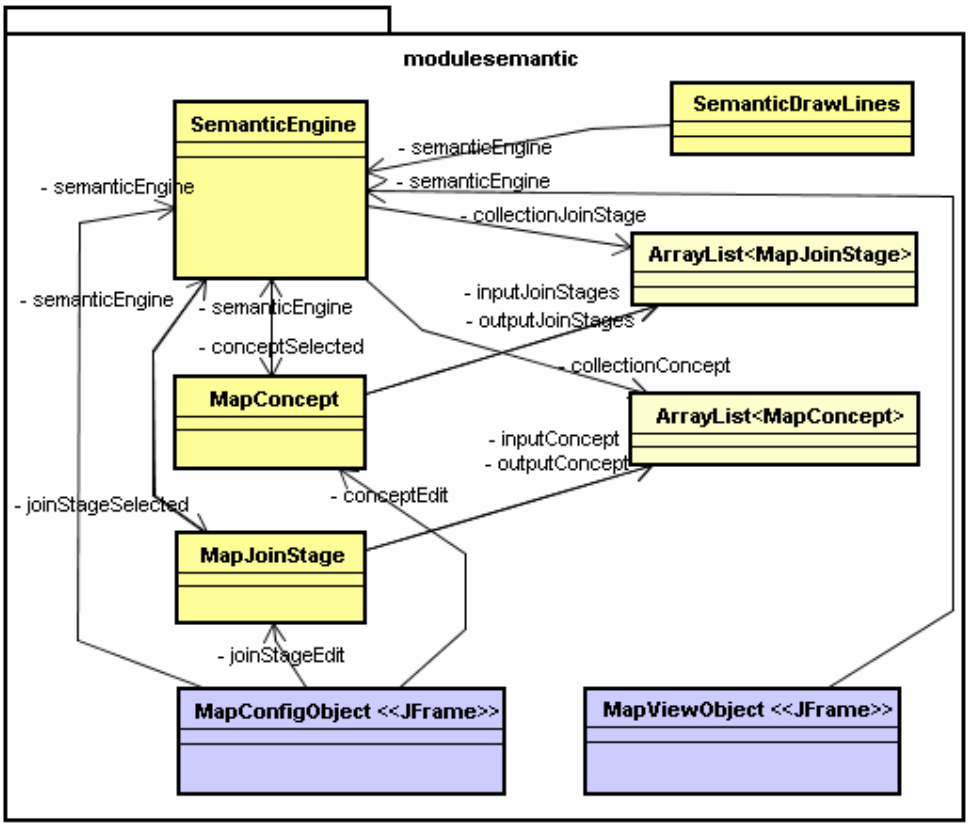

Figura 35. Diagrama de classes do Editor de Conhecimento Semântico

Em conformidade com os padrões apresentado nas seções anteriores, a classe *SemanticEngine* representa a mecânica do módulo.

Este módulo descreve o conhecimento semântico da tarefa baseado nas tradicionais formas de descrição de mapas conceituais. Esses descritores tradicionais, assim como o Editor de Conhecimento Semântico do S. PERERE possuem apenas duas ferramentas: Conceitos e Estágios de Ligação.

Os Conceitos são representados pela classe *MapConcept,* e os Estágios de Ligação são representados pela classe *MapJoinStage.*

Cada uma dessas classes de objetos possuem duas coleções (*ArrayList)* que guardam instâncias do outro objeto, esquerda e direita. A Tabela 13 apresenta esta afirmação.

Relação à esquerda	Classe de Objeto	Relação à direita
ArrayList<MapJointage>	MapConcept	ArrayList<MapJointage>
ArrayList<MapConcept>	MapJoinStage	ArrayList< MapConcept >

Tabela 13. Estrutura dos objetos do mapa conceitual

As linhas de ligação entre os objetos são construídas através da classe *SemanticDrawLines.*

Tanto o objeto *MapConcept* como o *MapJoinStage* possuem apenas um atributo funcional, o *Value,* que guarda seu respectivo valor no formato *String.*

A interface gráfica para configuração desses objetos no mapa é representada pela classe *MapConfigObject,* como mostra a Figura 36.

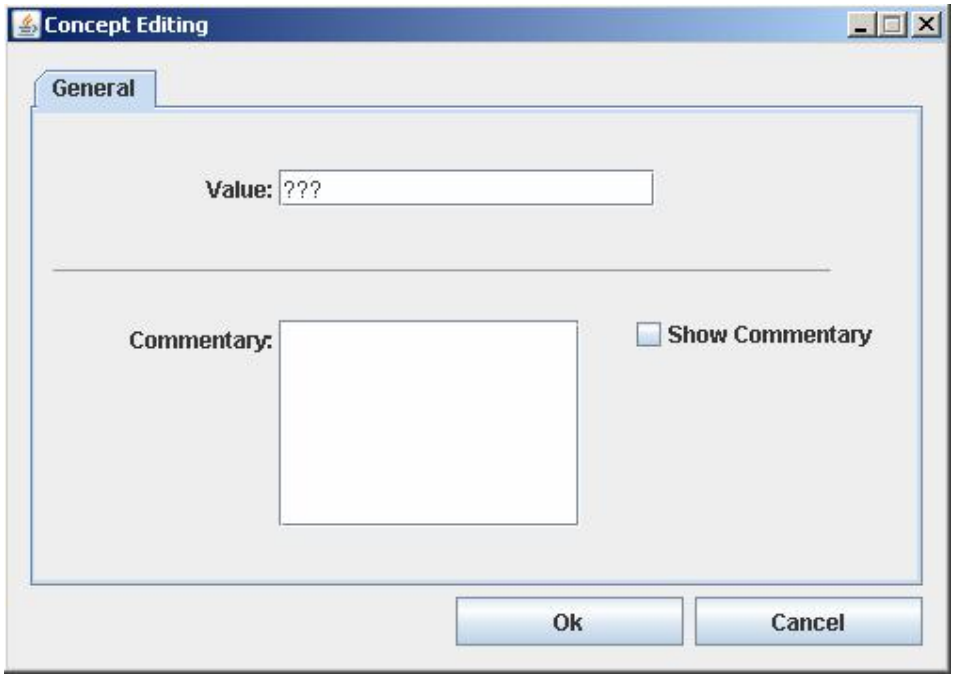

Figura 36. Configuração de objeto semântico

A visualização das propriedades dos objetos segue o mesmo padrão apresentado no módulo Editor de Interface e Editor de Tarefa. Sua interface gráfica é representada pela classe *MapViewObject,* conforme mostra a Figura 37.

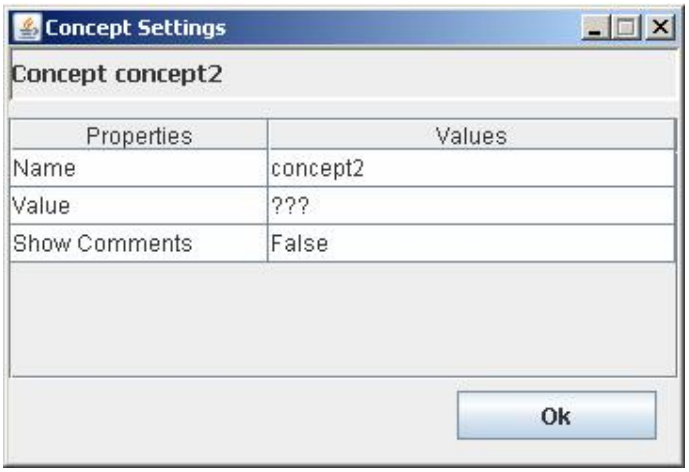

Figura 37. Visualização das propriedades do objeto semântico

Nota-se que a complexidade dos objetos deste módulo é bem menor que a complexidade dos objetos dos módulos: Editor de Interface e Editor de Tarefas, porém, a complexidade do módulo Editor de Conhecimento Semântico está na especificação dos comportamentos elementares das tarefas.

Sabe-se que uma tarefa descrita no módulo Editor de Tarefas está associada a um comportamento elementar de Berliner; Angell; Shearer (1964). Assim definiu-se que cada comportamento possui uma forma de descrever seu conhecimento semântico.

A definição dos comportamentos elementares é uma tarefa bastante difícil, já que suas definições estão intimamente ligadas à entrada da arquitetura cognitiva ACT-R. Assim, não faz parte do escopo deste trabalho definir todos os comportamentos, porém, foi definido apenas o comportamento Calcular.

Neste trabalho os exemplos que ilustram a descrição dos mapas conceituais deste módulo estão apoiados nas especificações do comportamento Calcular. Em trabalhos futuros serão criadas as especificações dos demais comportamentos.

4.5.1. Comportamento Calcular

Apresenta-se a especificação semântica do comportamento Calcular. Segundo Begosso (2005), este comportamento representa a realização de cálculos binários simples da aritmética (adição, subtração, divisão e multiplicação).

Sabendo-se que para cada comportamento elementar o elemento humano deve ser considerado pela arquitetura cognitiva ACT-R, assim, definiu-se que o comportamento calcular é representado pelo ACT-R como uma pessoa executando cálculo através das regras primitivas da aritmética.

No caso da adição o conhecimento semântico se apóia nas regras *vai um (go-one),* e no caso da subtração se apóia nas regras *empresta um (loans).*

A fórmula semântica definida para o comportamento Calcular é a seguinte: **Calculate (display1 operator display2).**

Os atributos *display1* e *display2* são identificados no momento em que a tarefa é criada no Editor de Tarefas, são relacionados aos displays de interface. Já o atributo *operator* é identificado no mapa conceitual descrito pelo módulo Editor de Conhecimento Semântico.

O mapa conceitual que representa o conhecimento do comportamento Calcular deve levar em consideração tanto o atributo *operator* como os valores dos respectivos displays de interface. Assim dada a tarefa representada pela Figura 38.

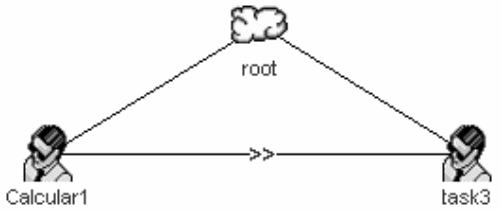

Figura 38. Exemplo de tarefa Calcular

Define-se o mapa conceitual no módulo Editor de Conhecimento Semântico, como ilustra a Figura 39.

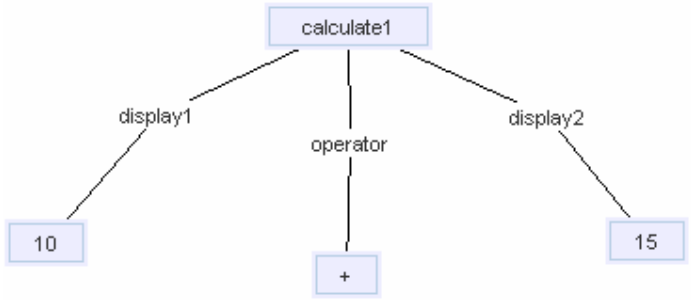

Figura 39. Mapa conceitual do comportamento Calcular

Os valores do mapa conceitual e transformado pelo S. PERERE em valores da memória declarativa da arquitetura ACT-R, esses valores serão apresentados nas próximas seções. A seguir será demonstrado a compilação deste módulo.

4.5.2. Compilação do conhecimento semântico

A compilação do conhecimento semântico consiste em validar os valores do mapa conceitual de acordo com cada comportamento elementar, conforme apresentou-se na seção 4.5.1. A Figura 40 prova esta relação.

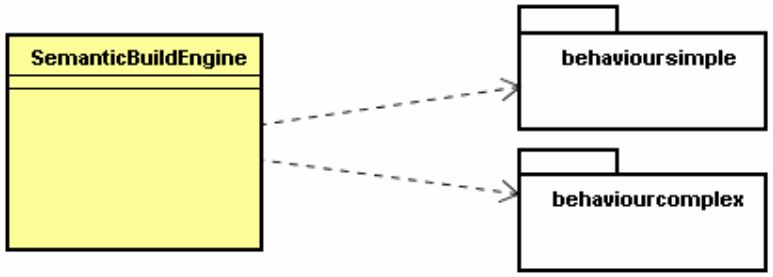

Figura 40. Diagrama de classes da compilação do módulo Descritor de Conhecimento Semântico

A classe *SemanticBuildEngine* controla os mecanismos de compilação, que acessam as classes dos comportamentos elementares agrupadas dos packages *behavioursimple* e *behavioursomplex.*

Apresentou-se os módulos que compõe a entrada de dados no S. PERERE, responsáveis por descrever Projetos SPR. Na próxima seção será apresentado o ambiente de desenvolvimento integrado do S. PERERE.

4.6. AMBIENTE DE DESENVOLVIMENTO INTEGRADO DO S. PERERE

Um dos principais resultados deste trabalho é a construção de um ambiente de desenvolvimento integrado do S. PERERE. Neste ambiente integram-se os módulos: Editor de Interface, Editor de Tarefas e Editor de Conhecimento Semântico em uma única interface gráfica representada pela classe *Main*.

Os mecanismos de cada módulo são processados separadamente, conforme apresentam as seções 4.3, 4.4 e 4.5, mas suas IDE's (Integrated Development Environment) são unificadas.

O que separa cada módulo são as abas em que eles estão inseridos, assim, ao selecioná-lo, automaticamente o sistema ajusta os menus e os botões correspondentes. A Interface gráfica da classe *Main* pode ser observada nas figuras 41, 42 e 43.

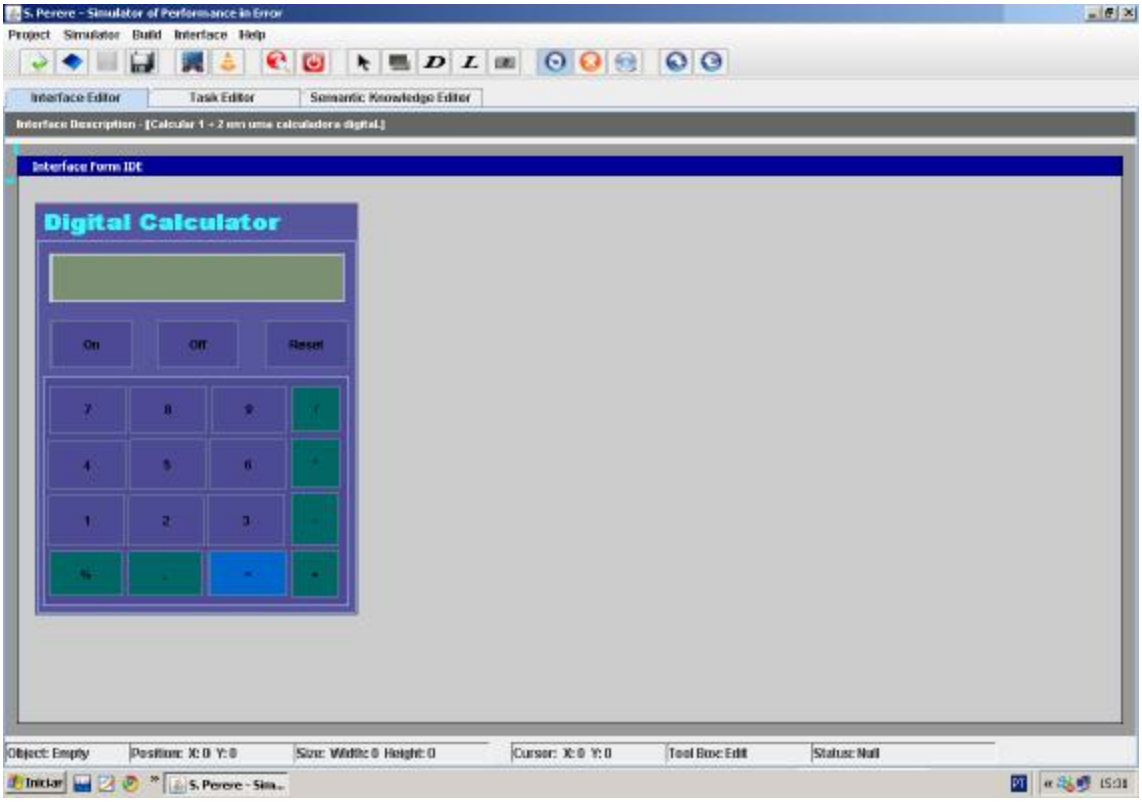

Figura 41. IDE do Editor de Interface

A Figura 41 ilustra a IDE do módulo Editor de Interface. Observa-se que uma calculadora digital foi definida pelo usuário neste módulo.

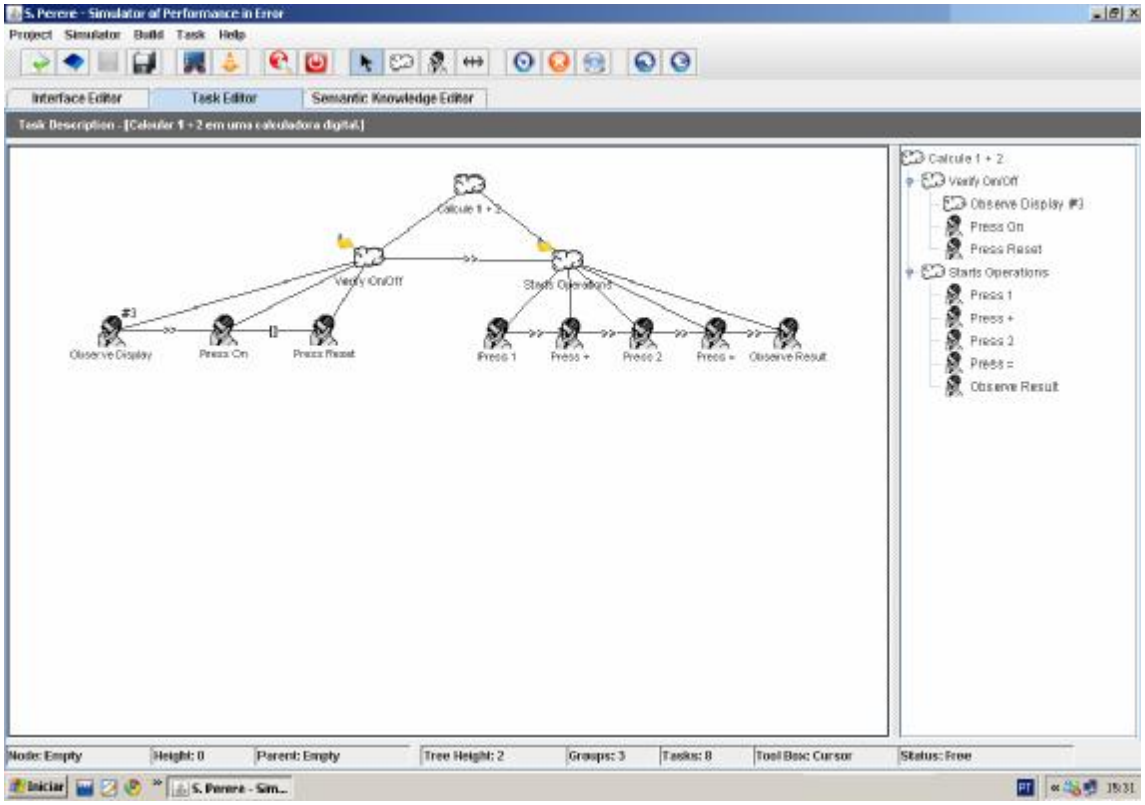

Figura 42. IDE do Editor de Tarefas

A Figura 42 ilustra a IDE do módulo Editor de Tarefas, demonstrando uma árvore de tarefa condizente com a interface apresentada na Figura 41. Esta IDE também possui recurso para visualizar a árvore verticalmente.

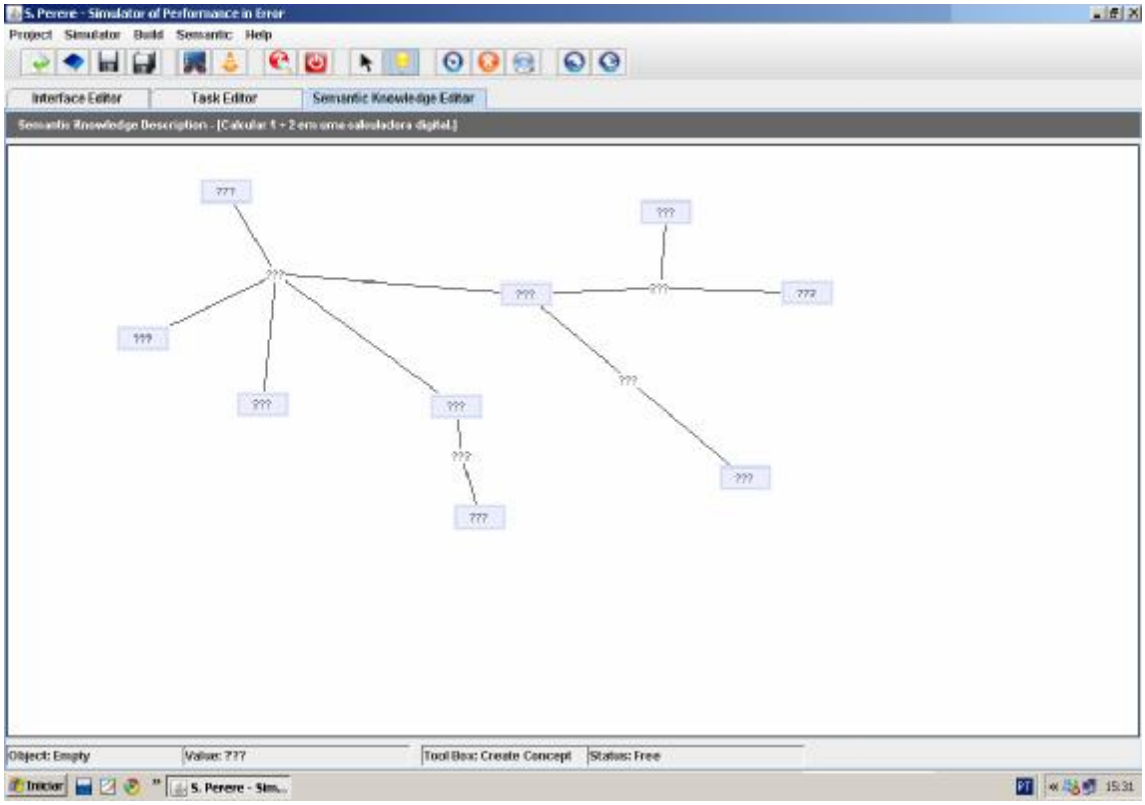

Figura 43. IDE do Editor de Conhecimento Semântico

A Figura 43 ilustra a IDE do módulo Editor de Conhecimento Semântico. Este exemplo não esta condizente com os exemplos apresentados nas figuras 41 e 42 porque nem todos os comportamentos elementares estão definidos.

A próxima seção apresenta a compilação de Projetos SPR.

4.6.1. Compilação de Projetos SPR

A compilação do Projeto SPR consiste no acionamento automático dos mecanismos de compilação de cada módulo: Editor de Interface, Editor de Tarefas e Editor de Conhecimento Semântico.

O compilador de cada módulo é acionado e os traços são reunidos em uma única interface gráfica.

A próxima seção apresenta a geração de código ACT-R.

4.7. GERAÇÃO DE CÓDIGO ACT-R

O S. PERERE possuí mecanismos para gerar traços de entrada da arquitetura cognitiva ACT-R, de acordo com a sintaxe estabelecida por Dan Bothell (2007).

Esses traços correspondem ao conhecimento humano sobre a execução de alguma tarefa. Esses conhecimentos são gerados pelo S. PERERE através das informações descritas pelo usuário nos módulos: Editor de Interface, Editor de Tarefas e Editor de Conhecimento Semântico.

A ordem das tarefas é identificada pelo S. PERERE través de uma execução da árvore de tarefas no Simulador de Tarefas em modo automático. Esta execução produz uma seqüência de tarefas, sobre a qual os traços do ACT-R são gerados.

Esses traços estão baseados na sintaxe da linguagem Lisp, sobre a qual a arquitetura ACT-R foi construída.

O S. PERERE, conforme mencionado nos capítulos anteriores, gera traços de execução das tarefas de forma correta e de forma perturbada.

O capítulo seguinte demonstra de forma mais clara a geração do código ACT-R através do experimento utilizado neste trabalho.

5. EXPERIMENTO REALIZADO

Neste capítulo será apresentado o experimento realizado na nova versão do S. PERERE, provando assim os benefícios dos recursos propostos neste trabalho.

Este experimento trata-se do exemplo de uma tarefa de interação homem-computador.

Levando em consideração que somente o comportamento elementar Calcular está definido, o exemplo trata de uma tarefa simples que simula o ser humano, a partir de uma interface computacional, realizando duas tarefas matemáticas: Soma e Subtração.

A Figura 44 apresenta a interface do experimento, construída no Editor de Interface.

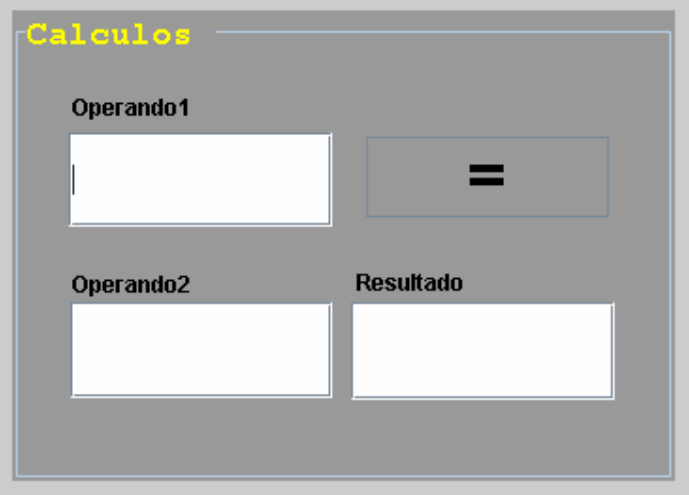

Figura 44. Interface do experimento

Nesta interface contém um objeto *Group*, três D*isplays*, três *Labels* e um *Button.* Através desta descreveu-se uma árvore tarefas, conforme mostra a Figura 45.

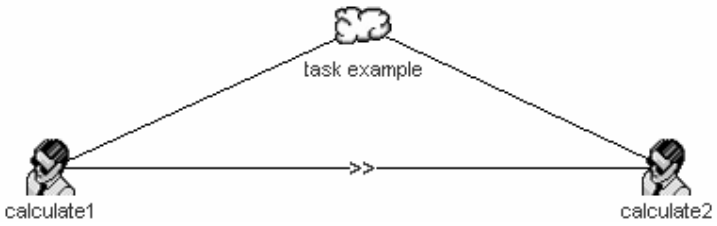

Figura 45. Tarefas do experimento

A árvore de tarefas apresentada na Figura 45 foi descrita no módulo Editor de Tarefas, em conformidade com a interface apresentada na Figura 44.

Na árvore contem duas tarefas, ambas pertencentes ao comportamento Calcular, a primeira intitulada como *calculate1* e a segunda intitulada como *calculate2.* O operador temporal (Enable) existente entre as tarefas assegura que a tarefa de adição seja executada antes da tarefa de subtração.

De acordo com a fórmula definida para o comportamento calcular (**Calculate (display1 operator display2)**), as duas tarefas possuem o mesmo valor até o momento: **Calculate (Operando1 operator Operando2).** Estes valores são alterados à medida que o Projeto SPR é construído pelo usuário.

O próximo passo é a definição do mapa conceitual referente a cada tarefa. Estes mapas podem ser observados na Figura 46.

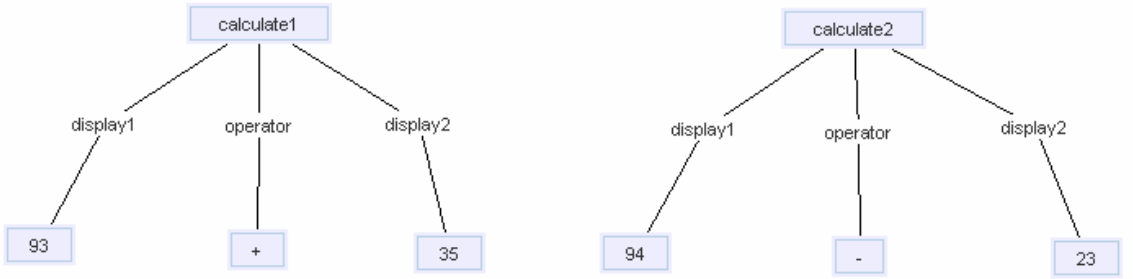

Figura 46. Conhecimento Semântico do experimento

A Figura 46 apresenta os dois mapas conceituais que representam os conhecimentos semânticos das tarefas *calculate1* e *calculate2.*

Definiu-se o Projeto SPR deste experimento através dos três módulos de entrada de dados no S. PERERE. Os valores relacionados às formulas de cada tarefa está representado na Tabela 14.

Tarefa	Comportamento	Fórmula
clculate1	Calcular	Calculate (93 + 53)
clculate2	Calcular	Calculate (94 – 23)

Tabela 14. Valores compilados das tarefas do experimento

Os valores relacionados às formulas das tarefas correspondem ao resultado da compilação do Projeto SPR.

Uma vez descrito o Projeto SPR referente ao experimento, o usuário acessa o recurso *Generate Producer for ACT-R* encontrado no menu *Build* da interface gráfica da classe *Main.*

Esta opção possibilita o acesso ao módulo pré-processador, que tem o papel de gerar a sintaxe de entrada da arquitetura ACT-R de acordo com o Projeto SPR. A sintaxe ACT-R só é gerada se o Projeto SPR estiver compilado e não possuir nenhum erro.

A Figura 47 ilustra a interface gráfica para geração de produções ACT-R.

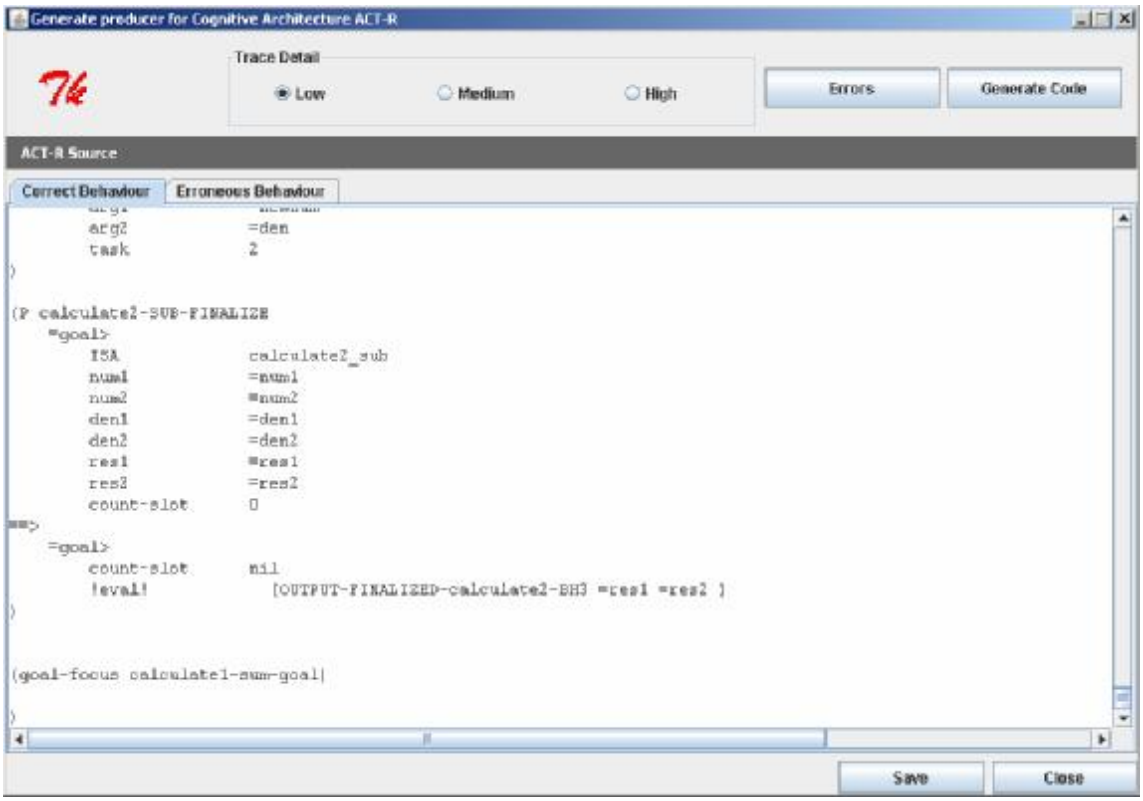

Figura 47. Geração de produções ACT-R

A opção **Trace Detail** configura o nível de detalhe dos traços de saída da arquitetura cognitiva ACT-R. A opção **Low** representa um baixo nível de detalhamento dos traços, a opção **Médium** especifica um nível de detalhamento médio e a opção **High** especifica um alto nível de detalhamento dos traços.

Para a geração do código perturbado o usuário deve acessar a opção **Errors** que por sua vez abre a interface gráfica conforme ilustra a Figura 48.

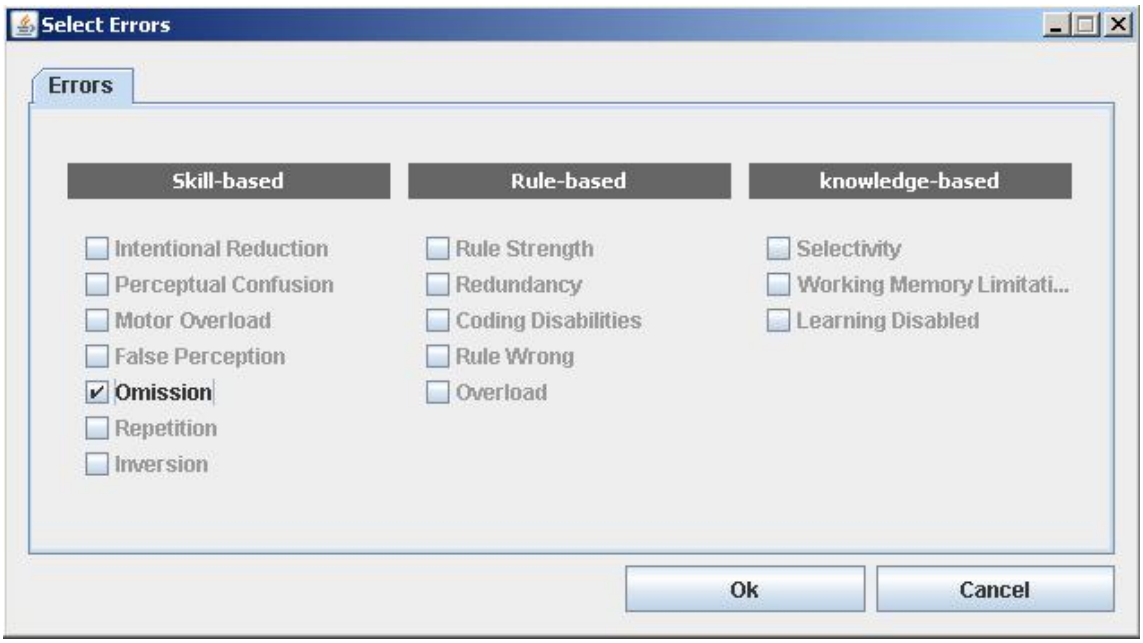

Figura 48. Seleção de erros

A seleção de erros permite que o usuário escolha qual erro o módulo Disparador usará para perturbar a tarefa. Eles estão organizados na tela de acordo com o modelo de Rasmussem (1983).

Neste trabalho foi implementado somente o erro de Omissão, que consiste em omitir um passo da tarefa.

No tocante ao experimento apresentado, o usuário seleciona o erro de omissão, que por sua vez gera o traço de entrada do ACT-R de forma perturbada.

Neste momento o código ACT-R na forma correta e na forma perturbada foram gerados e representados na interface gráfica semelhante a Figura 47. Esses códigos ACT-R estão representados nos Apêndices 1 e 2.

Para que a arquitetura ACT-R execute esses códigos, o usuário deve acionar o comando **Save** representado na interface da Figura 47, que permite a gravação dos arquivos na extensão *.lisp. É necessário abrir a interface da arquitetura ACT-R e carregar estes arquivos, um de cada vez. Sua saída é representada pelos traços de compilação, conforme apresenta os Apêndices 3 e 4.

6. CONCLUSÕES

Demonstrou-se a importância de atentar-se para os problemas relacionados ao erro humano na interação do homem com sistemas computacionais críticos apresentando o S. PERERE, um simulador da ação humana que leva em consideração o erro humano na interação homem-computador.

Apresentou-se o S. PERERE desde a sua concepção por Begosso (2005), assim como os trabalhos que contribuíram para sua evolução, entre eles destacam-se os trabalhos de: Filgueiras; Vitti (2006), Cortez (2007) e Cortez; Begosso (2008).

Foi proposto neste trabalho uma modelagem e implementação orientada a objetos para o S. PERERE, enfatizando os módulos responsáveis pela entrada de dados no simulador. Também foi implementado o comportamento elementar Calcular e o mecanismo de erro Omitir.

Uma das grandes conquistas desde trabalho foi a definição de uma interface de desenvolvimento integrado para o S. PERERE, facilitando assim a construção de futuros trabalhos e dando ao simulador características de qualidade no processo de desenvolvimento.

A construção do S. PERERE pode ser dividida em cinco etapas: A primeira etapa se caracteriza pela construção da sua base teórica por Begosso (2005). A segunda etapa é a proposta deste trabalho, dando ao S. PERERE uma estrutura totalmente orientada a objetos. A terceira etapa é a definição de todos os comportamentos elementares de Berliner; Angell; Shearer (1964) e de todos os mecanismos de erro humano de Reason (1990), de acordo com a arquitetura cognitiva ACT-R. A quarta etapa consiste na construção de mecanismos capaz de fazer com que a arquitetura ACT-R funcione de forma oculta, carregando por sua vez o modelo gerado pelo S. PERERE. Na quinta etapa deve ser construídos mecanismos baseados na linguagem lisp, dentro da arquitetura ACT-R, capaz de salvar seus resultados em um arquivo para posterior leitura e processamento pelo S. PERERE.

A primeira e a segunda etapa já estão concluídas, assim, como trabalhos futuros, devem ser desenvolvidas as outras três etapas, e por fim, sugere-se a realização de um caso de uso, provando as suas qualidades e os benefícios propostos.

APÊNDICE 1 – CODIGO ACT-R DO EXPERIMENTO – PRODUÇÃO CORRETA

```
(defvar *num1_BH3* nil)
(defvar *den1_BH3* nil)
(defvar *num2_BH3* nil)
(defvar *den2_BH3* nil)
(defvar *num3_BH3* nil)
(defvar *den3_BH3* nil)
(defvar *num4_BH3* nil)
(defvar *den4_BH3* nil)
(defvar *num5_BH3* nil)
(defvar *den5_BH3* nil)
(defvar *num6_BH3* nil)
(defvar *den6_BH3* nil)
(defvar *num7_BH3* nil)
(defvar *den7_BH3* nil)
(defvar *num8_BH3* nil)
(defvar *den8_BH3* nil)
(defvar *num9_BH3* nil)
(defvar *den9_BH3* nil)
(defvar *num10_BH3* nil)
(defvar *den10_BH3* nil)

(defun BEGIN ()
    (reset)
    (pm-run 10))

(defun OUTPUT-SLOT-STARTED-BH3 (count arg1 arg2 symbol)
    (case symbol
        (1 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S + ~S) STARTED" count arg1 arg2))
        (2 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S - ~S) STARTED" count arg1 arg2))
        (3 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S / ~S) STARTED" count arg1 arg2))
        (4 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S * ~S) STARTED" count arg1 arg2))))

(defun OUTPUT-SLOT-INCREMENT-BH3 (count result)
    (format t "SPR.-->  INCLEMENT COUNT [ ~S ]  --  NEW RESULT [ ~S ]" count result))

(defun OUTPUT-SLOT-COUNT-BH3 (count)
    (format t "SPR.-->  NEW COUNT [ ~S ]" count))

(defun OUTPUT-GO-ONE-DETECTED-BH3 ()
    (format t "SPR.-->  GO-ONE DETECTED"))

(defun OUTPUT-ASSIGN-GO-ONE-BH3 ()
    (format t "SPR.-->  ASSIGN GO-ONE"))

(defun OUTPUT-LOANS-DETECTED-BH3 ()
    (format t "SPR.-->  LOANS DETECTED"))

(defun OUTPUT-ASSIGN-LOANS-BH3 ()
    (format t "SPR.-->  ASSIGN LOANS"))

(defun OUTPUT-SLOT-FINALIZED-BH3 (count result)
    (format t "SPR.-->  CALCULE SLOT [ ~S ] FINALIZED  --  NEW RESULT [ ~S ]" count result))


(defun OUTPUT-STARTED-calculate1-BH3 (num1 num2 den1 den2 )
    (setf *num1_BH3* num1)
    (setf *num2_BH3* num2)
    (setf *den1_BH3* den1)
    (setf *den2_BH3* den2)

    (format t "SPR.-->  NEW TASK ELEMENT SELECTED~%")
    (format t "SPR.-->  PROCESS:               COGNITIVE~%")
    (format t "SPR.-->  TASK NAME:             calculate1~%")
    (format t "SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE~%")
    (format t "SPR.-->  FORMULE:               calculate ( display1 operator display2 )~%")
    (format t "SPR.-->  calculate (~S~S + ~S~S) STARTED" num1 num2 den1 den2 ))

(defun OUTPUT-FINALIZED-calculate1-BH3 (res0 res1 res2 )
```

```
    (if (= res0 0)
        (format t "SPR.-->  calculate (~S~S + ~S~S) FINALIZED -- RESULT [ ~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res1 res2 )
        (format t "SPR.-->  calculate (~S~S + ~S~S) FINALIZED -- RESULT [ ~S~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res0 res1 res2 )))

(defun OUTPUT-STARTED-calculate2-BH3 (num1 num2 den1 den2 )
    (setf *num1_BH3* num1)
    (setf *num2_BH3* num2)
    (setf *den1_BH3* den1)
    (setf *den2_BH3* den2)

    (format t "SPR.-->  NEW TASK ELEMENT SELECTED~%")
    (format t "SPR.-->  PROCESS:               COGNITIVE~%")
    (format t "SPR.-->  TASK NAME:             calculate2~%")
    (format t "SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE~%")
    (format t "SPR.-->  FORMULE:               calculate ( display1 operator display2 )~%")
    (format t "SPR.-->  calculate (~S~S - ~S~S) STARTED" num1 num2 den1 den2 ))

(defun OUTPUT-FINALIZED-calculate2-BH3 (res1 res2 )
    (if (= res1 0)
        (format t "SPR.-->  calculate (~S~S - ~S~S) FINALIZED -- RESULT [ ~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res2 )
        (format t "SPR.-->  calculate (~S~S - ~S~S) FINALIZED -- RESULT [ ~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res1 res2 )))

(defun OUTPUT-ERROR-1 ()
    (format t "SPR.-->  SKILL-BASED -- INTENTIONAL REDUCTION ERROR"))

(defun OUTPUT-ERROR-2 ()
    (format t "SPR.-->  SKILL-BASED -- PERCEPTUAL CONFUSION ERROR"))

(defun OUTPUT-ERROR-3 ()
    (format t "SPR.-->  SKILL-BASED -- MOTOR OVERLOAD ERROR"))

(defun OUTPUT-ERROR-4 ()
    (format t "SPR.-->  SKILL-BASED -- FALSE PERCEPTION ERROR"))

(defun OUTPUT-ERROR-5 ()
    (format t "SPR.-->  SKILL-BASED -- OMISSION ERROR"))

(defun OUTPUT-ERROR-6 ()
    (format t "SPR.-->  SKILL-BASED -- REPETITION ERROR"))

(defun OUTPUT-ERROR-7 ()
    (format t "SPR.-->  SKILL-BASED -- INVERSION ERROR"))

(defun OUTPUT-ERROR-8 ()
    (format t "SPR.-->  RULE-BASED -- RULE STRENGTH ERROR"))

(defun OUTPUT-ERROR-9 ()
    (format t "SPR.-->  RULE-BASED -- REDUNDANCY ERROR"))

(defun OUTPUT-ERROR-10 ()
    (format t "SPR.-->  RULE-BASED -- CODING DISABILITIES ERROR"))

(defun OUTPUT-ERROR-11 ()
    (format t "SPR.-->  RULE-BASED -- RULE WRONG ERROR"))

(defun OUTPUT-ERROR-12 ()
    (format t "SPR.-->  RULE-BASED -- OVERLOAD ERROR"))

(defun OUTPUT-ERROR-13 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- SELECTIVITY ERROR"))

(defun OUTPUT-ERROR-14 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- JOB MEMORY LIMITATION ERROR"))

(defun OUTPUT-ERROR-15 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- LEARNING DISABLED ERROR"))

(clear-all)

(define-model SPERERE-TASK

    (sgp :esc t :lf .05 :trace-detail LOW)
```

```
    (print "SIMULATOR OF PERFORMANCE IN ERROR - S. PERERE [TASK KNOWLEDGE]")
    (print "TRACE-DETAIL:      LOW")

(chunk-type number_rule_sum first second go-one)
(chunk-type number_rule_sub den num newnum loans)
(chunk-type count-order first second)
(chunk-type count-order-desc first second)
(chunk-type calculate1_sum num1 num2 den1 den2 res0 res1 res2 count-slot go-one task arg1 arg2
result count)
(chunk-type calculate2_sub num1 num2 den1 den2 res1 res2 count-slot loans task arg1 arg2
result count)

(add-dm
(sum-rule0 ISA number_rule_sum first 0 second 0 go-one 0)
(sum-rule1 ISA number_rule_sum first 1 second 1 go-one 0)
(sum-rule2 ISA number_rule_sum first 2 second 2 go-one 0)
(sum-rule3 ISA number_rule_sum first 3 second 3 go-one 0)
(sum-rule4 ISA number_rule_sum first 4 second 4 go-one 0)
(sum-rule5 ISA number_rule_sum first 5 second 5 go-one 0)
(sum-rule6 ISA number_rule_sum first 6 second 6 go-one 0)
(sum-rule7 ISA number_rule_sum first 7 second 7 go-one 0)
(sum-rule8 ISA number_rule_sum first 8 second 8 go-one 0)
(sum-rule9 ISA number_rule_sum first 9 second 9 go-one 0)
(sum-rule10 ISA number_rule_sum first 10 second 0 go-one 1)
(sum-rule11 ISA number_rule_sum first 11 second 1 go-one 1)
(sum-rule12 ISA number_rule_sum first 12 second 2 go-one 1)
(sum-rule13 ISA number_rule_sum first 13 second 3 go-one 1)
(sum-rule14 ISA number_rule_sum first 14 second 4 go-one 1)
(sum-rule15 ISA number_rule_sum first 15 second 5 go-one 1)
(sum-rule16 ISA number_rule_sum first 16 second 6 go-one 1)
(sum-rule17 ISA number_rule_sum first 17 second 7 go-one 1)
(sum-rule18 ISA number_rule_sum first 18 second 8 go-one 1)
(sum-rule19 ISA number_rule_sum first 19 second 9 go-one 1)

(sub-rule0_0 ISA number_rule_sub den 0 num 0 newnum 0 loans 0)
(sub-rule0_1 ISA number_rule_sub den 0 num 1 newnum 1 loans 0)
(sub-rule0_2 ISA number_rule_sub den 0 num 2 newnum 2 loans 0)
(sub-rule0_3 ISA number_rule_sub den 0 num 3 newnum 3 loans 0)
(sub-rule0_4 ISA number_rule_sub den 0 num 4 newnum 4 loans 0)
(sub-rule0_5 ISA number_rule_sub den 0 num 5 newnum 5 loans 0)
(sub-rule0_6 ISA number_rule_sub den 0 num 6 newnum 6 loans 0)
(sub-rule0_7 ISA number_rule_sub den 0 num 7 newnum 7 loans 0)
(sub-rule0_8 ISA number_rule_sub den 0 num 8 newnum 8 loans 0)
(sub-rule0_9 ISA number_rule_sub den 0 num 9 newnum 9 loans 0)

(sub-rule1_0 ISA number_rule_sub den 1 num 0 newnum 10 loans 1)
(sub-rule1_1 ISA number_rule_sub den 1 num 1 newnum 1 loans 0)
(sub-rule1_2 ISA number_rule_sub den 1 num 2 newnum 2 loans 0)
(sub-rule1_3 ISA number_rule_sub den 1 num 3 newnum 3 loans 0)
(sub-rule1_4 ISA number_rule_sub den 1 num 4 newnum 4 loans 0)
(sub-rule1_5 ISA number_rule_sub den 1 num 5 newnum 5 loans 0)
(sub-rule1_6 ISA number_rule_sub den 1 num 6 newnum 6 loans 0)
(sub-rule1_7 ISA number_rule_sub den 1 num 7 newnum 7 loans 0)
(sub-rule1_8 ISA number_rule_sub den 1 num 8 newnum 8 loans 0)
(sub-rule1_9 ISA number_rule_sub den 1 num 9 newnum 9 loans 0)

(sub-rule2_0 ISA number_rule_sub den 2 num 0 newnum 10 loans 1)
(sub-rule2_1 ISA number_rule_sub den 2 num 1 newnum 11 loans 1)
(sub-rule2_2 ISA number_rule_sub den 2 num 2 newnum 2 loans 0)
(sub-rule2_3 ISA number_rule_sub den 2 num 3 newnum 3 loans 0)
(sub-rule2_4 ISA number_rule_sub den 2 num 4 newnum 4 loans 0)
(sub-rule2_5 ISA number_rule_sub den 2 num 5 newnum 5 loans 0)
(sub-rule2_6 ISA number_rule_sub den 2 num 6 newnum 6 loans 0)
(sub-rule2_7 ISA number_rule_sub den 2 num 7 newnum 7 loans 0)
(sub-rule2_8 ISA number_rule_sub den 2 num 8 newnum 8 loans 0)
(sub-rule2_9 ISA number_rule_sub den 2 num 9 newnum 9 loans 0)

(sub-rule3_0 ISA number_rule_sub den 3 num 0 newnum 10 loans 1)
(sub-rule3_1 ISA number_rule_sub den 3 num 1 newnum 11 loans 1)
(sub-rule3_2 ISA number_rule_sub den 3 num 2 newnum 12 loans 1)
(sub-rule3_3 ISA number_rule_sub den 3 num 3 newnum 3 loans 0)
(sub-rule3_4 ISA number_rule_sub den 3 num 4 newnum 4 loans 0)
(sub-rule3_5 ISA number_rule_sub den 3 num 5 newnum 5 loans 0)
(sub-rule3_6 ISA number_rule_sub den 3 num 6 newnum 6 loans 0)
(sub-rule3_7 ISA number_rule_sub den 3 num 7 newnum 7 loans 0)
(sub-rule3_8 ISA number_rule_sub den 3 num 8 newnum 8 loans 0)
(sub-rule3_9 ISA number_rule_sub den 3 num 9 newnum 9 loans 0)
```

```
(sub-rule4_0 ISA number_rule_sub den 4 num 0 newnum 10 loans 1)
(sub-rule4_1 ISA number_rule_sub den 4 num 1 newnum 11 loans 1)
(sub-rule4_2 ISA number_rule_sub den 4 num 2 newnum 12 loans 1)
(sub-rule4_3 ISA number_rule_sub den 4 num 3 newnum 13 loans 1)
(sub-rule4_4 ISA number_rule_sub den 4 num 4 newnum 4 loans 0)
(sub-rule4_5 ISA number_rule_sub den 4 num 5 newnum 5 loans 0)
(sub-rule4_6 ISA number_rule_sub den 4 num 6 newnum 6 loans 0)
(sub-rule4_7 ISA number_rule_sub den 4 num 7 newnum 7 loans 0)
(sub-rule4_8 ISA number_rule_sub den 4 num 8 newnum 8 loans 0)
(sub-rule4_9 ISA number_rule_sub den 4 num 9 newnum 9 loans 0)

(sub-rule5_0 ISA number_rule_sub den 5 num 0 newnum 10 loans 1)
(sub-rule5_1 ISA number_rule_sub den 5 num 1 newnum 11 loans 1)
(sub-rule5_2 ISA number_rule_sub den 5 num 2 newnum 12 loans 1)
(sub-rule5_3 ISA number_rule_sub den 5 num 3 newnum 13 loans 1)
(sub-rule5_4 ISA number_rule_sub den 5 num 4 newnum 14 loans 1)
(sub-rule5_5 ISA number_rule_sub den 5 num 5 newnum 5 loans 0)
(sub-rule5_6 ISA number_rule_sub den 5 num 6 newnum 6 loans 0)
(sub-rule5_7 ISA number_rule_sub den 5 num 7 newnum 7 loans 0)
(sub-rule5_8 ISA number_rule_sub den 5 num 8 newnum 8 loans 0)
(sub-rule5_9 ISA number_rule_sub den 5 num 9 newnum 9 loans 0)

(sub-rule6_0 ISA number_rule_sub den 6 num 0 newnum 10 loans 1)
(sub-rule6_1 ISA number_rule_sub den 6 num 1 newnum 11 loans 1)
(sub-rule6_2 ISA number_rule_sub den 6 num 2 newnum 12 loans 1)
(sub-rule6_3 ISA number_rule_sub den 6 num 3 newnum 13 loans 1)
(sub-rule6_4 ISA number_rule_sub den 6 num 4 newnum 14 loans 1)
(sub-rule6_5 ISA number_rule_sub den 6 num 5 newnum 15 loans 1)
(sub-rule6_6 ISA number_rule_sub den 6 num 6 newnum 6 loans 0)
(sub-rule6_7 ISA number_rule_sub den 6 num 7 newnum 7 loans 0)
(sub-rule6_8 ISA number_rule_sub den 6 num 8 newnum 8 loans 0)
(sub-rule6_9 ISA number_rule_sub den 6 num 9 newnum 9 loans 0)

(sub-rule7_0 ISA number_rule_sub den 7 num 0 newnum 10 loans 1)
(sub-rule7_1 ISA number_rule_sub den 7 num 1 newnum 11 loans 1)
(sub-rule7_2 ISA number_rule_sub den 7 num 2 newnum 12 loans 1)
(sub-rule7_3 ISA number_rule_sub den 7 num 3 newnum 13 loans 1)
(sub-rule7_4 ISA number_rule_sub den 7 num 4 newnum 14 loans 1)
(sub-rule7_5 ISA number_rule_sub den 7 num 5 newnum 15 loans 1)
(sub-rule7_6 ISA number_rule_sub den 7 num 6 newnum 16 loans 1)
(sub-rule7_7 ISA number_rule_sub den 7 num 7 newnum 7 loans 0)
(sub-rule7_8 ISA number_rule_sub den 7 num 8 newnum 8 loans 0)
(sub-rule7_9 ISA number_rule_sub den 7 num 9 newnum 9 loans 0)

(sub-rule8_0 ISA number_rule_sub den 8 num 0 newnum 10 loans 1)
(sub-rule8_1 ISA number_rule_sub den 8 num 1 newnum 11 loans 1)
(sub-rule8_2 ISA number_rule_sub den 8 num 2 newnum 12 loans 1)
(sub-rule8_3 ISA number_rule_sub den 8 num 3 newnum 13 loans 1)
(sub-rule8_4 ISA number_rule_sub den 8 num 4 newnum 14 loans 1)
(sub-rule8_5 ISA number_rule_sub den 8 num 5 newnum 15 loans 1)
(sub-rule8_6 ISA number_rule_sub den 8 num 6 newnum 16 loans 1)
(sub-rule8_7 ISA number_rule_sub den 8 num 7 newnum 17 loans 1)
(sub-rule8_8 ISA number_rule_sub den 8 num 8 newnum 8 loans 0)
(sub-rule8_9 ISA number_rule_sub den 8 num 9 newnum 9 loans 0)

(sub-rule9_0 ISA number_rule_sub den 9 num 0 newnum 10 loans 1)
(sub-rule9_1 ISA number_rule_sub den 9 num 1 newnum 11 loans 1)
(sub-rule9_2 ISA number_rule_sub den 9 num 2 newnum 12 loans 1)
(sub-rule9_3 ISA number_rule_sub den 9 num 3 newnum 13 loans 1)
(sub-rule9_4 ISA number_rule_sub den 9 num 4 newnum 14 loans 1)
(sub-rule9_5 ISA number_rule_sub den 9 num 5 newnum 15 loans 1)
(sub-rule9_6 ISA number_rule_sub den 9 num 6 newnum 16 loans 1)
(sub-rule9_7 ISA number_rule_sub den 9 num 7 newnum 17 loans 1)
(sub-rule9_8 ISA number_rule_sub den 9 num 8 newnum 18 loans 1)
(sub-rule9_9 ISA number_rule_sub den 9 num 9 newnum 9 loans 0)

(count0 ISA count-order first 0 second 1)
(count1 ISA count-order first 1 second 2)
(count2 ISA count-order first 2 second 3)
(count3 ISA count-order first 3 second 4)
(count4 ISA count-order first 4 second 5)
(count5 ISA count-order first 5 second 6)
(count6 ISA count-order first 6 second 7)
(count7 ISA count-order first 7 second 8)
(count8 ISA count-order first 8 second 9)
(count9 ISA count-order first 9 second 10)
```

```
(count10 ISA count-order first 10 second 11)
(count11 ISA count-order first 11 second 12)
(count12 ISA count-order first 12 second 13)
(count13 ISA count-order first 13 second 14)
(count14 ISA count-order first 14 second 15)
(count15 ISA count-order first 15 second 16)
(count16 ISA count-order first 16 second 17)
(count17 ISA count-order first 17 second 18)
(count18 ISA count-order first 18 second 19)

(count0-desc ISA count-order-desc first 0 second nil)
(count1-desc ISA count-order-desc first 1 second 0)
(count2-desc ISA count-order-desc first 2 second 1)
(count3-desc ISA count-order-desc first 3 second 2)
(count4-desc ISA count-order-desc first 4 second 3)
(count5-desc ISA count-order-desc first 5 second 4)
(count6-desc ISA count-order-desc first 6 second 5)
(count7-desc ISA count-order-desc first 7 second 6)
(count8-desc ISA count-order-desc first 8 second 7)
(count9-desc ISA count-order-desc first 9 second 8)
(count10-desc ISA count-order-desc first 10 second 9)
(count11-desc ISA count-order-desc first 11 second 10)
(count12-desc ISA count-order-desc first 12 second 11)
(count13-desc ISA count-order-desc first 13 second 12)
(count14-desc ISA count-order-desc first 14 second 13)
(count15-desc ISA count-order-desc first 15 second 14)
(count16-desc ISA count-order-desc first 16 second 15)
(count17-desc ISA count-order-desc first 17 second 16)
(count18-desc ISA count-order-desc first 18 second 17)

(calculate1-sum-goal ISA calculate1_sum num1 9 num2 3 den1 3 den2 5 )
(calculate2-sub-goal ISA calculate2_sub num1 9 num2 4 den1 2 den2 3 )
)

(P calculate1-SUM-SLOT-START
    =goal>
        ISA             calculate1_sum
        count-slot      =count-slot
        arg1            =arg1
        arg2            =arg2
        result          nil
        count           0
==>
    =goal>
        task            1
        result          =arg1
        count           0
    +retrieval>
        ISA             count-order
        first           =arg1
        !eval!          (OUTPUT-SLOT-STARTED-BH3 =count-slot =arg1 =arg2 1)
)

(P calculate1-SUM-SLOT-FINALIZE
    =goal>
        ISA             calculate1_sum
        count-slot      =count-slot
        count           =arg2
        arg1            =arg1
        arg2            =arg2
        result          =result
        task            1
==>
    =goal>
        count           0
        task            nil
    +retrieval>
        ISA             number_rule_sum
        first           =result
        !eval!          (OUTPUT-SLOT-FINALIZED-BH3 =count-slot =result)
)

(P calculate1-SUM-SLOT-INCREMENT
    =goal>
        ISA             calculate1_sum
        count           =count
        - arg2          =count
```

```
        result          =result
        task            1
    =retrieval>
        ISA             count-order
        first           =result
        second          =newresult
==>
    =goal>
        result          =newresult
        task            2
    +retrieval>
        isa             count-order
        first           =count
        !eval!          (OUTPUT-SLOT-INCREMENT-BH3 =count =newresult)
)

(P calculate1-SUM-SLOT-COUNT
    =goal>
        ISA             calculate1_sum
        result          =result
        count           =count
        task            2
    =retrieval>
        ISA             count-order
        first           =count
        second          =newcount
==>
    =goal>
        count           =newcount
        task            1
    +retrieval>
        isa             count-order
        first           =result
        !eval!          (OUTPUT-SLOT-COUNT-BH3 =newcount)
)

(P calculate1-SUM-START
    =goal>
        ISA             calculate1_sum
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res0            nil
        res1            nil
        res2            nil
        count           nil
==>
    =goal>
        res1            =num1
        res2            =num2
        count-slot      2
        go-one          0
        !eval!          (OUTPUT-STARTED-calculate1-BH3 =num1 =num2 =den1 =den2 )
)

(P calculate1-SUM-IDENTIFIER-SLOT1
    =goal>
        ISA             calculate1_sum
        num1            =num1
        den1            =den1
        count-slot      1
        go-one          0
        count           nil
==>
    =goal>
        arg1            =num1
        arg2            =den1
        result          nil
        count           0
)

(P calculate1-SUM-IDENTIFIER-SLOT2
    =goal>
        ISA             calculate1_sum
        num2            =num2
        den2            =den2
```

```
        count-slot      2
        go-one          0
        count           nil
==>
    =goal>
        arg1            =num2
        arg2            =den2
        result          nil
        count           0
)

(P calculate1-SUM-PROCESS-RESULT-SLOT1
    =goal>
        ISA             calculate1_sum
        count-slot      1
        count           0
    =retrieval>
        ISA             number_rule_sum
        second          =second
        go-one          =go-one
==>
    =goal>
        res1            =second
        res0            =go-one
        count-slot      0
        count           nil
)

(P calculate1-SUM-PROCESS-RESULT-SLOT2
    =goal>
        ISA             calculate1_sum
        count-slot      2
        go-one          0
        count           0
    =retrieval>
        ISA             number_rule_sum
        second          =second
        go-one          =go-one
==>
    =goal>
        res2            =second
        count-slot      1
        go-one          =go-one
        count           nil
)

(P calculate1-SUM-DETECT-GO-ONE-SLOT2
    =goal>
        ISA             calculate1_sum
        num1            =num1
        count-slot      1
        go-one          1
        count           nil
==>
    =goal>
        arg1            =num1
        arg2            1
        count-slot      2
        result          nil
        count           0
        !eval!          (OUTPUT-GO-ONE-DETECTED-BH3)
)

(P calculate1-SUM-ASSIGN-GO-ONE-SLOT2
    =goal>
        ISA             calculate1_sum
        count-slot      2
        go-one          1
        count           0
    =retrieval>
        ISA             number_rule_sum
        first           =first
==>
    =goal>
        num1            =first
        count-slot      1
        go-one          0
```

```
        count           nil
        !eval!          (OUTPUT-ASSIGN-GO-ONE-BH3)
)

(P calculate1-SUM-FINALIZE
    =goal>
        ISA             calculate1_sum
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res0            =res0
        res1            =res1
        res2            =res2
        count-slot      0
        count           nil
==>
    +goal>
        ISA             calculate2_sub
        num1            9
        num2            4
        den1            2
        den2            3
        res1            nil
        res2            nil
        count-slot      nil
        loans           nil
        task            nil
        arg1            nil
        arg2            nil
        result          nil
        count           nil
        !eval!          (OUTPUT-FINALIZED-calculate1-BH3 =res0 =res1 =res2 )
)

(P calculate2-SUB-SLOT-START
    =goal>
        ISA             calculate2_sub
        count-slot      =count-slot
        task            2
        arg1            =arg1
        arg2            =arg2
==>
    =goal>
       task           3
      result          =arg1
      count           0
    +retrieval>
        isa             count-order-desc
        first           =arg1
        !eval!          (OUTPUT-SLOT-STARTED-BH3 =count-slot =arg1 =arg2 2)
)

(P calculate2-SUB-SLOT-FINALIZE
    =goal>
        ISA             calculate2_sub
        count-slot      =count-slot
        count           =arg2
        arg1            =arg1
        arg2            =arg2
        result          =result
        task            3
==>
    =goal>
        task            nil
        arg1            nil
        arg2            nil
        count           nil
        !eval!          (OUTPUT-SLOT-FINALIZED-BH3 =count-slot =result)
)

(P calculate2-SUB-SLOT-INCREMENT
    =goal>
        ISA             calculate2_sub
        result          =result
        count           =count
        - arg2          =count
```

```
        task            3
    =retrieval>
        ISA             count-order-desc
        first           =result
        second          =newresult
==>
    =goal>
        result          =newresult
        task            4
    +retrieval>
        isa             count-order
        first           =count
       !eval!         (OUTPUT-SLOT-INCREMENT-BH3 =count =newresult)
)

(P calculate2-SUB-SLOT-COUNT
    =goal>
        ISA             calculate2_sub
        result          =result
        count           =count
        task            4
    =retrieval>
        ISA             count-order
        first           =count
        second          =newcount
==>
    =goal>
        count           =newcount
        task            3
    +retrieval>
        isa             count-order-desc
        first           =result
       !eval!            (OUTPUT-SLOT-COUNT-BH3 =newcount)
)

(P calculate2-SUB-START
    =goal>
        ISA             calculate2_sub
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res1            nil
        res2            nil
        count-slot      nil
        arg1            nil
==>
    =goal>
        count-slot      2
        loans           0
        !eval!          (OUTPUT-STARTED-calculate2-BH3 =num1 =num2 =den1 =den2 )
)

(P calculate2-SUB-IDENTIFIER-SLOT1
    =goal>
        ISA             calculate2_sub
        num1            =num1
        den1            =den1
        count-slot      1
        loans           0
        task            nil
==>
    =goal>
        arg1            =num1
        arg2            =den1
        result          =num1
        task            2
    +retrieval>
        ISA             number_rule_sub
        den             =den1
        num             =num1

)(P calculate2-SUB-IDENTIFIER-SLOT2
    =goal>
        ISA             calculate2_sub
        num2            =num2
        den2            =den2
```

```
        count-slot      2
        loans           0
        task            nil
==>
    =goal>
        task            1
        count           nil
    +retrieval>
        ISA             number_rule_sub
        den             =den2
        num             =num2
)

(P calculate2-SUB-PROCESS-RESULT-SLOT1
    =goal>
        ISA             calculate2_sub
        res1            nil
        count-slot      1
        task            nil
        result          =result
==>
    =goal>
        count-slot      0
        res1            =result
        arg1            nil
        arg2            nil
        result          nil
)

(P calculate2-SUB-PROCESS-RESULT-SLOT2
    =goal>
        ISA             calculate2_sub
        res2            nil
        count-slot      2
        task            nil
        result          =result
==>
    =goal>
        count-slot      1
        res2            =result
        arg1            nil
        arg2            nil
        result          nil
)

(P calculate2-SUB-DETECT-LOANS-SLOT2
    =goal>
        ISA             calculate2_sub
        num1            =num1
        res1            nil
        count-slot      1
        loans           1
        task            nil
        result          nil
==>
    =goal>
        count-slot      2
    +retrieval>
        ISA             count-order-desc
        first           =num1
        !eval!          (OUTPUT-LOANS-DETECTED-BH3)
)

(P calculate2-SUB-ASSIGN-LOANS-SLOT2-CASE1
    =goal>
        ISA             calculate2_sub
        count-slot      2
        loans           1
        task            nil
        result          nil
    =retrieval>
        ISA             count-order-desc
        first           =first
        second          =second
==>
    =goal>
        num1            =second
```

```
        count-slot      1
        loans           0
        result          nil
        !eval!          (OUTPUT-ASSIGN-LOANS-BH3)
)

(P calculate2-SUB-VERIFY-SLOT2
    =goal>
        ISA             calculate2_sub
        count-slot      2
        task            1
        count           nil
    =retrieval>
        ISA             number_rule_sub
        den             =den
        newnum          =newnum
        loans           =loans
==>
    =goal>
        num2            =newnum
        loans           =loans
        arg1            =newnum
        arg2            =den
        task            2
)

(P calculate2-SUB-FINALIZE
    =goal>
        ISA             calculate2_sub
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res1            =res1
        res2            =res2
        count-slot      0
==>
    =goal>
        count-slot      nil
        !eval!            (OUTPUT-FINALIZED-calculate2-BH3 =res1 =res2 )
)


(goal-focus calculate1-sum-goal)

)
```

APÊNDICE 2 – CODIGO ACT-R DO EXPERIMENTO – PRODUÇÃO PERTURBADA

```
(defvar *num1_BH3* nil)
(defvar *den1_BH3* nil)
(defvar *num2_BH3* nil)
(defvar *den2_BH3* nil)
(defvar *num3_BH3* nil)
(defvar *den3_BH3* nil)
(defvar *num4_BH3* nil)
(defvar *den4_BH3* nil)
(defvar *num5_BH3* nil)
(defvar *den5_BH3* nil)
(defvar *num6_BH3* nil)
(defvar *den6_BH3* nil)
(defvar *num7_BH3* nil)
(defvar *den7_BH3* nil)
(defvar *num8_BH3* nil)
(defvar *den8_BH3* nil)
(defvar *num9_BH3* nil)
(defvar *den9_BH3* nil)
(defvar *num10_BH3* nil)
(defvar *den10_BH3* nil)

(defun BEGIN ()
    (reset)
    (pm-run 10))

(defun OUTPUT-SLOT-STARTED-BH3 (count arg1 arg2 symbol)
    (case symbol
        (1 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S + ~S) STARTED" count arg1 arg2))
        (2 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S - ~S) STARTED" count arg1 arg2))
        (3 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S / ~S) STARTED" count arg1 arg2))
        (4 (format t "SPR.-->  CALCULE SLOT [ ~S ] (~S * ~S) STARTED" count arg1 arg2))))

(defun OUTPUT-SLOT-INCREMENT-BH3 (count result)
    (format t "SPR.-->  INCLEMENT COUNT [ ~S ]  --  NEW RESULT [ ~S ]" count result))

(defun OUTPUT-SLOT-COUNT-BH3 (count)
    (format t "SPR.-->  NEW COUNT [ ~S ]" count))

(defun OUTPUT-GO-ONE-DETECTED-BH3 ()
    (format t "SPR.-->  GO-ONE DETECTED"))

(defun OUTPUT-ASSIGN-GO-ONE-BH3 ()
    (format t "SPR.-->  ASSIGN GO-ONE"))

(defun OUTPUT-LOANS-DETECTED-BH3 ()
    (format t "SPR.-->  LOANS DETECTED"))

(defun OUTPUT-ASSIGN-LOANS-BH3 ()
    (format t "SPR.-->  ASSIGN LOANS"))

(defun OUTPUT-SLOT-FINALIZED-BH3 (count result)
    (format t "SPR.-->  CALCULE SLOT [ ~S ] FINALIZED  --  NEW RESULT [ ~S ]" count result))


(defun OUTPUT-STARTED-calculate1-BH3 (num1 num2 den1 den2 )
    (setf *num1_BH3* num1)
    (setf *num2_BH3* num2)
    (setf *den1_BH3* den1)
    (setf *den2_BH3* den2)

    (format t "SPR.-->  NEW TASK ELEMENT SELECTED~%")
    (format t "SPR.-->  PROCESS:               COGNITIVE~%")
    (format t "SPR.-->  TASK NAME:             calculate1~%")
    (format t "SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE~%")
    (format t "SPR.-->  FORMULE:               calculate ( display1 operator display2 )~%")
    (format t "SPR.-->  calculate (~S~S + ~S~S) STARTED" num1 num2 den1 den2 ))

(defun OUTPUT-FINALIZED-calculate1-BH3 (res0 res1 res2 )
```

```
    (if (= res0 0)
        (format t "SPR.-->  calculate (~S~S + ~S~S) FINALIZED -- RESULT [ ~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res1 res2 )
        (format t "SPR.-->  calculate (~S~S + ~S~S) FINALIZED -- RESULT [ ~S~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res0 res1 res2 )))

(defun OUTPUT-STARTED-calculate2-BH3 (num1 num2 den1 den2 )
    (setf *num1_BH3* num1)
    (setf *num2_BH3* num2)
    (setf *den1_BH3* den1)
    (setf *den2_BH3* den2)

    (format t "SPR.-->  NEW TASK ELEMENT SELECTED~%")
    (format t "SPR.-->  PROCESS:               COGNITIVE~%")
    (format t "SPR.-->  TASK NAME:             calculate2~%")
    (format t "SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE~%")
    (format t "SPR.-->  FORMULE:               calculate ( display1 operator display2 )~%")
    (format t "SPR.-->  calculate (~S~S - ~S~S) STARTED" num1 num2 den1 den2 ))

(defun OUTPUT-FINALIZED-calculate2-BH3 (res1 res2 )
    (if (= res1 0)
        (format t "SPR.-->  calculate (~S~S - ~S~S) FINALIZED -- RESULT [ ~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res2 )
        (format t "SPR.-->  calculate (~S~S - ~S~S) FINALIZED -- RESULT [ ~S~S ]" *num1_BH3*
*num2_BH3* *den1_BH3* *den2_BH3*  res1 res2 )))

(defun OUTPUT-ERROR-1 ()
    (format t "SPR.-->  SKILL-BASED -- INTENTIONAL REDUCTION ERROR"))

(defun OUTPUT-ERROR-2 ()
    (format t "SPR.-->  SKILL-BASED -- PERCEPTUAL CONFUSION ERROR"))

(defun OUTPUT-ERROR-3 ()
    (format t "SPR.-->  SKILL-BASED -- MOTOR OVERLOAD ERROR"))

(defun OUTPUT-ERROR-4 ()
    (format t "SPR.-->  SKILL-BASED -- FALSE PERCEPTION ERROR"))

(defun OUTPUT-ERROR-5 ()
    (format t "SPR.-->  SKILL-BASED -- OMISSION ERROR"))

(defun OUTPUT-ERROR-6 ()
    (format t "SPR.-->  SKILL-BASED -- REPETITION ERROR"))

(defun OUTPUT-ERROR-7 ()
    (format t "SPR.-->  SKILL-BASED -- INVERSION ERROR"))

(defun OUTPUT-ERROR-8 ()
    (format t "SPR.-->  RULE-BASED -- RULE STRENGTH ERROR"))

(defun OUTPUT-ERROR-9 ()
    (format t "SPR.-->  RULE-BASED -- REDUNDANCY ERROR"))

(defun OUTPUT-ERROR-10 ()
    (format t "SPR.-->  RULE-BASED -- CODING DISABILITIES ERROR"))

(defun OUTPUT-ERROR-11 ()
    (format t "SPR.-->  RULE-BASED -- RULE WRONG ERROR"))

(defun OUTPUT-ERROR-12 ()
    (format t "SPR.-->  RULE-BASED -- OVERLOAD ERROR"))

(defun OUTPUT-ERROR-13 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- SELECTIVITY ERROR"))

(defun OUTPUT-ERROR-14 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- JOB MEMORY LIMITATION ERROR"))

(defun OUTPUT-ERROR-15 ()
    (format t "SPR.-->  KNOWLEDGE-BASED -- LEARNING DISABLED ERROR"))

(clear-all)

(define-model SPERERE-TASK

    (sgp :esc t :lf .05 :trace-detail LOW)
```

```
    (print "SIMULATOR OF PERFORMANCE IN ERROR - S. PERERE [TASK KNOWLEDGE]")
    (print "TRACE-DETAIL:      LOW")

(chunk-type number_rule_sum first second go-one)
(chunk-type number_rule_sub den num newnum loans)
(chunk-type count-order first second)
(chunk-type count-order-desc first second)
(chunk-type calculate1_sum num1 num2 den1 den2 res0 res1 res2 count-slot go-one task arg1 arg2
result count)
(chunk-type calculate2_sub num1 num2 den1 den2 res1 res2 count-slot loans task arg1 arg2
result count)

(add-dm
(sum-rule0 ISA number_rule_sum first 0 second 0 go-one 0)
(sum-rule1 ISA number_rule_sum first 1 second 1 go-one 0)
(sum-rule2 ISA number_rule_sum first 2 second 2 go-one 0)
(sum-rule3 ISA number_rule_sum first 3 second 3 go-one 0)
(sum-rule4 ISA number_rule_sum first 4 second 4 go-one 0)
(sum-rule5 ISA number_rule_sum first 5 second 5 go-one 0)
(sum-rule6 ISA number_rule_sum first 6 second 6 go-one 0)
(sum-rule7 ISA number_rule_sum first 7 second 7 go-one 0)
(sum-rule8 ISA number_rule_sum first 8 second 8 go-one 0)
(sum-rule9 ISA number_rule_sum first 9 second 9 go-one 0)
(sum-rule10 ISA number_rule_sum first 10 second 0 go-one 1)
(sum-rule11 ISA number_rule_sum first 11 second 1 go-one 1)
(sum-rule12 ISA number_rule_sum first 12 second 2 go-one 1)
(sum-rule13 ISA number_rule_sum first 13 second 3 go-one 1)
(sum-rule14 ISA number_rule_sum first 14 second 4 go-one 1)
(sum-rule15 ISA number_rule_sum first 15 second 5 go-one 1)
(sum-rule16 ISA number_rule_sum first 16 second 6 go-one 1)
(sum-rule17 ISA number_rule_sum first 17 second 7 go-one 1)
(sum-rule18 ISA number_rule_sum first 18 second 8 go-one 1)
(sum-rule19 ISA number_rule_sum first 19 second 9 go-one 1)

(sub-rule0_0 ISA number_rule_sub den 0 num 0 newnum 0 loans 0)
(sub-rule0_1 ISA number_rule_sub den 0 num 1 newnum 1 loans 0)
(sub-rule0_2 ISA number_rule_sub den 0 num 2 newnum 2 loans 0)
(sub-rule0_3 ISA number_rule_sub den 0 num 3 newnum 3 loans 0)
(sub-rule0_4 ISA number_rule_sub den 0 num 4 newnum 4 loans 0)
(sub-rule0_5 ISA number_rule_sub den 0 num 5 newnum 5 loans 0)
(sub-rule0_6 ISA number_rule_sub den 0 num 6 newnum 6 loans 0)
(sub-rule0_7 ISA number_rule_sub den 0 num 7 newnum 7 loans 0)
(sub-rule0_8 ISA number_rule_sub den 0 num 8 newnum 8 loans 0)
(sub-rule0_9 ISA number_rule_sub den 0 num 9 newnum 9 loans 0)

(sub-rule1_0 ISA number_rule_sub den 1 num 0 newnum 10 loans 1)
(sub-rule1_1 ISA number_rule_sub den 1 num 1 newnum 1 loans 0)
(sub-rule1_2 ISA number_rule_sub den 1 num 2 newnum 2 loans 0)
(sub-rule1_3 ISA number_rule_sub den 1 num 3 newnum 3 loans 0)
(sub-rule1_4 ISA number_rule_sub den 1 num 4 newnum 4 loans 0)
(sub-rule1_5 ISA number_rule_sub den 1 num 5 newnum 5 loans 0)
(sub-rule1_6 ISA number_rule_sub den 1 num 6 newnum 6 loans 0)
(sub-rule1_7 ISA number_rule_sub den 1 num 7 newnum 7 loans 0)
(sub-rule1_8 ISA number_rule_sub den 1 num 8 newnum 8 loans 0)
(sub-rule1_9 ISA number_rule_sub den 1 num 9 newnum 9 loans 0)

(sub-rule2_0 ISA number_rule_sub den 2 num 0 newnum 10 loans 1)
(sub-rule2_1 ISA number_rule_sub den 2 num 1 newnum 11 loans 1)
(sub-rule2_2 ISA number_rule_sub den 2 num 2 newnum 2 loans 0)
(sub-rule2_3 ISA number_rule_sub den 2 num 3 newnum 3 loans 0)
(sub-rule2_4 ISA number_rule_sub den 2 num 4 newnum 4 loans 0)
(sub-rule2_5 ISA number_rule_sub den 2 num 5 newnum 5 loans 0)
(sub-rule2_6 ISA number_rule_sub den 2 num 6 newnum 6 loans 0)
(sub-rule2_7 ISA number_rule_sub den 2 num 7 newnum 7 loans 0)
(sub-rule2_8 ISA number_rule_sub den 2 num 8 newnum 8 loans 0)
(sub-rule2_9 ISA number_rule_sub den 2 num 9 newnum 9 loans 0)

(sub-rule3_0 ISA number_rule_sub den 3 num 0 newnum 10 loans 1)
(sub-rule3_1 ISA number_rule_sub den 3 num 1 newnum 11 loans 1)
(sub-rule3_2 ISA number_rule_sub den 3 num 2 newnum 12 loans 1)
(sub-rule3_3 ISA number_rule_sub den 3 num 3 newnum 3 loans 0)
(sub-rule3_4 ISA number_rule_sub den 3 num 4 newnum 4 loans 0)
(sub-rule3_5 ISA number_rule_sub den 3 num 5 newnum 5 loans 0)
(sub-rule3_6 ISA number_rule_sub den 3 num 6 newnum 6 loans 0)
(sub-rule3_7 ISA number_rule_sub den 3 num 7 newnum 7 loans 0)
(sub-rule3_8 ISA number_rule_sub den 3 num 8 newnum 8 loans 0)
(sub-rule3_9 ISA number_rule_sub den 3 num 9 newnum 9 loans 0)
```

```
(sub-rule4_0 ISA number_rule_sub den 4 num 0 newnum 10 loans 1)
(sub-rule4_1 ISA number_rule_sub den 4 num 1 newnum 11 loans 1)
(sub-rule4_2 ISA number_rule_sub den 4 num 2 newnum 12 loans 1)
(sub-rule4_3 ISA number_rule_sub den 4 num 3 newnum 13 loans 1)
(sub-rule4_4 ISA number_rule_sub den 4 num 4 newnum 4 loans 0)
(sub-rule4_5 ISA number_rule_sub den 4 num 5 newnum 5 loans 0)
(sub-rule4_6 ISA number_rule_sub den 4 num 6 newnum 6 loans 0)
(sub-rule4_7 ISA number_rule_sub den 4 num 7 newnum 7 loans 0)
(sub-rule4_8 ISA number_rule_sub den 4 num 8 newnum 8 loans 0)
(sub-rule4_9 ISA number_rule_sub den 4 num 9 newnum 9 loans 0)

(sub-rule5_0 ISA number_rule_sub den 5 num 0 newnum 10 loans 1)
(sub-rule5_1 ISA number_rule_sub den 5 num 1 newnum 11 loans 1)
(sub-rule5_2 ISA number_rule_sub den 5 num 2 newnum 12 loans 1)
(sub-rule5_3 ISA number_rule_sub den 5 num 3 newnum 13 loans 1)
(sub-rule5_4 ISA number_rule_sub den 5 num 4 newnum 14 loans 1)
(sub-rule5_5 ISA number_rule_sub den 5 num 5 newnum 5 loans 0)
(sub-rule5_6 ISA number_rule_sub den 5 num 6 newnum 6 loans 0)
(sub-rule5_7 ISA number_rule_sub den 5 num 7 newnum 7 loans 0)
(sub-rule5_8 ISA number_rule_sub den 5 num 8 newnum 8 loans 0)
(sub-rule5_9 ISA number_rule_sub den 5 num 9 newnum 9 loans 0)

(sub-rule6_0 ISA number_rule_sub den 6 num 0 newnum 10 loans 1)
(sub-rule6_1 ISA number_rule_sub den 6 num 1 newnum 11 loans 1)
(sub-rule6_2 ISA number_rule_sub den 6 num 2 newnum 12 loans 1)
(sub-rule6_3 ISA number_rule_sub den 6 num 3 newnum 13 loans 1)
(sub-rule6_4 ISA number_rule_sub den 6 num 4 newnum 14 loans 1)
(sub-rule6_5 ISA number_rule_sub den 6 num 5 newnum 15 loans 1)
(sub-rule6_6 ISA number_rule_sub den 6 num 6 newnum 6 loans 0)
(sub-rule6_7 ISA number_rule_sub den 6 num 7 newnum 7 loans 0)
(sub-rule6_8 ISA number_rule_sub den 6 num 8 newnum 8 loans 0)
(sub-rule6_9 ISA number_rule_sub den 6 num 9 newnum 9 loans 0)

(sub-rule7_0 ISA number_rule_sub den 7 num 0 newnum 10 loans 1)
(sub-rule7_1 ISA number_rule_sub den 7 num 1 newnum 11 loans 1)
(sub-rule7_2 ISA number_rule_sub den 7 num 2 newnum 12 loans 1)
(sub-rule7_3 ISA number_rule_sub den 7 num 3 newnum 13 loans 1)
(sub-rule7_4 ISA number_rule_sub den 7 num 4 newnum 14 loans 1)
(sub-rule7_5 ISA number_rule_sub den 7 num 5 newnum 15 loans 1)
(sub-rule7_6 ISA number_rule_sub den 7 num 6 newnum 16 loans 1)
(sub-rule7_7 ISA number_rule_sub den 7 num 7 newnum 7 loans 0)
(sub-rule7_8 ISA number_rule_sub den 7 num 8 newnum 8 loans 0)
(sub-rule7_9 ISA number_rule_sub den 7 num 9 newnum 9 loans 0)

(sub-rule8_0 ISA number_rule_sub den 8 num 0 newnum 10 loans 1)
(sub-rule8_1 ISA number_rule_sub den 8 num 1 newnum 11 loans 1)
(sub-rule8_2 ISA number_rule_sub den 8 num 2 newnum 12 loans 1)
(sub-rule8_3 ISA number_rule_sub den 8 num 3 newnum 13 loans 1)
(sub-rule8_4 ISA number_rule_sub den 8 num 4 newnum 14 loans 1)
(sub-rule8_5 ISA number_rule_sub den 8 num 5 newnum 15 loans 1)
(sub-rule8_6 ISA number_rule_sub den 8 num 6 newnum 16 loans 1)
(sub-rule8_7 ISA number_rule_sub den 8 num 7 newnum 17 loans 1)
(sub-rule8_8 ISA number_rule_sub den 8 num 8 newnum 8 loans 0)
(sub-rule8_9 ISA number_rule_sub den 8 num 9 newnum 9 loans 0)

(sub-rule9_0 ISA number_rule_sub den 9 num 0 newnum 10 loans 1)
(sub-rule9_1 ISA number_rule_sub den 9 num 1 newnum 11 loans 1)
(sub-rule9_2 ISA number_rule_sub den 9 num 2 newnum 12 loans 1)
(sub-rule9_3 ISA number_rule_sub den 9 num 3 newnum 13 loans 1)
(sub-rule9_4 ISA number_rule_sub den 9 num 4 newnum 14 loans 1)
(sub-rule9_5 ISA number_rule_sub den 9 num 5 newnum 15 loans 1)
(sub-rule9_6 ISA number_rule_sub den 9 num 6 newnum 16 loans 1)
(sub-rule9_7 ISA number_rule_sub den 9 num 7 newnum 17 loans 1)
(sub-rule9_8 ISA number_rule_sub den 9 num 8 newnum 18 loans 1)
(sub-rule9_9 ISA number_rule_sub den 9 num 9 newnum 9 loans 0)

(count0 ISA count-order first 0 second 1)
(count1 ISA count-order first 1 second 2)
(count2 ISA count-order first 2 second 3)
(count3 ISA count-order first 3 second 4)
(count4 ISA count-order first 4 second 5)
(count5 ISA count-order first 5 second 6)
(count6 ISA count-order first 6 second 7)
(count7 ISA count-order first 7 second 8)
(count8 ISA count-order first 8 second 9)
(count9 ISA count-order first 9 second 10)
```

```
(count10 ISA count-order first 10 second 11)
(count11 ISA count-order first 11 second 12)
(count12 ISA count-order first 12 second 13)
(count13 ISA count-order first 13 second 14)
(count14 ISA count-order first 14 second 15)
(count15 ISA count-order first 15 second 16)
(count16 ISA count-order first 16 second 17)
(count17 ISA count-order first 17 second 18)
(count18 ISA count-order first 18 second 19)

(count0-desc ISA count-order-desc first 0 second nil)
(count1-desc ISA count-order-desc first 1 second 0)
(count2-desc ISA count-order-desc first 2 second 1)
(count3-desc ISA count-order-desc first 3 second 2)
(count4-desc ISA count-order-desc first 4 second 3)
(count5-desc ISA count-order-desc first 5 second 4)
(count6-desc ISA count-order-desc first 6 second 5)
(count7-desc ISA count-order-desc first 7 second 6)
(count8-desc ISA count-order-desc first 8 second 7)
(count9-desc ISA count-order-desc first 9 second 8)
(count10-desc ISA count-order-desc first 10 second 9)
(count11-desc ISA count-order-desc first 11 second 10)
(count12-desc ISA count-order-desc first 12 second 11)
(count13-desc ISA count-order-desc first 13 second 12)
(count14-desc ISA count-order-desc first 14 second 13)
(count15-desc ISA count-order-desc first 15 second 14)
(count16-desc ISA count-order-desc first 16 second 15)
(count17-desc ISA count-order-desc first 17 second 16)
(count18-desc ISA count-order-desc first 18 second 17)

(calculate1-sum-goal ISA calculate1_sum num1 9 num2 3 den1 3 den2 5 )
(calculate2-sub-goal ISA calculate2_sub num1 9 num2 4 den1 2 den2 3 )
)

(P calculate1-SUM-SLOT-START
    =goal>
        ISA             calculate1_sum
        count-slot      =count-slot
        arg1            =arg1
        arg2            =arg2
        result          nil
        count           0
==>
    -goal>
    !eval!          (OUTPUT-ERROR-5)
)

(P calculate1-SUM-SLOT-FINALIZE
    =goal>
        ISA             calculate1_sum
        count-slot      =count-slot
        count           =arg2
        arg1            =arg1
        arg2            =arg2
        result          =result
        task            1
==>
    =goal>
        count           0
        task            nil
    +retrieval>
        ISA             number_rule_sum
        first           =result
        !eval!          (OUTPUT-SLOT-FINALIZED-BH3 =count-slot =result)
)

(P calculate1-SUM-SLOT-INCREMENT
    =goal>
        ISA             calculate1_sum
        count           =count
        - arg2          =count
        result          =result
        task            1
    =retrieval>
        ISA             count-order
        first           =result
        second          =newresult
```

```
==>
    =goal>
        result          =newresult
        task            2
    +retrieval>
        isa             count-order
        first           =count
        !eval!          (OUTPUT-SLOT-INCREMENT-BH3 =count =newresult)
)

(P calculate1-SUM-SLOT-COUNT
    =goal>
        ISA             calculate1_sum
        result          =result
        count           =count
        task            2
    =retrieval>
        ISA             count-order
        first           =count
        second          =newcount
==>
    =goal>
        count           =newcount
        task            1
    +retrieval>
        isa             count-order
        first           =result
        !eval!          (OUTPUT-SLOT-COUNT-BH3 =newcount)
)

(P calculate1-SUM-START
    =goal>
        ISA             calculate1_sum
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res0            nil
        res1            nil
        res2            nil
        count           nil
==>
    =goal>
        res1            =num1
        res2            =num2
        count-slot      2
        go-one          0
        !eval!          (OUTPUT-STARTED-calculate1-BH3 =num1 =num2 =den1 =den2 )
)

(P calculate1-SUM-IDENTIFIER-SLOT1
    =goal>
        ISA             calculate1_sum
        num1            =num1
        den1            =den1
        count-slot      1
        go-one          0
        count           nil
==>
    =goal>
        arg1            =num1
        arg2            =den1
        result          nil
        count           0
)

(P calculate1-SUM-IDENTIFIER-SLOT2
    =goal>
        ISA             calculate1_sum
        num2            =num2
        den2            =den2
        count-slot      2
        go-one          0
        count           nil
==>
    =goal>
        arg1            =num2
```

```
        arg2            =den2
        result          nil
        count           0
)

(P calculate1-SUM-PROCESS-RESULT-SLOT1
    =goal>
        ISA             calculate1_sum
        count-slot      1
        count           0
    =retrieval>
        ISA             number_rule_sum
        second          =second
        go-one          =go-one
==>
    =goal>
        res1            =second
        res0            =go-one
        count-slot      0
        count           nil
)

(P calculate1-SUM-PROCESS-RESULT-SLOT2
    =goal>
        ISA             calculate1_sum
        count-slot      2
        go-one          0
        count           0
    =retrieval>
        ISA             number_rule_sum
        second          =second
        go-one          =go-one
==>
    =goal>
        res2            =second
        count-slot      1
        go-one          =go-one
        count           nil
)

(P calculate1-SUM-DETECT-GO-ONE-SLOT2
    =goal>
        ISA             calculate1_sum
        num1            =num1
        count-slot      1
        go-one          1
        count           nil
==>
    =goal>
        arg1            =num1
        arg2            1
        count-slot      2
        result          nil
        count           0
        !eval!          (OUTPUT-GO-ONE-DETECTED-BH3)
)

(P calculate1-SUM-ASSIGN-GO-ONE-SLOT2
    =goal>
        ISA             calculate1_sum
        count-slot      2
        go-one          1
        count           0
    =retrieval>
        ISA             number_rule_sum
        first           =first
==>
    =goal>
        num1            =first
        count-slot      1
        go-one          0
        count           nil
        !eval!          (OUTPUT-ASSIGN-GO-ONE-BH3)
)

(P calculate1-SUM-FINALIZE
    =goal>
```

```
        ISA             calculate1_sum
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res0            =res0
        res1            =res1
        res2            =res2
        count-slot      0
        count           nil
==>
    +goal>
        ISA             calculate2_sub
        num1            9
        num2            4
        den1            2
        den2            3
        res1            nil
        res2            nil
        count-slot      nil
        loans           nil
        task            nil
        arg1            nil
        arg2            nil
        result          nil
        count           nil
        !eval!          (OUTPUT-FINALIZED-calculate1-BH3 =res0 =res1 =res2 )
)

(P calculate2-SUB-SLOT-START
    =goal>
        ISA             calculate2_sub
        count-slot      =count-slot
        task            2
        arg1            =arg1
        arg2            =arg2
==>
    =goal>
       task           3
       result          =arg1
       count           0
    +retrieval>
        isa             count-order-desc
        first           =arg1
        !eval!          (OUTPUT-SLOT-STARTED-BH3 =count-slot =arg1 =arg2 2)
)

(P calculate2-SUB-SLOT-FINALIZE
    =goal>
        ISA             calculate2_sub
        count-slot      =count-slot
        count           =arg2
        arg1            =arg1
        arg2            =arg2
        result          =result
        task            3
==>
    =goal>
        task            nil
        arg1            nil
        arg2            nil
        count           nil
        !eval!          (OUTPUT-SLOT-FINALIZED-BH3 =count-slot =result)
)

(P calculate2-SUB-SLOT-INCREMENT
    =goal>
        ISA             calculate2_sub
        result          =result
        count           =count
        - arg2          =count
        task            3
    =retrieval>
        ISA             count-order-desc
        first           =result
        second          =newresult
==>
```

```
    =goal>
        result          =newresult
        task            4
    +retrieval>
        isa             count-order
        first           =count
       !eval!         (OUTPUT-SLOT-INCREMENT-BH3 =count =newresult)
)

(P calculate2-SUB-SLOT-COUNT
    =goal>
        ISA             calculate2_sub
        result          =result
        count           =count
        task            4
    =retrieval>
        ISA             count-order
        first           =count
        second          =newcount
==>
    =goal>
        count           =newcount
        task            3
    +retrieval>
        isa             count-order-desc
        first           =result
       !eval!            (OUTPUT-SLOT-COUNT-BH3 =newcount)
)

(P calculate2-SUB-START
    =goal>
        ISA             calculate2_sub
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res1            nil
        res2            nil
        count-slot      nil
        arg1            nil
==>
    =goal>
        count-slot      2
        loans           0
        !eval!          (OUTPUT-STARTED-calculate2-BH3 =num1 =num2 =den1 =den2 )
)

(P calculate2-SUB-IDENTIFIER-SLOT1
    =goal>
        ISA             calculate2_sub
        num1            =num1
        den1            =den1
        count-slot      1
        loans           0
        task            nil
==>
    =goal>
        arg1            =num1
        arg2            =den1
        result          =num1
        task            2
    +retrieval>
        ISA             number_rule_sub
        den             =den1
        num             =num1

)

(P calculate2-SUB-IDENTIFIER-SLOT2
    =goal>
        ISA             calculate2_sub
        num2            =num2
        den2            =den2
        count-slot      2
        loans           0
        task            nil
==>
```

```
    =goal>
        task            1
        count           nil
    +retrieval>
        ISA             number_rule_sub
        den             =den2
        num             =num2
)

(P calculate2-SUB-PROCESS-RESULT-SLOT1
    =goal>
        ISA             calculate2_sub
        res1            nil
        count-slot      1
        task            nil
        result          =result
==>
    =goal>
        count-slot      0
        res1            =result
        arg1            nil
        arg2            nil
        result          nil
)

(P calculate2-SUB-PROCESS-RESULT-SLOT2
    =goal>
        ISA             calculate2_sub
        res2            nil
        count-slot      2
        task            nil
        result          =result
==>
    =goal>
        count-slot      1
        res2            =result
        arg1            nil
        arg2            nil
        result          nil
)

(P calculate2-SUB-DETECT-LOANS-SLOT2
    =goal>
        ISA             calculate2_sub
        num1            =num1
        res1            nil
        count-slot      1
        loans           1
        task            nil
        result          nil
==>
    =goal>
        count-slot      2
    +retrieval>
        ISA             count-order-desc
        first           =num1
        !eval!          (OUTPUT-LOANS-DETECTED-BH3)
)

(P calculate2-SUB-ASSIGN-LOANS-SLOT2-CASE1
    =goal>
        ISA             calculate2_sub
        count-slot      2
        loans           1
        task            nil
        result          nil
    =retrieval>
        ISA             count-order-desc
        first           =first
        second          =second
==>
    =goal>
        num1            =second
        count-slot      1
        loans           0
        result          nil
        !eval!          (OUTPUT-ASSIGN-LOANS-BH3)
```

```
)

(P calculate2-SUB-VERIFY-SLOT2
    =goal>
        ISA             calculate2_sub
        count-slot      2
        task            1
        count           nil
    =retrieval>
        ISA             number_rule_sub
        den             =den
        newnum          =newnum
        loans           =loans
==>
    =goal>
        num2            =newnum
        loans           =loans
        arg1            =newnum
        arg2            =den
        task            2
)

(P calculate2-SUB-FINALIZE
    =goal>
        ISA             calculate2_sub
        num1            =num1
        num2            =num2
        den1            =den1
        den2            =den2
        res1            =res1
        res2            =res2
        count-slot      0
==>
    =goal>
        count-slot      nil
        !eval!            (OUTPUT-FINALIZED-calculate2-BH3 =res1 =res2 )
)

(goal-focus calculate1-sum-goal)

)
```

APÊNDICE 3 – TRAÇOS DO ACT-R – REFERENTE À PRODUÇÃO DO APÊNDICE 1

```
; Loading C:\Documents and Settings\CORTEZTH\Desktop\CORRECT.lisp

"SIMULATOR OF PERFORMANCE IN ERROR - S. PERERE [TASK KNOWLEDGE]"
"TRACE-DETAIL:      LOW"
Model Reloaded

#|## Model C:/Documents and Settings/CORTEZTH/Desktop/CORRECT.lisp loaded. ##|#
0.000   GOAL                  SET-BUFFER-CHUNK GOAL CALCULATE1-SUM-GOAL REQUESTED NIL
     0.050   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-START
SPR.-->  NEW TASK ELEMENT SELECTED
SPR.-->  PROCESS:               COGNITIVE
SPR.-->  TASK NAME:             calculate1
SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE
SPR.-->  FORMULE:               calculate ( display1 operator display2 )
SPR.-->  calculate (93 + 35) STARTED
     0.100   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-IDENTIFIER-SLOT2
     0.150   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-START
SPR.-->  CALCULE SLOT [ 2 ] (3 + 5) STARTED
     0.200   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT3
     0.250   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 0 ]  --  NEW RESULT [ 4 ]
     0.300   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT0
     0.350   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 1 ]
     0.400   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT4
     0.450   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 1 ]  --  NEW RESULT [ 5 ]
     0.500   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT1
     0.550   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 2 ]
     0.600   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT5
     0.650   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 2 ]  --  NEW RESULT [ 6 ]
     0.700   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT2
     0.750   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 3 ]
     0.800   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT6
     0.850   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 3 ]  --  NEW RESULT [ 7 ]
     0.900   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT3
     0.950   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 4 ]
     1.000   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT7
     1.050   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 4 ]  --  NEW RESULT [ 8 ]
     1.100   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT4
     1.150   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 5 ]
     1.200   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT8
     1.200   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-FINALIZE
SPR.-->  CALCULE SLOT [ 2 ] FINALIZED  --  NEW RESULT [ 8 ]
     1.250   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL SUM-RULE8
     1.300   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-PROCESS-RESULT-SLOT2
     1.350   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-IDENTIFIER-SLOT1
     1.400   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-START
SPR.-->  CALCULE SLOT [ 1 ] (9 + 3) STARTED
     1.450   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT9
     1.500   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 0 ]  --  NEW RESULT [ 10 ]
     1.550   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT0
     1.600   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 1 ]
     1.650   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT10
     1.700   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 1 ]  --  NEW RESULT [ 11 ]
     1.750   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT1
     1.800   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 2 ]
```

```
     1.850   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT11
     1.900   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 2 ]  --  NEW RESULT [ 12 ]
     1.950   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT2
     2.000   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-COUNT
SPR.-->  NEW COUNT [ 3 ]
     2.050   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT12
     2.050   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-FINALIZE
SPR.-->  CALCULE SLOT [ 1 ] FINALIZED  --  NEW RESULT [ 12 ]
     2.100   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL SUM-RULE12
     2.150   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-PROCESS-RESULT-SLOT1
     2.200   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-FINALIZE
SPR.-->  calculate (93 + 35) FINALIZED -- RESULT [ 128 ]
     2.200   GOAL                   SET-BUFFER-CHUNK GOAL CALCULATE2_SUB0
     2.250   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-START
SPR.-->  NEW TASK ELEMENT SELECTED
SPR.-->  PROCESS:               COGNITIVE
SPR.-->  TASK NAME:             calculate2
SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE
SPR.-->  FORMULE:               calculate ( display1 operator display2 )
SPR.-->  calculate (94 - 23) STARTED
     2.300   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-IDENTIFIER-SLOT2
     2.350   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL SUB-RULE3_4
     2.400   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-VERIFY-SLOT2
     2.450   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-START
SPR.-->  CALCULE SLOT [ 2 ] (4 - 3) STARTED
     2.500   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT4-DESC
     2.550   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 0 ]  --  NEW RESULT [ 3 ]
     2.600   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT0
     2.650   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-COUNT
SPR.-->  NEW COUNT [ 1 ]
     2.700   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT3-DESC
     2.750   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 1 ]  --  NEW RESULT [ 2 ]
     2.800   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT1
     2.850   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-COUNT
SPR.-->  NEW COUNT [ 2 ]
     2.900   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT2-DESC
     2.950   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 2 ]  --  NEW RESULT [ 1 ]
     3.000   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT2
     3.050   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-COUNT
SPR.-->  NEW COUNT [ 3 ]
     3.100   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT1-DESC
     3.100   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-FINALIZE
SPR.-->  CALCULE SLOT [ 2 ] FINALIZED  --  NEW RESULT [ 1 ]
     3.150   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-PROCESS-RESULT-SLOT2
     3.200   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-IDENTIFIER-SLOT1
     3.250   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL SUB-RULE2_9
     3.250   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-START
SPR.-->  CALCULE SLOT [ 1 ] (9 - 2) STARTED
     3.300   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT9-DESC
     3.350   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 0 ]  --  NEW RESULT [ 8 ]
     3.400   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT0
     3.450   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-COUNT
SPR.-->  NEW COUNT [ 1 ]
     3.500   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT8-DESC
     3.550   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-SLOT-INCREMENT
SPR.-->  INCLEMENT COUNT [ 1 ]  --  NEW RESULT [ 7 ]
     3.600   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT1
     3.650   PROCEDURAL              PRODUCTION-FIRED CALCUvents left to procesLATE2-SUB-SLOT-
COUNT
SPR.-->  NEW COUNT [ 2 ]
     3.700   DECLARATIVE            SET-BUFFER-CHUNK RETRIEVAL COUNT7-DESC
     3.700   PROCEDURAL              PRODUCTION-FIRED CALCUvents left to procesLATE2-SUB-SLOT-
FINALIZE
SPR.-->  CALCULE SLOT [ 1 ] FINALIZED  --  NEW RESULT [ 7 ]
     3.750   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-PROCESS-RESULT-SLOT1
     3.800   PROCEDURAL             PRODUCTION-FIRED CALCULATE2-SUB-FINALIZE
SPR.-->  calculate (94 - 23) FINALIZED -- RESULT [ 71 ]vents left to proces
     3.800   ------                 Stopped because no events left to process
```

APÊNDICE 4 – TRAÇOS DO ACT-R – REFERENTE À PRODUÇÃO DO APÊNDICE 2

```
; Loading C:\Documents and Settings\CORTEZTH\Desktop\ERRONEOUS.lisp

"SIMULATOR OF PERFORMANCE IN ERROR - S. PERERE [TASK KNOWLEDGE]"
"TRACE-DETAIL:      LOW"
Model Reloaded

#|## Model C:/Documents and Settings/CORTEZTH/Desktop/ERRONEOUS.lisp loaded. ##|#
0.000   GOAL                   SET-BUFFER-CHUNK GOAL CALCULATE1-SUM-GOAL REQUESTED NIL
     0.050   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-START
SPR.-->  NEW TASK ELEMENT SELECTED
SPR.-->  PROCESS:               COGNITIVE
SPR.-->  TASK NAME:             calculate1
SPR.-->  ELEMENTARY BEHAVIOUR:   CALCULATE
SPR.-->  FORMULE:               calculate ( display1 operator display2 )
SPR.-->  calculate (93 + 35) STARTED
     0.100   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-IDENTIFIER-SLOT2
     0.150   PROCEDURAL             PRODUCTION-FIRED CALCULATE1-SUM-SLOT-START
SPR.-->  SKILL-BASED -- OMISSION ERROR
     0.150   ------                 Stopped because no events left to process
```

ANEXO 1 – CLASSIFICAÇÃO DOS COMPORTAMENTOS ELEMENTARES

Comportamento Elementar	Processo	Nível de desempenho	Erro
Comparar	Cognitivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Deficiências na codificação
			Regra errada
			Sobrecarga
		Conhecimento	Seletividade
			Limitação da memória de trabalho
			Aprendizado deficiente

Tabela 15. Classificação do comportamento Comparar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Lembrar	Cognitivo	Habilidade	Redução de intencionalidade
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Deficiências na codificação
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 16. Classificação do comportamento Lembrar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Calcular	Cognitivo	Habilidade	Redução de intencionalidade
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Deficiências na codificação
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 17. Classificação do comportamento Calcular (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Decidir	Cognitivo	Habilidade	Redução de intencionalidade
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Deficiências na codificação
			Regra errada
			Sobrecarga
		Conhecimento	Seletividade
			Limitação da memória de trabalho
			Aprendizado deficiente

Tabela 18. Classificação do comportamento Decidir (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Escolher	Cognitivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Deficiências na codificação
			Regra errada
			Sobrecarga
		Conhecimento	Seletividade
			Limitação da memória de trabalho
			Aprendizado deficiente

Tabela 19. Classificação do comportamento Escolher (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Verificar	Cognitivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Deficiências na codificação
			Regra errada
			Sobrecarga
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 20. Classificação do comportamento Verificar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Interpolar	Cognitivo	Habilidade	Redução de intencionalidade
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Deficiências na codificação
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 21. Classificação do comportamento Interpolar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Monitorar	Perceptivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 22. Classificação do comportamento Monitorar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Observar	Perceptivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 23. Classificação do comportamento Observar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Localizar	Perceptivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 24. Classificação do comportamento Localizar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Explorar	Perceptivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 25. Classificação do comportamento Explorar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Ler	Perceptivo	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Percepção falsa
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Redundância
			Deficiências na codificação
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 26. Classificação do comportamento Ler (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Teclar	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 27. Classificação do comportamento Teclar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Clicar	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 28. Classificação do comportamento Clicar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Posicionar	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 29. Classificação do comportamento Posicionar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Mover	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 30. Classificação do comportamento Mover (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Ajustar	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 31. Classificação do comportamento Ajustar (Begosso, 2005)

Comportamento Elementar	Processo	Nível de desempenho	Erro
Instalar	Motor	Habilidade	Redução de intencionalidade
			Confusão perceptiva
			Sobrecarga motora
			Omissão
			Repetição
			Inversão
		Regra	Força de regra
			Regra errada
		Conhecimento	Seletividade
			Aprendizado deficiente

Tabela 32. Classificação do comportamento Instalar (Begosso, 2005)

7. REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSON, J.R. **Rules of the mind**. Hillsdale: Lea, 1993. 320 p.

ANDERSON, J. R.; LEBIERE, C. **The Atomic Components of Thought**. Mahwah: Lea, 1998. 490 p.

BEGOSSO, L. C. **S. PERERE:** Simulator of Performance on Error. Tese de outorado. São Paulo: USP, 2005.

BEGOSSO, L. C., FILGUEIRAS, L. V. L. Human Error Simulation as an Aid to HCI.

Dan Bothell. **ACT-R 6.0 Reference Manual,** Working Draft. USA, 2007.

Design for Critical Systems. **Simpósio sobre Fatores Humanos em Sistemas Computacionais**, Natal - RN, 2006.

BERLINER, D. C.; ANGELL, D.; SHEARER, J. Behaviors, measures and instruments for performance evaluation in simulated environments. In: **SYMPOSIUM AND WORKSHOP ON THE QUANTIFICATION OF HUMAN PERFORMANCE**, 1., Albuquerque-New Mexico, 1964. Anais. Albuquerque: The University of New Mexico, 1964. p.277-296.

BOOCH, G. RUNBAUGH, J. JACOBSON, I. **UML – Guia do Usuário**, Rio de Janeiro, Campus, 2000.

BUDIU, R. Pittsburgh. **Página do Grupo de Pesquisa da Arquitetura Cognitiva ACT-R**. Disponível em: <http://act-r.psy.cmu.edu>. Acesso em: 25 de abr. 2008.

BYRNE, M. D.; ANDERSON, J. R. Enhancing ACT-R's Perceptual-Motor Abilities. In: **Annual Conference of the Cognitive Science Society**, 19., Palo Alto, 1997. Proceedings Nineteenth Annual Conference of the Cognitive Science society. Palo Alto: 1997. p.116.

CORTEZ, T.H. – **Uma proposta de novas funcionalidades para o simulador do desempenho humano S. PERERE**. *Programa de Iniciação Científica*. Assis - SP: FEMA – Fundação Educacional do Município de Assis, 2007.

CORTEZ, T.H., BEGOSSO, L. C, *Simulador do Desempenho Humano S. PERERE,* **Anais Completos do IHC 2008** – 21-24 de outubro, Porto Alegre, Brasil.

CTTE, ***Su*pport for Developing and Analysing Task Models for Interactive System Design**. <http://giove.isti.cnr.it/ctte.html> Acesso em: 15 de mai. 2008.

FILGUEIRAS, L.V.L. **APIS**: Método para Análise e Projeto de Interfaces Homem-Computador visando confiabilidade humana. Tese de doutorado. São Paulo: Usp, 1996.

FILGUEIRAS, L.V.L., VITTI, L.R., *Modelagem de Tarefas para Simulação do Desempenho Humano em Erro,* **Anais Estendidos do IHC 2006** – 19-22 de novembro, Natal - RN, Brasil.

GAMMA, E.; HELM, R.; JOHNSON, R. et al.; 2000. Padrões de Projeto : **Soluções Reutilizáveis de Software Orientado a Objetos.** Porto Alegre : Bookman.

LEIDEN, K. et al. **A review of human performance models for the prediction of human error**. Moffett Field: National Aeronautics and Space Administration System-Wide Accident Prevention Program Ames Research Center, 2001. Disponível em <http://human-factors.arc.nasa.gov/ihi/hcsl/HPM_pubs/HumanErrorModels.pdf> Acesso em: 26 de abr. 2008.

MASSON, M.; KONING, Y. How to manage human error in aviation maintenance? The example of a JAR66-HF education and training programme. **Cognition, Technology & Work**, n.3, p.189-204, 2001

CTTE: **support for developing and analyzing task models for interactive system design** Mori, G.; Paterno, F.; Santoro, C. Software Engineering, IEEE Transactions on Volume 28, Issue 8, Aug 2002 Page(s): 797 - 813

RASMUSSEN, J. Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and other distinctions in human performance models. **IEEE: Transactions on Systems, Man, and Cybernetics**, v.13, n.3, p.257-266, 1983.

REASON, J. **Human Error**. Cambridge: Cambridge University Press, 1990.

REASON, J.; MADDOX, M.E. **Human Error**. <http://hfskyway.faa.gov/HFAMI.> Acesso em: 02 de abr. 2008.

RITTER, F.E. et al. **Techniques for Modeling Human Performance in Synthetic Environments**: a supplementary review. Ft. Belvoir: Defense Technical Information Center, 2002. Disponível em: <http://iac.dtic.mil/hsiac>. Acesso em: 02 de mai. 2008.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W. et al.; 1994. **Modelagem e Projetos** Baseados em Objetos. Rio de Janeiro : Campus.

TURBAN, E. **Knowledge acquisition and validation. In: Turban, E. Expert Systems and Applied Artificial Intelligence.** New York: Macmillan Publishing, 1992. p.117-166.

ZHU, J.; JOSSMAN, P.; 1999**. Application of Design Patterns for Object-Oriented Modeling of Power Systems.** IEEE Transactions on Power Systems, New York, v. 14, n. 2 (May), p. 532-537.

Wickens, C. D. **Engineering Psychology and Human Performance.** New York: HarperCollins, 1992.