LUIZ CARLOS BEGOSSO

**S. PERERE - Uma Ferramenta Apoiada por Arquiteturas Cognitivas para o
Estudo da Confiabilidade Humana**

Texto apresentado à Escola Politécnica
da Universidade de São Paulo para
obtenção do Título de Doutor em
Engenharia Elétrica.

São Paulo
2005

LUIZ CARLOS BEGOSSO

**S. PERERE - Uma Ferramenta Apoiada por Arquiteturas Cognitivas para o
Estudo da Confiabilidade Humana**

Texto apresentado à Escola Politécnica
da Universidade de São Paulo para
obtenção do Título de Doutor em
Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais.

Orientadora:
Prof^a Dr^a Lucia Vilela Leite Filgueiras

São Paulo
2005

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 30 de novembro de 2005.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Begosso, Luiz Carlos

**S. PERERE – uma ferramenta apoiada por arquiteturas cognitivas para o estudo da confiabilidade humana / L.C. Begosso. - ed.rev. -- São Paulo, 2005.
p. 203**

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Interface homem-computador I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

À minha esposa Izabel e meus filhos Rafael e Renata

AGRADECIMENTOS

A Deus, por permitir o cumprimento desta etapa em minha vida.

Agradeço à Profª Drª Lucia Vilela Leite Filgueiras, pela orientação, motivação e paciência, que certamente foram necessárias para a conclusão deste trabalho.

À Fundação Educacional do Município de Assis, pelo apoio e incentivo.

Ao meu aluno e amigo Fábio Luís Fujii Silveira - Chumeu, pela valiosa ajuda na construção do simulador.

Aos meus amigos professores e funcionários da FEMA, pelo incentivo e pelas preciosas contribuições.

Aos meus pais, Luiz e Nézia pela constante motivação aos estudos, desde a infância.

Ao Sr. Galiano e D. Jandira pelo apoio.

Ao meu irmão Nino, pelas horas dedicadas para a revisão deste trabalho.

Em especial à minha esposa Izabel, companheira e amiga,
e aos meus filhos Rafael e Renata pela compreensão, apoio e motivação.

RESUMO

A construção de modelos representa uma perspectiva interessante para buscar a compreensão sobre aspectos do comportamento humano. A partir de modelos do desempenho humano e do estudo das características do erro humano, este trabalho propõe o S. PERERE (Simulation of Performance in Error), um simulador computacional do comportamento humano cujo objetivo é produzir de forma aleatória estados de erro humano. O simulador recebe como entrada a especificação do comportamento a ser simulado e produz como saída o comportamento afetado de erro. Para a geração dos estados de erro o S. PERERE possui um mecanismo perturbador do comportamento e também um mecanismo de disparo das perturbações. A construção do simulador está baseada na arquitetura cognitiva ACT-R (Atomic Components of Thought – Rational). A utilização prevista do S. PERERE é seu acoplamento com simuladores de processos de forma a permitir a verificação do impacto dos erros humanos nestes processos.

ABSTRACT

The construction of models represents an interesting perspective to reach the understanding about human behaviour aspects. From human performance models and from the study of error characteristics, this work purposes the S.PERERE (Simulation of Performance in Error), a human behaviour computational simulator, which main objective is to produce, in a random way, human error states. The simulator receives, as entrance, the specification of the behaviour to be simulated, and produces, as output, the behaviour affected by the error. For the generation of error states, S.PERERE has a behaviour disturber mechanism and also a mechanism to start the perturbations. The construction of the simulator is based on ACT-R (Atomic Components of Thought – Rational) cognitive architecture. The use of S.PERERE is under its coupling to process simulators, allowing the verification of human errors impact on theses processes.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1. INTRODUÇÃO.....	1
1.1. Objetivos.....	1
1.2. Motivação.....	6
1.3. Trabalhos correlatos.....	8
1.4. Estrutura do trabalho.....	11
 2. MODELOS DE DESEMPENHO HUMANO E TÉCNICAS DE SIMULAÇÃO.....	 13
2.1. Modelos.....	13
2.2. Definições.....	15
2.3. Modelos cognitivos.....	15
2.4. Arquiteturas cognitivas.....	16
2.4.1. COGNET – Cognition as Network of Tasks.....	18
2.4.2. EPIC – Executive-Processes Interactive Control.....	20
2.4.2.1. GOMS.....	22
2.4.3. ACT-R.....	26
2.4.4. Soar.....	28
2.5. Comparação entre as arquiteturas cognitivas.....	30
2.6. A escolha da arquitetura cognitiva.....	30
 3. ERRO HUMANO.....	 32
3.1. Definições do erro humano.....	32
3.2. Modelo de Rasmussen.....	35
3.2.1. Intencionalidade.....	37
3.3. Classificação do erro humano.....	39
3.4. Taxonomia do erro humano.....	42

3.4.1. Desempenho no nível de habilidade.....	42
3.4.2. Desempenho no nível de regra.....	45
3.4.3. Desempenho no nível de conhecimento.....	48
3.5. Considerações finais.....	51
 4. PROPOSTA DO S. PERERE – SIMULATION OF PERFORMANCE IN ERROR	 53
4.1. Objetivos do S. PERERE.....	53
4.2. Proposta do S. PERERE.....	54
4.2.1. Unidades de comportamento.....	56
4.2.2. Estrutura do S. PERERE.....	59
4.2.2.1. Memória declarativa.....	60
4.2.2.2. Memória procedural.....	61
4.2.2.3. Módulo motor e perceptivo.....	62
4.2.2.4. Módulo pré-processador.....	62
4.2.2.5. Módulo disparador.....	63
4.2.2.6. Módulo perturbador.....	65
4.3. Especificações das perturbações.....	65
4.4. Considerações finais.....	71
 5. IMPLEMENTAÇÃO DO S. PERERE UTILIZANDO O ACT-R.....	 73
5.1. Detalhamento da arquitetura do ACT-R.....	73
5.2. Os componentes do ACT-R.....	77
5.2.1. Memória declarativa.....	77
5.2.2. Memória procedural.....	78
5.2.3. Módulo perceptivo.....	79
5.2.4. Módulo motor.....	82
5.3. Implementação do S. PERERE sobre o ACT-R.....	83
5.3.1. Especificação dos comportamentos elementares do processo cognitivo.....	85

5.3.2. Especificação dos comportamentos elementares do processo perceptivo.....	93
5.3.3. Especificação dos comportamentos elementares do processo motor.....	102
5.3.4. Implementação do disparador.....	110
5.4. Considerações finais.....	111
 6. EXEMPLO DE APLICAÇÃO DO S. PERERE NO ALINHAMENTO DE ROTAS NO METRÔ DE SÃO PAULO.....	 112
6.1. A tarefa-exemplo: alinhamento de rota no Metrô-SP.....	112
6.2. Descrição do procedimento operacional real do alinhamento de rotas.....	113
6.3. Simulação do operador alinhando a rota Linha 1 – Azul – caso correto.....	116
6.3.1. Construção da memória declarativa para os objetos.	117
6.3.2. Construção da memória declarativa para a tarefa – caso correto.....	119
6.3.3. Execução do modelo.....	120
6.3.4. Resultados da execução do caso correto.....	122
6.4. Resultados da execução do caso perturbado.....	133
6.4.1. Perturbação 1: omissão em verificação.....	133
6.4.2. Perturbação 2: omissão em teclar.....	134
6.4.3. Perturbação 3: omissão em monitorar.....	135
6.4.4. Perturbação 4: repetição em verificar.....	136
6.4.5. Perturbação 5: repetição em teclar.....	137
6.4.6. Perturbação 6: repetição em monitorar.....	138
6.4.7. Perturbação 7: confusão em teclar.....	138
6.4.8. Perturbação 8: engano em teclar.....	139
6.5. Discussão dos resultados.....	141

7. CONCLUSÕES.....	143
--------------------	-----

ANEXO A – Código ACT-R para cada comportamento elementar.....	148
--	-----

ANEXO B – Comportamentos elementares associados a erros....	185
---	-----

Referências bibliográficas.....	198
---------------------------------	-----

LISTA DE FIGURAS

Figura 1 – Modelo geral do desempenho humano.....	16
Figura 2 - Arquitetura Cognitiva genérica.....	17
Figura 3 – Encadeamento entre os Conceitos de Falha, Erro e Disfunção.....	34
Figura 4 – Significado do termo “erro”.....	35
Figura 5 – Fluxo de informação do S. PERERE	54
Figura 6 – Estrutura do S. PERERE.....	60
Figura 7 – Lógica de uso da arquitetura cognitiva ACT-R.....	74
Figura 8 – Organização da informação no ACT-R.....	75
Figura 9 – Estrutura de suporte para o S. PERERE.....	84
Figura 10 – Cabeçalho para o preenchimento de parâmetros.....	117
Figura 11 – Interface do S. PERERE.....	121
Figura 12 – Interface do ACT-R.....	122

LISTA DE TABELAS

Tabela 1	Características comparativas entre as Arquiteturas Cognitivas.....	30
Tabela 2	Manifestações de Deslizes.....	39
Tabela 3	Erros em relação à intenção e ao estágio cognitivo	41
Tabela 4	Modos de falha de acordo com os três níveis de desempenho.....	43
Tabela 5	Taxonomia de Berliner, Angell e Shearer – Processo Cognitivo ...	57
Tabela 6	Taxonomia de Berliner, Angell e Shearer – Processo Perceptivo...	58
Tabela 7	Taxonomia de Berliner, Angell e Shearer – Processo Motor	58
Tabela 8	Taxonomia de Berliner, Angell e Shearer – Processo Comunicação	59
Tabela 9	Modos de falha especificados para o S. PERERE.....	67
Tabela 10	Caso normal para o estado inicial do alinhamento de rotas.....	114
Tabela 11	Descrição dos procedimentos para o alinhamento de rota – Linha 1 – Azul.....	116

LISTA DE ABREVIATURAS E SIGLAS

ACT-R	Atomic Components of Thought – Rational
COGNET	Cognition as a Network of Tasks
CPU	Central Processing Unit
EPIC	Executive-Processes Interactive Control
GOMS	Goals, Operators, Methods and Selection Rules
IHC	Interação Homem Computador
RTMT	Real time multi tasking
S.PERERE	Simulation of Performance in Error

1. INTRODUÇÃO

O objetivo deste trabalho é a definição e o desenvolvimento de um simulador da ação humana que leve em consideração o erro, na interação homem-computador.

O simulador da ação humana proposto neste trabalho, denominado S. PERERE (**S**imulation of **P**erformance in **E**rror), parte da especificação de um comportamento humano e produz este comportamento afetado de erro. A viabilidade da proposta do simulador é mostrada pela implementação de um sistema computacional apoiado em uma arquitetura cognitiva denominada ACT-R.

As seções seguintes descrevem em maior detalhe os objetivos da pesquisa, a motivação do autor em desenvolvê-la, suas contribuições e a estrutura deste trabalho.

1.1. Objetivos

O erro humano pode ser considerado como causa ou fator contribuinte para uma série de incidentes e acidentes industriais, de acordo com Reason (1990): Bhopal em 1984, Chernobyl e Challenger em 1986 são exemplos clássicos.

Erros humanos que causaram tais acidentes são entendidos como combinações de vários fatores, entre eles, as limitações do processamento humano de informação e as características de projeto que necessitam de integração com habilidades humanas. O nível de automação faz aumentar o grau de complexidade dos sistemas e erros cometidos por operadores em ambientes complexos tendem a resultar em consequências mais graves, de acordo com Reason; Maddox (2003).

Observa-se, de modo geral, que as pessoas sempre cometem erros. O erro é uma condição normal do comportamento humano e que às vezes, tais situações possuem até mesmo um papel positivo (MASSON; KONING, 2001):

- Ajudam as pessoas a se adaptarem às particularidades situacionais de uma tarefa;
- Ajudam as pessoas a aprenderem mais sobre seu desempenho em relação à tarefa.

Na grande maioria das situações, portanto, o próprio indivíduo que cometeu o erro aprende e corrige sua atitude. Os problemas acontecem quando os erros não são detectados e passam por todas as barreiras construídas pelos projetistas. Ao se

aliarem a condições especiais do ambiente ou do processo crítico, esses erros contribuem para eventos indesejáveis.

Joshi; Kaufman; Giras (2001) apresentaram um trabalho sobre sistema de controle de trens e destacaram que análises pós-acidente demonstram que o erro humano tem um papel significativo na contribuição de acidentes. Assim o comportamento humano que desempenha um papel importante sobre a segurança de sistema deve ser estudado em profundidade.

Por conta disto, algumas iniciativas de reduzir o erro humano têm sido exatamente a melhoria do projeto da interação homem-computador. Os projetistas desenvolvem mecanismos de proteção e recuperação do erro para as tarefas humanas, sejam motoras, perceptivas ou de tomada de decisão. Por exemplo, em alguns casos estes mecanismos de proteção podem ser desde protetores de botões (onde o operador deve desobstruir o botão antes de acioná-lo, evitando o acionamento acidental de uma função crítica) até sistemas de apoio às decisões críticas do usuário. Naturalmente, estes mesmos mecanismos de proteção podem aumentar a complexidade da interação homem-computador. Assim, a engenharia obteve sucesso apenas parcial na tentativa de reduzir a probabilidade de erro humano, mesmo porque a complexidade do comportamento cognitivo leva a múltiplas possibilidades de erro humano e nem sempre se podem desenvolver todos os mecanismos necessários ou se conhecem todas as possibilidades.

Testes simples de interfaces são insuficientes para descobrir a grande variedade, complexidade e o momento exato em que ocorrem os erros. Infelizmente, os usuários descobrirão estes erros interagindo com a interface. Assim, um dos objetivos desta tese é explorar a diversidade do erro humano em situações de interação com sistemas críticos.

Estudiosos como Hollan; Hutchins; Kirsh (2000) acreditam que o indivíduo não deva ser tratado isoladamente do mundo em que vive. Ele, o indivíduo, faz parte de um contexto social que desenvolve relações entre processos internos e externos de comportamento. Tais relações envolvem a coordenação de muitas escalas de tempo diferentes entre recursos internos - memória, atenção, execução – e recursos externos – objetos, artefatos, e materiais manuais que completam o cotidiano.

Um exemplo da importância da compreensão do contexto ocorre na investigação de acidentes aeronáuticos. Os peritos sabem que as tarefas-padrão foram executadas inúmeras vezes pela tripulação, em cada etapa do voo. Em geral, a tripulação é bem treinada. Assim, eles devem procurar por dados do acidente que revelem a natureza dos eventos que desencadearam o acidente. Se houve erro humano, é necessário compreender a exata relação entre o fato e o contexto – interno e externo - para determinar as modificações necessárias ao projeto daquele sistema.

Diversos estudos, entre eles os de Norman (1983) e Reason (1990) apontam para a necessidade de entender, classificar e prever o erro nas suas mais diversas formas e combinações, na tentativa de minimizar seus efeitos que, quando sobre sistemas críticos, podem causar perdas irreparáveis, de vidas humanas e de bens materiais. Assim, outro objetivo desta tese é tratar o erro humano como uma expressão da variabilidade humana relacionada ao contexto interno e externo.

O foco da ação humana, neste trabalho, será a operação de sistemas computacionais. Nesta situação, segundo Norman; Draper (1986), o erro pode ser entendido como um desvio de uma atividade humana na interação com software ou hardware que conduz negativamente a uma falha de operação.

O produto final deste trabalho, o simulador S. PERERE, desenvolve uma relação do homem, ainda que simulado, e o computador. Autores como Alart et al. (1998) descreveram a cooperação homem-computador-máquina em três níveis.

O primeiro nível denominado *co-atividade*, apresenta um simples relacionamento onde homem e máquina não tenham necessariamente que interagir, porém, a realização das tarefas de ambos pode interferir positiva ou negativamente nesta relação. Por exemplo, considere um usuário e uma máquina utilizando recursos da CPU. Ao iniciar uma computação “pesada” o usuário ou a máquina monopolizarão tais recursos computacionais interferindo de forma negativa nesta relação.

O segundo nível, *colaboração*, é uma forma de operação na qual o usuário e a máquina desenvolvem uma certa contribuição entre si, a fim de alcançarem objetivos específicos de suas naturezas. Por exemplo, a máquina pode realizar tarefas do usuário de forma a auxiliá-lo no diagnóstico ou recuperação de dados.

O terceiro e último nível, *cooperação*, é uma forma de interação na qual o usuário e a máquina são encarregados de atingirem uma meta comum e que nem um e nem o outro podem solucionar sozinhos. O diálogo entre o usuário e a máquina deve ser parte de um esforço conjunto, com objetivos claramente definidos e compartilhados por ambos os componentes. Desta forma, a cooperação existente nesse nível representa uma evolução da interação homem-máquina e determina uma relação funcional entre os agentes como membros de uma mesma equipe.

Os dois últimos níveis de cooperação refletem os estágios de possíveis utilizações esperadas na relação homem-computador. Espera-se que este trabalho possa servir de base para incrementar esta relação.

A tecnologia tem demonstrado estar preparada para a construção de sistemas que interajam com usuários segundo os critérios estabelecidos nos três níveis de cooperação. Os estudos para projetos de sistemas tolerantes ao erro serão incrementados na medida em que houver o conhecimento de quando e porque as limitações, no processo cognitivo, ocorrem. Os problemas surgem na medida em que nem sempre é possível prever, no projeto, todas as situações que possam ser geradas durante a interação homem-computador.

Neste contexto, torna-se imprescindível atentar de forma sistemática para a ação humana, em situações de risco e que envolvam a relação homem-computador.

O objetivo desta pesquisa é produzir um simulador da ação humana que leve em consideração o erro, para ser aplicado ao estudo da confiabilidade humana na operação de sistemas computacionais.

O simulador da ação humana que é proposto neste trabalho reflete na construção de um sistema computacional, denominado S. PERERE (**S**imulation of **P**erformance in **E**rror), que simula o desempenho humano afetado de erros. Para a construção do S. PERERE pretende-se utilizar o suporte computacional de modelos cognitivos existentes na literatura.

Este simulador poderá contribuir com estudo de conseqüências importantes para a confiabilidade operacional, auxiliando o projetista de IHC a minimizar as possibilidades de ocorrência de erros durante a interação.

Acredita-se que o simulador do comportamento humano que inclui comportamentos errôneos tem potencial para contribuir com os estudos do erro humano em três situações:

- Primeiro, o simulador pode ser acoplado a um simulador de processos (por exemplo, de uma planta química ou de uma aeronave) para fazer o papel de operador deste processo. Programado para executar repetida e aleatoriamente tarefas humanas afetadas de erro em tempo acelerado ao invés do tempo real, o simulador pode produzir situações indesejáveis ao processo. O acompanhamento retroativo desta situação pode apontar para cadeias de eventos envolvendo a operação humana, bastante úteis para a análise da confiabilidade humana em aplicações críticas. Em geral, estes estudos são limitados devido à dificuldade de se considerarem falhas múltiplas. Um simulador executando um grande número de combinações em tempo acelerado, por exemplo, pode representar uma importante ferramenta para o processo de estudo da confiabilidade humana.
- Segundo, em treinamento da operação de sistemas críticos, principalmente naqueles em que vários operadores realizam papéis colaborativos sobre o sistema automatizado e existe a necessidade de supervisão, o simulador substitui um (ou mais) dos colaboradores, provocando situações errôneas ao parceiro. Neste caso, o simulador pode contribuir para que seja possível avaliar a resposta de sistemas que deveriam ser tolerantes ao erro humano e dos próprios treinandos. Por exemplo, durante o treinamento de um controlador de tráfego aéreo, o simulador (desempenhando o papel do piloto mal-comportado) interage com o controlador que está sendo treinado, cometendo vários erros ao cumprir as tarefas especificadas, com o propósito de testar a capacidade do controlador de detectar e reagir ao erro humano;
- Finalmente, a observação do resultado do simulador, listando as inúmeras possibilidades de erros humanos associadas a uma dada tarefa, pode contribuir para os projetos de IHC, na medida em que um analista avaliar cada possível situação e projetar o sistema interativo para permitir maior

observabilidade das situações de erro e maior recuperabilidade destes erros na interface estudada.

Neste trabalho, o S. PERERE é usado na última situação, ou seja, lista as possíveis ocorrências de erros associados à realização de uma tarefa. Esta forma foi a selecionada para que o autor possa ter controle sobre todas as possibilidades de erro em uma dada tarefa.

Um simulador do desempenho humano deve possuir mecanismos para simular diversos tipos de comportamento: deve poder representar conhecimento, deve poder realizar tarefas, deve poder perceber o ambiente e atualizar sua consciência sobre este contexto (consciência situacional), deve poder agir sobre o ambiente. Para o atendimento deste objetivo, torna-se necessário utilizar uma arquitetura cognitiva que produza unidades de comportamentos elementares e que possa ser integrada ao S. PERERE. As arquiteturas cognitivas são implementações computacionais de aspectos inerentes aos sistemas cognitivos, perceptivos e motores do ser humano.

Assim, o esforço do desenvolvimento do S.PERERE e o núcleo da contribuição deste trabalho ficam concentrados na simulação do comportamento associado ao conceito de erro.

1.2. Motivação

Apesar dos avanços nas áreas de desenvolvimento de sistemas e de hardware, tem-se notado que apenas recentemente surgiram preocupações com o aparato cognitivo do operador humano na interação com o software e com o hardware.

Putzer; Onken (2003) alertam para o fato de que a contínua introdução de funções automáticas em aeronaves, por exemplo, resulta no aumento da complexidade do ambiente que por natureza já é complexo. Analistas de segurança aeronáuticos têm relatado que, em alguns casos, a automação apresenta-se como fator de diminuição da segurança do avião.

Nesta mesma linha de preocupação, LaSala (2002) e Cacciabue (1997), indicam que o elemento humano deveria ser considerado desde o princípio do projeto e continuamente através das fases de operação e manutenção. Para estes autores, a equipe responsável pelo desenvolvimento do sistema deveria ser composta de

pessoal com experiência em engenharia orientada a fatores humanos e engenheiros de hardware e software. Desta forma os projetos seriam concebidos com o foco no usuário do produto, minimizando problemas que advêm da interação com a máquina.

Conforme ressaltam Reason; Maddox (2003), o aumento na proporção de incidentes e acidentes tributados ao erro humano, no caso específico da aviação, quando comparados aos causados por falhas em outros componentes do sistema pode ser atribuído a duas tendências. Elas refletem de certa forma, a motivação para o desenvolvimento deste trabalho. A primeira tendência é que, de modo geral, os componentes do sistema aeronáuticos, por exemplo, componentes hidráulicos, elétricos, de comunicação, etc., têm se tornado ao mesmo tempo mais sofisticados e seguros, o que aumenta a confiabilidade do equipamento. A segunda é que existem projetistas, engenheiros e executivos com poder de tomada de decisões, etc. que criam condições para promoção de erros tanto no solo quanto no ar. A combinação dessas tendências resulta num menor número de falhas nos equipamentos e, por outro lado, num aumento nos relatos de erros humanos provenientes da interação com tais equipamentos. A situação exposta para o caso da área aeronáutica se reproduz em outras áreas onde há sistemas críticos: nas áreas industriais, de transporte e crescentemente, na área médica (JOHNSON, 2002).

Pode-se observar que existe uma relação alarmante entre o número de falhas e a proporção contribuinte do erro humano. Entretanto, autoridades e cientistas estão debruçados em projetos que procuram melhorar a interação homem-computador minimizando assim os fatores contribuintes para graves acidentes aéreos (REASON; MADDUX, 2003).

A maneira pela qual este trabalho busca contribuir para oferecer possibilidades de minimização dos problemas na relação usuário-computador é no aumento da capacidade de prever erros humanos.

Nessa linha de pensamento, Reason (1990) destaca a necessidade de direcionar esforços e reflexões na tentativa de construir uma teoria de predição do erro que leve em consideração os três maiores elementos contribuintes na produção de um erro: a natureza da tarefa e as circunstâncias ambientais, natureza do indivíduo e os mecanismos que governam seu desempenho.

O comportamento humano apresenta facetas distintas de acordo com o contexto. As pessoas se comportam diferentemente em diferentes circunstâncias no mesmo ambiente (NIELSEN, 1996). Hollnagel (1993) afirma que os comportamentos inseguros contribuem direta e indiretamente em 90% dos acidentes e incidentes no local de trabalho. Estas diferenças de comportamento devem ser, na medida do possível, previstas na interação do operador com o sistema automático.

Motivado por este desafio de contribuir para a área da segurança e confiabilidade de sistemas críticos, é que se propõe a desenvolver este trabalho. Para isso, parte-se das seguintes premissas:

- É possível definir um conjunto de tarefas elementares que representam a interação homem-computador.
- Existe uma teoria do erro humano que estabeleça tipos de erros humanos em função das ações humanas.

A partir dessas premissas, o objetivo principal, conforme descrito anteriormente, é o de produzir um simulador da ação humana que leve em consideração o erro, na interação homem-computador.

Na próxima seção, serão descritos alguns trabalhos encontrados na literatura que possuem alguma relação com esta pesquisa.

1.3. Trabalhos correlatos

O desenvolvimento desse trabalho, neste momento, tenciona considerar o estado da arte das discussões (PEW; MAVOR, 1998); (RITTER et al., 2002), sobre a complexidade do comportamento humano focalizando a construção de modelos que contribuam para a predição de tais comportamentos errôneos do usuário na interação com sistemas automatizados.

A contribuição do trabalho é a construção de um simulador de comportamento humano que possa ser implementado e que gere erros sobre o comportamento simulado. A saída do simulador, afetada de erros, subsidiará a tomada de decisão do analista de confiabilidade ou de IHC sobre o encaminhamento a ser realizado após a geração dos comportamentos errôneos. Ressalta-se que não faz parte do escopo desta pesquisa aprofundar os estudos nas consequências dos erros

simulados para o sistema, mas sim ser um apoio a estes estudos. Este apoio é encontrado nas palavras de Norman (1983):

“The analysis of errors provide one set of considerations for the construction of a system that might minimize errors. There are several others factors that should be considered as well. First, people will make errors even in the best designed systems, even with the best of training and motivations. So, a corollary of the attempt to minimize errors is that one should try to minimize the effect of an error... Actions that can lead to difficulty should be difficult to do, perhaps requiring a set of steps, as in the required release of ‘safeties’ when the pilot wishes to eject from a military airplane, or at least, requiring a confirmation, as when requesting that all files on a computer directory be destroyed.” (NORMAN, 1983, p.257)

Uma atividade de pesquisa deve se calcar em trabalhos anteriores. Existem, na literatura, muitas pesquisas relacionadas à simulação computacional do desempenho humano e diversas pesquisas relacionadas com o erro humano. Esta seção tem por objetivo apresentar recentes estudos nesta área e mostrar em que essas pesquisas se assemelham e em que elas diferem do trabalho aqui realizado.

O trabalho mais próximo da proposta do S.PERERE encontrado na literatura é o simulador cognitivo de Itoh et al. (2001). Estes autores desenvolveram um simulador de desempenho do indivíduo responsável pelas tarefas de manobra de um navio. Para a obtenção da tarefa a simular, todo processo de construção para o simulador está baseado na análise de tarefas obtida em sessões de simulação de manobras de navios.

Segundo os autores o trabalho objetiva contribuir com a abordagem de simulação de modelos cognitivos para análise de risco na área de navegação marítima.

O *simulador cognitivo*, como foi denominado pelos autores, gera o desempenho cognitivo do navegador simulando erros humanos de observação/identificação de situações durante o curso de navegação e a tomada de decisões. Tais erros estão associados aos processos de percepção e cognição do navegador com pouca experiência na tarefa de manobrar o navio.

De acordo com Itoh et al. (2001), o *simulador cognitivo* produziu desempenho similar àqueles realizados por navegadores humanos. A similaridade entre os desempenhos citados pode considerar o *simulador cognitivo* como

contribuinte, na análise de risco, para a descrição e previsão do desempenho cognitivo do navegador.

A principal diferença entre o *simulador cognitivo* de Itoh e o S. PERERE é a capacidade deste último de generalizar o domínio de aplicação fornecendo saídas em erro a partir de uma tarefa genérica especificada como entrada. O simulador de Itoh concentra-se em simular tarefas de domínio específico sem a perspectiva de trocar o domínio. Outro aspecto importante que diferencia os dois simuladores é que o S. PERERE objetiva simular também o processo motor além dos processos perceptivo e cognitivo simulado por Itoh.

Em outra pesquisa recente e relacionada a este trabalho, Wood (2000) reformulou um modelo de desempenho humano bastante conhecido, o GOMS¹, para acrescentar a ele características de erro e recuperação, criando o EGOMS (Extending GOMS).

De acordo com o autor, o EGOMS representa três estados para o operador humano: o Normal (*Normal*), o Intermediário (*Quasi-Normal*) e o estado de Recuperação (*Recovery*).

Um indivíduo inicia a execução de sua tarefa no estado Normal. A qualquer tempo ele pode deixar de realizar a tarefa normal. Se não percebeu a mudança, passa para o estado Intermediário. A partir do instante que o usuário identifica o erro, o estado atual passa a ser o de Recuperação. Após a correção do erro, a execução da tarefa retorna ao estado Normal.

O propósito do EGOMS é reproduzir o tempo de resposta humano (desempenho) na execução de tarefas, considerando que erros surgirão durante a realização da tarefa e estes podem ser recuperados.

O trabalho de Wood considera imprescindível determinar os possíveis pontos de erro numa tarefa específica. O erro será recuperado apenas se existir um método para sua recuperação e naquele ponto onde ocorreu. O S. PERERE, diferentemente do EGOMS, objetiva gerar estados de erro, coerentes com a tarefa simulada, sem a preocupação de quantificar o desempenho, como elemento contribuinte para o desenvolvimento de projetos tolerantes ao erro humano.

¹ GOMS, acrônimo de Goals (Metas), Operators (Operadores), Methods (Métodos) e Selection Rules (Regras de Seleção) será estudado na Seção 2.4.2.1

Um outro trabalho importante na área da simulação do desempenho humano é Crew Activity Tracking System – CATS, desenvolvido por Todd J. Callantine (CALLANTINE, 2001) e (CALLANTINE, 2003). CATS é um sistema computacional que utiliza um modelo das atividades do operador para: prever as atividades do operador e interpretar suas ações. Encontram-se aplicações do CATS na área de controle de tráfego aéreo.

O sistema CATS pode ser resumidamente, descrito da seguinte maneira: ele recebe informação sobre o estado atual do sistema no qual está conectado, restrições de operação e as ações dos operadores que controlam o sistema. CATS utiliza um modelo de atividades de operadores, baseado no Modelo de Função Operacional (OFM) para prever as atividades que os operadores deveriam estar realizando num dado momento. Quando a ação dos operadores é detectada pelo CATS, ele interpreta tais ações combinando com seus métodos de predição de atividades de desempenho. Se as ações não combinarem com as atividades previstas pelo CATS ele as interpretará como erro de omissão ou erro de comissão.

O simulador de Callantine identifica a operação correta do operador de sistema, em tempo real, e está focado em detectar erros de omissão e comissão que estão no nível de realização da tarefa. Diferentemente do S. PERERE, CATS não gera erros sobre a tarefa executada.

1.4. Estrutura do Trabalho

Este trabalho está organizado em 7 capítulos, sendo o primeiro esta Introdução.

Os Capítulos 2 e 3 apresentam o resultado da pesquisa bibliográfica realizada, formando o embasamento teórico para o desenvolvimento deste trabalho. O Capítulo 2 apresenta o estado da arte dos modelos de desempenho e as técnicas de simulação do comportamento humano. O Capítulo 3 apresenta uma ampla discussão em torno do termo erro humano objetivando, também, determinar a definição adotada neste trabalho.

O Capítulo 4 apresenta a proposta do S. PERERE.

O Capítulo 5 apresenta a implementação da proposta do modelo S. PERERE, com base em uma implementação pré-existente de uma arquitetura cognitiva.

O Capítulo 6 apresenta um exemplo de aplicação do S. PERERE, visando mostrar sua capacidade de geração de erros humanos. Também são discutidos os resultados obtidos com esta implementação.

Finalmente, o Capítulo 7 apresenta as conclusões obtidas com a realização deste trabalho, bem como propostas para a continuação de seu desenvolvimento no futuro.

O Anexo A apresenta os códigos fontes, em ACT-R, da implementação dos comportamentos elementares da taxonomia de Berliner; Angell; Shearer (1964).

O Anexo B apresenta os comportamentos elementares associados a erros.

2. MODELOS DE DESEMPENHO HUMANO E TÉCNICAS DE SIMULAÇÃO

O 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.

Este capítulo tem por objetivo apresentar o embasamento teórico obtido através da pesquisa bibliográfica sobre modelos que reflitam o comportamento humano.

Na primeira seção, será estudado o papel dos modelos na ciência e sua representação. A construção de modelos baseados no conhecimento atual da cognição humana é uma tentativa de explicar alguns comportamentos humanos.

Assim, este capítulo traz, em seguida, a conceituação de modelos cognitivos, de arquiteturas cognitivas e um estudo sobre o estado da arte de algumas arquiteturas cognitivas.

2.1. Modelos

Em consonância com o objetivo de produzir um simulador da ação humana que leve em consideração o erro, para ser aplicado ao estudo da confiabilidade humana na operação de sistemas computacionais, considera-se de extrema importância a identificação de um modelo que represente não somente as propriedades cognitivas mas que especialize estas propriedades nas tarefas de interação homem-computador. Assim, inicia-se esta seção com uma discussão sobre o papel dos modelos na ciência e em seguida como utilizá-los na modelagem do comportamento humano.

Os modelos, de acordo com a visão do autor (Begosso, 1999), têm garantido o controle explicativo e preditivo de fatos do mundo real, e a matemática constitui-se numa poderosa ferramenta capaz de auxiliar o cientista na sua construção. As simulações² feitas sobre o modelo traduzem as idéias de analogia com o mundo real, em que o cientista impõe as diversas formas de modelização. Begosso (1999, p.37) ressalta que “os modelos têm vida e estrutura próprias que se distanciam da realidade

² Segundo Dupuy (1996), no contexto científico a simulação é um caso particular de modelização e consiste em reproduzir as funções de um sistema no computador.

formal. O cientista projeta sua mente no mundo real durante a atividade de modelização”.

Dupuy (1996) apresenta uma extensa discussão sobre modelos e indica que o modelo:

“[...] é como uma forma abstrata que vem encarnar-se ou realizar-se nos fenômenos[...] O modelo científico é uma imitação humana da natureza que o cientista logo toma como ‘modelo’ – no sentido comum – desta.” (DUPUY, 1996, p.23)

“Conhecer é produzir um modelo do fenômeno e efetuar sobre ele manipulações ordenadas. Todo conhecimento é reprodução, representação, repetição, simulação. Isso caracteriza o modo científico, racional do conhecimento.” (DUPUY, 1996, p.27)

O modelo é único no aspecto específico da investigação e, ao mesmo tempo, modela apenas uma parte do sistema. Lavery (1986) ressalta que uma das expectativas durante a construção do modelo é que ele seja capaz de fornecer uma base organizada do domínio do conhecimento específico cuja perspectiva parcial possa ser integrada para conseguir a compreensão do todo.

Conforme discutido anteriormente, os modelos são construídos pelo cientista objetivando buscar o entendimento de situações específicas presentes num certo contexto. Portanto os modelos apresentam duas facetas interessantes:

- Eles são geralmente limitados em seu domínio de aplicação;
- Porém suportam mais precisão quando comparados com a realidade formal e, em alguns casos, é possível realizar medições quantitativas (RITTER et al., 2000).

Gore; Corker (2000) enfatizam o fato de que modelar o desempenho humano,

“... is the process whereby human characteristics are embedded within a computer software structure that represents a virtual human operator interacting with a virtual operating environment.” (GORE; CORKER, 2000, p.4.B.3-1)

O modelamento da atividade humana objetiva atender, segundo Filgueiras (1996), a três propósitos. O primeiro é prever o comportamento humano de forma a calcular seu desempenho e orientar projetos de sistemas interativos. O segundo é simular o comportamento humano, a fim de testar sistemas interativos projetados no

primeiro propósito. Finalmente, é compreender o funcionamento da mente humana, simulando, computacionalmente, alguns comportamentos humanos inteligentes.

Este último propósito é um dos focos principais da Ciência Cognitiva, (BEGOSSO, 1999). Nesta abordagem, este trabalho utilizará modelos cognitivos com o objetivo de gerar modos simulados de comportamento humano.

2.2 Definições

Para melhor compreensão dos termos modelo cognitivo e arquitetura cognitiva, utilizados extensamente neste trabalho, esta seção tem por objetivo esclarecê-los.

Um modelo cognitivo fornece uma descrição limitada de como alguns aspectos da cognição humana estão estruturados. Estes modelos geralmente representam o processamento de informações numa tomada de decisão, no desempenho de uma tarefa perceptiva ou motora.

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). Neste sentido, as arquiteturas cognitivas oferecem uma plataforma para desenvolver modelos cognitivos e manter uma coerência teórica entre estes modelos.

Conforme apresentado no início desta seção, o modelo cognitivo quando implementado em programas computacionais, gera arquitetura cognitiva com o propósito de obter comportamento humano simulado.

2.3 Modelos cognitivos

Partindo da crença de que o ser humano tem a capacidade de processar informação e de que tal capacidade pode ser expressa num modelo geral de desempenho, Reason; Maddox (2003) propuseram o modelo representado pela Figura 1:

Fig. 1 – Modelo geral do desempenho humano (adaptado de REASON; MADDUX, 2003, p.03)

A Figura 1 consiste de três processos que, combinados, desempenham a realização de uma ação. O primeiro processo é a conscientização e percepção da informação. O segundo diz respeito ao processamento da informação e a tomada de decisão, ou seja, qual a ação que será executada. Finalmente, o terceiro processo, é a junção dos processos anteriores que proporcionará a execução da ação.

Um ponto importante que precisa ser destacado no modelo geral do desempenho humano é sua característica cíclica. Isto pode ser observado no momento da execução do terceiro processo (materialização da ação). Neste momento, o indivíduo está pronto para receber novos estímulos que o levarão a novos processos de conscientização e percepção.

2.4 Arquiteturas cognitivas

As arquiteturas cognitivas são desenvolvidas com o intuito de expressar o desempenho humano na forma de implementações computacionais. Elas apresentam um padrão genérico que foi representado por Wickens (1992), está ilustrado na Figura 2 e é descrito pelo autor como segue.

O *processo de percepção* transforma o estímulo do mundo externo em representações que podem ser operadas por processos cognitivos.

A memória consiste de dois componentes: a *memória de trabalho* que armazena informação temporariamente para o processo cognitivo efetuar transformações e a *memória de longa duração* que armazena grandes quantidades de informação que representam o conhecimento do sistema.

O processo de *cognição* envolve uma variedade de funções que objetivam transformar as informações vinda da memória em aprendizado, tomada de decisão, planejamento, etc.

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

Butterworth et al. (2000) escrevem que uma arquitetura cognitiva deve considerar, em sua estrutura, unidades de comportamento que se deseja estudar. Assim, é de se esperar que as diferentes arquiteturas cognitivas possuam componentes específicos, correspondentes a estas unidades de comportamento.

A arquitetura cognitiva deve, de maneira geral, abarcar um conjunto de processos humanos que reflitam características cognitivas, perceptivas e motoras. Estas características são, dependendo da arquitetura, passíveis de serem representadas em componentes computacionais que por sua vez refletem tipos de comportamento.

Fig. 2 – Arquitetura cognitiva genérica (adaptado de WICKENS, 1992, p.17)

Newell (1990) reforça a idéia apresentada anteriormente argumentando que as arquiteturas cognitivas deveriam ser capazes de explicar como todos os componentes da mente trabalham para produzir a própria mente:

“... a single system (mind) produces all aspects of behavior. It is one mind that minds them all. Even if the mind has parts, modules, components, or whatever, they all mesh together to produce behavior. Any bit of behavior has causal tendrils that extend back through large parts of the total cognitive system grounding in the environmental situation of some earlier times. If a theory covers only one part or component, it flirts with trouble from the start. It goes without saying that there are dissociations, independencies, impenetrabilities, and modularities. These all help to break the web of each bit behavior being shaped by an unlimited set of antecedents. So they are important to understand and help to make that theory simple enough to use. But they don't remove the necessity of a theory that provides the total picture and explains the role of the parts and why they exist” (NEWELL, 1990, p.17-18).

As palavras de Newell refletem sua busca de uma teoria capaz de congregar os aspectos mais expressivos da cognição humana.

O modelo cognitivo procurado deverá ser representado através de uma arquitetura cognitiva para que o simulador possa ser construído.

Para que isso seja possível, a arquitetura cognitiva candidata à construção do modelo aqui proposto deve explicar, num nível computacional ou teórico, aspectos cognitivos, perceptivos e motores do desempenho humano.

São exemplos destas arquiteturas o ACT-R (Atomic Components of Thought -Rational), proposto por Anderson (1993) e Anderson; Lebiere (1998); o Soar, proposto por Laird; Newell; Roseblom (1987) e Newell (1990); o Epic (Executive-Processes Interactive Control), proposto por Kieras; Meyer (2004) e o Cognet (Cognition as a Network of Tasks), proposto por Zachary et al. (1992).

Dada a importância destas arquiteturas nesta pesquisa, na próxima seção serão apresentadas algumas arquiteturas cognitivas que representam o estado da arte.

2.4.1. COGNET – Cognition as a Network of Tasks

Cognet desenvolvido por Zachary et al. (1992), é uma estrutura para criar e empregar modelos de operadores humanos utilizados em tarefas cognitivas. O principal uso da arquitetura cognitiva Cognet é para desenvolver modelos de usuários para interfaces inteligentes.

Conforme ressaltam Zachary et al. (1992) e Zachary; Le Mentec; Ryder (1996), a arquitetura cognitiva Cognet surgiu a partir da necessidade de diminuição da sobrecarga de concentração a qual são submetidos os operadores de sistemas

industriais. Nesta categoria de sistemas pode-se citar salas de controle de planta nuclear, de redes de telecomunicações, de sistemas de operações de armas, de controle de tráfego aéreo, etc. Geralmente os operadores destes sistemas precisam dividir sua atenção entre várias tarefas que são executadas simultaneamente e em tempo real. Assim, o Cognet pretende incorporar modelos inteligentes de usuário aos sistemas com os quais os operadores interagem.

Para atender o objetivo proposto os autores Zachary et al. (1992), propuseram um *framework* para representação de processos cognitivos de múltiplas tarefas em tempo real que denominaram de RTMT (*Real-Time Multi-Tasking*). Uma representação de processos cognitivos RTMT, para aplicação como modelo de usuário incorporado numa interface inteligente deve atender a três condições: psicológica, computacional e operacional.

Como condição psicológica deve-se considerar aspectos comportamentais e observáveis da interação homem computador. Este requisito não deve conter especificações de como o problema será solucionado ou otimizado, mas o que o operador humano deverá fazer em situações de RTMT.

Para a condição computacional deve-se considerar a necessidade de que apenas aspectos computáveis de RTMT sejam selecionados para incorporar o modelo de usuário na interface.

Finalmente, para a condição operacional deve-se considerar a inclusão de restrições do mundo real neste ambiente simulado.

Estrutura de funcionamento do Cognet

O Cognet está baseado na metáfora do pandemônio onde os processos cognitivos são compostos por demônios que gritam continuamente. Cada demônio é capaz de realizar alguns aspectos da cognição e chamam por atenção como uma oportunidade para que seu processo ocorra (ZACHARY et al., 1992).

Reservando as devidas proporções, em situações de múltiplas tarefas que precisam ser desempenhadas em tempo real com sistemas computacionais, a metáfora do pandemônio retrata a disputa pela atenção do operador na interação com tais sistemas.

Zachary et al. (1992), representam a tomada de decisão em processos RTMT de maneira similar à metáfora.

No Cognet, o operador é abstraído como uma rede de tarefas cognitivas. Cada tarefa representa uma estratégia para desempenhá-la ou para resolver algum aspecto do problema em geral. O fluxo de atenção de uma tarefa para outra é disparado por mudanças momentâneas no ambiente do problema. Tais mudanças podem ser o resultado de ações tomadas por outros operadores ou por outros agentes que resultam numa mudança no estado do ambiente.

As tarefas são realizadas por agentes independentes que objetivam resolver problemas. Cada tarefa pode ser parcialmente completada, interrompida por outra tarefa e num outro momento, ser reiniciada no ponto da interrupção.

O Cognet possui uma estrutura que permite a criação de modelos de desempenho humano que representam o conhecimento especialista e a manipulação de informação num dado cenário computacional. O componente de modelagem do Cognet apresenta uma interface de usuário chamada “quadro negro” onde o problema a ser solucionado é representado e armazenado em três bases de conhecimento, explicadas em mais detalhe a seguir: a base de percepção, a base de problema/declarativa e a base procedural/controle/ação (LEIDEN et al., 2001).

A base de conhecimento perceptivo é um mecanismo “capaz de perceber” e verificar o ambiente externo. A base de conhecimento declarativa refere-se a determinar os conceitos sobre uma dada situação. Por fim, a base de conhecimento procedural/controle/ação descreve o que pode ser realizado sobre uma situação em particular.

Embora o Cognet atenda a modelagem de processos cognitivos, esta arquitetura apresenta alguns pontos fracos que devem ser apontados: o software Cognet não pertence à classe de programas livres; a validação da arquitetura está restrita a aspectos psicológicos; e os modelos de percepção estão limitados e apresentam restrições na habilidade de modelar comportamento motor (LEIDEN et al., 2001).

2.4.2. EPIC – Executive-Processes Interactive Control

O EPIC é uma arquitetura cognitiva simbólica proposta por Kieras; Meyer (2004) que pretende modelar tarefas que utilizam o processamento perceptivo/motor,

tarefas em ambientes complexos com muitos eventos e objetos que devem ser manipulados, e o desempenho humano na realização destas múltiplas tarefas.

Card; Moran; Newell (1983) propuseram o modelo de processamento humano de informação (GOMS) que serviu de inspiração para o desenvolvimento do EPIC. Na próxima seção será estudado o modelo GOMS.

De acordo com Pew; Mavor (1998) o objetivo do EPIC é desenvolver uma teoria computacional de desempenho humano com o foco na realização de múltiplas tarefas. A teoria está sustentada por dois aspectos: no domínio da teoria e dos resultados de pesquisas da psicologia cognitiva e do desempenho humano; e na caracterização e na previsão quantitativa de sobrecarga mental, especialmente em situações que envolvam desempenho de múltiplas tarefas.

Para Leiden et al. (2001) a relevância do EPIC consiste em explorar a limitação do ser humano no modo pelo qual ele utiliza os sistemas perceptivo/motor e como eles são usados para sustentar atividades de pensamento e tomada de decisão.

Pelo fato da arquitetura não modelar processos cognitivos de aprendizagem, o foco do EPIC está nos estudos da limitação da capacidade audiovisual e motora dos seres humanos. Esta limitação se estende aos canais de saída (voz e mãos) e são representados num sistema computacional modelando tarefas interativas, focando diretamente na representação do desempenho de múltiplas tarefas. O objetivo principal do EPIC é explicar como os detalhes de estruturas de tarefas e requisitos podem ser conjugados durante o desempenho de múltiplas tarefas. Os modelos gerados pelo EPIC podem assim identificar pontos críticos na estrutura de tarefas. Ele é um importante instrumento para o estudo da melhoria do desempenho humano em situações que envolvam o desempenho de tarefas elementares perceptivas e motoras (KIERAS; MEYER, 1997). As tarefas elementares serão abordadas com mais detalhes no Capítulo 4 deste trabalho.

Conforme ressaltam Kieras; Meyer (2004), o modelo aborda o projeto de interface de usuário e fornece uma previsão estatística de desempenho humano, na realização de uma tarefa, durante a interação com um sistema. Estas previsões incluem tempo total para realizar a tarefa, o tempo e a seqüência de cada ação e também aspectos importantes que determinam a sobrecarga mental do indivíduo.

As aplicações do EPIC são relevantes para situações onde a velocidade na operação é crítica, pois envolve múltiplas modalidades perceptivas e motoras, na execução das tarefas.

Com relação à capacidade de previsão de erro, o foco da arquitetura, de acordo com a visão de Pew; Mavor (1998) está em modelar situações onde não exista alto grau de complexidade para a realização de múltiplas tarefas.

Por ser o EPIC, uma arquitetura cognitiva recentemente desenvolvida, nem todos os mecanismos psicológicos modelados por ela estão validados (LEIDEN et al., 2001).

Conforme mencionado anteriormente o EPIC é derivado do GOMS, desta forma, na próxima seção, será estudado o referido modelo para o processamento humano de informações.

Estrutura de funcionamento do EPIC

De acordo com Kieras; Meyer (1997), para o funcionamento de um modelo EPIC deve ser fornecido ao sistema: um conjunto de regras de produção que representam a tarefa a ser simulada; características físicas (por exemplo, cor, localização) dos objetos no ambiente; um conjunto de propriedades que instanciam a tarefa a ser simulada (por exemplo, eventos que ocorrem durante a realização da tarefa, valores de tempo que representam a duração dos eventos).

O produto final de um modelo EPIC é a previsão de tempo e seqüências de ações durante a realização de tarefas.

O EPIC possui a importante característica de pertencer à categoria de software livre.

2.4.2.1 Modelo GOMS

Esta teoria, desenvolvida por Card; Moran; Newell (1983) apresenta um estudo das habilidades cognitivas envolvidas na interação homem-computador. A teoria recebeu o nome de GOMS que significa: **G**oal (metas), **O**perators (operadores), **M**ethods (métodos) e **S**election Rules (regras de seleção).

Cada um dos componentes do Modelo GOMS, segundo a concepção de seus autores, pode ser definido como segue: O primeiro componente, a Meta, é

classificada como sendo uma estrutura simbólica que define um estado de operação que se deseja alcançar.

O segundo componente, Operadores, é composto por atos (procedimentos) elementares (perceptivo, motor ou cognitivo) cuja execução é necessária para mudar aspectos mentais do indivíduo ou para afetar o ambiente da tarefa. Um operador é definido por um efeito (saída) e duração específicos.

Como exemplo de Operador pode-se citar a digitação de um texto por um usuário de computador. A entrada é o texto a ser digitado, a saída é a seqüência de teclas no teclado e a duração (aproximada) é uma função linear do número de caracteres.

O terceiro componente, Métodos, descreve um procedimento para a realização de uma Meta. Esses Métodos são procedimentos que o indivíduo já desempenhou em algum momento, eles, entretanto, não são criados durante o desenvolvimento da tarefa. A descrição de um Método é um conjunto condicional de Metas e Operadores. Um Método que o indivíduo constrói a partir da experiência, análise e instrução anterior reflete a estrutura do ambiente da tarefa. Em síntese, um método traduz o exato conhecimento da seqüência de passos que é requerida para a realização de uma tarefa específica.

Finalmente, na tentativa de alcançar uma Meta, o quarto componente, Regras de Seleção, possibilitam encontrar mais de um Método que satisfaça a Meta desejada. A seleção de qual Método utilizar nem sempre é um processo demorado, pois, em alguns casos, existe apenas uma alternativa apropriada. No modelo GOMS, cada regra é do tipo: *Se condição é verdadeira na situação da tarefa Então utilize o Método M.*

O objetivo da teoria é qualificar e quantificar parâmetros de desempenho humano, na realização de tarefas bem especificadas, de interação homem-computador. Por exemplo, para uma certa tarefa, uma estrutura particular GOMS pode ser construída e usada para predizer o tempo necessário para completar a tarefa proposta.

O modelo GOMS, segundo seus autores, está baseado em uma estrutura do processamento humano de informação que pode ser descrito como um conjunto de três subsistemas: sistema perceptivo, sistema motor e pelo sistema cognitivo.

Card; Moran; Newell (1983) afirmam que a estrutura cognitiva de um indivíduo consiste de quatro componentes:

1. Um conjunto de Metas.
2. Um conjunto de Operadores.
3. Um conjunto de Métodos para alcançar as Metas.
4. Um conjunto de Regras de Seleção para escolher entre os métodos competitivos.

É importante ressaltar que as quatro unidades descritas acima estão relacionadas a atividades que uma pessoa deverá realizar. Assim, podem-se citar algumas características do modelo:

- Para uma dada tarefa, uma estrutura particular GOMS pode ser construída e utilizada para predizer o tempo necessário para completar a referida tarefa;
- Auxiliar na avaliação do sistema existente;
- Permitir que seja estabelecido um cenário do tipo passo-a-passo, de como os usuários interagem com o sistema.

Neste ponto, os autores indicam que o modelo GOMS desempenhará seu papel de processador humano quando as seguintes características estiverem presentes na realização da tarefa:

- Todas as unidades de tarefas devem ser definidas;
- O usuário que desempenha a tarefa deve possuir um certo nível de habilidade com a execução da mesma;
- A interface deve apresentar um alto grau de usabilidade de modo a minimizar as chances de incorrer em erro durante a realização da tarefa ;
- Espera-se que durante a execução da tarefa, o usuário execute-as de forma seqüencial;
- O usuário não poderá cometer erros durante a realização da tarefa.

O modelo, de acordo com Card; Moran; Newell (1983), não trata diretamente o comportamento em erro, desta forma é ignorada a possibilidade de erros, ainda que estes ocorram com relativa baixa frequência. Uma justificativa, dos autores do modelo, para permitir comportamentos errôneos na realização de uma tarefa, no modelo GOMS, é fornecida a seguir.

Para comportamentos livres de erro, Card; Moran; Newell (1983) afirmam que o modelo GOMS proporciona uma descrição completa do comportamento, medida no nível de metas, métodos e operadores. A partir de uma tarefa específica esta descrição pode ser detalhada numa sequência de operações (ocorrências de operadores). Pela associação de tempos com cada operador, o modelo fará a predição do tempo total para a realização da tarefa (se estes tempos são dados como distribuições, será feita uma predição estatística). Entretanto, o modelo não é apropriado quando ocorrem erros, ainda que, segundo os autores, o comportamento dos operadores seja suscetível a erros.

Mesmo aqueles usuários que exibem comportamentos especializados ao realizarem certas tarefas, não estão livres de cometerem erros e, com efeito, gastarem uma boa parte da realização da tarefa para corrigirem tais erros.

Na tentativa de explicar e adequar o modelo ao surgimento de erros cometidos pelos usuários na execução de tarefas, os autores explicam que o GOMS pode ser ligeiramente estendido para situações afetadas por comportamento errôneo. Primeiramente deve-se permitir que os usuários falhem e, em seguida, deve-se especificar como as falhas são corrigidas. O propósito de estender o modelo é o de observar que quando o operador falhar ele estará produzindo uma condição de erro na qual pode ser representada como uma *correção de meta* e esta meta será realizada através da *correção de método*.

De acordo com Card; Moran; Newell (1983), quando um usuário qualificado comete e corrige um erro, ele terá provocado os seguintes estágios:

- Erro: o usuário comete o engano;
- Detecção: o usuário torna-se ciente do erro;
- Reinicialização: o usuário reinicializa a tarefa para permitir a correção;
- Correção: o usuário desfaz o efeito do erro;
- Reassunção: o usuário retoma a atividade livre de erro.

Em resumo, é possível modelar os estágios citados acima, entretanto observa-se que o controle da estrutura GOMS perde sua essência na medida em que os erros, de maneira geral, são eventos de difícil previsão e quando ocorrem levam o usuário a iniciar uma correção sem empregar um método planejado.

O modelo GOMS, objetiva prever o tempo de realização de uma tarefa, claramente especificada, executada por um operador. A realização de tais tarefas deve ser livre de erro, pois estas variações comprometem a estimativa temporal de tempo para sua realização, objetivo do modelo. Entretanto, as formas de correção de erro, durante a realização da tarefa, podem ser incluídas na sua execução.

Nas próximas duas seções apresentam-se as arquiteturas cognitivas que têm por finalidade apoiar a proposta de construção de um simulador de erros sobre interfaces homem computador.

2.4.3. ACT-R – The Atomic Components of Thought – Rational

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.

Atualmente, conforme ressalta Budiu (2004), o ACT-R tem sido aplicado na construção de modelos que abrangem os seguintes domínios:

- Diferenças individuais;
- Aprendizado e memória;
- Resolução de problemas e tomada de decisão;
- Percepção e atenção;
- Desenvolvimento cognitivo.

O ACT-R possui três tipos de módulos:

- O módulo de interface, composto pelo módulo perceptivo que é responsável por identificar objetos, pelo módulo motor que controla as atividades motoras sobre o ambiente externo, módulo de audição e o módulo de fala. Estes dois últimos módulos fornecem ao ACT-R uma rudimentar capacidade de ouvir e de falar (BYRNE; ANDERSON, 1998);

- Os módulos de memória que distinguem dois tipos de conhecimento de longa duração: conhecimento declarativo e conhecimento procedural;
- O módulo de metas que tem por objetivo sustentar um comportamento cognitivo mantendo o estado interno do que fora planejado a ser executado naquele momento.

A coordenação que estrutura estes módulos é realizada por um sistema central de produção. Uma importante função do sistema de produção de regras é a atualização do estado atual do sistema.

Estrutura de funcionamento do ACT-R

O ACT-R representa dois tipos de conhecimento: o conhecimento declarativo e o conhecimento procedural. Estes tipos de conhecimento são mantidos em duas memórias que carregam os nomes correspondentes aos conhecimentos.

Neste trabalho adotam-se as definições de memória declarativa e memória procedural, amplamente utilizadas pelas ciências cognitivas, propostas por Anderson; Lebiere (1998).

O conhecimento declarativo corresponde às coisas de que as pessoas têm consciência e que podem descrever ou explicar para outras pessoas.

O conhecimento procedural corresponde ao conhecimento que as pessoas demonstram em seus comportamentos, dos quais elas não têm consciência. Por exemplo, não se pode descrever (ou possuir plena consciência dos processos, pensamentos, intenções) o ato de falar ou de expressar um sentimento de alegria.

O conhecimento é representado nas duas memórias (procedural e declarativa) de formas distintas.

Na memória declarativa, o conhecimento é representado através de *chunks*. Os *chunks* são elementos básicos da memória declarativa e são definidos por duas características: tipo ou categoria (por exemplo, um carro) e *slot* ou atributo de categoria (por exemplo, cor, modelo, ano do carro). Eles representam o conhecimento que um indivíduo possui para a resolução de um problema.

Na memória procedural, o conhecimento é representado através de produções (também denominadas de regras de produção). Estas produções são caracterizadas

por uma sentença do tipo SE-ENTÃO que objetiva controlar o comportamento do modelo (ANDERSON; LEBIERE, 1998).

Assim, *chunks* e produções são as estruturas básicas para a representação do conhecimento no ACT-R.

2.4.4. Soar

Soar é uma arquitetura cognitiva que tem como objetivo geral o desenvolvimento de sistemas inteligentes. Ele fornece uma estrutura computacional na qual o conhecimento é codificado e utilizado para produzir ações que procuram atender às metas. O Soar pertence à categoria de software livre.

De acordo com Laird (2004), o Soar foi concebido para construir entidades autônomas que buscam solucionar problemas e que utilizam grandes quantidades de informações.

Para Newell (1990), a partir de uma perspectiva biológica, evidencia-se a existência de uma forte relação entre comportamentos motor, sensorial e cognitivo. Desta forma emerge uma dificuldade para lidar com todos estes aspectos num só sistema. Atualmente o Soar não dispõe de mecanismos para representação de comportamentos sensoriais e motores. Ele está preparado para construir modelos de aprendizagem.

As principais características do modelo Soar podem ser assim expressas, de acordo com Newell (1990), Lehman; Laird; Rosenbloom (2004) e Cunha; Giraffa (2003):

- As tarefas são representadas num espaço de problema. Estes espaços de problemas são conjuntos de possíveis estados e operadores que ocorrem na tentativa de alcançar uma meta.
- O sistema de produção de regras atualiza a memória de longa duração. Nesta memória está armazenado o conhecimento para implementar e guiar as buscas por entre os espaços de problema do qual o sistema é capaz de lidar. Em outras palavras, esta memória armazena o conhecimento que traduz a realidade sobre situações no mundo.

- O sistema de produção interage com a memória de trabalho que contém situações para resoluções de problemas. Ela é composta por fragmentos de informação (*chunk*), ou seja, cálculos intermediários, operadores (é a parte aplicável de uma estrutura sentença) e metas.
- O procedimento para tomada de decisão está baseado em linguagem de preferências, por exemplo: aceito/rejeitado, melhor/indiferente/pior.
- Finalmente, as metas dirigem todos os comportamentos do sistema.

Em síntese, o Soar demonstra quatro funções primitivas para a resolução de um problema: selecionar um espaço de problema; selecionar um estado daqueles disponíveis diretamente; selecionar um operador; e aplicar o operador para obter um novo estado.

O Soar pode simular erros do processo cognitivo gerando estados de erro tanto na memória de trabalho quanto na memória de longa duração, porém com relação aos possíveis erros gerados para os comportamentos dos sistemas perceptivos e motor a arquitetura não está preparada. Os mecanismos psicológicos modelados pela arquitetura estão validados (LEIDEN et al., 2001).

Estrutura de funcionamento do Soar

Num modelo Soar o conhecimento inicial, para a realização de uma tarefa, é codificado por um especialista, através de regras de produção. O estado inicial do modelo, por sua vez, é representado pela codificação de variáveis que representam a situação do mundo num instante inicial, na memória de trabalho.

O resultado final da execução de um modelo Soar pode ser obtido de duas formas: o usuário codifica o estado final desejado e aguarda o final da execução de modo a verificar a solução encontrada pelo sistema; ou o modelo é executado durante um período de tempo e o modelador interrompe-o objetivando avaliar a saída proporcionada pelo Soar.

2.5. Comparação das arquiteturas cognitivas

Em resumo, nota-se que são muitos os esforços para obter uma teoria unificada que represente aspectos significativos da cognição humana através de arquiteturas cognitivas.

Uma característica desse esforço é que as arquiteturas especializaram-se em modelar comportamentos cognitivos, enquanto que os comportamentos perceptivos e motores surgem com maior profundidade em algumas (Act-r e Epic) e com menos detalhes em outras (Soar e Cognet).

A Tabela 1 apresenta algumas comparações entre as arquiteturas estudadas nesta seção. Tais comparações proporcionam uma visão geral das principais características de cada arquitetura.

Arquitetura Cognitiva	Processos			Software aberto	Validação dos mecanismos psicológicos modelados pela arquitetura
	Cognitivo	Perceptivo	Motor		
ACT-R	x	x	x	x	x
COGNET	x				
EPIC	x	x	x	x	
Soar	x			x	x

Tabela 1 - Características comparativas entre as Arquiteturas Cognitivas

2.6. A escolha da arquitetura cognitiva

O comportamento humano inteligente é dinâmico, complexo e envolve variáveis que, em boa parte das vezes, não são possíveis de serem controladas.

O ACT-R é o melhor dos candidatos estudados aos requisitos explicitados no parágrafo anterior. Ele é considerado arquitetura cognitiva, pois têm como objetivo central o desenvolvimento de sistemas inteligentes, amparado por uma estrutura computacional que “codifica” o conhecimento sobre uma tarefa específica e o utiliza para produzir ações durante o desempenho da tarefa gerando comportamentos. Desta forma, possui as características para apoiar construção do simulador aqui proposto.

As razões pelas quais optou-se por escolher a arquitetura cognitiva ACT-R foram:

- É apropriada para construir modelos que representam o conhecimento armazenado e adquirido para gerar ações que buscam atingir objetivos previamente definidos;
- A forma de representação do conhecimento está documentada e pronta para ser utilizada;
- Possui mecanismos de aprendizagem;
- Modela os processos cognitivos, perceptivos e motores;
- Diversos autores entre eles Jones; Ritter; Wood (2000), Leiden et al. (2001), Pew; Mavor (1998) e Ritter et al. (2002) afirmam que esta arquitetura está entre as que mais apresentam mecanismos psicológicos validados;

E finalmente, o software está disponível e suficientemente aberto para ser modificado.

Para atender os objetivos propostos no início deste trabalho, que é o de produzir um simulador da ação humana que leve em consideração o erro, faz-se necessário, no próximo capítulo estudar o erro e sua classificação segundo especialistas da área.

3. ERRO HUMANO

De maneira geral, observa-se que o comportamento humano é elemento contribuinte para a diminuição da eficiência e segurança de diversos sistemas críticos. Entretanto é importante observar que não é possível dispensar o elemento humano desta interação.

No capítulo anterior, foram descritas algumas arquiteturas cognitivas que têm como características fundamentais a simulação de aspectos da cognição humana, objetivando contribuir para os estudos nesta área.

Este capítulo tem por objetivo o estudo do erro humano. Para compreender a natureza e as formas de manifestação do erro, realizou-se pesquisa bibliográfica, enfocando os assuntos seguintes, apresentados nas seções deste capítulo.

O capítulo inicia com as definições para a expressão “erro humano”. Em seguida, é apresentado o modelo de Rasmussen que representa características de comportamento do operador durante a realização de uma tarefa. Estas características são divididas em três níveis de desempenho que podem ser afetados de erros. As causas de erro humano, em geral, são explicadas segundo estes três níveis de desempenho e o conhecimento deste modelo é fundamental para o entendimento das classificações dos erros. Em seguida, apresenta-se a classificação do erro humano proposta por Reason (1990). Esta classificação é importante, pois conduz a construção do simulador S. PERERE no tocante aos tipos de erro que ele produzirá.

Apesar de existirem diversas formas de estudar o comportamento humano, este trabalho está focado sob a perspectiva da interação humana com sistemas automatizados, através das interfaces homem-computador.

3.1. Definições do erro humano

Nesta seção serão apresentadas as definições do termo “erro humano” visando garantir o correto uso do termo.

Antes de iniciar qualquer reflexão a respeito do erro humano, faz-se necessário apresentar a definição de erro. Por ser o erro estudado a partir de diferentes perspectivas (engenheiros, psicólogos, sociólogos, etc.) é natural que as análises e conseqüentes definições do termo tenham também concepções distintas. Na literatura são encontradas inúmeras tentativas de definir “erro humano”,

entretanto observa-se não existir concordância, entre os autores, numa definição única para o termo.

Inicia-se esta discussão, antes de definir erro humano, tomando-se a definição para o termo erro. Ferreira (1988, p. 259) define erro, no senso comum, como

“Ato ou efeito de errar. Juízo falso; desacerto, engano. Incorreção, inexatidão. Desvio de bom caminho; desregramento, falta”.

Muitos autores propuseram definições para o erro humano, entre eles pode-se citar Sanders; McCormick (1987) que definiram o termo como uma decisão ou comportamento humano inadequado ou indesejável que reduz ou tem potencial para redução da eficácia, segurança ou do desempenho do sistema.

Hollnagel (1993) procurou definir o erro considerando aquelas ações humanas que provocam conseqüências indesejáveis:

“An erroneous action can be defined as an action which fails to produce the expected result and/or which produces an unwanted consequence.” (HOLLNAGEL, 1993, p.29)

Uma outra definição importante do erro humano é aquela proposta por Reason (1990):

“Error will be taken as a generic term to encompass all those occasions in which a planned sequence of mental or physical activities fails to achieve its intended outcome, and when these failures cannot be attributed to the intervention of some chance agency.” (Reason, 1990, p.09)

Ações errôneas abrangem toda a situação onde uma seqüência planejada de atividades físicas ou mentais falhou na obtenção de um resultado, e estas falhas não podem ser atribuídas às intervenções de causas externas. As ações humanas são intencionais (REASON, 1990).

Uma outra caracterização do termo erro humano está apoiada nas discussões da confiabilidade, que abriga o aspecto da confiabilidade humana, no qual se insere este trabalho. Conforme ressalta Almeida Júnior (2003), na acepção da teoria da Confiabilidade, o erro pertence à relação entre os conceitos de falha e disfunção e os três conceitos podem ser encadeados como ilustra a Figura 3:

Fig. 3 – Encadeamento entre os Conceitos de Falha, Erro e Disfunção (extraído de Almeida Júnior, 2003, p.14)

Para Almeida Júnior (2003), a **falha** de um componente pode ser causada por problemas mecânicos ou elétricos (no caso de hardware) ou por algum componente lógico do sistema (no caso de software). Geralmente, as falhas nos componentes podem conduzir a **erro** que se configura como problema que se estende ao módulo ou um subsistema ao qual o componente pertence. As falhas que conduziram ao erro podem, neste encadeamento de eventos, acarretar a **disfunção** do sistema que se configura como sendo o desvio de comportamento daquele especificado.

Em sua discussão sobre a expressão “erro humano”, Hollnagel (1998) apresenta diferenças no uso do termo “erro” que, segundo ele, pode significar ou a causa de alguma coisa, o evento propriamente dito, ou a consequência da ação. A Figura 4 ilustra esta discussão. Observe-se da figura, que o termo erro pode, na literatura, estar relacionado a qualquer elemento da hierarquia de Almeida Junior.

A visão de Almeida Junior, na opinião de Filgueiras (2004) é particularmente adequada ao entendimento do comportamento humano, visto que nesta hierarquia, falhas humanas são as causas psicológicas e fisiológicas que comprometem o desempenho humano. Estas falhas podem eventualmente se manifestar como erro humano, que é o comportamento que se afasta do especificado. Finalmente, como o elemento humano é parte de um sistema, composto ainda por hardware, software e tarefas, o erro humano pode ocasionar a disfunção do sistema. Não parece fazer sentido, pelo menos no escopo deste trabalho, a disfunção humana, *human failure*, no contexto da confiabilidade operacional, pois seria algo equivalente à invalidez física ou psíquica, ou mesmo a morte.

Fig. 4 – Significado do termo “erro” (adaptado de HOLLNAGEL, 1998, p.24)

Para efeito dos estudos neste trabalho, será considerado erro humano segundo a proposta de Swain; Guttman (1983), que está no contexto da confiabilidade humana.

Para estes autores, a confiabilidade humana é a probabilidade de que uma pessoa:

1. Execute corretamente alguma atividade exigida pelo sistema, dentro de um tempo requerido (se o tempo for fator limitante) e,
2. Não realize nenhuma atividade estranha que degrade o sistema.

Uma tarefa bem sucedida não é apenas aquela que, durante sua execução, alcançou a meta proposta. Sob a ótica da confiabilidade humana, a tarefa bem sucedida é aquela que atingiu o resultado esperado da missão, dentro das condições especificadas, e sem criar perturbações indevidas no processo.

Na próxima seção será estudado o modelo de Rasmussen que apresenta características de comportamento do operador durante a realização de uma tarefa.

3.2. Modelo de Rasmussen

A proposta de Rasmussen (1983) é um modelo do processamento humano de informações que tem por objetivo explicar o comportamento humano e geralmente é utilizado para estudar atividades de erro humano que foram causadas durante processos de resolução de problemas. Este modelo está baseado em estudos de psicologia do comportamento.

Rasmussen dividiu o modelo em três níveis de atividades desempenhadas por um operador humano numa determinada tarefa:

1. O nível de habilidade (*skill-based*)
2. O nível de regras (*rule-based*)
3. O nível de conhecimento (*knowledge-based*)

A seqüência dos níveis de desempenho (*skill-rule-knowledge*) corresponde a um aumento, nesta ordem, da complexidade da atividade cognitiva exigida para com o ambiente em que o operador está inserido ou para com a tarefa que ele realiza.

Resumidamente, os níveis de desempenho propostos por Rasmussen (1983) podem ser assim definidos:

- **nível de habilidade** (*skill-based*): está associado a tarefas que necessitam de habilidades manuais. Geralmente estas habilidades são frutos da prática rotineira de uma atividade, fazendo com que o indivíduo produza resposta rapidamente na presença de estímulo;
- **nível de regras** (*rule-based*): está associado a tarefas que são governadas por situações predefinidas. O indivíduo utiliza regras existentes na base de conhecimento para a execução da ação;
- **nível de conhecimento** (*knowledge-based*): está associado com a realização de tarefas mais complexas, isto é, aquelas que não dependem de respostas instantâneas e nem de treinamento prévio para executá-las. Os níveis inferiores de desempenho (*skill* e *rule*) proporcionam ao indivíduo um acúmulo de experiência pelas atividades de raciocínio inferencial e treinamento para a execução da ação.

Este modelo da ação humana permite refletir sobre o fato de que diferentes atividades demandam diferentes níveis de atenção, treinamento ou habilidade manual. Entretanto erros podem surgir na medida em que as tarefas são executadas em cada um dos três níveis. Como mencionado anteriormente, de acordo com a natureza da tarefa, os níveis de desempenho, do mais simples para o mais elaborado, *skill-rule-knowledge*, apresentam substancial aumento do grau de complexidade da execução da atividade.

De acordo com Reason (1990), qualquer tentativa de definição de erro humano ou formas de classificação deve iniciar com uma consideração das variações do comportamento intencional humano.

Neste ponto, considera-se importante situar o papel da intencionalidade junto ao sistema cognitivo. Uma breve discussão em torno do termo é realizada na próxima seção.

3.2.1 Intencionalidade

A noção de intencionalidade permeia a variabilidade do comportamento humano e que pode conduzir ao erro. O erro acontece no contexto de uma tarefa, que é precedida pela intenção.

Searle (1997), assim como Reason (1990), argumentam favoravelmente a uma relação entre consciência e intencionalidade. Tal relação aponta para o fato de que toda ação humana é o resultado primeiro de processos intencionais disparados por características individuais de capacidades, aptidões e conhecimento geral possibilitando que os estados mentais atuem.

A abordagem de intencionalidade proposta por Searle é, na sua essência, naturalista. Para ele, estados, processos e eventos intencionais são como parte da história da vida biológica do indivíduo, tal qual a digestão, o crescimento, o envelhecimento pertencem a ordem natural de acontecimentos de fenômenos naturais.

Os indivíduos, de acordo com Searle (1997), possuem duas características importantes:

- 1- Capacidades mentais, disposições, atitudes, modos de comportamento, *know-how*, etc. que ele denominou de *Background*;
- 2- Um conjunto de crenças e desejos denominados de Rede.

Searle argumenta que a Rede precisa do *Background*, pois os elementos da Rede não são auto-interpretativos ou auto-aplicáveis. Assim, pode-se dizer que o *Background* consiste em um conjunto de capacidades, e a Rede é uma estrutura de estados intencionais.

Do ponto de vista do conjunto de capacidades de um indivíduo, Searle (1997) afirma:

“Ora, dentro desse conjunto de capacidades haverá algumas que uma pessoa adquiriu em forma de regras, fatos, etc., aprendidos conscientemente. Por exemplo, ensinaram-me as regras de beisebol, a norma de que nos EUA dirigimos no lado direito da pista, e o fato de que George Washington foi o primeiro presidente. Não me ensinaram nenhuma regra para andar, nem me ensinaram que objetos são sólidos. [...] Algumas de nossas capacidades nos permitem formular e aplicar regras, princípios, crenças etc. em nossas atividades conscientes. Ainda assim, porém, estas precisam de capacidades de *Background* para sua aplicação.” (SEARLE, 1997, p.271-272)

Finalmente, Searle (1997) enumera algumas leis de funcionamento do *Background* que esclarecem o papel da intencionalidade nas ações do indivíduo:

- “Em geral, *não há nenhuma ação sem percepção, nenhuma percepção sem ação.*
- *A intencionalidade ocorre num fluxo coordenado de ação percepção, e o Background é a condição de possibilidade das formas assumidas pelo fluxo [...]* O *Background* não apenas dá forma ao conteúdo intencional [...] mas, para começar, a existência do conteúdo intencional requer as capacidades de *Background*.
- *O Background é manifesto apenas quando há conteúdo intencional.*” (SEARLE, 1997, p. 278-280)

Em resumo, capacidades mentais, disposições, atitudes, modos de comportamento, etc., apenas podem manifestar-se quando existem fenômenos intencionais, como uma ação, uma percepção, um pensamento, etc., porém todos eles são intencionais. Assim, é importante ressaltar que o erro não é uma falha na intenção. A intenção é clara nas ações do indivíduo, porém quando da ocorrência de um erro, algum elemento de natureza informacional não estava presente ou se encontrava corrompido.

Sustentado pelos estudos de Searle, Reason (1990), salienta que as noções de intenção, de erro e suas conseqüências são inseparáveis. Desta forma, sob o ponto de vista da ação humana, quando o erro ocorrer isto se dará, em primeira instância, a partir de processos intencionais.

Uma vez estabelecida a relação dos processos intencionais com as ações dos indivíduos, na próxima seção será apresentada uma classificação para o erro humano.

3.3. Classificação do Erro Humano

Nesta seção, pretende-se apresentar uma classificação do erro humano com o propósito de situar os estudos nesta área.

Reason (1990) propôs que as ações errôneas podem ser de dois tipos: ações involuntárias e ações intencionais.

As **ações involuntárias** são aquelas que desviam das intenções planejadas e, portanto não atingem suas metas. Por exemplo, situações onde, ao se realizar uma tarefa, ela é executada de forma automática e quando se percebe o estado atual da realização, a ação desviou-se do caminho.

As **ações intencionais** ocorrem como planejadas e ainda podem ser julgadas como errôneas, se falharem na obtenção do resultado. O problema reside mais na adequação do plano do que na ação que é constituída por uma intenção prévia (sobre intencionalidade, v. Seção 3.2.1).

A proposta de classificação do erro humano desenvolvida por Reason (1990), apresenta duas categorias, a saber:

- deslizes (*slips/lapse*)
- enganos (*mistakes*)

De acordo com autor, os **deslizes** (*slips*) caracterizam-se quando a ação pretendida não atinge seu objetivo e, em geral, são provocados por uma falha na atenção dedicada durante a realização de uma tarefa. Deslizes podem manifestar-se como explicado na Tabela 2:

Manifestações	Características
Omissão	Quando um passo do plano deixa de ser executado.
Seleção indesejada	Objeto selecionado indevidamente, 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 sequencial	Execução, dos passos da tarefa, fora da ordem prevista.

Tabela 2– Manifestações de Deslizes (adaptado de FILGUEIRAS, 1996, p.31)

O lapso (*lapse*), classificado anteriormente como um subtipo de deslize, ocorre quando acontece falha na memorização da ação ou dos objetos.

O **engano** envolve a diferença entre a intenção prévia e as conseqüências pensadas para a realização de uma tarefa, cognitiva ou física.

Reason (1990) ressalta que erros que pertencem à categoria de **engano** (*mistake*) são considerados como deficiência ou falha no processo de julgamento e/ou inferência envolvida na seleção de um objetivo ou na especificação de meios para atingi-los, independentemente se as ações dirigidas pelo esquema de decisão foram executadas de acordo com o plano.

O **engano** ocorre quando da seleção do objetivo ou definição do plano para alcançá-lo não são adequados. Este tipo de erro está associado à necessidade de diagnóstico de uma situação e à tomada de decisões, argumenta Filgueiras (1996).

Em outras palavras, as ações podem ocorrer como planejadas, mas o plano não está adequado para completar com êxito o que fora tencionado. O **engano** está associado com a formulação do plano.

Para Reason; Maddox (2003), o **engano** pode ser classificado em duas classes de acordo com o nível de desempenho em que eles ocorrem:

a) **Engano no nível de regras:** envolve falha na seleção ou na aplicação de regras para a resolução de problemas. Em outras palavras, ou foi aplicada uma regra errada (não condizente com a situação) ou uma regra adequada foi erroneamente aplicada, favorecendo ao surgimento de situação inapropriada.

b) **Engano no nível de conhecimento:** é caracterizado pela necessidade de resolver novos problemas para os quais o indivíduo não possuía “regras prontas”. Ele deverá apresentar uma solução a partir de seus conhecimentos prévios.

Reason; Maddox (2003) sugerem que deslizes (*slips*) e lapsos (*lapses*) ocorrem quando o plano de ação pode estar perfeitamente adequado, mas a ação não ocorre como planejado. Por exemplo, alguém planejou executar uma tarefa de modo correto, mas algo ocorreu impossibilitando a realização apropriada daquela tarefa. Estes tipos de erro ocorrem no nível de habilidade do desempenho humano na realização de uma tarefa:

- **Deslizes no nível de habilidade:** estão relacionados ao conjunto de tarefas, tipicamente manuais, que são executadas rotineiramente, por

um longo período de tempo. Por exemplo, “tarefas de digitação, manipulação de um painel de controle de elevadores, etc” (REASON; MADDUX, 2003, p.5).

Similarmente, Reason (1990) ressalta que deslizes e lapsos são erros que ocorrem a partir de alguma falha na execução e/ou na seqüência de uma ação, independentemente se o plano que direcionava a ação estava ou não adequado para atingir o objetivo. Para esse mesmo autor, deslizes fazem parte de ocorrências de erros que conduzem a comportamentos observáveis, tais como, deslizes na escrita, deslizes na fala, deslizes na ação, etc. Em contra partida, lapsos são tipos de erros que envolvem, por exemplo, falhas de memória e que geralmente não demonstram comportamentos observáveis, a não ser para o próprio indivíduo que o experimentou. Por exemplo, o indivíduo preparou-se mentalmente para falar e no decorrer do discurso uma palavra ou uma frase lhe faltou. Pode ocorrer uma pequena pausa, fruto do esforço para recuperar o termo “perdido ou esquecido”, até retomar o fluxo do discurso, com ou sem o fragmento “esquecido”.

Em resumo, sob o ponto de vista do processador humano de informações, de acordo com o modelo de Rasmussen (v. Seção 3.2), pode-se classificar os tipos de erro segundo os três níveis de desempenho, conforme apresentado por Filgueiras (1996), na Tabela 3:

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

Tabela 3 – Erros em relação à intenção e ao estágio cognitivo (Extraído de FILGUEIRAS, 1996, p.32)

Na próxima seção objetiva-se apresentar uma taxonomia do erro humano que detalha a discussão deste capítulo.

3.4. Taxonomia do Erro Humano

Para entender a natureza dos erros, pesquisadores têm tentado classificá-los de acordo com suas similaridades e diferenças. Reason (1990) desenvolveu um modelo teórico denominado GEMS (Generic Error Modeling System) que objetiva explicar os erros humanos. GEMS está organizado a partir do modelo de processamento humano de informações de Rasmussen (1983) (v. Seção 3.2): o nível de habilidade (*skill-based*), o nível de regras (*rule-based*) e o nível de conhecimento (*knowledge-based*).

O modelo GEMS é aceito como a taxonomia mais completa de erro humano.

Wood (2000) argumenta em favor da utilização do GEMS porque este modelo descreve tipos de erros que, combinados com o modelo de processamento humano de informações de Rasmussen, fornece a taxonomia de erros mais abrangente.

Shorrock; Kirwan (2002) ressaltam algumas características importantes do GEMS: o modelo discrimina e classifica de forma abrangente os erros; o modelo descreve o desempenho humano dentro de uma estrutura teoricamente plausível; e a documentação disponível é adequada para o entendimento da técnica.

A taxonomia está estruturada de acordo com os níveis de desempenho propostos por Rasmussen (1983), com algumas subdivisões dentro de cada nível. Estas subdivisões são denominadas de modos de falha que podem assumir diferentes formas de erro. A Tabela 4 apresenta a taxonomia de Reason.

A relação dos erros exposta a seguir corresponde à visão de Reason sobre os comportamentos errôneos humanos. Estes comportamentos devem ser reproduzidos, de alguma forma, pelo simulador da ação humana em erro.

3.4.1. Desempenho no nível de habilidade (skill-based performance)

Desempenho neste nível de habilidade representa comportamentos automáticos, misturados com comportamentos conscientes.

Erros humanos relacionados ao desempenho no nível de habilidade
Deslize de dupla captura Omissão após interrupção Redução de intencionalidade Confusão perceptiva Erros de interferência Omissão Repetição Inversão
Erros humanos relacionados ao desempenho no nível de regra
Primeiras exceções Countersigns e nonsigns Sobrecarga informacional Força de regra Regras gerais Redundância Rigidez Deficiências na Codificação Ação deficiente <ul style="list-style-type: none"> Regras erradas Regras não elegantes Regras não aconselháveis
Erros humanos relacionados ao desempenho no nível de conhecimento
Seletividade Limitação da memória de trabalho Longe dos olhos longe da mente Viés de confirmação Excesso de confiança Revisão enviesada Correlação ilusória Efeito halo Problemas com causalidade Problemas com complexidade <ul style="list-style-type: none"> Problemas com feedback atrasado Consideração insuficiente do processo no tempo Dificuldades com desenvolvimento exponencial Pensamento em série causal e não em redes causais Divagação temática Enquistar

Tabela 4 - Modos de falha de acordo com os três níveis de desempenho (de Reason, 1990, p. 69)

Deslizes de dupla captura

Ao realizar uma tarefa rotineira ou em um ambiente familiar, no momento de tomada de decisão, algum evento externo ou interno desvia a atenção do indivíduo que toma a decisão automaticamente, por força do hábito.

Omissão após interrupção

O indivíduo ao realizar uma tarefa rotineira é interrompido por algum evento e ao tentar retornar ao que estava fazendo não consegue lembrar-se em que ponto estava antes da interrupção, deixando de executar um passo. Por exemplo, em alguns casos quando o telefone toca, as pessoas atendem e esquecem onde elas estavam na realização da tarefa antes do telefone tocar ocasionando a omissão de algum passo da tarefa.

Redução de intencionalidade

Este tipo de comportamento em erro representa que o indivíduo possui uma meta para atingir, porém, por algum motivo, ele se esquece do objetivo proposto. Por exemplo, o indivíduo vai até a garagem para pegar alguma coisa, mas esqueceu por que ele teve que chegar até lá.

Confusão perceptiva

Normalmente, em situações rotineiras, os indivíduos realizam tarefas sem prestar a devida atenção naquilo que estão fazendo. Por exemplo abrir a porta do carro com a chave de casa; pressionar um botão similar ou numa posição próxima àquela que estava acostumado a acionar.

Erros de interferência

Apresenta-se na forma de “misturas” de fala, de ação ou na transposição de ações dentro da mesma sequência, produzindo uma mistura comportamental. Ele envolve a realização de certas ações não intencionais, porém muito parecidas com aquelas tencionadas. No exemplo de Reason (1990), “Na pressa de terminar o serviço de casa tomei um banho, coloquei as plantas no quarto e minha roupa-de-baixo na janela”.

Omissão

O indivíduo, durante a realização de uma tarefa, omite o próximo passo que deveria ser realizado para a tarefa, ou ainda ele pode omitir toda a realização da tarefa a partir de um certo ponto. Isto em geral ocorre quando falha a verificação do estado entre duas ações, requerida para determinar quando a primeira ação está completa e a próxima deverá começar.

Repetição

Este tipo de comportamento em erro ocorre quando assume-se que uma tarefa não seja tão longa quanto ela realmente é e o passo entre a localização assumida e a localização atual (dentro da sequência da ação) é repetido. Por exemplo, o indivíduo repete o passo atual da tarefa ou no caso extremo pode repetir todos os passos da tarefa, executando-a novamente.

Inversão

Inversão ocorre quando a sequência original de execução da tarefa é invertida. Por exemplo: um operador precisa acionar as chaves A, B e C sequencialmente. Uma verificação não adequada das chaves a acionar ele inverterá a ordem de acionamento.

3.4.2. Desempenho no nível de regra (rule-based performance)

De acordo com Reason (1990), regras consistem num conjunto de condições e ações. Os seres humanos atribuem um valor que está baseado na frequência de sucesso de aplicação da regra. Quando todas as condições são encontradas para uma regra, a ação é desempenhada e a força associada à regra é aumentada. Se não existir uma combinação perfeita para qualquer regra, a regra mais forte com a condição mais parecida é utilizada. Há a má aplicação de boas regras, isto é, as regras são válidas mas sua aplicação não é oportuna e a aplicação de regras ruins, que são situações onde o indivíduo lança mão de regras ou ações erradas para aquela situação.

Primeiras exceções

Este erro diz respeito à primeira vez em que o indivíduo encontra uma exceção para a regra geral que ele possui. Neste caso se a regra geral tem sido aplicada com sucesso no passado, ela também será aplicada para a situação de exceção, conduzindo ao erro. A regra mais forte com a combinação parcial mais próxima é disparada, causando um erro porque a regra correta nunca foi aprendida.

Countersigns e nonsigns

Ocorre quando uma informação contraditória ou omissa causa o disparo da regra errada. Reason argumenta que quanto mais condições forem atendidas, mais disponível a regra se torna para o disparo permitindo que regras mais fortes sejam disparadas ao invés da regra correta.

Exemplos deste tipo de erro ocorrem em ambientes onde múltiplos alarmes são disparados confundindo o operador humano sobre a real natureza do que está ocorrendo.

Sobrecarga informacional

Sobrecarga informacional ocorre devido à grande quantidade de informação em determinadas situações que impede a avaliação adequada de todas as possíveis regras que são aplicáveis àquela situação. Geralmente estas situações excedem a habilidade cognitiva do indivíduo para resolver problemas, não considerando todas as regras necessárias para resolvê-los.

Um exemplo deste tipo de erro é aquele em que, existem cinco possíveis condições para que um indivíduo resolva um problema. Ele considera apenas algumas destas condições e aplica a regra ignorando as demais condições.

Força de regra

Em casos onde não exista uma combinação perfeita de regras para determinada situação, é provável que a regra mais parecida e com a mais alta força seja disparada. Em outras palavras, quanto mais vezes uma regra for disparada, mais forte ela será considerada e sempre que houver combinação parcial das condições

para determinada situação, o sistema cognitivo tende a disparar a regra mais forte em detrimento da mais fraca.

Regras gerais

Regras gerais são consideradas mais fortes que regras específicas, pois as primeiras atendem uma parcela maior de problemas no mundo. Este tipo de erro está associado com a informação ausente numa certa condição e esta ausência distingue a regra geral de uma regra específica, assim a regra geral será disparada, pois ela tem a maior força.

Redundância

Aquela informação que não é necessária para combinação de uma regra, mas ela geralmente está presente. Na ausência de informação perfeita, a ocorrência de informação redundante pode levar ao disparo de regra errada.

Rigidez

Refere-se à preferência para aplicar regras que foram utilizadas com sucesso no passado, quando regras mais simples e eficazes estejam disponíveis. Por exemplo, pessoas rolam a tela, numa página de internet, procurando por certa informação. Se optassem por utilizar a função de “busca” talvez encontre a informação desejada mais facilmente.

Deficiências na codificação

Quando as condições de uma regra que representam características de uma situação particular não foram codificadas em sua totalidade, ou foram mal representadas.

Ação deficiente

Este tipo de desempenho em erro ocorre quando a ação da regra foi aprendida incorretamente. Assume-se que a condição tenha sido aprendida corretamente, mas as ações não foram apropriadas para o objetivo pretendido.

Ações deficientes estão divididas em:

- a) **Regras erradas:** a ação realizada está completamente errada;
- b) **Regras não elegantes:** quando regras antigas são usadas mesmo se regras melhores estão disponíveis;
- c) **Regras não aconselháveis:** são aquelas regras que funcionam para o objetivo pretendido, mas podem trazer riscos ou conseqüências negativas. Geralmente são aplicadas por indivíduos que utilizam soluções não aconselháveis para economizar tempo.

3.4.3. Desempenho no nível de conhecimento (knowledge-based performance)

Desempenho no nível de conhecimento é caracterizado por raciocínio complexo e resolução de problema em circunstâncias imprevistas. Erros que ocorrem neste nível acontecem por limitações de recurso, conhecimento incompleto, ou conhecimento incorreto. Isto significa que as pessoas falham ao pensar corretamente porque elas ou não têm a informação necessária ou não tem conhecimento para resolver o problema, ou elas não têm condições para resolver o problema mesmo com informação precisa.

Seletividade

Durante a realização de uma tarefa, o indivíduo focaliza sua atenção em informações não relevantes ao invés de prestar atenção nas características corretas.

A exatidão do desempenho sobre a tarefa depende diretamente do foco da atenção do indivíduo. Numa atividade de observação de instrumentos, o operador pode prestar atenção em características menos relevantes do contexto deixando de atentar para aspectos verdadeiramente importantes naquele momento.

Limitação da memória de trabalho

Quando um problema excede os recursos da capacidade da memória de trabalho do indivíduo. A carga ou pressão cognitiva sobre a memória de trabalho varia de acordo com a forma em que o problema é apresentado ao indivíduo.

Longe dos olhos longe da mente

Este tipo de erro acontece quando é dada muita atenção a fatos que vêm prontamente à mente ou não é dada atenção suficiente a fatos que não estão imediatamente presentes. Exemplos deste tipo de erro podem ser encontrados no trabalho de Fischhoff; Slovic; Lichtenstein (1978).

Viés de Confirmação

Quando o indivíduo constrói hipóteses preliminares a partir de uma base de informação com dados iniciais. Estas hipóteses cruzarão com situações em que os dados sejam melhores. Tais situações geram ambigüidade, o que conduzirá o indivíduo a favorecer a hipótese inicialmente formulada.

Excesso de confiança

Indivíduos excessivamente confiantes tendem a justificar o curso de ação escolhido focando a evidência que favorece sua escolha e desconsiderando sinais contraditórios. Esta tendência é composta por viés de confirmação demonstrado num plano de ação. Pessoas que cometem este tipo de erro resistem na mudança do plano, mesmo que haja informação indicando que as ações planejadas são improváveis de alcançar o objetivo ou que o objetivo é irreal.

Revisão enviesada

O indivíduo acredita que todos os fatores, numa tomada de decisão, foram considerados quando, de fato, eles não foram.

Correlação ilusória

Quando existe dificuldade para reconhecer ou entender quando duas ou mais relações variam conjuntamente. Exemplo de tal correlação é que a turbulência do ar tende a aumentar com a temperatura. Reconhecer tais relações é difícil para muitas pessoas.

Efeito halo

Indivíduos têm dificuldades em processar, de forma independente, duas classificações diferentes das mesmas pessoas ou objetos. Por exemplo, alguém possui uma visão sobre outra pessoa ou objeto. Esta visão é tão forte e marcante que quando expressa em público, contamina também outras pessoas, ou seja, o ouvinte passará a comungar com a visão expressada.

Problemas com causalidade

Indivíduos tendem a simplificar em demasia as relações de causa e efeito. Como consequência desta simplificação de causalidade, elas preparar-se-ão para um número menor de eventualidades que certamente ocorrerão no futuro. Geralmente o conhecimento prévio do resultado sobre um evento aumenta a probabilidade de perceber antecipadamente aquele desfecho. Este fato conduz as pessoas a superestimar suas habilidades de influenciar eventos futuros.

Problemas com complexidade

Problemas com complexidade referem-se à dificuldade humana em resolver problemas em ambientes com tarefas complexas. Os erros que ocorrem em ambientes complexos segundo a taxonomia são:

a) Problemas com *feedback* atrasado

Em situações onde o *feedback* proporcionado ao usuário não é imediato, surgirão problemas para estes usuários construírem um modelo preditivo para a situação. O *feedback* atrasado impõe às pessoas perda de sincronia entre o desenvolvimento da tarefa e o desenrolar dos acontecimentos.

b) Consideração insuficiente de processos no tempo

Pessoas que apresentam este comportamento de erro estão mais interessadas no estado atual das coisas do que em considerar como ocorreu seu processo de desenvolvimento.

c) Dificuldades com desenvolvimentos exponenciais

Pessoas que apresentam dificuldades em lidar com o desenvolvimento exponencial parecem não ter sentimento intuitivo para processos que se desenvolvem exponencialmente, tanto para crescimento quanto para o declínio.

d) Pensamento em série causal e não em redes causais

Pessoas que lidam com sistemas complexos apresentam tendência de pensamento seqüencial. Elas percebem os principais efeitos de suas ações para alcançar uma meta, mas ficam sem saber os efeitos dessas ações sobre o resto do sistema.

e) Divagação temática

Pessoas que mudam de assunto para assunto muito rapidamente, tratando cada um deles superficialmente. Geralmente a divagação temática está associada à dificuldade que as pessoas apresentam para tratar um assunto, abandonando-o e em seguida iniciando estudos sobre outros temas.

f) Enquistar

Enquistar diz respeito a considerar assuntos ou temas de forma excessivamente profunda, isto é, as pessoas que apresentam este tipo de comportamento apegam-se a detalhes, deixando de lado questões mais importantes referentes ao tema estudado.

3.5 Considerações finais

Neste capítulo foram discutidos os seguintes tópicos: o conceito de erro humano, a partir da visão de diversos autores; e um modelo do processamento humano de informação, o modelo de Rasmussen. Este modelo fornece uma descrição de três níveis de atividades de desempenho humano. Os erros gerados por um operador, durante a realização de uma tarefa, podem ser classificados nos níveis propostos por Rasmussen (1983).

Estudou-se também a classificação do erro proposta por Reason (1990) que objetiva classificar para os tipos conhecidos de erro os três níveis de desempenho de Rasmussen (1983). Essa taxonomia é importante para este trabalho pois oferece suporte teórico para a obtenção do sistema gerador de erros.

Observe-se, entretanto, que há comportamentos em erro descritos por Reason cujo efeito é idêntico, embora as causas possam ser diferentes. Assim, em um simulador cuja aplicação não é o entendimento da ação humana, mas a exploração da confiabilidade de interação homem-computador, pode-se agrupar alguns destes comportamentos.

No próximo capítulo, apresenta-se a proposta do simulador. Entre outros aspectos, são mostradas as especificações adotadas para a simulação dos mecanismos de erro.

4. PROPOSTA DO S. PERERE – SIMULATION OF PERFORMANCE IN ERROR

Este capítulo tem por objetivo apresentar a proposta do S. PERERE em atendimento aos objetivos explicitados no início deste trabalho que são: explorar a diversidade do erro humano em situações de interação com sistemas críticos; tratar o erro humano como uma expressão da variabilidade humana; e produzir um simulador da ação humana que leve em consideração o erro.

O capítulo inicia com os objetivos do S. PERERE e uma visão das entradas e saídas do sistema. Em seguida, são apresentados os comportamentos elementares para os processos cognitivo, perceptivo, motor e de comunicação. Destaca-se também a estrutura do S. PERERE bem como a descrição de cada um dos módulos componentes do simulador. A partir da taxonomia do erro humano proposta por Reason (1990), estudada no Capítulo 3, elaborou-se novo conjunto de modos de falha bem como a especificação para cada um deles.

4.1. Objetivos do S. PERERE

Sabe-se que seres humanos, de maneira geral, cometem erros. Sabe-se também alguns dos tipos de erros que são frequentemente cometidos por eles na execução de suas tarefas (v. Seção 3.3).

Idealmente, os sistemas que necessitam da interação homem-computador deveriam levar em consideração, em seus projetos, a maior diversidade possível dos erros que podem ocorrer durante a interação. Em geral, o projetista ou analista de confiabilidade considera apenas os erros mais críticos: ou aqueles cuja ocorrência sabidamente leva o sistema a situações indesejáveis, obtidas das análises de risco, ou ainda aqueles cuja ocorrência é sabidamente provável, pelo histórico dos fatos. No entanto, nem sempre é o erro apontado nas análises preliminares como mais crítico que causa acidentes.

Deseja-se que o S. PERERE contribua para o estudo da confiabilidade operacional. A análise dos erros humanos é tolhida pela própria limitação do desempenho humano: é limitada a capacidade humana de traçar hipóteses de erro e de avaliar suas conseqüências.

Sendo uma ferramenta computacional, o S. PERERE gera formas de comportamento humano em erro em número muito maior e com combinações que o tratamento humano não permitiria por ser uma tarefa muito longa e eventualmente anti-econômica.

Dada uma tarefa, o S. PERERE gera perturbações no comportamento do operador que são correspondentes aos erros humanos.

4.2. Proposta do S. PERERE

No início deste trabalho foram discutidas as dificuldades em considerar o erro humano na interação do homem com sistemas automatizados.

A proposta ora apresentada ressalta o fato de que é possível prever qualitativamente, em boa parte dos casos, erros que podem ocorrer da interação homem-computador, se for possível analisar o comportamento humano em situações para as quais existam erros associados.

Objetiva-se também definir a abrangência do sistema S. PERERE.

A visão de “caixa-preta” do simulador é representada na Figura 5 que destaca o fluxo de informação associado às entradas e às saídas.

Fig. 5 – Fluxo de informação do S. PERERE

Com relação à entrada de dados, o S. PERERE deve permitir que esta seja composta por tarefas, descrita através de seus elementos, representando o

conhecimento sobre o comportamento correto (esperado, postulado) de uma pessoa, realizando aquela tarefa.

O S. PERERE deve também permitir que seja informado o estado inicial da tarefa. Isto permite ao sistema reconhecer uma situação, um ponto de partida para a execução da tarefa. Os elementos de tarefa serão discutidos na Seção 4.2.2.1, entretanto pode-se adiantar que tais elementos serão atomizados como unidades de comportamentos elementares.

Com relação à saída, o S. PERERE gera, a partir da entrada, o comportamento do operador afetado de perturbações, bem como a relação das perturbações ocorridas. Esta relação de saída inclui múltiplas ocorrências de erro para uma mesma entrada, ou seja, o comportamento do operador realizando uma tarefa pode ser perturbado de diversas maneiras.

Dependendo da aplicação a ser dada aos resultados da simulação, o usuário do S. PERERE deve poder configurar o disparo do mecanismo de perturbação (modo de funcionamento). Se a intenção do estudo for, por exemplo, a melhoria do projeto da interação homem-computador, interessará ao analista a relação de todos os possíveis erros em uma dada tarefa. Neste caso, a geração de erros do S. PERERE deve funcionar de forma combinatória.

Se a aplicação do S. PERERE for uma aplicação de treinamento, em que o simulador substitui um operador humano, a geração de erros do S. PERERE deve ser aleatória no tempo, porém pode ser condicionada pelo instrutor em termos dos tipos de erros produzidos.

Se a aplicação do S. PERERE for a análise da confiabilidade humana através da conexão da ferramenta a um simulador de processos, a geração do erro humano deve ser aleatória no tempo e no tipo de erro gerado.

De acordo com as premissas anunciadas nos objetivos deste trabalho, os elementos de entrada do S. PERERE deverão ser definidos segundo a taxonomia de Berliner; Angell; Shearer (1964), apresentada na próxima seção.

4.2.1. Unidades de Comportamento

Conforme motivação estabelecida no início deste trabalho (v. Seção 1.2), é possível definir um conjunto de tarefas elementares que representam a interação homem-computador.

A premissa de representar o comportamento humano através de unidades elementares de comportamento pode não ser a forma mais adequada, do ponto de vista das arquiteturas cognitivas. No entanto, esta é a forma como as tarefas são descritas nos procedimentos operacionais sobre sistemas críticos; portanto esta representação é considerada a mais adequada para a entrada de dados ao simulador.

Berliner; Angell; Shearer (1964) propuseram uma taxonomia para modelos de processamento de informação que traduz, através de verbos, unidades de comportamento em relação aos processos de percepção, cognição, ação e comunicação. O objetivo da taxonomia, segundo os autores, é auxiliar na descrição de procedimentos operacionais.

Isaac et al. (2000) ressaltam que esta classificação é útil para explicar comportamentos que são comuns em tarefas que exijam algum tipo de interação com sistemas automatizados, principalmente com sistemas de controle de processos.

Filgueiras; Melnikoff (1995) usaram esta taxonomia para descrever procedimentos operacionais dos quais seriam extraídos objetos da interface homem-computador de sistemas de controle de processos.

As unidades de comportamento, expressas na taxonomia de Berliner; Angell; Shearer (1964) são apresentadas nas Tabelas 5 a 8.

PROCESSO COGNITIVO		
ATIVIDADE	COMPORTAMENTO ELEMENTAR	DEFINIÇÃO
Solução de problemas e tomada de decisão	Calcular	Determinar por processos matemáticos.
	Escolher	Selecionar após consideração de alternativas.
	Decidir	Chegar a uma conclusão com base em informação disponível.
	Comparar	Examinar características ou qualidades de dois ou mais objetos ou conceitos, com o propósito de identificar semelhanças ou diferenças.
Processamento de informações	Interpolar	Determinar ou estimar valor intermediário entre dois valores dados.
	Verificar	Confirmar.
	Lembrar	Reter informação (memória de curta duração) ou recuperar informação (memória de longa duração).

Tabela 5 – Taxonomia de Berliner, Angell e Shearer – Processo Cognitivo

PROCESSO PERCEPTIVO		
ATIVIDADE	COMPORTAMENTO ELEMENTAR	DEFINIÇÃO
Procurar e receber informação	Inspecionar	Examinar com cuidado, ver com espírito crítico.
	Observar	Atender visualmente à presença ou ao estado de um objeto, indicação ou evento.
	Ler	Examinar visualmente informação apresentada de forma simbólica.
	Monitorar	Acompanhar ao longo do tempo.
	Explorar	Examinar rapidamente mostradores para ter impressão geral.
	Detectar	Tomar consciência da presença/ausência do estímulo físico.
Identificar objetos, eventos e ações	Identificar	Reconhecer a natureza de um objeto ou indicação, de acordo com as características implícitas ou predeterminadas.
	Localizar	Procurar e determinar o lugar de um objeto.

Tabela 6 – Taxonomia de Berliner, Angell e Shearer – Processo Perceptivo

PROCESSO MOTOR		
ATIVIDADE	COMPORTAMENTO ELEMENTAR	DEFINIÇÃO
Simples/discreta	Mover	Mudar a localização de um objeto.
	Segurar	Aplicar pressão contínua sobre um controle.
	Empurrar/puxar	Exercer força para longe/perto do corpo do agente.
	Prender	Fixar um objeto a outro.
	Dar	Colocar um objeto sob posse de outrem, para uso dele.
	Remover	Destacar e mover fora de posição.
	Descartar	Livrar-se de objeto inútil ou desnecessário.
	Devolver	Retornar objeto a seu proprietário.
Complexa/contínua	Posicionar	Operar um controle que tem estados discretos.
	Ajustar	Operar um controle contínuo.
	Teclar	Operar um teclado.
	Instalar	Colocar no local ou posição indicada.

Tabela 7 – Taxonomia de Berliner, Angell e Shearer – Processo Motor

PROCESSO DE COMUNICAÇÃO		
ATIVIDADE	COMPORTAMENTO ELEMENTAR	DEFINIÇÃO
	Responder	Atender à solicitação de informação.
	Informar	Divulgar informação.
	Solicitar	Pedir informação.
	Registrar	Documentar algo, como na escrita.
	Ordenar	Solicitar ação.
	Receber	Ser destinatário de informação.

Tabela 8 – Taxonomia de Berliner, Angell e Shearer – Processo de Comunicação

A utilização da taxonomia de Berliner; Angell; Shearer (1964) é importante pelos seguintes motivos:

- Pela precisão lingüística na especificação dos verbos representativos do comportamento humano, o que facilitará a explicitação das tarefas por parte do usuário do S. PERERE;
- Por estabelecer a granularidade da especificação das tarefas em um nível suficientemente genérico para permitir a representação do comportamento humano em qualquer domínio;
- Por se julgar que os comportamentos humanos a serem afetados de erro no S. PERERE podem ser descritos segundo esta taxonomia.

Destaca-se que neste trabalho, optou-se por focar os estudos nos processos cognitivos, perceptivos e motores da taxonomia de Berliner; Angell; Shearer (1964) e que o processo de comunicação não faz parte do escopo do S. PERERE.

Na próxima seção serão estudados os elementos participantes do simulador S. PERERE.

4.2.2. Estrutura do S. PERERE

Inspirado no padrão genérico das arquiteturas cognitivas, apresentado no Capítulo 2, o S. PERERE possui a estrutura ilustrada pela Figura 6.

A Figura 6 apresenta as entradas, as saídas, os módulos de memória, os módulos motor e perceptivo, o pré-processador que interpreta a taxonomia de Berliner, o módulo perturbador da tarefa e o módulo disparador das perturbações.

Neste ponto é importante ressaltar as idéias de Butterworth et al. (2000), discutidas no Capítulo 2, que apontaram para o fato de que diferentes arquiteturas cognitivas possuem componentes específicos.

No caso do S. PERERE, estes componentes são o Pré-processador, o Módulo Perturbador e o Módulo Disparador, que estão em destaque.

Fig. 6 – Estrutura do S. PERERE

Nas próximas seções objetiva-se explicar a arquitetura cognitiva e a arquitetura do S. PERERE.

4.2.2.1. Conhecimento Declarativo

O conhecimento declarativo, representado dentro da Memória de longa-duração, na Figura 6, será denominado neste trabalho de Memória Declarativa. Neste

tipo de memória são armazenadas as informações que representam as tarefas a serem simuladas. Conforme Tate; Hendler; Drummond (1990), uma tarefa é definida como uma possível solução para resolver um problema específico. A tarefa é composta por elementos, chamados de operadores, os quais caracterizam uma classe de ações, contendo um conjunto de variáveis e/ou constantes que descrevem ações específicas.

Assim, para o contexto deste trabalho, as tarefas ou os conjuntos de elementos são entendidos como sendo a receita, ou a descrição, de algo que deve ser realizado. Estas tarefas são compostas por elementos de tarefas que Tate e seus colegas denominaram de operadores.

Os elementos de tarefas, por sua vez, dizem respeito a cada passo da receita (ou da descrição) que compõe a referida tarefa. Assim, adota-se que o passo da tarefa é um comportamento elementar da taxonomia de Berliner; Angell; Shearer (1964), descritos na Seção 4.2.1.

O conhecimento declarativo é alimentado como fluxo de entrada, na Figura 6, junto ao Pré-processador, denominado de *Conhecimento da tarefa*. Este conhecimento é fornecido pelo ator “usuário”, um especialista da tarefa a ser simulada. A memória declarativa precisa ser ainda alimentada com o conhecimento situacional, correspondente ao estado inicial das variáveis que armazenam o conhecimento para a realização da tarefa. Este conhecimento deve ser atualizado pela própria execução da tarefa.

4.2.2.2. Conhecimento Procedural

O conhecimento procedural, também representado dentro da Memória de longa-duração, na Figura 6, será denominado neste trabalho como Memória Procedural. Esta memória tem por objetivo armazenar as regras necessárias para a resolução de uma dada tarefa.

O conhecimento procedural é representado através de regras de produção do tipo condição-ação. A condição especifica um padrão a ser verificado na memória declarativa e nas variáveis que representam o estado atual do sistema, que também estão na memória declarativa. Este padrão deve estar presente nas condições, da regra de produção, para que as ações da regra possam ser aplicadas.

Esta memória estabelece que o sistema utilize o conhecimento adquirido pela memória declarativa e *como* tal conhecimento será empregado pelo S. PERERE para atender às metas propostas.

É importante salientar que o conhecimento procedural, assim como o conhecimento declarativo, também possui destaque na construção do S. PERERE. Isto deve-se ao fato de que, além de representar a forma de utilização correta do conhecimento armazenado, para a realização de tarefas, o conhecimento procedural pode estar afetado de erro.

4.2.2.3. Módulo Motor e Módulo Perceptivo

O módulo motor, representado na Figura 6 com o nome de *comportamento motor*, e o módulo perceptivo, representado com o nome de *percepção*, são canais que deverão estar sincronizados com o meio ambiente. No caso do S. PERERE, o ambiente é a interface homem-computador de um sistema de controle de processos.

O módulo perceptivo tem a função de verificar, no ambiente, informações de objetos relacionados a uma tarefa, tais como cor, posição, tamanho e tipo (valor textual ou numérico) e enviar essas informações para a memória declarativa.

O módulo motor tem a função de desempenhar comportamentos motores discretos ou contínuos relacionados a uma tarefa, descrita na memória procedural.

4.2.2.4. Módulo Pré-processador

O módulo pré-processador é um componente do S. PERERE e, portanto não faz parte da arquitetura cognitiva. Este módulo foi definido para simplificar a descrição das tarefas pelo usuário.

O módulo pré-processador recebe como entrada, do especialista da tarefa a ser simulada, o conhecimento da tarefa que é transformado em conhecimento declarativo. Este módulo recebe também como entrada o estado inicial da tarefa e o conhecimento de resolução da tarefa. A partir destas entradas o pré-processador realiza associação dos comportamentos do operador que serão simulados com aqueles que estão armazenados no repositório *comportamentos elementares*. No repositório *comportamentos elementares* estão armazenadas as regras de produção para gerar o comportamento a ser simulado, de maneira sintaticamente correta para a

execução na arquitetura cognitiva ACT-R. Após esta associação o pré-processador armazena no repositório *tarefa* o comportamento correto do operador simulado na realização da tarefa.

Embora não seja objetivo deste trabalho, o módulo do pré-processador também gera a tarefa a ser simulada em sua forma correta, ou seja, sem perturbações. Isto é representado pelo fluxo tracejado denominado *Fluxo Normal* que tem origem no pré-processador e destino na arquitetura cognitiva, sem a interferência do módulo perturbador.

4.2.2.5. Módulo Disparador

O estudo da confiabilidade humana requer uma reflexão sobre as probabilidades de erro associadas a execução de uma tarefa, pelo operador.

De acordo com LaSala (2002), existem quatro fontes de dados responsáveis pelo fornecimento de subsídios para a determinação das probabilidades de erro humano. Estas fontes são classificadas como:

- Estudos laboratoriais: são derivados de situações artificiais e não representam o mundo real;
- Dados de simuladores: dados provenientes de simuladores são melhores, pois tendem a representar o mundo com maior precisão, entretanto eles são geralmente coletados a partir de aplicações específicas o que nem sempre permite que tais dados sejam extrapolados para outras situações;
- Dados de campo: são os dados mais realistas para o estudo da confiabilidade humana. Entretanto nem sempre são coletados de forma sistemática e muitas vezes o ambiente físico e psicológico, no qual os dados foram gerados, não são analisados;
- Opinião de especialistas: opinião fornecida por especialistas é uma técnica a ser utilizada quando os dados, obtidos em outras fontes, devem ser modificados para serem aplicados em situações específicas. Os especialistas muito têm a contribuir em virtude de suas experiências na área.

Observa-se na literatura a existência de vários modelos que abarcam o estudo da confiabilidade humana. LaSala (1998, p. 365) pondera que tais modelos surgem como ondas pois: “several models will be proposed in a given time, will be

evaluated, and will be followed by calls for new models”. Entretanto, poucos modelos sobreviveram ao tempo, um bom exemplo é THERP (Techniques for Error Rate Prediction) proposto por Swain; Guttman (1983).

THERP é uma técnica freqüentemente aplicada para a predição da confiabilidade do desempenho humano. De acordo com LaSala (2002), esta técnica fornece taxas do erro humano que possibilitam avaliar a degradação de um sistema homem-computador, provavelmente causada por erro humano, associado com fatores tais como confiabilidade de equipamentos, procedimentos e outros. THERP também leva em conta a influência do ambiente, como favorável ou não ao erro, no cálculo das probabilidades. Entretanto, em alguns casos sua aplicação pode tornar-se difícil na tentativa de acomodar todos os fatores que são considerados importantes numa dada situação.

Em resumo, nota-se que muitos dos dados sobre o desempenho humano são altamente específicos e nem sempre é possível transferi-los para situações que extrapolam as circunstâncias nas quais eles foram obtidos.

A partir da discussão realizada aqui sobre os valores de probabilidade associados ao erro, é de interesse destacar que tais valores devem ser considerados na fase de projeto e nas reflexões que antecedem a concepção de uma interface homem-computador.

Neste trabalho, para a concepção do S. PERERE manteve-se o foco na geração do erro, independentemente da probabilidade de sua ocorrência. Cabe ao projetista de IHC, conhecedor da importância destes valores probabilísticos, analisar e refletir sobre a saída do sistema aqui proposto e o impacto dos erros gerados na interação homem-computador.

O módulo disparador do S. PERERE deve representar a mecânica de disparo do erro, uma vez que a geração das perturbações não pode ser totalmente aleatória.

Isto se deve ao fato de que nem todo tipo de erro ocorrerá a qualquer momento e em qualquer situação. Por exemplo, não é possível que seja disparado um erro do processo motor (por exemplo, teclar) enquanto o sistema desempenha um comportamento do processo perceptivo (por exemplo, localizar). Em outras palavras, o S. PERERE não deve disparar um erro que não pertença à categoria da ação que está sendo desempenhada.

O disparador realiza o sorteio de uma perturbação possível para determinada situação e informa ao perturbador sobre a perturbação escolhida.

4.2.2.6. Módulo Perturbador

O módulo perturbador é o núcleo do simulador S. PERERE. Ele é responsável por simular e afetar de erro os comportamentos elementares dos módulos motor e perceptivo da arquitetura cognitiva e simular o comportamento do operador realizando a tarefa afetada de erros. O módulo deve apresentar em sua estrutura mecanismos de produção para simulação dos processos cognitivos, perceptivos e motores. O módulo perturbador não é um elemento tradicional das arquiteturas cognitivas.

Após o recebimento da perturbação escolhida pelo módulo do disparador, o perturbador, a partir do repositório de *erros*, envia ao pré-processador, via repositório *tarefa*, a perturbação a ser incluída em determinado comportamento, produzindo como saída a produção perturbada. Em outras palavras, pode-se dizer que o perturbador gera situações errôneas que afetam os comportamentos simulados para a realização da tarefa.

Em outro modo de execução, o perturbador também envia ao pré-processador instruções para gerar como saída uma relação de perturbações sobre determinado comportamento simulado.

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: comportamento correto, comportamento perturbado e relação de perturbações.

Uma vez que foram apresentados os principais elementos da arquitetura do simulador, na próxima seção apresentam-se as especificações das perturbações geradas pelo S. PERERE.

4.3. Especificação das perturbações

Nesta seção, objetiva-se especificar as perturbações que o módulo perturbador deve simular.

No Capítulo 3, discutiu-se a classificação do erro humano proposta por Reason (1990). Anunciou-se que tal classificação apresenta restrições no tocante a sobreposição de erros, ou seja, classificação distinta para erros muito próximos. Esta sobreposição acontece, principalmente, porque há diferentes causas, ou mecanismos, que levam ao erro.

De acordo com Hollnagel (1993):

“There are two fundamentally different ways to consider erroneous actions. One is with regard to their manifestation or **phenotype**, i.e., how they appear in overt action, how they can be observed, hence the empirical basis for their classification. The other is with regard to their cause or **genotype**, i.e. the functional characteristics of the human cognitive system that are assumed to be a contributing cause for the erroneous actions.” (HOLLNAGEL, 1993, p.69, destaques do autor)

Como o objetivo deste simulador não é estudar o comportamento humano, mas seu efeito do ponto de vista da confiabilidade operacional, na maioria dos casos a causa (genótipo) é menos importante que o efeito (fenótipo).

Assim, com o propósito de agrupar a implementação daqueles erros que apresentam proximidade na definição, optou-se por generalizá-los num único mecanismo que produza o mesmo efeito.

Com relação ao erro de “omissão após interrupção”, do nível de habilidade, considera-o como sendo caso especial do erro de “omissão”, assim adota-se agrupá-los como “omissão”.

No nível de regra, considera-se como sendo da mesma classe de erros: “primeiras exceções”, “força de regra”, “regras gerais” e “rigidez”. Tal agrupamento deve-se ao fato de que a alteração dos níveis de ativação das regras está presente em todos estes mecanismos. Adota-se, para o agrupamento, o termo genérico “força de regra”.

A característica comum entre os erros “countersign e nonsigns”, “redundância” e “sobrecarga informacional”, é a presença de informação redundante ou desnecessária na formação da regra. Para designar o agrupamento entre estes mecanismos de erro, adota-se o termo “redundância”.

Ainda com relação ao nível de regra considera-se que “regras erradas”, “regras não elegantes”, e “regras não aconselháveis” possuem em comum o fato de

que a condição codificada para tais regras não foi realizada adequadamente. Assim adota-se o agrupamento sob o termo “regra errada”.

No nível de conhecimento, adota-se “limitação da memória de trabalho” e “revisão enviesada” como sendo do mesmo tipo. A característica comum entre eles é falta de consideração de alguns fatores relevantes numa tomada de decisão, pois a memória de trabalho está sobrecarregada. Normalmente esta situação é configurada em situações onde exista a realização de múltiplas tarefas e em ambientes complexos. Assim adota-se o agrupamento sob o termo “limitação da memória de trabalho”.

Assim, constrói-se a Tabela 9 com os modos de falha de acordo com os níveis de desempenho de Rasmussen que o S. PERERE é capaz de simular.

Desempenho no nível de habilidade
Redução de intencionalidade Confusão perceptiva Sobrecarga motora* Percepção falsa* Omissão Repetição Inversão
Desempenho no nível de regra
Força de regra Redundância Deficiências na Codificação Regra errada Sobrecarga
Desempenho no nível de conhecimento
Seletividade Limitação da memória de trabalho Aprendizado deficiente*

* novos mecanismos de erro

Tabela 9 – Modos de falha especificados para o S. PERERE, a partir de (REASON, 1990)

É importante salientar que todos os mecanismos de erro do nível de regra foram contemplados na especificação do perturbador, considerando-se o agrupamento descrito anteriormente. Entretanto alguns dos mecanismos do nível de habilidade e do nível de conhecimento não foram contemplados para a especificação do perturbador do S. PERERE. Para o nível de habilidade, não foram contemplados: “deslize de dupla captura” e “erros de interferência”. Para o nível de conhecimento, não foram contemplados: “longe dos olhos longe da mente”, “viés de confirmação”, “excesso de confiança”, “correlação ilusória”, “efeito halo”, “problemas com causalidade” e “problemas com complexidade”.

Os motivos pelos quais estes mecanismos foram descartados da especificação do perturbador do S. PERERE são: as arquiteturas cognitivas estudadas não estão preparadas para simular tais mecanismos em sua totalidade; o nível de granularidade da definição destes mecanismos é muito alto, ou seja, existe grande dificuldade para abstraí-los num nível razoável para simulação.

Observe-se na Tabela 9 a inserção de três novos tipos de mecanismos de erro. Estes mecanismos surgiram ao longo desta pesquisa e considera-se importante descrevê-los, neste momento:

Aprendizado deficiente

Indivíduos ao realizarem uma tarefa que necessite de tomada de decisão, apresentam comportamento deficiente não sabendo qual atitude tomar no momento da decisão.

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.

Uma vez estabelecida a abrangência do perturbador, passa-se a especificação propriamente dita.

A memória procedural deve ser afetada por erros do tipo engano e deslize em alguns níveis de desempenho como: nível de habilidade e nível de regra, gerando perturbações. Essas perturbações podem provocar alterações no desenvolvimento da ação do indivíduo simulado.

No nível de habilidade, erros do tipo deslize são simulados pelas seguintes perturbações:

Sobrecarga motora

O perturbador deve selecionar aleatoriamente um elemento de tarefa do processo motor, executa uma ação e antes de concluí-la, dispara uma produção que solicita a realização de outra ação.

Omissão

O perturbador deve selecionar aleatoriamente um elemento de tarefa e omitir o disparo da produção que executaria aquela tarefa.

Repetição

O perturbador deve selecionar aleatoriamente um elemento de tarefa repetindo sua execução logo em seguida.

Inversão

O perturbador deve selecionar aleatoriamente um elemento de tarefa e inverter sua ordem com o elemento imediatamente seguinte.

Confusão perceptiva

O perturbador deve disparar regra de produção após objeto atendido visualmente e selecionar um elemento fisicamente próximo àquele observado.

Redução de intencionalidade

O perturbador deve selecionar aleatoriamente um elemento de tarefa e disparar produção que modifique o estado atual da meta daquela produção para valor nulo.

O tipo de erro engano, no nível de regra deve gerar as seguintes perturbações:

Deficiências na codificação

O perturbador deve disparar produção, substituindo a produção atual, por outra em que modifique parâmetros da condição.

Regra Errada

O perturbador deve alterar o ponto onde a produção é disparada substituindo por uma outra produção associada ao mesmo conjunto de variáveis.

Força de regra

O perturbador deve alterar o nível de ativação da regra, recuperando regra que não seja a mais aplicável àquela situação.

Sobrecarga

O perturbador deve considerar, num processo decisório, para uma dada situação, $n-1$ regra.

Redundância

O perturbador deve selecionar aleatoriamente regra de produção e acrescentar, à esta regra, variável que será verificada em conjunto com as demais.

Para a memória declarativa, pode-se produzir perturbações do tipo engano (as duas primeiras especificações a seguir) e lapso (a terceira especificação), nos níveis de conhecimento e habilidade, respectivamente, provocando o seguinte:

Aprendizado deficiente

O perturbador deve selecionar aleatoriamente um elemento de tarefa modificando o conteúdo de um atributo.

Seletividade

O perturbador, num processo de recuperação de porção de informação, deve recuperar aleatoriamente outra porção de informação que não a desejada, caracterizando recuperação de informação não relevante.

Limitação da memória de trabalho

O perturbador deve considerar, num processo decisório, para uma dada situação, apenas a última regra.

É possível representar o conhecimento incompleto sobre uma tarefa. Além da geração automática de perturbações, o usuário do S. PERERE alimenta manualmente a memória declarativa com erros do tipo lapso, como entrada, configurando, por exemplo, falhas na memorização de um dado conhecimento. Este tipo de erro pode conduzir a ações que acontecem diferentemente daquilo que fora planejado. Neste caso, o sistema pode ter armazenado a informação incorreta ou pode não ter armazenado uma informação importante (informação correta) para determinada situação.

No módulo perceptivo, o tipo de erro deslize, no nível de habilidade, gera a seguinte perturbação:

Percepção falsa

O perturbador deve selecionar aleatoriamente um elemento de tarefa do processo perceptivo, atender visualmente a uma informação e não apresentá-la.

4.4. Considerações finais

Neste capítulo foi discutida a arquitetura do S. PERERE, detalhando os requisitos de cada um dos seus elementos.

Apresentou-se também a especificação do módulo perturbador com os erros humanos gerados pelo simulador S. PERERE.

No próximo capítulo objetiva-se detalhar os componentes do ACT-R que apóiam a implementação do S. PERERE bem como, com base na especificação aqui realizada, a implementação propriamente dita.

5. IMPLEMENTAÇÃO DO S. PERERE UTILIZANDO O ACT-R

Este capítulo tem por objetivo apresentar a implementação do simulador S. PERERE.

Na primeira seção apresenta-se com mais detalhes, a descrição da teoria ACT-R, já apresentada no Capítulo 2 e cujo modelo computacional foi usado neste trabalho.

Na terceira seção detalha-se o funcionamento dos componentes da arquitetura.

Na quarta seção apresenta-se a implementação do S. PERERE sobre a arquitetura ACT-R.

5.1. Detalhamento da arquitetura do ACT-R

Para mostrar o uso do ACT-R como base para a construção do S. PERERE é necessária uma descrição mais detalhada desta arquitetura.

O ACT-R, desenvolvido por Anderson (1993), Anderson; Lebiere (1998) e aperfeiçoado por Anderson et al. (2004), é uma arquitetura cognitiva que pretende explicar a cognição humana através do desenvolvimento de estruturas de conhecimento fornecendo uma descrição integrada de alguns aspectos da cognição humana.

A Figura 7, adaptada de Budiu (2004), ilustra a lógica de uso do ACT-R.

A forma pela qual o ACT-R foi construído, ressalta Budiu (2004), reflete estudos sobre a cognição humana e estes estudos estão baseados em fatos derivados de experimentos psicológicos. Um programa ACT-R pode ser construído, utilizando-se a sintaxe da arquitetura, para verificar ou estudar um domínio cognitivo, perceptivo ou motor. Quando o desempenho do programa gerado convergir para a característica humana modelada, diz-se que o modelo ACT-R representa ou simula tais características naturais.

Anderson et al. (2004) descrevem a arquitetura ACT-R como um conjunto de módulos, em que cada um é responsável por um tipo diferente de processamento de informação.

Fig. 7 - Lógica de uso da Arquitetura Cognitiva ACT-R (adaptado de Budiu (2004))

A Figura 8, adaptada de Anderson et al. (2004, p.2) ilustra alguns módulos do sistema.

Fig. 8 – Organização da informação no ACT-R

O ACT-R possui três tipos de módulos:

- O módulo visual que é 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 ACT-R/PM, desempenham o papel de interface do sistema com o mundo real.
- Os módulos de memória que distinguem dois tipos de conhecimento de longa duração: conhecimento declarativo (memória declarativa) e conhecimento procedural (memória procedural).
- O 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. 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*.

Com o intuito de esclarecer este funcionamento, Anderson et al. (2004) escrevem que geralmente as pessoas não estão cientes de todas as informações que surgem em seus campos visuais. Elas apenas prestam atenção a um objeto ou situação de que tenham interesse. Da mesma forma, as pessoas não estão cientes de todas as informações que estão armazenadas na memória de longa duração. Elas atentam para aquela porção de informação que foi recuperada da memória de longa duração. Estas situações enfatizam o fato de que, na Figura 8, as informações são colocadas, retiradas dos *buffers* e distribuídas entre os módulos.

O ACT-R assume a hipótese de que cada ciclo da cognição humana é completado após decorridos aproximadamente 50ms. Assim, cada ciclo de cognição, na arquitetura, é caracterizado pelas seguintes etapas: os *buffers* armazenam representações determinadas pelo mundo exterior e pelos módulos internos, os padrões nestes *buffers* são reconhecidos, produções são disparadas, e os *buffers* são atualizados para o início de outro ciclo.

A característica híbrida do ACT-R é misturar processamento paralelo e serial dentro de cada um dos módulos. Anderson et al. (2004) ressaltam que enquanto o sistema visual está processando em paralelo as informações do campo visual, o sistema declarativo está executando uma busca entre as memórias para atender solicitação de outro módulo. Porém o sistema possui dois níveis de limitação estrutural no tocante ao processamento: primeiro, o conteúdo de um *buffer* está limitado a um *chunk*³, isto é, apenas uma memória pode ser recuperada de uma vez ou apenas um objeto pode ser codificado, naquele instante, a partir do módulo visual; segundo, apenas uma produção é selecionada, por vez, a cada ciclo de disparo de regras.

³ Neste caso, *chunk* está relacionado a uma porção de informação que é recuperada no *buffer*.

5.2. Os Componentes do ACT-R

Esta seção objetiva apresentar os componentes do ACT-R: as memórias declarativa e procedural e os módulos perceptivo e motor.

5.2.1. Memória Declarativa

O conhecimento declarativo é representado através de *chunks* que são responsáveis por armazenar a menor unidade de informação neste tipo de conhecimento. (ACT-R, 2002).

No ACT-R, o conhecimento declarativo é representado da seguinte forma geral:

nome-do-chunk:

isa ⁴	categoria
slot1	atributo1 da categoria
slot2	atributo2 da categoria
slotn	atributon da categoria

Os fatos de uma situação são codificados através dos *chunks* garantindo que uma estrutura maior de conhecimento declarativo seja formada a partir destas unidades de informação. Por exemplo, o fato de que o número dois somado ao número seis é igual a oito, é representado da seguinte maneira:

Fato2+6:

Isa	adição
Parcela1	two
Parcela2	six
Soma	eight

Two

Isa	numero
Value	2

⁴ Isa é equivalente a “is a” (“é um”) e corresponde ao tipo do *chunk*.

Three

Isa numero

Value 3

Six

Isa numero

Value 6

Eight

Isa numero

Value 8

(chunk-type adição parcela1 parcela2 soma)

(chunk-type numero value)

Fato2+6 é o nome do *chunk* que pertence à categoria **adicao**. **Adicao** é um tipo de *chunk* declarado por *chunk-type* que estabelece os *slots* pertencentes a este tipo. Situação similar ocorre com o tipo de *chunk* **numero**. Os valores atribuídos ao **Fato2+6** também são *chunks* do tipo **numero** e cada qual compreendido pelo seu respectivo valor numérico.

5.2.2. Memória Procedural

O conhecimento procedural é representado através de regras de produção do tipo Condição (Se) – Ação (Então). A condição SE, consiste de uma especificação de *chunks* em vários *buffers*. A ação ENTÃO, diz respeito às modificações daqueles *chunks* ou solicitações para outros *chunks*.

Os *buffers* podem ser entendidos como a interface entre a memória procedural e os módulos, visual, motor e a memória declarativa do ACT-R.

Ressalta-se que cada *buffer* armazena um *chunk* por vez e as ações de uma produção afetam o conteúdo de um determinado *buffer*. O ACT-R apresenta quatro *buffers*: de metas (*goal*) que armazena o objetivo atual da produção; de recuperação (*retrieval*) que armazena um *chunk* da memória declarativa; perceptivo (*visual*) que armazena um objeto no contexto visual, a localização do objeto e o estado do módulo

perceptivo; e motor (*motor*) que armazena informação sobre a disponibilidade do buffer e requisita ações que serão desempenhadas por este módulo (ACT-R, 2002).

A forma geral de uma regra de produção é dada por:

(p nome-da-produção

Lista de *buffers* a serem verificados

==>

Lista de *buffers* a serem modificados

)

Onde: p é o comando para criar uma produção

==> então (é o lado da ação)

O teste de *buffers* consiste de um conjunto de padrões que são verificados com os conteúdos dos *buffers* atuais. Se os padrões testados forem iguais, então a produção pode ser selecionada. É possível que mais de uma produção seja selecionada e a partir delas, apenas uma será escolhida para disparar a ação da produção.

5.2.3. Módulo Perceptivo

O módulo perceptivo é composto pelo módulo de *buffer* visual que é subdividido em três *buffers*: o *buffer* visual (*visual*), o *buffer* de localização visual (*visual-location*) e o *buffer* de estado visual (*visual-state*).

De acordo com ACT-R (2002), o *buffer* visual armazena *chunk* que representa um objeto no contexto visual. O *buffer* de localização visual armazena *chunk* que representa a localização do objeto no contexto visual. E, finalmente, o *buffer* de estado visual armazena *chunk* que representa o estado interno do módulo visual.

O exemplo abaixo mostra uma produção que determina a localização de uma letra na tela.

(p verificar-letra

=goal>

Isa read-letters

State find-location

=visual-location>

Isa visual-location

=visual-state>

Isa module-state

Modality free

Se a meta está no estado find-location e existe uma localização visual no buffer visual-location e o módulo de visão está livre

==>

+visual>

Isa visual-object

Screen-pos =visual-location

=goal>

State attend

Então
Adicionar a localização visual em screen-pos e mudar a meta para “atendida”.

)

Onde:

- =goal> é o teste do *buffer* de metas
- =visual-location> é o teste do *buffer* de localização visual
- =visual-state> é o teste do *buffer* de estado visual
- após o símbolo ==>, se o nome do *buffer* é precedido com “=” então a ação é para modificar seu conteúdo. Se o nome do *buffer* é precedido com “+” então a ação é para requisitar o módulo do *buffer* (ACT-R, 2002).

O comando *attended* codifica se o sistema não percebeu o objeto (*nil*) ou se já visualizou o objeto (*t*) ou ainda, se já visualizou o objeto que foi recentemente colocado na tela (*new*).

A sintaxe do ACT-R e de suas extensões é complexa para ser apresentada nesta tese. Em resumo, esta produção solicita que um objeto visual seja colocado no *buffer* visual e modifica o estado da meta para *attend* (atendido).

Inicialmente é realizado teste para verificar a existência de um *chunk* do tipo *visual-location* no *buffer visual-location*. O próximo teste é realizado no *buffer* de

estado visual (*visual-state*) para verificar, no *chunk* do tipo *module-state*, se o módulo de visão está disponível.

No lado da ação (após o símbolo $\Rightarrow$) é solicitado que a posição do objeto visual (*screen-pos*) que é a localização visual no *buffer visual-location*, seja colocada no *buffer* visual. Esta solicitação causa no módulo visual mudança de atenção para uma localização específica, codificando o objeto desta posição e colocando-o no *buffer* visual.

O *buffer* visual apresenta o seguinte conteúdo:

Text177

Isa	text
Screen-pos	Loc178
Value	“h”
Status	nil
Color	black
Height	10
Width	7

O *chunk* é do tipo texto, pois uma letra foi codificada na tela. O *slot screen-pos* armazena a localização do *chunk* para aquele objeto e faz atribuição do *chunk* no *buffer* localização visual (*visual-location*) como fora especificado na produção. O *slot value* armazena o texto codificado na tela, neste caso a letra “h”.

O conteúdo do *chunk* Loc178 no *buffer* de localização visual (*visual-location*) é:

Loc178

Isa	visual-location
Screen-x	130
Screen-y	160
Attended	t

É importante observar que o *slot attended* teve seu conteúdo alterado para t indicando que o objeto e sua localização possuem a atenção do sistema.

5.2.4. Módulo Motor

O padrão de representação motora realizado pelo ACT-R/PM diz respeito ao controle do movimento das mãos.

De acordo com Byrne (2001) o módulo motor pressupõe um teclado virtual, representando um teclado tradicional de computador, no qual todos os comandos resultam em pressionamento de teclas. A ação motora é representada através de diversos estilos:

- Teclar com firmeza (*punch*): corresponde ao pressionamento de uma tecla que está logo abaixo de um dedo selecionado;
- Teclar levemente (*peck*): corresponde ao movimento contínuo de um dedo até a localização indicada no teclado seguido pelo pressionamento da tecla;
- Teclar levemente e retornar (*peck-recoil*): movimento similar ao anterior, a diferença está no fato de que após o pressionamento da tecla o dedo retorna para a posição inicial;
- Deslocar (*ply*): mover o mouse para uma determinada posição na tela;
- Apontar-a-mão (*point-hand*): mover a mão para uma nova localização.

Existem dois *buffers* associados ao módulo motor: o *buffer* manual (*manual*) e o *buffer* de estado manual (*manual-state*).

O *buffer* manual é utilizado para requisitar ações que serão desempenhadas pelas mãos. Assim como no módulo visual, antes de qualquer solicitação de ação deve-se verificar se este *buffer* está disponível.

A verificação do estado do *buffer* é muito importante, pois o módulo de visão/motor e o sistema de produção operam em paralelo. Tal operação demanda 85 ms para completar a atenção a um objeto e pode ocorrer que uma outra produção requisite a mudança de atenção antes que o módulo de visão/motor complete a primeira solicitação. Esta situação conduz ao que se chama de “*jamming*” do módulo perceptivo/motor causando um erro no sistema (ACT-R, 2002).

O *buffer* de estado manual armazena informação sobre o estado deste *buffer*. O módulo motor pode apresentar dois estados: *busy* (ocupado) ou *free* (livre).

A produção abaixo é um exemplo do processo motor onde uma tecla é pressionada:


```

(p pressionar
  =goal>
    Isa      read-letters
    Letter   =letter
    State    respond
  }         Existe uma letra
            codificada em letter
  =manual-state>
    Isa      module-state
    Modality free
  }         e
            o buffer de estado
            manual está livre
==>

    +manual>
      Isa      press-key
      Key      =letter
    }         então
            é solicitado pressionar a tecla
            correspondente ao slot letter
            e a meta é trocada para stop
    =goal>
      State    stop
)

```

Esta produção dispara quando o estado da meta é *respond* e o *buffer* de estado manual indica que as mãos não estão em ação. Então uma solicitação é feita para que seja pressionada a tecla correspondente à letra do *slot letter* em *goal* e o estado final é alterado para *stop*. O tipo de ação solicitada para as mãos está especificada no *slot isa* do *buffer* manual. A solicitação *press-key* assume que as mãos estão localizadas sobre o teclado e os dedos retornarão à posição inicial após o pressionamento da tecla.

5.3. Implementação do S. PERERE sobre o ACT-R

Esta seção inicia com uma discussão acerca da necessidade da especificação de comportamentos elementares na implementação do S. PERERE.

Conforme estudado na Seção 5.1, a arquitetura cognitiva ACT-R fornece apoio computacional para a implementação do S. PERERE. Desta forma, para a construção das memórias declarativa e procedural e dos módulos motor e perceptivo, no S. PERERE é utilizada a estrutura disponível no ACT-R/PM.

Visando interferir minimamente no software do ACT-R/PM, de forma a não reduzir sua confiabilidade, os módulos do S. PERERE foram implementados de forma isolada do ACT-R/PM, alimentando a arquitetura cognitiva com arquivos de dados. Devido a esta independência, foi possível a implementação do módulo perturbador, do módulo disparador e do módulo pré-processador utilizando a linguagem de programação C++.

Sobre cópia da Figura 6, que apresentou a arquitetura do S. PERERE, observam-se os elementos em destaque da Figura 9 que representam a estrutura do ACT-R conforme utilizada para a implementação do S. PERERE.

Fig. 9 - Estrutura de suporte para o S. PERERE

A próxima seção apresenta os comportamentos elementares da taxonomia de Berliner; Angell; Shearer (1964), que são especificados na linguagem do ACT-R. Os comportamentos estão inicialmente descritos em português estruturado. Observe-se que para alguns comportamentos elementares não foi possível elaborar um código de conversão, pois até o momento, o desenvolvimento da arquitetura cognitiva ACT-R não permite tais especificações.

A descrição dos comportamentos elementares segue a ordem de classificação proposta pelos autores da taxonomia: processo cognitivo, processo perceptivo e processo motor conforme apresentação na Seção 4.2.1.

5.3.1 Especificação dos comportamentos elementares do processo cognitivo

Nesta seção apresenta-se a forma pela qual o conhecimento declarativo é aceito na estrutura do S. PERERE. Apresenta-se também o pseudocódigo e a especificação genérica, em ACT-R, de cada um dos comportamentos elementares.

As especificações em ACT-R dos comportamentos elementares de Berliner é uma das principais contribuições desta pesquisa. Estas especificações genéricas contribuem para flexibilidade do S. PERERE na aplicação em domínios diferentes. Obviamente algumas restrições devem ser atendidas. Esta seção estabelece tais restrições que são importantes para flexibilizar o simulador S. PERERE.

Os comportamentos associados ao processo cognitivo referem-se a dois tipos de atividades:

- Aquelas em que o ser humano desenvolve para solução de problemas e tomada de decisão e;
- Aquelas em que o ser humano as utiliza para o processamento de informações.

Os comportamentos do primeiro tipo são calcular, escolher, decidir, comparar. Os comportamentos do segundo tipo de atividade são: interpolar, verificar e lembrar.

Estes comportamentos são descritos nos quadros a seguir, que estão estruturados da seguinte forma: descrição do comportamento elementar, conhecimento declarativo e pseudocódigo. Os códigos ACT-R gerados para cada comportamento elementar estão listados no Anexo A.

Calcular

O comportamento elementar de *Calcular* simula o cálculo trivial realizado por um ser humano. A trivialidade do cálculo é proposital, uma vez que seres humanos realizam as operações básicas da aritmética (adição, subtração, multiplicação e divisão) mentalmente, isto é, sem lançar mão de recursos externos.

Conhecimento declarativo:

(solucao isa solucao op)

(term isa term num passo 1)

(term1 isa term num2 passo 2)

Onde:

solucao, term e term1: nome dos *chunks*

num e num2: argumentos numéricos que serão utilizados para o cálculo

op: operador aritmético (+, -, *, /)

passo: seqüência da operação

Pseudocódigo:

CALCULAR (num, op, num2)

 Result = num op num2

Retorna (result)

Onde:

result: resultado da operação

Escolher

O comportamento de *Escolher* diz respeito à preferência por um objeto após a consideração de alternativas. A especificação abaixo simula o ser humano visualizando objetos, em seguida, verifica se o atributo procurado está nos objetos visualizados. Em caso afirmativo retornará o objeto escolhido.

Conhecimento declarativo:

(passo1 isa escolha passo 1 status marca_visao)

(passo2 isa escolha passo 2 status visao_ok)

(passo3 isa escolha passo 3 status obj_visto)

(passo4 isa escolha passo 4 status resultado)

Onde:

passo1..passo4: nome dos *chunks*

passo: seqüência de execução

status: estado atual para cada porção de conhecimento

Pseudocódigo:

ESCOLHER (atributo)

Atender visualmente objeto na tela

SE objeto = atributo

Então result = “Escolhido”

Senão result = falso

Retorna (result)

Onde:

atributo: característica procurada no objeto.

result: o objeto escolhido.

Decidir

O comportamento de *Decidir* reflete a deliberação para uma certa situação com base em informação disponível.

Conhecimento declarativo:

(decidir1 isa numero sequencia 1 digito valor1)
 (decidir2 isa numero sequencia 2 digito valor2)
 (decidir3 isa numero sequencia 3 digito valor3)
 (decidir4 isa operador value 3 sequencia seqop1)
 (decidir5 isa operador value 2 sequencia seqop2)

Pseudocódigo:

DECIDIR (valor1, valor2, valor3, operador1, operador2, seqop1, seqop2)

 Se (valor1 operador1 valor2) operador2 valor3

 Então result = verdadeiro

 Senão result = falso

Retorna (result)

Onde:

valor1, valor2, valor3: valores numéricos

operador1, operador2: operadores lógicos

seqop1, seqop2: seqüência entre os operadores para a tomada de decisão

result: resultado da decisão

Comparar

Este comportamento de *Comparar* representa a observação de atributos de dois objetos, a fim de reconhecer semelhanças entre eles.

Conhecimento declarativo:

(compara1 isa comparar atributo ver_obj objeto 1)

(compara2 isa comparar atributo ver_obj objeto 2)

Pseudocódigo:

COMPARAR (objeto1, objeto2, atributo1, atributo2)

 Atender visualmente objeto1

 Atender visualmente objeto2

 Se (objeto2 possuir atributo1) e (objeto2 possuir atributo2)

 Então result = verdadeiro

 Senão result = falso

Retorna (result)

Onde:

Objeto1 e objeto2: objetos comparados

Atributo1 e atributo2: atributos do objeto1

Result: retorna *verdadeiro* se os atributos comparados existirem em objeto2 e *falso* em caso contrário.

Interpolar

O comportamento simulado de *Interpolar* determina o valor médio entre dois valores. Para que isto ocorra é necessário efetuar a leitura dos números e em seguida proceder ao cálculo de interpolação propriamente dito.

Conhecimento declarativo:

(solucao isa solucao op + op1 /)

(term isa term num passo 1)

(term1 isa term num2 passo 2)

Onde:

solução, term e term1: nome dos *chunks*

solução e term: categoria do chunk (após o *isa*)

num e num2: valores numéricos que serão utilizados para a interpolação

op e op1: operadores

passo: seqüência de execução

Pseudocódigo:

INTERPOLAR (num, num2)

Result = (num + num2)/2

Retorna (result)

Onde:

result: resultado da interpolação.

Verificar

O comportamento de *Verificar* busca confirmar visualmente se o estado de um objeto é o estado esperado. a exatidão de uma dada situação.

Conhecimento declarativo:

(verificar1 isa verificar atributo prep_mod)

(verificar2 isa verificar atributo ok obj "valor")

Onde:

verificar1 e verificar2: nome do *chunk*

verificar: categoria do *chunk*

atributo: estado atual para cada porção de conhecimento

obj: característica que se pretende verificar (cor, posição, tipo [numérico ou texto], valor)

Pseudocódigo:

VERIFICAR (objeto, valor)

 Atender visualmente objeto na tela

 Ler estado objeto

 Se estado = valor

 Então result = verdadeiro

 Senão result = falso

Retorna (result)

Onde:

result: resultado da verificação.

Lembrar

Definição: O comportamento de *Lembrar* tem por finalidade fazer vir à memória ou recuperar informação armazenada. A especificação aqui apresentada executa pesquisa, na memória do indivíduo simulado, de uma dada informação.

Conhecimento declarativo:

(lembrar1 isa lembrar atributo prep)

(lembrar2 isa lembrar atributo valor)

Onde:

lembrar1 e lembrar2: nome do chunk

valor: valor numérico que se deseja recuperar

Pseudocódigo:

LEMBRAR (valor)

 Pesquisar “valor”

 Se “valor” encontrado

 Então result = chunk

 Senão result = valor não recuperado

Retorna (result)

Onde:

valor: informação que se deseja recuperar

chunk: porção de informação na memória

5.3.2. Especificação dos comportamentos elementares do processo perceptivo

Os comportamentos do processo perceptivo estão associados a dois tipos de atividades:

- Atividades que resultam em procura, busca e recebimento de informação, tais como: inspecionar, observar, ler, monitorar, explorar e detectar;
- Atividades que identificam objetos, eventos e ações, tais como: identificar e localizar.

Inspecionar

O comportamento de *Inspecionar* refere-se ao exame minucioso e criterioso de um objeto no campo visual de um ser humano. A especificação abaixo simula a visualização do objeto na tela identificando suas características como cor, posição, tipo do objeto (um texto ou valor numérico) e o próprio objeto. O comportamento especificado retorna tais características após a inspeção.

Conhecimento declarativo:

(tarefa1 isa verifica atributo ver_obj)
 (tarefa2 isa verifica atributo prep_ver)
 (tarefa3 isa verifica atributo prep_conc valor "objeto" cor Black x 107 y 65)
 (tarefa4 isa verifica atributo ok testFim 0)

Onde:

tarefa1..tarefa4: nome dos *chunks*

verifica: categoria do chunk (após o *isa*)

atributo: estado atual para cada porção de conhecimento

valor: características que serão inspecionadas

Pseudocódigo:

INSPECIONAR (objeto)

Atender visualmente objeto na tela

Retorna (cor, posição, valor, tipo)

Onde:

Cor: é a cor do objeto

Posição: posição ocupada pelo objeto na tela (valor de x e y)

Valor: é o próprio objeto (aquilo que está impresso na tela)

Tipo: tipo do objeto (valor numérico ou texto)

Observar

O comportamento de *Observar* objetiva simular o fato do ser humano atender atentamente e visualmente à presença ou ao estado de um objeto, indicação ou evento no ambiente que o cerca. Na especificação abaixo o objeto é visualizado na tela e em seguida os valores de cor e tamanho do objeto são devolvidos como retorno.

Conhecimento declarativo:

(observar isa observar atributo ver_obj))

Onde:

observar: categoria do chunk (após o *isa*)

atributo: estado atual para cada porção de conhecimento

obj: objeto a ser observado

Pseudocódigo:

OBSERVAR (objeto)

 Atender visualmente objeto na tela

Retorna (valor, tipo)

Onde:

Valor: é o próprio objeto (aquilo que está impresso na tela)

Tipo: tipo do objeto (valor numérico ou texto)

Ler

O comportamento simulado de *Ler* tem por objetivo examinar visualmente informação textual apresentada no contexto visual daquele que faz a leitura. O retorno do comportamento consiste na palavra lida.

Conhecimento declarativo:

(le1 isa ler atributo ver_obj)

Onde:

le1: nome do chunk

ler: categoria do chunk

atributo: seqüência de execução

ver_obj: palavra lida

Pseudocódigo:

LER (palavra)

 Atender visualmente palavra na tela

Retornar (palavra)

Monitorar

O comportamento de *Monitorar* refere-se ao acompanhamento da execução de uma tarefa durante um determinado tempo. A especificação aqui relacionada localiza a posição do objeto no contexto visual, atende visualmente sua presença e acompanha tal objeto durante alguns instantes.

Conhecimento declarativo:

(monitorar1 isa verifica atributo ver_obj)
 (monitorar2 isa verifica atributo prep_ver)
 (monitorar3 isa verifica atributo prep_conc x v1 y v2)
 (monitorar4 isa verifica atributo ok testFim 0)
 (monitorar5 isa verifica atributo aguarde tempo valor)

Onde:

monitorar1..monitorar5: nome dos *chunks*

verifica: categoria do *chunk*

atributo: estado atual para cada porção de conhecimento

v1 e v2: posição de tela, em x e y respectivamente, onde será feita a monitoração

valor: tempo, em milissegundo, da duração da monitoração

*Pseudocódigo:***MONITORAR (objeto)**

Atender visualmente objeto na tela
 Recuperar “posição”
 tempo=1
 Enquanto tempo<=700 faça
 tempo=tempo+1
 ver objeto na tela em “posição”

Onde:

Posição: local do objeto na tela

Tempo: tempo, em milissegundo, transcorrido para visualizar o objeto na posição recuperada.

Explorar

O comportamento de *Explorar* diz respeito ao rápido exame de todos os componentes do contexto visual no qual o indivíduo está inserido. Este comportamento exprime a impressão geral que uma pessoa possui de um ambiente percorrendo visualmente toda sua extensão para obter informações.

Conhecimento declarativo:

(explora1 isa explora objeto objeto1 atributo passo1)

Onde:

explora: tipo do chunk

objeto: objeto a ser explorado na tela

atributo: posição do objeto

Pseudocódigo:

EXPLORAR (espaço_visual)

 Atender visualmente ao espaço_visual

Retorna (posição de todos objetos no espaço)

Onde:

Espaço_visual: toda a extensão da tela

Detectar

O comportamento de *Detectar* simula a percepção da presença ou da ausência de algum estímulo físico no ambiente. Nesta simulação, o indivíduo detecta a cor do objeto atendido visualmente.

Conhecimento declarativo:

(detecta1 isa detectar atributo ver_obj)

Onde:

detectar: tipo do chunk

atributo: estado atual para cada porção de conhecimento

obj: cor do objeto

Pseudocódigo:

DETECTAR (objeto)

 Atender visualmente objeto

Retorna (cor)

Identificar

O comportamento elementar de *Identificar* objetiva determinar a identidade de um objeto reconhecendo suas características predeterminadas. Na especificação apresentada a seguir, o objeto é localizado e visualizado. A seguir, as características do objeto em questão são comparadas com elementos implícitos ou predeterminados do ser humano simulado.

Conhecimento declarativo:

(identifica1 isa identificar atributo ver_obj)

Onde:

identificar: tipo do chunk

atributo: estado atual para cada porção de conhecimento

obj: característica do objeto

Pseudocódigo:

IDENTIFICAR (objeto, atributo1)

 Atender visualmente objeto

 Se objeto possuir (atributo1)

 então result = “identificado”

 senão result = “não identificado”

Retorna (result)

Onde:

atributo1: característica desejada identificar no objeto

Localizar

O comportamento de *Localizar* simula o ato do ser humano de inteirar-se do paradeiro de um objeto. A especificação abaixo inicialmente localiza a posição do objeto, atende visualmente a sua presença e retorna a posição encontrada.

Conhecimento declarativo:

(localiza1 isa localizar atributo ver_obj)

Onde:

localizar: tipo do chunk

atributo: estado atual para cada porção de conhecimento

obj: posição do objeto

Pseudocódigo:

LOCALIZAR (objeto)

 Atender visualmente objeto

Retorna (posição)

5.3.3. Especificação dos comportamentos elementares do processo motor

Os comportamentos do processo motor estão associados a dois tipos de atividades:

- Atividades motoras consideradas simples, tais como: mover, segurar, empurrar/puxar, prender, dar, remover, descartar e devolver;
- Atividades motoras consideradas complexas, tais como: posicionar, ajustar, teclar e instalar.

Nesta seção apresenta-se a forma pela qual o conhecimento declarativo é aceito na estrutura do S. PERERE e também a especificação genérica dos comportamentos elementares: mover e teclar.

Considera-se que os comportamentos do grupo de atividade motora simples (segurar, empurrar/puxar, prender, dar, remover, descartar e devolver) são variações do comportamento elementar mover, não havendo diferença significativa entre eles do ponto de vista da interação homem-computador. Deve-se considerar também que até o presente momento o ACT-R/PM não está preparado para implementar tais comportamentos.

Com relação àqueles comportamentos motores considerados complexos (posicionar, ajustar e instalar) a arquitetura cognitiva também não está preparada para implementá-los.

Para os casos em que não é possível obter o código de conversão ACT-R/PM, apresenta-se o pseudocódigo, na crença de que, num futuro próximo, tais comportamentos possam ser implementados.

Mover

O comportamento elementar de *Mover* objetiva simular o ser humano no ato de exercer movimento sobre um objeto. Na especificação abaixo ocorre a localização do objeto, sua visualização na tela, e um movimento com a mão direita sobre o mouse apontando para o referido objeto.

Conhecimento declarativo:

(chunk1 isa teclar status inicio)

(chunk2 isa visual-location screen-x X screen-y Y attended T)

(chunk3 isa visual-object screen-pos =chunk2)

Onde:

chunk1..chunk3: nomes dos *chunks*

X e Y: posição para a qual deseja-se mover o mouse

status: seqüência de operação

Pseudocódigo:

MOVER (objeto, posição)

 Atender visualmente *objeto* na tela

 Posicionar mão sobre o mouse

 Mover mouse até *posição*

Retornar (ok)

Segurar

O comportamento de *Segurar* está relacionado à aplicação de pressão contínua sobre um objeto. Este pseudocódigo consiste em determinar a posição de um objeto, atender visualmente ao objeto, executar movimento com a mão direita sobre ele e pressioná-lo com o dedo indicador por alguns instantes.

Pseudocódigo:

SEGURAR (objeto)

Atender visualmente objeto

Posicionar mão sobre o objeto

Segurar o objeto com o dedo

Retorna (ok)

Empurrar/puxar

O comportamento elementar de *Empurrar/puxar* consiste em deslocar um objeto para longe ou para perto do ser humano simulado. O pseudocódigo abaixo atende visualmente ao objeto, encontra a posição do objeto, posiciona as mãos do indivíduo simulado sobre o objeto, e finalmente desloca o objeto para a distância recebida como parâmetro.

Pseudocódigo:

EMPURRAR/PUXAR (objeto, distância)

Atender visualmente objeto

Determinar posição do objeto

Posicionar mão sobre o objeto

Deslocar o objeto aplicando “distância”

Retorna (ok)

Prender

O comportamento *Prender* objetiva fixar um objeto a outro. O pseudocódigo estabelece a determinação da posição do primeiro objeto e sua visualização. A seguir, ocorre a determinação do segundo objeto e também sua visualização. Finalmente ocorre o deslocamento do primeiro objeto em posição contígua ao segundo objeto.

Pseudocódigo:

PRENDER (objeto1, objeto2)

 Atender visualmente objeto1

 Determinar posição do objeto1

 Atender visualmente objeto2

 Determinar posição do objeto2

 Deslocar objeto1 para posição contigua a objeto2

Retorna (ok)

Dar e Devolver

O comportamento *Dar* significa disponibilizar um objeto a outro indivíduo e *Devolver* significa retornar ou dar de volta a seu proprietário aquilo que havia sido entregue. Ambos os comportamentos apresentam similaridade no que diz respeito a dar e retornar o objeto. Assim consideram-se, neste momento, que as especificações podem ser as mesmas. O pseudocódigo abaixo identifica visualmente um objeto, determina sua posição e desloca-o para nova localidade (dar) próximo à quem recebe tal objeto ou próximo a seu proprietário (devolver).

Pseudocódigo:

DAR (objeto1, posição1, objeto2, posição2) e DEVOLVER (objeto1, posição1, objeto2, posição2)

Atender visualmente objeto1

Determinar posição do objeto1

Atender visualmente objeto2

Determinar posição do objeto2

Deslocar objeto posição próxima de objeto2

Retorna (ok)

Remover e Descartar

O comportamento *Remover* tem por finalidade mover um objeto para fora de sua posição original e o comportamento de *Descartar* significa livrar-se de coisa inoportuna ou livrar-se de objeto desnecessário. Consideram-se estes comportamentos similares por este motivo o pseudocódigo abaixo atende ambos os comportamentos. Inicialmente ocorre a identificação visual do objeto e a determinação da sua posição em seguida o objeto é removido/descartado do contexto visual.

Pseudocódigo:

REMOVER (objeto) e DESCARTAR (objeto)

Atender visualmente objeto

Determinar posição do objeto

Remover/Descartar objeto

Retorna (ok)

Posicionar

O comportamento de *Posicionar* diz respeito à manipulação de um controle que possui estados discretos. A especificação deste comportamento atende visualmente ao objeto, determina sua localização, e movimenta-o de modo discreto para uma nova posição.

Pseudocódigo:

POSICIONAR (objeto, posição)

Atender visualmente objeto

Determinar posição do objeto

Movimentar de forma discreta 'objeto' para 'posição'

Retorna (ok)

Ajustar

O comportamento de *Ajustar* é uma variação do comportamento *Posicionar* anteriormente descrito. *Ajustar* refere-se à operação de um controle contínuo e sua especificação diz respeito a determinar a posição do objeto, atender visualmente a sua presença e movimenta-lo de modo contínuo para uma nova posição.

Pseudocódigo:

AJUSTAR (objeto, posição)

Atender visualmente objeto

Determinar posição do objeto

Movimentar de forma contínua 'objeto' para 'posição'

Retorna (ok)

Teclar

O comportamento elementar *Teclar* diz respeito ao ato de operar um teclado. A especificação apresentada simula o atendimento visual da tecla a ser pressionada, o posicionamento das mãos sobre o teclado e em seguida, com o dedo indicador da mão direita ocorre o pressionamento da tecla desejada.

Conhecimento declarativo:

(chunk1 isa teclar status inicio)

Onde:

chunk1: nome do *chunk*

status: sequência de operação

Pseudocódigo:

TECLAR (tecla)

Posicionar mão direita sobre o teclado

Pressionar *tecla* com o dedo indicador

Retorna (ok)

Instalar

O comportamento de *Instalar* simula a colocação de um objeto numa posição indicada. A especificação determina a posição inicial do objeto, atende visualmente ao objeto, recupera a posição para a instalação e, finalmente instala-o na posição indicada.

Pseudocódigo:

INSTALAR (objeto, posição)

 Atender visualmente objeto

 Determinar posição do objeto

 Instalar 'objeto' em 'posição'

Retorna (ok)

5.3.4. Implementação do Disparador

O módulo Disparador tem por finalidade informar ao Perturbador o momento oportuno para o disparo da perturbação. Isto significa dizer que nem todo tipo de erro poderá ocorrer a qualquer momento e em qualquer situação.

Conforme discutido na Seção 4.2.2.5, o Disparador deve verificar o estado atual do desempenho da tarefa e sortear um erro possível para aquela situação.

O S. PERERE permite que o usuário configure o mecanismo de disparo de duas formas:

1. Selecionando um tipo de erro, do nível de tarefa, para um comportamento específico. Por exemplo, o erro de omissão pode ser selecionado para o comportamento de teclar. Esta opção de funcionamento determina ao S. PERERE o sorteio de um conhecimento da memória declarativa, obviamente que seja composto pela ação motora de teclar, e produz a omissão do referido passo da tarefa;
2. Selecionando a opção para que todos os possíveis erros associados à tarefa simulada sejam gerados. Isto é realizado a partir da relação de erros associados a cada comportamento discutidos neste trabalho. A relação de erros associados a cada um dos comportamentos especificados neste trabalho encontra-se no Anexo B.

É importante considerar que para cada tipo de 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. Isto significa observar que, numa perspectiva simplificada, no caso de um comportamento possuir quatro erros associados a ele, o S. PERERE gerará quatro arquivos ACT-R, um para cada tipo de erro. Analisando sob a ótica da simulação de uma tarefa completa, pode ocorrer o caso em que a tarefa seja composta por três comportamentos elementares e associados a estes comportamentos existam três erros para cada um deles. Assim, o S. PERERE gerará nove arquivos ACT-R, com as respectivas perturbações.

5.5. Considerações finais

Neste capítulo foram discutidos os seguintes tópicos:

- A arquitetura cognitiva ACT-R, que apóia a construção do S. PERERE, e como ela interage com os componentes da arquitetura;
- As especificações genéricas dos comportamentos elementares, com os quais o usuário define a tarefa humana a ser simulada, com seus respectivos códigos em ACT-R, de cada um dos comportamentos elementares da taxonomia de Berliner; Angell; Shearer (1964).

A partir da discussão e especificação dos comportamentos elementares realizados neste capítulo é que apresenta-se, a seguir, um exemplo visando apresentar o funcionamento do simulador construído.

6. EXEMPLO DE APLICAÇÃO DO S. PERERE NO ALINHAMENTO DE ROTAS NO METRÔ DE SÃO PAULO

Este capítulo objetiva apresentar um exemplo de aplicação do S. PERERE, visando mostrar sua capacidade de geração de erros humanos. O exemplo selecionado foi baseado em um estudo de caso da operação de alinhamento de rotas no metrô de São Paulo, especificamente da Linha 1 – Azul.

O exemplo é apresentado em duas etapas. Inicialmente, apresenta-se a geração do comportamento humano correto, evidenciando a capacidade do S. PERERE obter um modelo ACT-R/PM executável a partir de uma descrição de tarefas feita com a taxonomia de Berliner. Com este exemplo, pretende-se mostrar como o usuário do S. PERERE deve escrever a tarefa a ser simulada. Na descrição do exemplo, apresenta-se a saída do ACT-R/PM correspondente à execução especificada da tarefa, denominada a seguir de “caso correto”, porque não é afetada de qualquer tipo de erro.

Em seguida, o mesmo exemplo é usado para mostrar a capacidade do S. PERERE perturbar a execução da tarefa. A saída, neste caso, é a mesma tarefa do exemplo anterior, porém afetada de erros. Denomina-se, aqui, este tipo de resultado de “caso perturbado”.

O objetivo do exemplo é mostrar a aplicabilidade do S. PERERE, assim o capítulo se encerra com uma discussão dos resultados em face dos propósitos do simulador.

6.1. A tarefa-exemplo: alinhamento de rota no Metrô-SP

A atividade usada como exemplo neste trabalho é uma tarefa de alinhamento de rotas no Metrô de São Paulo, mais especificamente na Linha 1 - Azul, no sistema de controle existente no Metrô no ano de 1996. O alinhamento de rotas é a operação de se estabelecer um determinado caminho para a circulação de trens.

Esta tarefa foi descrita por Filgueiras (1996) como estudo de caso em seu trabalho de desenvolvimento de método para análise e projeto de interfaces homem-computador, com o foco na confiabilidade humana.

A decisão de se usar esta tarefa específica reside no fato de que ela é composta por unidades de comportamentos que representam os processos cognitivo, perceptivo e motor. Além disso, a especificação de Filgueiras, para o alinhamento de rotas, tem o foco na descrição de uma interface homem-computador o que interessa diretamente aos propósitos do S. PERERE.

Sabe-se que, atualmente, alguns procedimentos para o alinhamento de rotas em questão foram modificados e que a descrição a seguir não representa o que hoje é realizado. Para efeito deste estudo de caso, esse fato torna-se irrelevante, pois o foco aqui é a simulação do desempenho humano associado a uma tarefa de interação homem-computador.

Conforme esclarece Filgueiras (1996), o alinhamento de rotas é um procedimento automatizado. Isto significa que, para a circulação comum dos trens, as rotas são previamente programadas. Em boa parte dos casos, o alinhamento pelo operador tem por objetivo atender a condições de exceção como proporcionar a entrada de um novo trem na via ou o recolhimento do mesmo, forçar a passagem do trem por outra via de sentido contrário, evitando um trecho em manutenção, ou ainda antecipar uma condição desejada de manobras.

6.2. Descrição do procedimento operacional real do alinhamento de rotas

Dada a meta do operador, de estabelecer um caminho para a circulação dos trens, o entendimento da tarefa depende da compreensão do meio para seu cumprimento. É importante, pois, descrever a interface que o operador manipula. Esta descrição refere-se ao ambiente real de operação.

A circulação de trens é monitorada pelo operador através de um painel sinótico, onde os trechos ocupados por trens são representados por leds acesos em vermelho (ocupado) ou verde (livre) e também por uma tela onde estes mesmos trechos são representados com o detalhe da posição das chaves de mudança de via e dos semáforos (“bloqueios”) que precedem os trechos onde estão estas chaves.

A atuação sobre as chaves é feita através de um teclado onde há teclas especiais, associadas às funções, além das demais teclas de um teclado QWERTY comum. Os comandos digitados neste teclado são exibidos no monitor, na forma de uma linha de comando com campos pré-definidos.

O procedimento operacional do alinhamento de rotas consiste, inicialmente, num conjunto de verificações, por parte do operador, sobre a condição atual do intertravamento. Denomina-se intertravamento ao trecho onde existe uma chave de mudança de via. A primeira verificação a ser realizada pelo operador consiste em checar o estado atual de ocupação do intertravamento que pode assumir o estado *livre* (preenchimento verde) ou o estado *ocupado* (preenchimento vermelho). Em seguida ele também verifica o modo de operação que pode assumir um dos seguintes estados: *central*, onde o operador tem o controle sobre o alinhamento; ou *automático*, isto é, o controle é do sistema. O operador deve garantir que o sistema esteja no estado *central*. O operador deve verificar mais duas situações: o bloqueio (semáforo) de saída que pode assumir o estado *normal* ou *proibido* e o bloqueio de entrada que também pode assumir o estado *normal* ou *proibido*. Estas verificações são importantes, pois são as que possibilitam, no caso normal, a continuidade do processo de alinhamento de rota. Por caso normal entende-se as situações verificadas anteriormente e resumidas na Tabela 10:

Objeto	Valor
Ocupação do intertravamento	Livre
Modo de operação	Central
Bloqueio de saída	Normal
Bloqueio de entrada	Normal

Tabela 10 - Caso normal para o estado inicial do alinhamento de rotas

Após as verificações descritas acima, o operador inicia um conjunto de ações motoras sobre o teclado funcional. O operador pressiona a tecla ROTA PEDE, tecla de função do teclado especial, para indicar o comando de alinhamento de rota. O acionamento desta tecla faz com que o sistema exiba, no terminal de vídeo, um cabeçalho para o preenchimento de parâmetros:

	XL-ENT	BLQ	XL-SAI	BLQ	MODO
ROTA PEDE	???	??	???	??	?

Em seguida, o operador inicia a digitação dos parâmetros que proporcionarão o alinhamento da rota desejada, indicados na interface por campos com pontos de interrogação.

Inicialmente o operador deve digitar o nome do intertravamento de entrada, XL-ENT, com três caracteres. Em seguida, são digitados dois caracteres referentes ao bloqueio de entrada, BLQ. O primeiro caractere de BLQ é a origem do intertravamento e o segundo é o bloqueio de saída da rota naquele intertravamento. Com relação ao campo XL-SAI, intertravamento de saída, são digitados três caracteres que indicam o destino do trem. O bloqueio de saída, BLQ, com dois caracteres, faz referência ao bloqueio da rota naquele intertravamento.

O operador necessita pressionar a tecla MODO indicando se o alinhamento deverá continuar após a passagem do trem ou se é alinhamento temporário. Após a alimentação do sistema com os dados que solicitam o alinhamento de rotas, o operador pressiona a tecla ENTER que submete a transação. Finalmente, ele monitora, durante certo tempo, a interface, acompanhando a efetivação do alinhamento de rota, que se dá pela mudança de posição da chave de desvio, para a nova rota solicitada.

A Tabela 11 resume o procedimento, explicado nesta seção, para o alinhamento da rota Linha 1 – Azul.

Seq.	Tarefa	Objeto	Atributo	Valor
01	Verifica	Intertravamento	Estado inicial	Verde/vermelho
02	Verifica	Intertravamento	Modo de operação	Central/automático
03	Verifica	Intertravamento	Bloqueio de saída	Normal/proibido
04	Verifica	Intertravamento	Bloqueio de entrada	Normal/proibido
05	Tecla	Comando	ROTA – PEDE	-
06	Tecla	Intertravamento	Identificação Entrada (XL – ENT)	-
07	Tecla	Intertravamento	Identif. bloqueio Entrada (BLQ)	-
08	Tecla	Intertravamento	Identificação Saída (XL – SAI)	-
09	Tecla	Intertravamento	Identif. bloqueio Saída (BLQ)	-
10	Tecla	Rota	Modo de alinhamento (MOD0)	Contínuo/Temporário
11	Tecla	Comando	ENTER	-
12	Monitora	Intertravamento	Estado final	Alinhada/ não alinhada

Tabela 11 - Descrição dos procedimentos para o alinhamento de rota – Linha 1 – Azul (adaptado de Filgueiras, 1996, p.213)

6.3 Simulação do operador alinhando a rota Linha 1 – Azul – caso correto

Esta seção tem por objetivo apresentar a simulação do operador humano, na interação com o sistema de alinhamento da rota, conforme a especificação do S. PERERE.

Supõe-se, nesta implementação, que o S. PERERE comunique-se com uma interface que reproduza as condições de monitoração e entrada de dados do problema. Esta interface é um ambiente de teste para o S. PERERE e consiste na Figura 10 abaixo:

Fig. 10 - Cabeçalho para o preenchimento de parâmetros (Adaptado de Filgueiras, 1996, p.211)

O operador simulado pelo S. PERERE desempenha comportamento relacionado na Tabela 11 verificando inicialmente os quatro estados do intertravamento, representados pelas Sequências de 01 a 04. Em seguida aciona a tecla ROTA – PEDE, explicitada na Sequência 05. A seguir, conforme ilustrado na Figura 10, desenvolve um conjunto de ações motoras de teclar para as entradas: XL-ENT, BLQ, XL-SAI e BLQ representadas pelas Sequências de 06 a 09 respectivamente. O acionamento da tecla Modo também é necessário, e corresponde à Sequência 10. Para submeter o alinhamento ao sistema, o operador aciona a tecla Enter na Sequência 11. Finalmente, o operador monitora as condições finais do procedimento executado, Sequência 12.

O usuário, para descrever uma tarefa a ser simulada pelo S. PERERE deve descrever o conhecimento declarativo para os comportamentos que se deseja simular, usando a taxonomia de Berliner. No caso deste exemplo os comportamentos simulados são: verificar, teclar e monitorar.

6.3.1 Construção da memória declarativa para os objetos

O conhecimento do operador, necessário para a realização da tarefa de alinhar a rota está armazenado na memória declarativa. Revisitando a Seção 4.2.2.1, onde discutiu-se a memória declarativa, é importante destacar que os *chunks* são destinados a representação do conhecimento que o indivíduo possui para resolver um problema.

Nesta específica aplicação, os objetos manipulados pelo operador são:

- estado inicial do intertravamento,
- estado do modo de operação do intertravamento
- estado do bloqueio de saída do intertravamento
- estado do bloqueio de entrada do intertravamento
- tecla ROTA – PEDE
- entrada de dados da Identificação de Entrada, da Identificação do Bloqueio de Entrada, da Identificação de Saída, da Identificação do Bloqueio de Saída e do Modo de Alinhamento
- tecla Enter
- e finalmente estado final do intertravamento.

Além destes objetos, são necessários os seguintes tipos de *chunks* para o exemplo:

(chunk-type verificar objeto atributo sequencia)

Este tipo de *chunk* especifica o uso de uma categoria de conhecimento na taxonomia de Berliner, neste caso é Verificar. O *slot* objeto diz respeito ao objeto (estados do intertravamento) que deverá ser verificado. O *slot* atributo armazena o estado do objeto e finalmente o *slot* sequência determina a sequência de execução de cada passo do comportamento verificar.

(chunk-type tecla objeto atributo sequencia)

O tipo de *chunk* tecla especifica o conhecimento sobre o comportamento motor do operador simulado. O *slot* objeto diz respeito a tecla a ser pressionada, o atributo armazena o estado atual da ação e o *slot* sequência representa a execução de cada passo deste comportamento.

(chunk-type monitor atributo tempo)

O tipo de *chunk* monitor especifica o conhecimento sobre o comportamento cognitivo do operador simulado de monitorar. O *slot* atributo representa o objeto a ser monitorado e por último, o *slot* tempo especifica o tempo total para a monitoração do objeto.

6.3.2 Construção da memória declarativa para a tarefa – caso correto

A tarefa de alinhamento de rota precisa ser descrita ao S. PERERE, para que ele possa afetá-la de erros. A tarefa deve ser descrita usando-se os termos da taxonomia de Berliner.

O conjunto de *chunks* listado abaixo representa o conhecimento declarativo do operador para a realização da tarefa de alinhar a rota Linha 1 – Azul. Para a inserção de *chunks* deve-se utilizar a palavra reservada *add-dm*, em seguida adiciona-se o respectivo conhecimento declarativo, gerando um arquivo texto.

(add-dm

(verificar1 isa verificar atributo prep_mod sequencia 1)

Atributo: verificar o estado inicial do intertravamento

Seqüência: corresponde a ordem de execução

(verificar2 isa verificar atributo prep_mod sequencia 2)

Atributo: verificar o estado do modo de operação

(verificar3 isa verificar atributo prep_mod sequencia 3)

Atributo: verificar o estado do bloqueio de saída

(verificar4 isa verificar atributo prep_mod sequencia 4)

Atributo: verificar o estado do bloqueio de entrada

(tarefa5 isa tecla objeto comando atributo "a")

Chunk relativo ao conhecimento para a digitação do comando ROTA – PEDE, conforme anunciado anteriormente, representa-se este comando pelo pressionamento da tecla *a*.

(tarefa6 isa tecla objeto intertravamento atributo "x-31")

Chunk relativo ao conhecimento para a digitação da Identificação de Entrada, neste exemplo, representa-se este conhecimento com o valor *x-31*.

(tarefa7 isa tecla objeto intertravamento atributo "be")

Chunk relativo ao conhecimento para a digitação do Bloqueio de Entrada, neste exemplo, representa-se este conhecimento com o valor *be*.

(tarefa8 isa tecla objeto intertravamento atributo "y52")

Chunk relativo ao conhecimento para a digitação da Identificação de Saída, neste exemplo, representa-se este conhecimento com o valor *y52*.

(tarefa9 isa tecla objeto intertravamento atributo "bs")

Chunk relativo ao conhecimento para a digitação do Bloqueio de Saída, neste exemplo, representa-se este conhecimento com o valor *bs*.

(tarefa10 isa tecla objeto rota atributo "c")

Chunk relativo ao conhecimento para a digitação do Modo de Alinhamento, neste exemplo, representa-se este conhecimento com o valor *c*.

(tarefa11 isa tecla objeto executa atributo "enter")

Chunk relativo ao conhecimento para a digitação do Comando ENTER.

(monitorar1 isa monitor atributo ver_obj)

(monitorar2 isa monitor atributo prep_ver)

(monitorar3 isa monitor atributo prep_conc)

(monitorar4 isa monitor atributo ok)

(monitorar5 isa monitor atributo aguarde tempo 13700)

Com relação aos *chunks* do tipo monitor (monitorar1 a 5), tem-se o que segue, na ordem: conhecimento visual preparando-se para ver o objeto a monitorar; teste do módulo visual; visualização do objeto; monitoração do objeto; marcação de tempo para o monitoramento.

Baseado nas especificações genéricas dos comportamentos elementares da taxonomia de Berliner; Angell; Shearer (1964), descritos nas seções de 5.4.1 a 5.4.3, construiu-se as produções geradoras dos comportamentos utilizados para a simulação do alinhamento da rota. De acordo com a Tabela 11, os comportamentos agregados à realização da tarefa são: verificar, teclar e monitorar.

6.3.3 Execução do modelo

Para executar o exemplo de aplicação, o analista deve seguir os seguintes passos:

- a) Executar o programa S. PERERE, ilustrado na Figura 11, escolhendo a opção "Gerador de Modelo Correto";
- b) Atribuir nome para o arquivo de saída;
- c) Escolher opção "Modelo com Monitoramento";
- d) Selecionar o botão "Gerar";

Fig. 11- Interface do S. PERERE

- e) Após a geração do arquivo, modo correto, carregar o ACT-R e abrir o arquivo gerado pelo S. PERERE, conforme ilustrado na Figura 12;
- f) Executar o arquivo gerado pelo S. PERERE digitando: (inicio) na linha de comando.

Fig. 12 - Interface do ACT-R

6.3.4 Resultados da execução do caso correto

A partir deste ponto lista-se, em *itálico*, a saída gerada pelo S. PERERE do comportamento correto para o alinhamento de rota. A saída listada e a explicação logo a seguir, dizem respeito a cada item da Sequência da Tabela 11. Para melhor entendimento, procurou-se adicionar algumas explicações extras no corpo da saída, as quais estão descritas em letras maiúsculas e em **negrito**. Esta saída denominada de *trace* é produzida pela execução do ACT-R sobre o comportamento simulado e as

mensagens maiúsculas e em negrito foram incorporadas à programação do exemplo para destacar a ocorrência dos eventos gerados pela simulação.

Observe-se que em alguns trechos da saída aparece a palavra reservada *Window* que representa o processo perceptivo simulado pelo ACT-R/PM.

A saída abaixo é o resultado do comportamento **verificar** o qual desempenha a checagem do *estado inicial do intertravamento*, representado na Tabela 11 pela Sequência 01. Os tempos listados a seguir são aqueles produzidos pelo ACT-R, que considera que cada ciclo da cognição humana é completado depois de decorridos aproximadamente 50ms (v. Seção 5.2). Observa-se que este tempo é variável e indica o início e o término de cada uma das atividades executadas pelo S. PERERE.

Na saída abaixo observe-se a presença de palavras reservadas. Estes termos indicam qual o módulo que está sendo utilizado naquele momento (por exemplo, no tempo 0.000 o módulo perceptivo está ativo), qual a produção que está sendo selecionada (no tempo 0.000, a produção *Intertravamento_passo_1*) e em seguida disparada (tempo 0.050), etc. As Seções 5.3.3 e 5.3.4 que tratam dos módulos perceptivo e motor respectivamente apresentam discussão sobre estes termos.

```
Time 0.000: Vision found LOC19
Time 0.000: Intertravamento_Passo_1 Selected
Time 0.050: Intertravamento_Passo_1 Fired
Time 0.050: Module :VISION running command FIND-LOCATION
Time 0.050: Vision found LOC19
Time 0.050: Intertravamento_Passo_2 Selected
Time 0.100: Intertravamento_Passo_2 Fired
Time 0.100: Module :VISION running command MOVE-ATTENTION
Time 0.185: Module :VISION running command ENCODING-COMPLETE
Time 0.185: Vision sees TEXT18
Time 0.235: Intertravamento_Verificou_Verde Selected
VERIFICOU INTERTRAVAMENTO --> VERDE
Time 0.235: Intertravamento_Verificou_Verde Fired
Time 0.285: Goal21 Retrieved
```

Uma vez que o estado inicial do intertravamento é *verde*, o operador pode continuar o procedimento. O próximo passo é verificar o *modo de operação* do sistema de alinhamento, apresentado na Tabela 11 pela Sequência 02. A saída abaixo representa esta situação:

```
Time 0.285: Intertravamento_Tarefa1 Selected
Merging chunk Goal21 into chunk Tarefa1
Time 0.335: Intertravamento_Tarefa1 Fired
```

```

Time 0.335: Modo_De_Operacao_Passo_1 Selected
Time 0.385: Modo_De_Operacao_Passo_1 Fired
Time 0.385: Module :VISION running command FIND-LOCATION
Time 0.385: Vision found LOC28
Time 0.420: Module :VISION running command ENCODING-COMplete
Time 0.420: Modo_De_Operacao_Passo_2 Selected
Time 0.470: Modo_De_Operacao_Passo_2 Fired
Time 0.470: Module :VISION running command MOVE-ATTENTION
Time 0.555: Module :VISION running command ENCODING-COMplete
Time 0.555: Vision sees TEXT26
Time 0.555: Modo_De_Operacao_Verificou_Automatico Selected
VERIFICOU MODO_DE_OPERACAO --> AUTOMATICO
Time 0.605: Modo_De_Operacao_Verificou_Automatico Fired
Time 0.605: Modo_De_Operacao_Passo_1 Selected
Time 0.655: Modo_De_Operacao_Passo_1 Fired
Time 0.655: Module :VISION running command FIND-LOCATION
Time 0.655: Vision found LOC33
Time 0.655: Modo_De_Operacao_Passo_2 Selected
Time 0.705: Modo_De_Operacao_Passo_2 Fired
Time 0.705: Module :VISION running command MOVE-ATTENTION
Time 0.790: Module :VISION running command ENCODING-COMplete
Time 0.790: Vision sees TEXT27
Time 0.840: Modo_De_Operacao_Verificou_Central Selected
Merging chunk Goal30 into chunk Goal22
VERIFICOU MODO_DE_OPERACAO --> CENTRAL
Time 0.840: Modo_De_Operacao_Verificou_Central Fired
Time 0.890: Goal35 Retrieved

```

É importante observar que no tempo 0.555, o operador verificou que o modo de operação estava em *automático*. Assim, no tempo 0.605 foi disparada produção que conduz à nova verificação do Modo de Operação. No tempo 0.840 o operador simulado viu o novo estado do Modo de Operação que assumiu *central* e que, de acordo com o caso normal para o alinhamento, apresentado na Tabela 10, o bloqueio de saída deve ser verificado.

O próximo passo é o de verificar o *bloqueio de saída*, representado pela Sequência 03 na Tabela 11. A saída abaixo representa esta situação:

```

Time 0.890: Modo_De_Operacao_Tarefa2 Selected
Merging chunk Goal35 into chunk Tarefa2
Time 0.940: Modo_De_Operacao_Tarefa2 Fired
Time 0.940: Bloqueio_De_Saida_Passo_1 Selected
Time 0.990: Bloqueio_De_Saida_Passo_1 Fired
Time 0.990: Module :VISION running command FIND-LOCATION
Time 0.990: Vision found LOC42
Time 1.025: Module :VISION running command ENCODING-COMplete
Time 1.025: Bloqueio_De_Saida_Passo_2 Selected
Time 1.075: Bloqueio_De_Saida_Passo_2 Fired
Time 1.075: Module :VISION running command MOVE-ATTENTION
Time 1.160: Module :VISION running command ENCODING-COMplete
Time 1.160: Vision sees TEXT40
Time 1.210: Bloqueio_De_Saida_Verificou_Normal Selected

```

VERIFICOU BLOQUEIO_DE_SAIDA --> NORMAL

Time 1.210: Bloqueio_De_Saida_Verificou_Normal Fired
 Time 1.260: Goal44 Retrieved

Observa-se que o *bloqueio de saída* foi verificado com o valor *normal*, no tempo 1.210. Isto conduz o operador para a última verificação que é o *bloqueio de entrada* (Sequência 04 na Tabela 11).

Time 1.260: Bloqueio_De_Saida_Tarefa3 Selected
 Merging chunk Goal44 into chunk Tarefa3
 Time 1.310: Bloqueio_De_Saida_Tarefa3 Fired
 Time 1.310: Bloqueio_De_Entrada_Passo_1 Selected
 Time 1.360: Bloqueio_De_Entrada_Passo_1 Fired
 Time 1.360: Module :VISION running command FIND-LOCATION
 Time 1.360: Vision found LOC51
 Time 1.395: Module :VISION running command ENCODING-COMPLETE
 Time 1.395: Bloqueio_De_Entrada_Passo_2 Selected
 Time 1.445: Bloqueio_De_Entrada_Passo_2 Fired
 Time 1.445: Module :VISION running command MOVE-ATTENTION
 Time 1.530: Module :VISION running command ENCODING-COMPLETE
 Time 1.530: Vision sees TEXT50
 Time 1.530: Bloqueio_De_Entrada_Verificou_Proibido2 Selected
VERIFICOU BLOQUEIO_DE_ENTRADA --> PROIBIDO2
 Time 1.580: Bloqueio_De_Entrada_Verificou_Proibido2 Fired
 Time 1.580: Bloqueio_De_Entrada_Passo_1 Selected
 Time 1.630: Bloqueio_De_Entrada_Passo_1 Fired
 Time 1.630: Module :VISION running command FIND-LOCATION
 Time 1.630: Vision found LOC56
 Time 1.630: Bloqueio_De_Entrada_Passo_2 Selected
 Time 1.680: Bloqueio_De_Entrada_Passo_2 Fired
 Time 1.680: Module :VISION running command MOVE-ATTENTION
 Time 1.765: Module :VISION running command ENCODING-COMPLETE
 Time 1.765: Vision sees TEXT49
 Time 1.815: Bloqueio_De_Entrada_Verificou_Normal2 Selected
 Merging chunk Goal53 into chunk Goal45
VERIFICOU BLOQUEIO_DE_ENTRADA --> NORMAL2
 Time 1.815: Bloqueio_De_Entrada_Verificou_Normal2 Fired
 Time 1.865: Goal58 Retrieved

No tempo 1.530 observa-se que o operador verificou que o *bloqueio de entrada* estava *proibido*. Esta situação o conduziu ao disparo de regra de produção, no tempo 1.580 para nova verificação deste bloqueio. Em 1.815 foi disparada regra para o bloqueio de estado *normal*, e no tempo 1.865 a meta foi atualizada para a execução do novo comportamento.

A seguir o operador aciona a tecla ROTA – PEDE (Sequência 05 na Tabela 11) o que garante o início da digitação dos parâmetros para o alinhamento da rota.

```

Time 1.865: Bloqueio_De_Entrada_Tarefa4 Selected
Merging chunk Goal58 into chunk Tarefa4
Time 1.915: Bloqueio_De_Entrada_Tarefa4 Fired
Time 1.915: Module :MOTOR running command PUNCH
Time 1.965: Module :MOTOR running command PREPARATION-COMPLETE
Time 2.015: Module :MOTOR running command INITIATION-COMPLETE
Time 2.025: Device running command OUTPUT-KEY
<< Window "Acionando Rota - Pede" got key #\a at time 2025 >>
Time 2.115: Module :MOTOR running command FINISH-MOVEMENT

```

Em 2.025 o operador pressiona a tecla *a* com o estilo de movimento *punch* (v. Seção 5.3.4) simulando o comando ROTA – PEDE.

Em seguida, inicia-se a digitação do parâmetro com o nome do intertravamento de entrada, com três caracteres (Sequência 06 da Tabela 11). Nesta simulação, adota-se que será digitado o valor *X3I* para o campo XL – ENT. A saída abaixo mostra a digitação do caractere *X*, *3* e *I* respectivamente nos tempos 2.825, 3.712 e 4.544.

```

Time 2.115: Teclar_Id_Ent_Mover_1 Selected
Merging chunk Goal59 into chunk Tarefa6
Time 2.165: Teclar_Id_Ent_Mover_1 Fired
Time 2.165: Module :MOTOR running command POINT-HAND-AT-KEY
Time 2.365: Module :MOTOR running command PREPARATION-COMPLETE
Time 2.415: Module :MOTOR running command INITIATION-COMPLETE
Time 2.515: Module :MOTOR running command MOVE-A-HAND
Time 2.565: Module :MOTOR running command FINISH-MOVEMENT
Time 2.565: Teclar_Id_Ent_Tecla_1 Selected
Time 2.615: Teclar_Id_Ent_Tecla_1 Fired
Time 2.615: Module :MOTOR running command PUNCH
Time 2.765: Module :MOTOR running command PREPARATION-COMPLETE
Time 2.815: Module :MOTOR running command INITIATION-COMPLETE
Time 2.825: Device running command OUTPUT-KEY
<< Window "Identificação de Entrada" got key #\x at time 2825 >>
Time 2.915: Module :MOTOR running command FINISH-MOVEMENT
Time 2.915: Teclar_Id_Ent_Mover_2 Selected
Time 2.965: Teclar_Id_Ent_Mover_2 Fired
Time 2.965: Module :MOTOR running command POINT-HAND-AT-KEY
Time 3.165: Module :MOTOR running command PREPARATION-COMPLETE
Time 3.215: Module :MOTOR running command INITIATION-COMPLETE
Time 3.402: Module :MOTOR running command MOVE-A-HAND
Time 3.452: Module :MOTOR running command FINISH-MOVEMENT
Time 3.452: Teclar_Id_Ent_Tecla_2 Selected
Time 3.502: Teclar_Id_Ent_Tecla_2 Fired
Time 3.502: Module :MOTOR running command PUNCH
Time 3.652: Module :MOTOR running command PREPARATION-COMPLETE
Time 3.702: Module :MOTOR running command INITIATION-COMPLETE
Time 3.712: Device running command OUTPUT-KEY
<< Window "Identificação de Entrada" got key #\3 at time 3712 >>
Time 3.802: Module :MOTOR running command FINISH-MOVEMENT
Time 3.802: Teclar_Id_Ent_Mover_3 Selected
Time 3.852: Teclar_Id_Ent_Mover_3 Fired

```

```

Time 3.852: Module :MOTOR running command POINT-HAND-AT-KEY
Time 4.052: Module :MOTOR running command PREPARATION-COMPLETE
Time 4.102: Module :MOTOR running command INITIATION-COMPLETE
Time 4.234: Module :MOTOR running command MOVE-A-HAND
Time 4.284: Module :MOTOR running command FINISH-MOVEMENT
Time 4.284: Teclar_Id_Ent_Tecla_3 Selected
Time 4.334: Teclar_Id_Ent_Tecla_3 Fired
Time 4.334: Module :MOTOR running command PUNCH
Time 4.484: Module :MOTOR running command PREPARATION-COMPLETE
Time 4.534: Module :MOTOR running command INITIATION-COMPLETE
Time 4.544: Device running command OUTPUT-KEY
<< Window "Identificação de Entrada" got key #\1 at time 4544 >>
Time 4.634: Module :MOTOR running command FINISH-MOVEMENT

```

A seguir passa-se a digitação, pelo operador simulado, da *identificação do bloqueio de entrada* com dois caracteres (Seqüência 07 da Tabela 11). Assume-se que o valor digitado para este campo será *BE*. Estes valores ocorrem nos tempos 5.490 e 6.363 respectivamente.

```

Time 4.634: Teclar_Id_Blq_Ent_Mover_1 Selected
Time 4.684: Teclar_Id_Blq_Ent_Mover_1 Fired
Time 4.684: Module :MOTOR running command POINT-HAND-AT-KEY
Time 4.884: Module :MOTOR running command PREPARATION-COMPLETE
Time 4.934: Module :MOTOR running command INITIATION-COMPLETE
Time 5.180: Module :MOTOR running command MOVE-A-HAND
Time 5.230: Module :MOTOR running command FINISH-MOVEMENT
Time 5.230: Teclar_Id_Blq_Ent_Tecla_1 Selected
Time 5.280: Teclar_Id_Blq_Ent_Tecla_1 Fired
Time 5.280: Module :MOTOR running command PUNCH
Time 5.430: Module :MOTOR running command PREPARATION-COMPLETE
Time 5.480: Module :MOTOR running command INITIATION-COMPLETE
Time 5.490: Device running command OUTPUT-KEY
<< Window "Bloqueio de Entrada" got key #\b at time 5490 >>
Time 5.580: Module :MOTOR running command FINISH-MOVEMENT
Time 5.580: Teclar_Id_Blq_Ent_Mover_2 Selected
Time 5.630: Teclar_Id_Blq_Ent_Mover_2 Fired
Time 5.630: Module :MOTOR running command POINT-HAND-AT-KEY
Time 5.830: Module :MOTOR running command PREPARATION-COMPLETE
Time 5.880: Module :MOTOR running command INITIATION-COMPLETE
Time 6.053: Module :MOTOR running command MOVE-A-HAND
Time 6.103: Module :MOTOR running command FINISH-MOVEMENT
Time 6.103: Teclar_Id_Blq_Ent_Tecla_2 Selected
Time 6.153: Teclar_Id_Blq_Ent_Tecla_2 Fired
Time 6.153: Module :MOTOR running command PUNCH
Time 6.303: Module :MOTOR running command PREPARATION-COMPLETE
Time 6.353: Module :MOTOR running command INITIATION-COMPLETE
Time 6.363: Device running command OUTPUT-KEY
<< Window "Bloqueio de Entrada" got key #\e at time 6363 >>
Time 6.453: Module :MOTOR running command FINISH-MOVEMENT

```

O operador passa a digitar a *identificação de saída*, XL – SAI, com três caracteres (Seqüência 08 da Tabela 11). Adotou-se que os valores digitados são Y52.

A digitação de cada um destes caracteres ocorre nos seguintes tempos: 7.244, 8.044 e 8.925.

```

Time 6.453: Teclar_Id_Saida_Mover_1 Selected
Time 6.503: Teclar_Id_Saida_Mover_1 Fired
Time 6.503: Module :MOTOR running command POINT-HAND-AT-KEY
Time 6.703: Module :MOTOR running command PREPARATION-COMPLETE
Time 6.753: Module :MOTOR running command INITIATION-COMPLETE
Time 6.934: Module :MOTOR running command MOVE-A-HAND
Time 6.984: Module :MOTOR running command FINISH-MOVEMENT
Time 6.984: Teclar_Id_Saida_Tecla_1 Selected
Time 7.034: Teclar_Id_Saida_Tecla_1 Fired
Time 7.034: Module :MOTOR running command PUNCH
Time 7.184: Module :MOTOR running command PREPARATION-COMPLETE
Time 7.234: Module :MOTOR running command INITIATION-COMPLETE
Time 7.244: Device running command OUTPUT-KEY
<< Window "Identificação de saída" got key #\y at time 7244 >>
Time 7.334: Module :MOTOR running command FINISH-MOVEMENT
Time 7.334: Teclar_Id_Saida_Mover_2 Selected
Time 7.384: Teclar_Id_Saida_Mover_2 Fired
Time 7.384: Module :MOTOR running command POINT-HAND-AT-KEY
Time 7.584: Module :MOTOR running command PREPARATION-COMPLETE
Time 7.634: Module :MOTOR running command INITIATION-COMPLETE
Time 7.734: Module :MOTOR running command MOVE-A-HAND
Time 7.784: Module :MOTOR running command FINISH-MOVEMENT
Time 7.784: Teclar_Id_Saida_Tecla_2 Selected
Time 7.834: Teclar_Id_Saida_Tecla_2 Fired
Time 7.834: Module :MOTOR running command PUNCH
Time 7.984: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.034: Module :MOTOR running command INITIATION-COMPLETE
Time 8.044: Device running command OUTPUT-KEY
<< Window " Identificação de saída " got key #\5 at time 8044 >>
Time 8.134: Module :MOTOR running command FINISH-MOVEMENT
Time 8.134: Teclar_Id_Saida_Mover_3 Selected
Time 8.184: Teclar_Id_Saida_Mover_3 Fired
Time 8.184: Module :MOTOR running command POINT-HAND-AT-KEY
Time 8.384: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.434: Module :MOTOR running command INITIATION-COMPLETE
Time 8.615: Module :MOTOR running command MOVE-A-HAND
Time 8.665: Module :MOTOR running command FINISH-MOVEMENT
Time 8.665: Teclar_Id_Saida_Tecla_3 Selected
Time 8.715: Teclar_Id_Saida_Tecla_3 Fired
Time 8.715: Module :MOTOR running command PUNCH
Time 8.865: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.915: Module :MOTOR running command INITIATION-COMPLETE
Time 8.925: Device running command OUTPUT-KEY
<< Window " Identificação de saída " got key #\2 at time 8925 >>
Time 9.015: Module :MOTOR running command FINISH-MOVEMENT

```

Em seguida o operador simulado executa a digitação da *identificação do bloqueio de saída* com dois caracteres (Sequência 09 da Tabela 11). Assume-se que o valor digitado para este campo será *BS*. Estes valores ocorrem nos tempos 9.850 e 10.737 respectivamente.

```

Time 9.015: Teclar_Id_Blq_Saida_Mover_1 Selected
Time 9.065: Teclar_Id_Blq_Saida_Mover_1 Fired
Time 9.065: Module :MOTOR running command POINT-HAND-AT-KEY
Time 9.265: Module :MOTOR running command PREPARATION-COMPLETE
Time 9.315: Module :MOTOR running command INITIATION-COMPLETE
Time 9.540: Module :MOTOR running command MOVE-A-HAND
Time 9.590: Module :MOTOR running command FINISH-MOVEMENT
Time 9.590: Teclar_Id_Blq_Saida_Tecla_1 Selected
Time 9.640: Teclar_Id_Blq_Saida_Tecla_1 Fired
Time 9.640: Module :MOTOR running command PUNCH
Time 9.790: Module :MOTOR running command PREPARATION-COMPLETE
Time 9.840: Module :MOTOR running command INITIATION-COMPLETE
Time 9.850: Device running command OUTPUT-KEY
<< Window "Verificando Blq de Entrada" got key #\b at time 9850 >>
Time 9.940: Module :MOTOR running command FINISH-MOVEMENT
Time 9.940: Teclar_Id_Blq_Saida_Mover_2 Selected
Time 9.990: Teclar_Id_Blq_Saida_Mover_2 Fired
Time 9.990: Module :MOTOR running command POINT-HAND-AT-KEY
Time 10.190: Module :MOTOR running command PREPARATION-COMPLETE
Time 10.240: Module :MOTOR running command INITIATION-COMPLETE
Time 10.427: Module :MOTOR running command MOVE-A-HAND
Time 10.477: Module :MOTOR running command FINISH-MOVEMENT
Time 10.477: Teclar_Id_Blq_Saida_Tecla_2 Selected
Time 10.527: Teclar_Id_Blq_Saida_Tecla_2 Fired
Time 10.527: Module :MOTOR running command PUNCH
Time 10.677: Module :MOTOR running command PREPARATION-COMPLETE
Time 10.727: Module :MOTOR running command INITIATION-COMPLETE
Time 10.737: Device running command OUTPUT-KEY
<< Window "Verificando Blq de Entrada" got key #\s at time 10737 >>
Time 10.827: Module :MOTOR running command FINISH-MOVEMENT

```

O modo de alinhamento, representado pela Sequência 10 da Tabela 11, deve ser pressionado. Como este comando pode assumir dois estados, *contínuo* ou *temporário*, adotou-se que o operador pressionará a tecla *c*. O pressionamento ocorre no tempo 11.537.

```

Time 10.827: Teclar_Md_Alinhamento_Mover_1 Selected
Time 10.877: Teclar_Md_Alinhamento_Mover_1 Fired
Time 10.877: Module :MOTOR running command POINT-HAND-AT-KEY
Time 11.077: Module :MOTOR running command PREPARATION-COMPLETE
Time 11.127: Module :MOTOR running command INITIATION-COMPLETE
Time 11.227: Module :MOTOR running command MOVE-A-HAND
Time 11.277: Module :MOTOR running command FINISH-MOVEMENT
Time 11.277: Teclar_Md_Alinhamento_Tecla_1 Selected
Time 11.327: Teclar_Md_Alinhamento_Tecla_1 Fired
Time 11.327: Module :MOTOR running command PUNCH
Time 11.477: Module :MOTOR running command PREPARATION-COMPLETE
Time 11.527: Module :MOTOR running command INITIATION-COMPLETE
Time 11.537: Device running command OUTPUT-KEY
<< Window "Modo" got key #\c at time 11537 >>
Time 11.627: Module :MOTOR running command FINISH-MOVEMENT

```

Encerrando o comportamento motor, o operador necessita submeter a transação ao sistema de alinhamento de rota. Isso ocorre com o pressionamento da tecla ENTER (Seqüência 11 da Tabela 11). Nesta simulação a tecla foi pressionada no tempo 12.666.

```
Time 11.627: Teclar_Executa_Mover_1 Selected
Time 11.677: Teclar_Executa_Mover_1 Fired
Time 11.677: Module :MOTOR running command POINT-HAND-AT-KEY
Time 11.877: Module :MOTOR running command PREPARATION-COMPLETE
Time 11.927: Module :MOTOR running command INITIATION-COMPLETE
Time 12.356: Module :MOTOR running command MOVE-A-HAND
Time 12.406: Module :MOTOR running command FINISH-MOVEMENT
Time 12.406: Teclar_Executa_Tecla_1 Selected
Time 12.456: Teclar_Executa_Tecla_1 Fired
Time 12.456: Module :MOTOR running command PUNCH
Time 12.541: Module :VISION running command ENCODING-COMPLETE
Time 12.606: Module :MOTOR running command PREPARATION-COMPLETE
Time 12.656: Module :MOTOR running command INITIATION-COMPLETE
Time 12.666: Device running command OUTPUT-KEY
<< Window "Enter" got key ENTER at time 12666 >>
Time 12.756: Module :MOTOR running command FINISH-MOVEMENT
```

Finalmente, o operador necessita monitorar (Seqüência 12 da Tabela 11) a seqüência digitada para o alinhamento de rotas objetivando obter resposta, do sistema, que confirme a transação. A listagem abaixo mostra a monitoração do estado final do alinhamento da rota.

```
Time 12.756: Monitorar_Passo_1 Selected
Time 12.806: Monitorar_Passo_1 Fired
Time 12.806: Module :VISION running command FIND-LOCATION
Time 12.806: Vision found LOC87
Time 12.806: Monitorar_Passo_2 Selected
Time 12.856: Monitorar_Passo_2 Fired
Time 12.856: Module :VISION running command MOVE-ATTENTION
Time 12.941: Module :VISION running command ENCODING-COMPLETE
Time 12.941: Vision sees TEXT86
Time 13.041: Monitorar_Verificou_Objeto Selected
Merging chunk Goal83 into chunk Monitorar4
Monitoração Ok
POS-X 160
POS-Y 145
Time 12.991: Monitorar_Verificou_Objeto Fired
Time 12.991: Monitorar Selected
MONITORANDO...
Time 13.041: Monitorar Fired
Time 13.041: Monitorar Selected
MONITORANDO...
Time 13.091: Monitorar Fired
Time 13.091: Monitorar Selected
MONITORANDO...
```



```

Time 13.141: Monitorar Fired
Time 13.141: Monitorar Selected
MONITORANDO...
Time 13.191: Monitorar Fired
Time 13.191: Monitorar Selected
MONITORANDO...
Time 13.241: Monitorar Fired
Time 13.241: Monitorar Selected
MONITORANDO...
Time 13.291: Monitorar Fired
Time 13.291: Monitorar Selected
MONITORANDO...
Time 13.341: Monitorar Fired
Time 13.341: Monitorar Selected
MONITORANDO...
Time 13.391: Monitorar Fired
Time 13.391: Monitorar Selected
MONITORANDO...
Time 13.441: Monitorar Fired
Time 13.441: Monitorar Selected
MONITORANDO...
Time 13.491: Monitorar Fired
Time 13.491: Monitorar Selected
MONITORANDO...
Time 13.541: Monitorar Fired
Time 13.541: Monitorar_Completa Selected
Time 13.591: Monitorar_Completa Fired
Time 13.591: Monitorar_Passo_1 Selected
Time 13.641: Monitorar_Passo_1 Fired
Time 13.641: Module :VISION running command FIND-LOCATION
Time 13.641: Vision found LOC93
Time 13.676: Module :VISION running command ENCODING-COMplete
Time 13.676: Vision sees TEXT92
Time 13.676: Monitorar_Passo_2 Selected
Time 13.726: Monitorar_Passo_2 Fired
Time 13.726: Module :VISION running command MOVE-ATTENTION
Time 13.826: Monitorar_Verificou_Objeto Selected
Monitoração Ok
POS-X 160
POS-Y 145
Time 13.776: Monitorar_Verificou_Objeto Fired
Time 13.776: Monitorar_Completa Selected
Time 13.811: Module :VISION running command ENCODING-COMplete
Time 13.811: Vision sees TEXT92
Time 13.826: Monitorar_Completa Fired
Time 13.826: Monitorar_Passo_1 Selected
Time 13.876: Monitorar_Passo_1 Fired
Time 13.876: Module :VISION running command FIND-LOCATION
Time 13.876: Checking for silent events.
Time 13.876: * Nothing to run: No productions, no events.

```

A monitoração tem início no tempo 13.041. Transcorreram 0.375 segundos após o operador pressionar a tecla ENTER, que ocorreu em 12.666, e este processo termina depois de decorridos 0.785 segundos. Na prática, o operador olha duas

vezes, em momentos diferentes, para a interface buscando informações sobre o resultado da solicitação do alinhamento.

Salienta-se que a cada execução da simulação em questão, os tempos totais podem apresentar valores distintos, pois os estados das variáveis assumem valores diferentes, o que pode tornar o processo mais lento ou mais rápido. Isto é, o operador simulado pode encontrar junto aos estados iniciais do alinhamento, situação onde não seja permitido o caso normal discutido na Seção 6.2. Assim, o operador desempenhará comportamento de verificação até que estas condições assumam o caso normal para o alinhamento. Situações como esta promove um acréscimo no tempo total do desempenho da tarefa.

Observe-se que se um simulador do processo de circulação dos trens fosse conectado ao S. PERERE, poderia receber os comandos gerados pelo operador simulado, “escutando” as linhas que representam a digitação das teclas. De fato, extraindo da simulação as linhas “Window” que representam as teclas digitadas pelo operador simulado, têm-se a sequência:

```
<< Window "Acionando Rota - Pede" got key #\a at time 2025 >>
<< Window "Identificação de Entrada" got key #\x at time 2825 >>
<< Window "Identificação de Entrada" got key #\3 at time 3712 >>
<< Window "Identificação de Entrada" got key #\l at time 4544 >>
<< Window "Bloqueio de Entrada" got key #\b at time 5490 >>
<< Window "Bloqueio de Entrada" got key #\e at time 6363 >>
<< Window "Identificação de saída" got key #\y at time 7244 >>
<< Window " Identificação de saída " got key #\5 at time 8044 >>
<< Window " Identificação de saída " got key #\2 at time 8925 >>
<< Window "Verificando Blq de Entrada" got key #\b at time 9850 >>
<< Window "Verificando Blq de Entrada" got key #\s at time 10737 >>
<< Window "Modo" got key #\c at time 11537 >>
<< Window "Enter" got key ENTER at time 12666 >>
```

Esta sequência corresponde à digitação do seguinte texto:

ax31bey52bsc<enter>

Na próxima seção apresenta-se o alinhamento de rota da Linha 1 – Azul afetado de erro.

6.4 Resultados da execução do caso perturbado

Esta seção tem por objetivo explicar a simulação do operador humano, na interação com o sistema de alinhamento da rota afetado de erro. A descrição do procedimento operacional é o mesmo adotado na seção anterior. O conhecimento do operador simulado para a realização da tarefa de alinhar a rota afetada de erro é o mesmo descrito na Seção 6.3.

Para a apresentação dos resultados do S. PERERE, adota-se também a estratégia de analisar a saída padrão do ACT-R.

Na simulação apresentada a seguir, as perturbações foram geradas de acordo com a configuração de disparo discutidas na Seção 5.3.4.

Para maior controle nos resultados da simulação, optou-se por considerar neste exemplo, um tipo de erro para cada comportamento da tarefa em questão.

Apresentam-se a seguir os resultados de 08 (oito) diferentes execuções da tarefa de alinhamento de rota, discutindo-se o efeito de cada uma no processo de alinhamento.

6.4.1 Perturbação 1: omissão em verificação

Nesta execução, selecionou-se a geração de perturbações de **omissão** para os comportamentos elementares de verificar. Observe-se que o módulo disparador do S. PERERE sorteia um dos comportamentos para perturbação.

A seguir apresenta-se a saída do S. PERERE desta execução.

```
Time 1.445: Bloqueio_De_Saida_Verificou_Normal Selected
Merging chunk Goal44 into chunk Goal36
VERIFICOU BLOQUEIO_DE_SAIDA --> NORMAL
Time 1.445: Bloqueio_De_Saida_Verificou_Normal Fired
Time 1.495: Goal49 Retrieved
Time 1.495: Bloqueio_De_Saida_Tarefa4 Selected
Merging chunk Goal49 into chunk Tarefa4
Time 1.545: Bloqueio_De_Saida_Tarefa4 Fired
Time 1.545: Module :MOTOR running command PUNCH
Time 1.595: Module :MOTOR running command PREPARATION-
COMPLETE
Time 1.645: Module :MOTOR running command INITIATION-COMPLETE
Time 1.655: Device running command OUTPUT-KEY
Omitiu a Verificação do Bloqueio de Entrada
<< Window "Verificando Blq de Saída" got key #\a at time 1655
```

Observa-se que após a verificação do Bloqueio de Saída (tempo 1.545) o próximo passo seria verificar o Bloqueio de Entrada. Neste momento o S. PERERE

gerou perturbação, causando a omissão deste elemento de tarefa. Esta situação está caracterizada pelo pressionamento da tecla *a*, no tempo 1.655, simulando o ROTA – PEDE.

Isto significa que o operador simulado digita os comandos de alinhamento de rota sem perceber o real estado do bloqueio de entrada. Na prática, três situações são decorrentes deste fato:

- Se o bloqueio de entrada estiver normal, este erro não terá conseqüências. Esta é uma situação bastante comum, em que erros humanos não produzem conseqüências indesejáveis.
- Se o bloqueio de entrada estiver proibido e o sistema de controle dos trens estiver preparado para não aceitar um comando errado nesta situação, da mesma forma, o erro não tem conseqüências por conta de um projeto bem feito.
- Se o bloqueio de entrada estiver proibido e o sistema de controle dos trens não estiver preparado para rejeitar o comando fora do momento correto, neste caso, alguma situação indesejável poderá ocorrer.

6.4.2 Perturbação 2: omissão em teclar

Nesta execução, selecionou-se a geração de perturbações de **omissão** para os comportamentos elementares de teclar. Novamente, o disparador sorteia um dos comportamentos de teclar da tarefa especificada para esta execução particular.

A próxima saída diz respeito a omissão do comportamento teclar:

```
<< Window "Verificando Blq de Entrada" got key #\2 at time
8960
Time 9.050: Module :MOTOR running command FINISH-MOVEMENT
Time 9.050: Teclar_Md_Alinhamento_Mover_1 Selected
Time 9.100: Teclar_Md_Alinhamento_Mover_1 Fired
Time 9.100: Module :MOTOR running command POINT-HAND-AT-KEY
Time 9.300: Module :MOTOR running command PREPARATION-
COMPLETE
Time 9.350: Module :MOTOR running command INITIATION-
COMPLETE
Time 9.537: Module :MOTOR running command MOVE-A-HAND
Time 9.587: Module :MOTOR running command FINISH-MOVEMENT
Time 9.587: Teclar_Md_Alinhamento_Tecla_1 Selected
Time 9.637: Teclar_Md_Alinhamento_Tecla_1 Fired
Time 9.637: Module :MOTOR running command PUNCH
Time 9.787: Module :MOTOR running command PREPARATION-
COMPLETE
```

```

Time 9.837: Module :MOTOR running command INITIATION-
COMPLETE
Time 9.847: Device running command OUTPUT-KEY
Omitiu teclar o Bloqueio de Saída
<< Window "Verificando Blq de Entrada" got key #\c at time
9847

```

É importante atentar, nesta saída, que no tempo 9.050 a tecla pressionada foi o último dígito referente à Identificação do Intertravamento de Saída, o número 2, que é representada por *Y52*. O elemento omitido foi a Identificação do Bloqueio de Saída, representado por *BS*. Esta situação caracteriza-se pelo pressionamento da tecla *c* do Modo de Alinhamento.

Isto significa que o operador simulado digita a seguinte sequência como resultado da tarefa:

ax31bey52c<enter>

Como resultado desta digitação, as seguintes situações podem ocorrer:

- Se o sistema de controle de trens estiver preparado para uma análise da sintaxe dos comandos, o que é bastante provável, este erro provocará no sistema uma mensagem de erro de sintaxe.
- Se, no pior caso, o sistema não estiver preparado para esta análise, pode acontecer alguma consequência indesejável. Naturalmente, as consequências dos erros humanos dependem do contexto em que estes erros ocorrem.

6.4.3 Perturbação 3: omissão em monitorar

A seguir, observa-se a omissão do comportamento monitorar:

```

<< Window "Iniciando Monitoramento" got key ENTER at time 12666 >>
Omitiu o monitoramento
Time 12.756: Module :MOTOR running command FINISH-MOVEMENT
Time 12.756: Checking for silent events.
Time 12.756: * Nothing to run: No productions, no events.
12.756

```

No tempo 12.666 foi pressionada a tecla Enter e em seguida deveria ter ocorrido a monitoração do alinhamento. Pelo fato da omissão deste comportamento, após a finalização do processo motor de teclar o Enter, o sistema não encontrou

nenhuma outra regra de produção e finaliza. Como resultado desta omissão da monitoração, as seguintes situações podem ocorrer:

- Se o sistema emitir mensagem de que a rota foi alinhada, provavelmente, o operador continuará o desempenho de sua tarefa, após a omissão;
- Se o sistema emitir a mensagem de que a rota não foi alinhada, alguma situação indesejada pode ocorrer, pois dependerá da reação do operador após a omissão do monitoramento.

6.4.4 Perturbação 4: repetição em verificar

Em seguida foram geradas as perturbações de repetição para os comportamentos elementares de verificar.

Na saída abaixo observa-se que as produções *Modo_De_Operacao_Passo_1* e *Modo_De_Operacao_Passo_2* foram disparadas duas vezes caracterizando a repetição do comportamento de verificar:

```
Time 0.570: Modo_De_Operacao_Passo_1 Selected
Time 0.620: Modo_De_Operacao_Passo_1 Fired
Time 0.620: Module :VISION running command FIND-LOCATION
Time 0.620: Vision found LOC33
Time 0.655: Module :VISION running command ENCODING-COMplete
Time 0.655: Modo_De_Operacao_Passo_2 Selected
Time 0.705: Modo_De_Operacao_Passo_2 Fired
Time 0.705: Module :VISION running command MOVE-ATTENTION
Time 0.755: Goal34 Retrieved
Time 0.755: Repeticao_Verifica Selected
Time 0.790: Module :VISION running command ENCODING-COMplete
Time 0.790: Vision sees TEXT31
Time 0.805: Repeticao_Verifica Fired
Time 0.805: Module :VISION running command CLEAR
Time 0.805: Vision found LOC39
Time 0.805: Modo_De_Operacao_Passo_1 Selected
Time 0.855: Module :VISION running command CHANGE-STATE
Time 0.855: Modo_De_Operacao_Passo_1 Fired
Time 0.855: Module :VISION running command FIND-LOCATION
Time 0.855: Vision found LOC39
Time 0.855: Modo_De_Operacao_Passo_2 Selected
Merging chunk Goal36 into chunk Goal27
Time 0.905: Modo_De_Operacao_Passo_2 Fired
```

As produções *Modo_De_Operacao_Passo_1* e *Modo_De_Operacao_Passo_2* foram disparadas nos tempos 0.620 e 0.705, respectivamente, na primeira vez. Depois novamente nos tempos 0.855 e 0.905, respectivamente. Ressalta-se que a

produção Repeticao_Verifica disparada no tempo 0.805 é a responsável pela perturbação.

Neste caso, a implicação deste tipo de erro, para o sistema, reside no caso de haver restrição de tempo para conclusão do procedimento de alinhamento, que ficou dilatado por conta da repetição.

6.4.5 Perturbação 5: repetição em teclado

A próxima saída diz respeito à repetição do comportamento teclado:

```
<< Window " Modo de Alinhamento " got key #\c at time 11302 >>
Time 11.392: Module :MOTOR running command FINISH-MOVEMENT
Time 11.392: Repeticao_Verifica Selected
Time 11.442: Repeticao_Verifica Fired
Time 11.442: Teclar_Md_Alinhamento_Mover_1 Selected
Merging chunk Goal380 into chunk Goal377
Time 11.492: Teclar_Md_Alinhamento_Mover_1 Fired
Time 11.492: Module :MOTOR running command POINT-HAND-AT-KEY
Time 11.692: Module :MOTOR running command PREPARATION-COMPLETE
Time 11.742: Module :MOTOR running command INITIATION-COMPLETE
Time 11.842: Module :MOTOR running command MOVE-A-HAND
Time 11.892: Module :MOTOR running command FINISH-MOVEMENT
Time 11.892: Teclar_Md_Alinhamento_Tecla_1 Selected
Merging chunk Goal381 into chunk Goal378
Time 11.942: Teclar_Md_Alinhamento_Tecla_1 Fired
Time 11.942: Module :MOTOR running command PUNCH
Time 12.092: Module :MOTOR running command PREPARATION-COMPLETE
Time 12.142: Module :MOTOR running command INITIATION-COMPLETE
Time 12.152: Device running command OUTPUT-KEY
<< Window " Modo de Alinhamento " got key #\c at time 12152 >>
```

Observa-se que a tecla *c*, referente ao modo de alinhamento, foi pressionada duas vezes: nos tempos 11.302 e 12.152.

As seguintes situações podem ocorrer a partir do erro cometido:

- Se o sistema estiver preparado para uma análise da sintaxe dos comandos, este erro provocará no sistema uma mensagem de erro de sintaxe.
- Se, após a primeira digitação, o campo do modo de alinhamento perder o foco provavelmente não ocorra consequência alguma para o sistema, pois, a próxima ação é submeter os dados de entrada, pressionando a tecla Enter.

No entanto, a repetição de teclas em outras circunstâncias podem causar erros na entrada de dados. Em alguns casos, a tecla Enter está associada a um foco. Repetir a pode provocar a execução não desejada de algum evento do sistema.

6.4.6 Perturbação 6: repetição em monitorar

A próxima perturbação está relacionada com o comportamento de monitorar, o qual também foi perturbado por repetição:

```
OK OBJETO MONITORADO COM SUCESSO
POS-X 160
POS-Y 145
Time 13.026: Monitorar_Verificou_Objeto Fired
Time 13.026: Monitorar Selected
...
OK OBJETO MONITORADO COM SUCESSO
POS-X 160
POS-Y 145
Time 13.861: Monitorar_Verificou_Objeto Fired
Time 13.861: Monitorar_Completa Selected
```

O comportamento de monitorar foi executado nos tempos: 13.026 e 13.861.

A consequência deste erro para o sistema é análoga àquela produzida pelo erro de repetição da verificação, isto é, haverá prejuízo para a tarefa se o tempo for fator limitante.

6.4.7 Perturbação 7: confusão em teclar

Nesta simulação, foi gerado um erro de confusão para o comportamento de teclar. A saída abaixo mostra esta perturbação:

```
Time 11.992: Module :MOTOR running command PREPARATION-COMplete
Time 12.042: Module :MOTOR running command INITIATION-COMplete
Time 12.052: Device running command OUTPUT-KEY
<< Window "Modo de Alinhamento" got key #\v at time 12052 >>
Time 12.142: Module :MOTOR running command FINISH-MOVEMENT
Time 12.142: Teclar_Executa_Mover_1 Selected
Time 12.192:
```

Observa-se que a tecla *c* deveria ter sido pressionada para o Modo de Alinhamento. Entretanto, pelo fato do disparo de uma perturbação o operador simulado confundiu-se pressionando a tecla *v* que é fisicamente vizinha à tecla que deveria ter sido pressionada.

Devem-se considerar as seguintes situações para o erro cometido:

- Se o sistema estiver preparado para uma análise da sintaxe dos comandos, este erro provocará no sistema uma mensagem de erro de sintaxe.
- Se o sistema não estiver preparado para detectar tal erro, provavelmente ele assumirá Modo de Alinhamento *default* para aquela rota. Esta situação pode conduzir o sistema de trens a condição indesejada e perigosa.

6.4.8 Perturbação 8: engano em teclar

Finalmente, foi gerado um erro do tipo engano no comportamento de teclar.

Observa-se que no tempo 6.453, o módulo motor encerrou o movimento para a digitação do último caractere do Bloqueio de Entrada.

```
Time 6.453: Module :MOTOR running command FINISH-MOVEMENT
Time 6.453: Id_Engano_Mover_1 Selected
Time 6.503: Id_Engano_Mover_1 Fired
Time 6.503: Module :MOTOR running command POINT-HAND-AT-KEY
Time 6.703: Module :MOTOR running command PREPARATION-COMPLETE
Time 6.753: Module :MOTOR running command INITIATION-COMPLETE
Time 6.934: Module :MOTOR running command MOVE-A-HAND
Time 6.984: Module :MOTOR running command FINISH-MOVEMENT
Time 6.984: Id_Engano_Tecla_1 Selected
Time 7.034: Id_Engano_Tecla_1 Fired
Time 7.034: Module :MOTOR running command PUNCH
Time 7.184: Module :MOTOR running command PREPARATION-COMPLETE
Time 7.234: Module :MOTOR running command INITIATION-COMPLETE
Time 7.244: Device running command OUTPUT-KEY
```

<< Window "Identificação de Saída" got key #\y at time 7244 >>

```
Time 7.334: Module :MOTOR running command FINISH-MOVEMENT
Time 7.334: Id_Engano_Mover_2 Selected
Time 7.384: Id_Engano_Mover_2 Fired
Time 7.384: Module :MOTOR running command POINT-HAND-AT-KEY
Time 7.584: Module :MOTOR running command PREPARATION-COMPLETE
Time 7.634: Module :MOTOR running command INITIATION-COMPLETE
Time 7.734: Module :MOTOR running command MOVE-A-HAND
Time 7.784: Module :MOTOR running command FINISH-MOVEMENT
Time 7.784: Id_Engano_Tecla_2 Selected
Time 7.834: Id_Engano_Tecla_2 Fired
Time 7.834: Module :MOTOR running command PUNCH
Time 7.984: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.034: Module :MOTOR running command INITIATION-COMPLETE
Time 8.044: Device running command OUTPUT-KEY
```

<< Window " Identificação de Saída " got key #\7 at time 8044 >>

```
Time 8.134: Module :MOTOR running command FINISH-MOVEMENT
```

```

Time 8.134: Id_Engano_Mover_3 Selected
Time 8.184: Id_Engano_Mover_3 Fired
Time 8.184: Module :MOTOR running command POINT-HAND-AT-KEY
Time 8.384: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.434: Module :MOTOR running command INITIATION-COMPLETE
Time 8.615: Module :MOTOR running command MOVE-A-HAND
Time 8.665: Module :MOTOR running command FINISH-MOVEMENT
Time 8.665: Id_Engano_Tecla_3 Selected
Time 8.715: Id_Engano_Tecla_3 Fired
Time 8.715: Module :MOTOR running command PUNCH
Time 8.865: Module :MOTOR running command PREPARATION-COMPLETE
Time 8.915: Module :MOTOR running command INITIATION-COMPLETE
Time 8.925: Device running command OUTPUT-KEY

<< Window " Identificação de Saída " got key #\0 at time 8925 >>

Time 9.015: Module :MOTOR running command FINISH-MOVEMENT

```

No tempo 6.503 a regra de produção geradora do erro foi disparada alterando a meta do operador simulado que era de digitar o valor y52 como a Identificação da Saída. Tal perturbação levou-o a digitar os valores y70, nos tempos 7.244, 8.044 e 8.925.

Para que o S. PERERE produza a geração desta perturbação a meta cognitiva do operador simulado deve ser alterada. Foi o que ocorreu, conduzindo-o para a digitação de outro valor para a Identificação de Saída. O disparador poderia ter sorteado outro campo de entrada de dados.

Desta perturbação retira-se que:

- Se o sistema de alinhamento estiver preparado para verificar os valores dos intertravamentos de entrada e saída, uma mensagem de erro será exibida;
- De outro modo, se o sistema considerar que a entrada de dados corresponde a um valor válido, provavelmente a rota seja alinhada para outro trem. Esta situação caracteriza-se pela presença de estado de erro latente.

O exemplo apresentado mostra que a utilização do S. PERERE para a geração de perturbações pode contribuir para o estudo retrospectivo de sistemas com o foco na interação homem-computador, direcionando o projeto da interface a partir da análise dos erros gerados.

Na próxima seção discute-se os resultados obtidos a partir da implementação do S. PERERE.

6.5. Discussão dos resultados

As simulações geradas, neste capítulo, sobre a tarefa do alinhamento de rotas permitem refletir, inicialmente, sobre o exemplo adotado para verificar o potencial do S. PERERE.

Considera-se que, apesar da simplicidade com que se especificou a tarefa em questão, o exemplo mostra um aspecto relevante da pesquisa, no qual foi possível simular comportamentos dos três processos que este trabalho traçou como objetivos: o processo cognitivo, o processo perceptivo e o processo motor.

A simulação do processo cognitivo foi realizada através do comportamento elementar verificar, o qual consistiu de verificar o estado inicial do intertravamento tomando a decisão de prosseguir ou não, conforme o resultado obtido.

A simulação do processo perceptivo foi realizada através do comportamento elementar de monitorar, o qual consistiu de monitorar, durante certo tempo, o estado final do alinhamento de rota, isto é, o operador simulado percebia, observando o terminal de vídeo, o alinhamento ou não da rota.

A simulação do processo motor foi realizada através do comportamento elementar de teclar o qual consistiu em efetuar ações motoras sobre um teclado virtual. Através do exemplo, pode-se observar que tais ações foram constantemente executadas, utilizando diversos estilos de movimento.

A tarefa de alinhamento de rota foi concebida com três comportamentos elementares: verificar (cognitivo), teclar (motor) e monitorar (perceptivo). Com o propósito de incorporar outras características aos comportamentos elementares utilizados neste exemplo, adicionaram-se funcionalidades visuais ao comportamento teclar. Por exemplo, para a simulação deste comportamento, além dos movimentos inerentes a este tipo de procedimento, implementou-se regras de produção que, antes de pressionar a tecla desejada, o indivíduo simulado devesse atendê-la visualmente. Para o ACT-R, este procedimento não é necessário, ou seja, o indivíduo simulado não precisa “enxergar” o teclado antes de pressionar algum botão.

A simulação do exemplo para o caso perturbado mostrou-se muito positiva, no sentido de que foi possível gerar perturbações para oito casos: omissão na

verificação, omissão em teclar, omissão em monitorar, repetição em verificar, repetição em teclar, repetição em monitorar, confusão em teclar e engano em teclar.

Os resultados obtidos a partir das simulações em erro da tarefa de alinhamento de rota convergiram para o atendimento dos três principais objetivos deste trabalho.

O primeiro objetivo, que é o de explorar a diversidade do erro humano em situações de interação com sistemas críticos, foi atendido através da possibilidade de geração de diversos tipos de erro pelo S. PERERE.

Pelo mesmo motivo, o segundo objetivo, que é o de simular o comportamento humano em erro como uma expressão da variabilidade humana, também foi atendido.

Finalmente, também foi atendido o terceiro objetivo desse trabalho, que é o desenvolvimento de um simulador da ação humana que leve em consideração o erro, para ser aplicado ao estudo da confiabilidade humana na operação de sistemas computacionais.

Embora o exemplo implementado seja bastante limitado, já que simulou oito casos de erro humano, isto se deve ao fato de o exemplo ter intenções de explicar o desempenho humano simulado na realização de uma tarefa.

Entende-se que o simulador S. PERERE possui limitações, entre as quais destaca-se o fato dele possuir relação de dependência com a arquitetura cognitiva ACT-R, forçando o analista a ter esse conhecimento adicional.

Concluindo, pode-se afirmar a importância do S. PERERE por contribuir com o estudo do erro humano, listando possíveis erros humanos associados a realização de uma tarefa, podendo contribuir para projetos de IHC, conforme descrito no Capítulo 1.

7. CONCLUSÕES

O ponto inicial deste trabalho foi a investigação de que erros humanos podem ser considerados como causa ou fator contribuinte para uma série de acidentes e incidentes nas mais diversas áreas onde seres humanos interagem com sistemas automatizados.

Determinou-se que o foco desta pesquisa está na operação de sistemas computacionais e delimitaram-se os objetivos da seguinte forma:

- Objetivos gerais:
 - Explorar a diversidade do erro humano em situações de interação com sistemas críticos;
 - Tratar o erro humano como uma expressão da variabilidade humana.
- Objetivo específico:
 - Produzir um simulador da ação humana que leve em consideração o erro, para ser aplicado ao estudo da confiabilidade humana na operação de sistemas computacionais.

Na busca para o atendimento dos objetivos da pesquisa, partiu-se das seguintes premissas: é possível definir um conjunto de tarefas elementares que representam a interação homem-computador; e existe uma teoria do erro humano que estabelece tipos de erros humanos em função das ações humanas.

Em seguida, discutiu-se o papel das arquiteturas cognitivas como propostas para a concepção e implementação do simulador tencionado bem como a justificativa pela escolha da arquitetura cognitiva ACT-R que serviu como base para a construção do simulador.

O ACT-R foi a arquitetura cognitiva de apoio para o desenvolvimento do S. PERERE por atender a diversos aspectos considerados fundamentais, entre os quais destacam-se: ser apropriado para a construção de modelos que representam o conhecimento armazenado; sua forma de representação do conhecimento estar documentada e disponível para uso; modelar os processos cognitivos, perceptivos e motores; e por ser software disponível e suficientemente aberto para ser modificado.

Na crença de que uma teoria do erro humano estabelecesse tipos de erros humanos em função das ações humanas é que discutiu-se a classificação proposta por Reason (1990) a partir do modelo de processamento humano de informações de Rasmussen (1983).

Uma vez estabelecida a teoria do erro humano a ser considerada na pesquisa, definiu-se a taxonomia de Berliner; Angell; Shearer (1964) para tratar os comportamentos elementares que são comuns em tarefas que exijam algum tipo de interação com sistemas automatizados.

Uma das principais contribuições desta pesquisa é que para cada comportamento da taxonomia de Berliner construiu-se especificação genérica em português estruturado e seu respectivo código de conversão em ACT-R. Ressalta-se que para alguns comportamentos não foi possível obter o código de conversão por limitação da linguagem, entretanto espera-se que num futuro próximo, com o desenvolvimento da arquitetura cognitiva ACT-R, tais comportamentos possam ser implementados.

Desenvolveu-se, a partir da classificação de erros de Reason, uma especificação dos mecanismos de perturbação para os comportamentos elementares. Apesar do apelo intuitivo da categorização de erros na taxonomia de Reason, a existência de alguns erros pode ser considerada como subtipos ou como pertencentes a outras categorias de erro. A partir desta visão é que optou-se por agrupar erros que apresentam proximidade na definição objetivando generaliza-los em mecanismos únicos e que produzam o mesmo efeito.

Uma outra contribuição importante deste trabalho é que alguns erros que não estavam na classificação de Reason foram incorporados na especificação do perturbador. Destacam-se os seguintes mecanismos de erro propostos por este trabalho: aprendizado deficiente, sobrecarga motora e percepção falsa.

Algumas das perturbações especificadas neste trabalho foram implementadas no S. PERERE e utilizadas no exemplo de aplicação (v. Capítulo 6).

A seguir, apresentou-se a proposta do S. PERERE e discutiu-se os componentes do sistema e sua estrutura de funcionamento. Com relação aos componentes do simulador, três deles merecem destaque: o módulo perturbador, o módulo pré-processador e o módulo disparador.

O módulo perturbador representa o núcleo do S. PERERE gerando mecanismos de produção para a simulação dos processos cognitivos, perceptivos e motores. Tais mecanismos de produção são especificações genéricas dos comportamentos elementares de Berliner; Angell; Shearer (1964). A generalidade da descrição dos comportamentos elementares permite ao analista flexibilidade para configuração do S. PERERE na simulação de uma tarefa. Entretanto o fato de generalizar o sistema para outras situações o analista precisará de conhecimento específico do ACT-R para tal generalização. Na próxima seção este tópico voltará à discussão.

O módulo pré-processador foi definido para simplificar, ao usuário do S. PERERE, a entrada de dados que descreve a tarefa a ser simulada. A partir da entrada de dados, este módulo gera a memória declarativa e as regras de produção necessárias para a simulação do desempenho humano. O pré-processador também recebe do módulo perturbador regras de produção afetadas de erro para inclusão no comportamento a ser simulado.

O módulo disparador foi concebido para gerar disparos sob o desempenho da tarefa e em sintonia com o módulo perturbador, sorteando um erro possível para aquele comportamento que esteja sendo simulado. A configuração do disparador é realizada de duas formas: geração de um tipo de erro para um comportamento específico; e a geração de todos os possíveis erros associados a uma tarefa. No estágio em que se encontra o S. PERERE, apenas a geração de erro para comportamento específico foi implementada e discutida no exemplo de aplicação.

Foram encontradas, na literatura, diversas pesquisas relacionadas à simulação computacional do desempenho humano e algumas delas relacionadas com o erro humano. Dentre estes trabalhos discutiu-se: o simulador cognitivo de Itoh et al. (2001) que desenvolveram um simulador de desempenho do indivíduo responsável pelas tarefas de manobras de um navio; o EGOMS proposto por Wood (2000), que tem por finalidade reproduzir o tempo de resposta humano (desempenho) na execução de tarefas; e CATS desenvolvido por Callantine (2001) que utiliza um modelo das atividades do operador para prever as atividades do operador e interpretar suas ações.

Até onde foi realizada a revisão bibliográfica, este trabalho é o único na utilização de comportamentos elementares para, sobre eles, gerar perturbações.

O desenvolvimento deste trabalho foi de fundamental importância para a área de confiabilidade humana, auxiliando no estudo sobre operação de sistemas computacionais, gerando um simulador da ação humana que leve em consideração o erro.

Trabalhos Futuros

Durante o desenvolvimento deste trabalho foram realizadas algumas limitações em sua abrangência objetivando manter o foco nos estudos da geração do comportamento em erro. Entretanto foram percebidas linhas de pesquisa que podem dar continuidade ao estudo aqui realizado. A seguir, listam-se algumas linhas de pesquisa relacionadas ao tema abordado neste trabalho:

- Recuperação do erro: desenvolver estudos sobre formas de recuperação do erro após a geração da perturbação sobre o comportamento simulado. Recuperar o erro em cada um dos três níveis do modelo de processamento humano de informações de Rasmussen (1983), tende a apresentar o mesmo grau de dificuldade quando da sua execução. Considera-se este fato importante, pois aquilo que o indivíduo já adquiriu através da experiência, de reflexões, etc., e que pertencem ao nível de conhecimento (*knowledge-based*) estabeleceu um lugar em sua mente. Assim, acredita-se que a recuperação de erro no nível de conhecimento tende a ser mais difícil que nos níveis anteriores. A recuperação do erro permitirá ao projetista de IHC ou ao analista de confiabilidade analisar e conceber alternativas de melhoria no projeto.
- Inclusão do processo de comunicação: comportamentos elementares do processo de comunicação da taxonomia de Berliner; Angell; Shearer (1964) que não foram tratados nesta pesquisa podem ser incluídos em futuros trabalhos e também em novas versões do S. PERERE. Isto se aplica em situações de treinamento da operação de sistemas críticos, onde existam vários operadores e que precisem desenvolver comunicação verbal dentro do grupo. Atualmente as arquiteturas cognitivas ACT-R e EPIC fornecem um rudimentar apoio ao processo de comunicação.

- Extensão do S. PERERE: desenvolver estudos com o intuito de integrar a aplicação S. PERERE a outras arquiteturas cognitivas como, por exemplo, Soar, EPIC e COGNET. De modo geral, as características e funcionalidades destas arquiteturas cognitivas podem contribuir para a melhoria do S. PERERE.
- Construção de interface para especificação da tarefa: desenvolver estudos para a ampliação do S. PERERE, entre os quais a inclusão de processo interativo para especificação da tarefa a ser simulada. Isto tende a facilitar a declaração do conhecimento declarativo do indivíduo simulado para a inserção no sistema, além de facilitar o seu uso por usuários que não tenham conhecimentos sobre o ACT-R.
- Integração do S. PERERE a outras aplicações: desenvolver estudos com o propósito de submeter o S. PERERE a duas outras situações de uso descritas no início do trabalho, ou seja, acoplá-lo a um simulador de processos objetivando que o S. PERERE desempenhe o papel de operador deste processo e também acoplá-lo a sistemas utilizados para treinamento de aplicações críticas.

Anexo A - Código ACT-R para cada comportamento elementar

Calcular

```
(defvar *numero1* nil)
(defvar *numero2* nil)
(defvar *op* nil)

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

(defun teste(numero)
  (setf *numero1* numero)
  (print *numero1*)
)

(defun teste2(numero)
  (setf *numero2* numero)
  (print *numero2*)
)

(defun resolucao(operador)
  (setf *op* operador)
  (case *op*
    (1 (format t "~%0 resultado da operação de Adição é .: ~S.~%"
      (+ *numero1* *numero2*)))
    (2 (format t "~%0 resultado da operação de Subtração é .:
      ~S.~%" (- *numero1* *numero2*)))
    (3 (format t "~%0 resultado da operação de Divisão é .: ~S.~%"
      (/ *numero1* *numero2*)))
    (4 (format t "~%0 resultado da operação de Multiplicação é .:
      ~S.~%" (* *numero1* *numero2*)))
  )
)

(clear-all)
(pm-reset)

(chunk-type term passo num num2)
(chunk-type solucao op status)

(add-dm
(solucao isa solucao status ativo op 4)
(term isa term num 3 passo 1)
(term1 isa term num2 6 passo 2))

(p carrega_num1
=goal>
  ISA      term
  num      =num
  passo    1
!eval! (teste =num)
==>
+goal>
  ISA      term
  num2     6
  passo    2
```

```

!output!          (Num1 carregado com sucesso!!!)
)

(p carrega_num2
  =goal>
    ISA          term
    num2          =num2
    passo         2
!eval! (teste2 =num2)
==>
  +goal>
    ISA          solucao
    op           4
!output!          (Num2 carregado com sucesso!!!)
)

(p solucao_operação
  =goal>
    ISA          solucao
    op           =op
!eval! (resolucao =op)
==>
  -goal>
!output!          (Resolucao do Calculo)
)

(goal-focus term)

```

Escolher

```

(defun inicio ()

  (let* ((letters (permute-list '("objeto1" "objeto2" "objeto3"))))

    (op1 (first letters))
    (op2 (second letters))
    (op3 (third letters))

    (window (open-exp-window "Escolha")))

    (var1 op1)
    (var2 op2)
    (var3 op3))

  (reset)
  (pm-install-device window)

  (case (random 3)
    (0 (setf var1 op1))
    (1 (setf var2 op2))
    (2 (setf var3 op3)))

  (add-text-to-exp-window :text var1 :x 55 :y 55)
  (add-text-to-exp-window :text var2 :x 130 :y 55)
  (add-text-to-exp-window :text var2 :x 55 :y 80)

  (pm-proc-display)

```

```

    (pm-run 10)
  )
)
(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type escolha passo status)
(add-dm
  (passo1 isa escolha passo 1 status marca_visao)
  (passo2 isa escolha passo 2 status visao_ok)
  (passo3 isa escolha passo 3 status obj_visto)
  (passo4 isa escolha passo 4 status resultado))
(p escolha_1
  =goal>
    ISA      escolha
    passo    1
    status   marca_visao
==>
  =goal>
    ISA      escolha
    passo    2
    status   visao_ok
  +visual-location>
    ISA      visual-location
    attended nil
  +retrieval>
    ISA      escolha
    passo    2
    status   visao_ok
)

(p escolha_2
  =goal>
    ISA      escolha
    passo    2
    status   visao_ok
  =retrieval>
    ISA      escolha
    passo    2
    status   visao_ok
  =visual-location>
    ISA      visual-location
  =visual-state>
    ISA      module-state
    modality free
==>
  +visual>
    ISA      visual-object
    screen-pos =visual-location
  =goal>
    ISA      escolha
    passo    3
    status   obj_visto

```

```

+retrieval>
  ISA      escolha
  passo    3
  status    obj_visto
)

(p escolha_3
=goal>
  ISA      escolha
  passo    3
  status    obj_visto
=retrieval>
  ISA      escolha
  passo    3
  status    obj_visto
==>
=goal>
  ISA      escolha
  passo    4
+retrieval>
  ISA      escolha
  passo    4
  status    resultado
)

(p escolhido_não
=goal>
  ISA      escolha
  passo    4
=retrieval>
  ISA      escolha
  passo    4
  status    resultado
=>visual>
  ISA      TEXT
  value     " "
==>
-goal>
-retrieval>
+visual-state>
  ISA      module-state
  modality  free
!output!   (Objeto NÃO escolhido!!!!)
)

(p escolhido_sim
=goal>
  ISA      escolha
  passo    4
=retrieval>
  ISA      escolha
  passo    4
  status    resultado
=>visual>
  ISA      TEXT
  value     "objeto1"
  color     Black
==>
-goal>

```

```

-retrieval>
+visual-state>
      ISA                      module-state
      modality                free
!output!      (Objeto escolhido!!!!)
)
(setf *actr-enabled-p* t)
(pm-set-cursor-position 50 37)
(goal-focus passol)

```

Decidir

```

(defun inicio ()
  (defvar *digito1* nil)
  (defvar *digito2* nil)
  (defvar *digito3* nil)
  (defvar *op1* nil)
  (reset)
  (pm-run 10)
)
(defun setDigit (op val)
  (case op
    (1 (setf *digito1* val))
    (2 (setf *digito2* val))
    (3 (setf *digito3* val))
  )
)
(defun setOp (op)
  (setf *op1* op)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;value 1 --> Sinal de =;;;;;;;;;;;;;;;;
;;value 2 --> Sinal de >;;;;;;;;;;;;;;;;
;;value 3 --> Sinal de <;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun setTest (op2)
  (case op2
    (1 (case *op1*
        (1 (if (= *digito1* *digito2*)
              (if (= *digito2* *digito3*)
                  (format t "~%~%Decisão verdadeira~%" )
                  0
                )
              0
            )
        )
        (2 (if (> *digito1* *digito2*)
              (if (= *digito2* *digito3*)
                  (format t "~%~%Decisão verdadeira~%" )
                  0
                )
              0
            )
        )
        (3 (if (< *digito1* *digito2*)
              (if (= *digito2* *digito3*)
                  (format t "~%~%Decisão verdadeira~%" )
                  0
                )
              )
        )
    )
  )
)

```

```

0
)
)
)
)
(2 (case *op1*
  (1 (if (= *digito1* *digito2*)
    (if (< *digito2* *digito3*)
      (format t "~%~%Decisão verdadeira~%")
      0
    )
  )
)
)
(2 (if (> *digito1* *digito2*)
  (if (< *digito2* *digito3*)
    (format t "~%~%Decisão verdadeira~%")
    0
  )
)
)
(3 (if (< *digito1* *digito2*)
  (if (< *digito2* *digito3*)
    (format t "~%~%Decisão verdadeira~%")
    0
  )
)
)
)
)
(3 (case *op1*
  (1 (if (= *digito1* *digito2*)
    (if (> *digito2* *digito3*)
      (format t "~%~%Decisão verdadeira~%")
      0
    )
  )
)
)
(2 (if (> *digito1* *digito2*)
  (if (> *digito2* *digito3*)
    (format t "~%~%Decisão verdadeira~%")
    0
  )
)
)
(3 (if (< *digito1* *digito2*)
  (if (> *digito2* *digito3*)
    (format t "~%~%Decisão verdadeira~%")
    0
  )
)
)
)
)

```

```

    )
  )
)

(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type numero digito sequencia)
(chunk-type operador value sequencia)
(add-dm
  (decidir1 isa numero sequencia 1 digito 30)
  (decidir2 isa numero sequencia 2 digito 30)
  (decidir3 isa numero sequencia 3 digito 50)
  (decidir4 isa operador value 3 sequencia 1)
  (decidir5 isa operador value 2 sequencia 2))
(p decide_passo_1
  =goal>
    ISA          numero
    digito       =num
    sequencia    1
    !eval!      (setDigit 1 =num)
==>
  =goal>
    ISA          numero
    sequencia    2
  +retrieval>
    ISA          numero
    sequencia    2
)
(p decide_passo_2
  =goal>
    ISA          numero
    sequencia    2
  =retrieval>
    ISA          numero
    sequencia    2
    digito       =num
    !eval!      (setDigit 2 =num)
==>
  +retrieval>
    ISA          numero
    sequencia    3
  =goal>
    ISA          numero
    sequencia    3
)
(p decide_passo_3
  =goal>
    ISA          numero
    sequencia    3
  =retrieval>
    ISA          numero

```



```

        sequencia      3
        digito          =num
        !eval!          (setDigit 3 =num)
==>
    +retrieval>
        ISA            operador
        sequencia      1
    )
    (p decide_passo_4
    =retrieval>
        ISA            operador
        sequencia      1
        value          =val1
        !eval!          (setOp =val1)
==>
    +retrieval>
        ISA            operador
        sequencia      2
    )
    (p decide_passo_5
    =retrieval>
        ISA            operador
        sequencia      2
        value          =val2
        !eval!          (setTest =val2)
==>
    -retrieval>
    -goal>
    )
    (setf *actr-enabled-p* t)
    (goal-focus decidir1)

```

Comparar

```

(defun inicio ()
  (defvar *posx* nil)
  (defvar *posy* nil)
  (defvar *value* nil)
  (defvar *type* nil)
  (defvar *cor* nil)
  (defvar *posx2* nil)
  (defvar *posy2* nil)
  (defvar *value2* nil)
  (defvar *type2* nil)
  (defvar *cor2* nil)
  (let* ((letters (permute-list '("OBJETO" "OBJETO2"))))
    (op1 (first letters))
    (op2 (second letters))
    (window (open-exp-window "Experimento Comparar"
                             :visible t
                             :width 200
                             :height 200))
    (var1 op1)
    (var2 op2))
    (reset)
    (pm-install-device window)
    (case (random 5)

```

```

    (0 (add-text-to-exp-window :text var1 :x 55 :y 55 :color
'blue)
      (add-text-to-exp-window :text var2 :x 85 :y 85 :color
'yellow))
    (1 (add-text-to-exp-window :text var1 :x 55 :y 55 :color
'black)
      (add-text-to-exp-window :text var2 :x 85 :y 85 :color
'green))
    (2 (add-text-to-exp-window :text var1 :x 55 :y 55 :color 'red)
      (add-text-to-exp-window :text var2 :x 85 :y 85 :color
'black))
    (3 (add-text-to-exp-window :text var1 :x 55 :y 55 :color
'green)
      (add-text-to-exp-window :text var2 :x 85 :y 85 :color
'red))
    (4 (add-text-to-exp-window :text var1 :x 55 :y 55 :color
'yellow)
      (add-text-to-exp-window :text var2 :x 85 :y 85 :color
'blue))
  )
  (pm-proc-display)
  (pm-run 10)
)
)
(defun impComparacao ()
  (format t "~%~%Resultados para comparação dos objetos
atendidos~%")
  (format t "~%Primeiro Objeto Atendido .: ")
  (format t "~%Localização X .: ~S e Y .: ~S"*posx* *posy*)
  (format t "~%Valor .: ~S"*value*)
  (format t "~%Cor .: ~S"*cor*)
  (format t "~%Tipo .: ~S"*type*)
  (format t "~%~%Segundo Objeto Atendido .: ")
  (format t "~%Localização X .: ~S e Y .: ~S"*posx2* *posy2*)
  (format t "~%Valor .: ~S"*value2*)
  (format t "~%Cor .: ~S"*cor2*)
  (format t "~%Tipo .: ~S~%~%"*type2*)
)
(defun setFeaturesObj2 (x y cor tipo v)
  (setf *posx2* x)
  (setf *posy2* y)
  (setf *value2* v)
  (setf *type2* tipo)
  (setf *cor2* cor)
)
(defun setFeaturesObj1 (x y cor tipo v)
  (setf *posx* x)
  (setf *posy* y)
  (setf *value* v)
  (setf *type* tipo)
  (setf *cor* cor)
)
)
(clear-all)
(pm-reset)
(sgp
 :LF 0.05
 :ESC t
 :ga 0

```

```

)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type comparar atributo objeto x y cor tipo valor)
(add-dm
  (compara1 isa comparar atributo ver_obj objeto 1)
  (compara2 isa comparar atributo ver_obj objeto 2))
(p comparar_passo_1
  =goal>
    ISA      comparar
    atributo  ver_obj
    objeto    1
  =visual-state>
    ISA      module-state
    modality  free
==>
  =goal>
    ISA      comparar
    atributo  prep_ver
    objeto    1
  +visual-location>
    ISA      visual-location
    attended  nil
)
(p comparar_passo_2
  =goal>
    ISA      comparar
    atributo  prep_ver
    objeto    1
  =visual-location>
    ISA      visual-location
  =visual-state>
    ISA      module-state
    modality  free
==>
  +visual>
    ISA      visual-object
    screen-pos =visual-location
  +goal>
    ISA      comparar
    atributo  obj_ok
    objeto    1
)
(p comparar_passo_3
  =visual-state>
    ISA      module-state
    modality  free
  =goal>
    ISA      comparar
    atributo  obj_ok
    objeto    1
  =>visual>
    ISA      TEXT
    value     =valor
  =>visual-location>
    ISA      visual-location
    screen-x  =x
    screen-y  =y
    kind      =tipo

```

```

        color                =cor
==>
=goal>
    ISA                    comparar
    atributo              ver_obj
    objeto                2
+retrieval>
    ISA                    comparar
    atributo              ver_obj
    objeto                2
    !eval!                (setFeaturesObj1 =x =y =cor =tipo =valor)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(p comparar_passo_4
=visual-state>
    ISA                    module-state
    modality              free
=goal>
    ISA                    comparar
    atributo              ver_obj
    objeto                2
=retrieval>
    ISA                    comparar
    atributo              ver_obj
    objeto                2
==>
=goal>
    ISA                    comparar
    atributo              prep_ver
    objeto                2
+visual-location>
    ISA                    visual-location
    attended              nil
)
(p comparar_passo_5
=goal>
    ISA                    comparar
    atributo              prep_ver
    objeto                2
=visual-location>
    ISA                    visual-location
=visual-state>
    ISA                    module-state
    modality              free
==>
+visual>
    ISA                    visual-object
    screen-pos            =visual-location
=goal>
    ISA                    comparar
    atributo              obj_ok
    objeto                2
)
(p comparar_passo_6
=visual-state>
    ISA                    module-state
    modality              free
=goal>
    ISA                    comparar

```

```

        atributo      obj_ok
        objeto        2
=>visual>
    ISA              TEXT
    value            =valor
=>visual-location>
    ISA              visual-location
    screen-x         =x
    screen-y         =y
    kind              =tipo
    color             =cor
==>
    =goal>
        ISA          comparar
        atributo      nil
        objeto        nil
        !eval!        (setFeaturesObj2 =x =y =cor =tipo =valor)
        !eval!        (impComparacao)
    )
    (setf *actr-enabled-p* t)
    (goal-focus compar1)

```

Interpolar

```

    (defvar *numero1* nil)
    (defvar *numero2* nil)

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

    (defun teste(numero)
        (setf *numero1* numero)
        (print *numero1*)
    )

    (defun teste2(numero)
        (setf *numero2* numero)
        (print *numero2*)
    )

    (defun resolucao()
        (format t "~%0 resultado da interpolação [ $\sim S + \sim S$ ]/2 é  $\sim S. \sim \%$ "
            *numero1* *numero2* (/ (+ *numero1* *numero2*) 2 ))
    )

    (clear-all)
    (pm-reset)

    (chunk-type term passo num num2)
    (chunk-type solucao op op1)

    (add-dm
        (solucao isa solucao op + op1 /)
        (term isa term num 3 passo 1)
        (term1 isa term num2 6 passo 2))

    (p carrega_num1

```

```

=goal>
  ISA      term
  num      =num
  passo    1
!eval! (teste =num)
==>
+goal>
  ISA      term
  num2     6
  passo    2
!output! (Num1 carregado com sucesso!!!)
)

(p carrega_num2
=goal>
  ISA      term
  num2     =num2
  passo    2
!eval! (teste2 =num2)
==>
+goal>
  ISA      solucao
  op       +
  op1      /
!output! (Num2 carregado com sucesso!!!)
)

(p solucao_operação
=goal>
  ISA      solucao
  op       +
  op1      /
!eval! (resolucao)
==>
-goal>
)

(goal-focus term)

```

Verificar

```

(defun inicio ()
  (let* ((letters (permute-list '("objeto1" "objeto2"))))
    (op1 (first letters))
    (op2 (second letters))
    (window (open-exp-window "Verificar"
                             :visible t
                             :width 400
                             :height 400))

    (var1 op1)
    (var2 op2))
  (reset)
  (pm-install-device window)
  (case (random 2)
    (0 (setf var1 op1))
    (1 (setf var2 op2)))
  (add-text-to-exp-window :text var1 :x 55 :y 55)
  (add-text-to-exp-window :text var2 :x 130 :y 55)
  (pm-proc-display)

```

```

        (pm-run 10)
    )
)
(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type verificar objeto atributo obj)
(add-dm
  (verificar1 isa verificar atributo prep_mod)
  (verificar2 isa verificar atributo ok obj "objeto1"))
(p verificar_passo_1
  =goal>
    ISA          verificar
    atributo      prep_mod
==>
  =goal>
    ISA          verificar
    atributo      prep_ver
  +visual-location>
    ISA          visual-location
    attended nil
)

(p verificar_passo_2
  =goal>
    ISA          verificar
    atributo      prep_ver
  =visual-location>
    ISA          visual-location
  =visual-state>
    ISA          module-state
    modality      free
==>
  +visual>
    ISA          visual-object
    screen-pos    =visual-location
  +goal>
    ISA          verificar
    atributo      ok
)
(p verificar_verificou_objeto2
  =goal>
    ISA          verificar
    atributo      ok
  =>visual>
    ISA          TEXT
    value         "objeto2"
==>
  -goal>
    !output!      (Objeto não apresenta características
esperadas)
)
(p verificar_verificou_objeto1

```

```

=goal>
  ISA          verificar
  atributo     ok
=>visual>
  ISA          TEXT
  value        =obj
==>
  -goal>
  !output!     (Verificação ok!)
)
(setf *actr-enabled-p* t)
(goal-focus verificar1)

```

Lembrar

```

(defun inicio (valor)
  (defvar *valor* nil)
  (setf *valor* valor)
  (reset)
  (pm-run 10)
)
(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type lembrar objeto atributo obj)
(add-dm
  (lembrar1 isa lembrar atributo prep)
  (lembrar2 isa lembrar atributo 1))
(p lembrar_passo_1
  =goal>
    ISA          lembrar
    atributo     prep
==>
  +retrieval>
    ISA          lembrar
    atributo     (!eval! (print *valor*))
  +goal>
    ISA          lembrar
    atributo     (!eval! (print *valor*))
)
(p lembrar_passo_2
  =goal>
    ISA          lembrar
    atributo     1
  =retrieval>
    ISA          lembrar
    atributo     1
==>
  -goal>
  -retrieval>
    !output!     (Valor retornado com sucesso!!!)
)
(setf *actr-enabled-p* t)

```



```
(goal-focus lembrar1)
```

Inspeccionar

```
(defvar *valor* nil)
(defvar *cor* nil)
(defvar *x* nil)
(defvar *y* nil)
(defvar *valorLoc* nil)
(defvar *corLoc* nil)
(defvar *xLoc* nil)
(defvar *yLoc* nil)

(defun inicio ()
  (let* ((letters (permute-list '("objeto"))))

    (opl (first letters))

    (window (open-exp-window "Verifica opção"))

    (var1 opl))

  (reset)
  (pm-install-device window)

  (add-text-to-exp-window :text var1 :x 85 :y 55)

  (pm-proc-display)
  (pm-run 10)
)

(defun setPars (v c corX corY)
  (setf *valor* v)
  (setf *cor* c)
  (setf *x* corX)
  (setf *y* corY)
)

(defun setParsLoc (v c corX corY)
  (setf *valorLoc* v)
  (setf *corLoc* c)
  (setf *xLoc* corX)
  (setf *yLoc* corY)
)

(defun verificar()
  (if (= *x* *xLoc*)
    (if (= *y* *yLoc*)
      1 0)
    0)
)

(clear-all)
(pm-reset)
(sgp
 :LF 0.05
 :ESC t
 :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
```

```

(chunk-type verifica atributo valor cor x y testFim)
(add-dm
  (tarefa1 isa verifica atributo ver_obj)
  (tarefa2 isa verifica atributo prep_ver)
  (tarefa3 isa verifica atributo prep_conc valor "objeto" cor Black x
107 y 65)
  (tarefa4 isa verifica atributo ok testFim 0))
(p verificacao_passo_1
  =goal>
    ISA      verifica
    atributo  ver_obj
==>
  =goal>
    ISA      verifica
    atributo  prep_ver
  +visual-location>
    ISA      visual-location
    attended  nil
  +retrieval>
    ISA      verifica
    atributo  prep_ver
)
(p verificacao_passo_2
  =goal>
    ISA      verifica
    atributo  prep_ver
  =retrieval>
    ISA      verifica
    atributo  prep_ver
  =visual-location>
    ISA      visual-location
  =visual-state>
    ISA      module-state
    modality  free
==>
  +visual>
    ISA      visual-object
    screen-pos =visual-location
  =goal>
    ISA      verifica
    atributo  prep_conc
  +retrieval>
    ISA      verifica
    atributo  prep_conc
)
(p verificacao_passo_3
  =goal>
    ISA      verifica
    atributo  prep_conc
  =retrieval>
    ISA      verifica
    atributo  prep_conc
    valor    =valor
    cor      =cor
    x        =x
    y        =y
    !eval!   (setPars =valor =cor =x =y)
==>
  =goal>

```

```

        ISA          verifica
        atributo      ok
+retrieval>
        ISA          verifica
        atributo      ok
        testFim      0
)
(p verificacao_verificou_objeto
=goal>
        ISA          verifica
        atributo      ok
=>visual>
        ISA          TEXT
        value         =valor
        color         =cor
=>visual-location>
        ISA          visual-location
        screen-x      =x
        screen-y      =y
        attended      t
        !eval!        (setParsLoc =valor =cor =x =y)
=retrieval>
        ISA          verifica
        atributo      ok
        < testFim     (!eval! (verificar))
==>
=goal>
-retrieval>
!output! (OK objeto inspecionado com sucesso)
!output! (Cor Preto)
!output! (Pos-x 107)
!output! (Pos-y 65)
!output! (Valor "objeto")
!output! (Tipo Texto)
)
(setf *actr-enabled-p* t)
(goal-focus tarefal)

```

Observer

```

(defun inicio ()

  (let* ((letters (permute-list '("OBJETO")))
        (op1 (first letters))
        (window (open-exp-window "Experimento Observar"
                                :visible t
                                :width 200
                                :height 200))

        (var1 op1))
    (reset)
    (pm-install-device window)
    (add-text-to-exp-window :text var1 :x 85 :y 55)
    (pm-proc-display)
    (pm-run 10)
  )
)

(defun impResult (obj tipagem)
  (format t "~%~%0 objeto identificado foi: ~S."obj)

```

```

        (format t "~%E seu tipo é.: ~S.~%"tipagem)
    )

    (clear-all)
    (pm-reset)
    (sgp
      :LF 0.05
      :ESC t
      :ga 0
    )
    (sgp :v t)
    (pm-set-params :real-time t :show-focus t)
    (chunk-type observar atributo obj tp)
    (add-dm
      (observe! isa observar atributo ver_obj))
    (p observar_passo_1
      =goal>
        ISA          observar
        atributo      ver_obj
      =visual-state>
        ISA          module-state
        modality      free
    ==>
      =goal>
        ISA          observar
        atributo      prep_ver
      +visual-location>
        ISA          visual-location
        attended      nil
    )
    (p observar_passo_2
      =goal>
        ISA          observar
        atributo      prep_ver
      =visual-location>
        ISA          visual-location
      =visual-state>
        ISA          module-state
        modality      free
    ==>
      +visual>
        ISA          visual-object
        screen-pos    =visual-location
      =goal>
        ISA          observar
        atributo      obj_ok
    )
    (p observar_passo_3
      =visual-state>
        ISA          module-state
        modality      free
      =goal>
        ISA          observar
        atributo      obj_ok
      =>visual>
        ISA          TEXT
        value         =obj
      =>visual-location>
        ISA          visual-location
    )

```

```

        kind                =tp
        !eval!              (impResult =obj =tp)
==>
    -goal>
)
(setf *actr-enabled-p* t)
(goal-focus observat)

```

Ler

```

(defun inicio (str1)

    (setf window (open-exp-window "Experimento Ler"
                                   :visible t
                                   :width 200
                                   :height 200))

    (reset)
    (pm-install-device window)
    (add-text-to-exp-window :text str1 :x 85 :y 55)
    (pm-proc-display)
    (pm-run 10)
)
(defun impResult (obj)
    (format t "~%~%0 texto Lido foi: ~S."obj)
)

(clear-all)
(pm-reset)
(sgp
    :LF 0.05
    :ESC t
    :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type ler atributo obj)
(add-dm
    (le1 isa ler atributo ver_obj))
(p ler_passo_1
    =goal>
        ISA          ler
        atributo      ver_obj
    =visual-state>
        ISA          module-state
        modality      free
==>
    =goal>
        ISA          ler
        atributo      prep_ver
    +visual-location>
        ISA          visual-location
        attended      nil
)
(p ler_passo_2
    =goal>
        ISA          ler
        atributo      prep_ver
    =visual-location>

```

```

        ISA          visual-location
    =visual-state>
        ISA          module-state
        modality     free
==>
    +visual>
        ISA          visual-object
        screen-pos   =visual-location
    =goal>
        ISA          ler
        atributo     obj_ok
    )
    (p ler_passo_3
      =visual-state>
        ISA          module-state
        modality     free
    =goal>
        ISA          ler
        atributo     obj_ok
    =>visual>
        ISA          TEXT
        value        =obj
        !eval!       (impResult =obj)
==>
    -goal>
    )
    (setf *actr-enabled-p* t)
    (goal-focus 1e1)

```

Monitorar

```

(defun inicio ()
  (defvar *x* nil)
  (defvar *y* nil)
  (defvar *xLoc* nil)
  (defvar *yLoc* nil)

  (defvar *flag* nil)

  (let* ((letters (permute-list '("objeto"))))

    (opl (first letters))

    (window (open-exp-window "Verifica opção"))

    (var1 opl))

  (reset)
  (pm-install-device window)

  (add-text-to-exp-window :text var1 :x 85 :y 55)

  (setf *flag* 1)
  (pm-proc-display)
  (pm-run 10)
  )
)

```

```

(defun inicio_denovo ()

  (let* ((letters (permute-list '("objeto")))
         (op1 (first letters))
         (window (open-exp-window "Verifica opção"))
         (var1 op1))

    (pm-install-device window)
    (add-text-to-exp-window :text var1 :x 85 :y 56)

    (setf *flag* (+ 1 *flag*))
    (pm-proc-display)

  )
)
(defun setPars (corX corY)
  (setf *x* corX)
  (setf *y* corY)
)
(defun setParsLoc (corX corY)
  (setf *xLoc* corX)
  (setf *yLoc* corY)
)
(defun verificar()
  (if (= *flag* 1)
      (if (= *x* *xLoc*)
          (if (= *y* *yLoc*)
              1 0)
          0)
      (if (= *x* *xLoc*)
          (if (= *yLoc* (+ *y* 1))
              1 0)
          0)
  )
)
)
(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type verifica atributo tempo x y testFim)
(add-dm
  (monitorar1 isa verifica atributo ver_obj)
  (monitorar2 isa verifica atributo prep_ver)
  (monitorar3 isa verifica atributo prep_conc x 107 y 65)
  (monitorar4 isa verifica atributo ok testFim 0)
  (monitorar5 isa verifica atributo aguarde tempo 700))
(p monitorar_passo_1
  =goal>
    ISA      verifica
    atributo  ver_obj
==>
  =goal>

```

```

        ISA      verifica
        atributo  prep_ver
+visual-location>
        ISA      visual-location
        attended nil
)
(p monitorar_passo_2
=goal>
        ISA      verifica
        atributo  prep_ver
=visual-location>
        ISA      visual-location
=visual-state>
        ISA      module-state
        modality  free
==>
+visual>
        ISA      visual-object
        screen-pos =visual-location
=goal>
        ISA      verifica
        atributo  prep_conc
+retrieval>
        ISA      verifica
        atributo  prep_conc
        x        =x
        y        =y
)
(p monitorar_passo_3
=goal>
        ISA      verifica
        atributo  prep_conc
=retrieval>
        ISA      verifica
        atributo  prep_conc
        x        =x
        y        =y
        !eval!    (setPars =x =y)
==>
+retrieval>
        ISA      verifica
        atributo  ok
        testFim   =testFim
=goal>
        ISA      verifica
        atributo  ok
)
(p monitorar_verificou_objeto
=goal>
        ISA      verifica
        atributo  ok
=>visual-location>
        ISA      visual-location
        screen-x   =x
        screen-y   =y
        attended   t
        !eval!     (setParsLoc =x =y)
=retrieval>
        ISA      verifica

```



```

        atributo      ok
        < testFim      (!eval! (verificar))
==>
+goal>
    ISA              verifica
    atributo         aguarde
    tempo            700

    !output! (OK objeto inspecionado com sucesso)
    !output! (Pos-x 107)
    !output! (Pos-y 65)
)
(p monitorar
=goal>
    ISA              verifica
    atributo         aguarde
==>
=goal>
    ISA              verifica
    atributo         aguarde
!output! (Aguarde)
)
(p monitorar_completa
=goal>
    ISA              verifica
    atributo         aguarde
    < tempo          (!eval! (pm-get-time))
==>
=goal>
    ISA              verifica
    atributo         ver_obj
    !eval!           (if(< *flag* 2) (inicio_denovo) nil)
)
(setf *actr-enabled-p* t)
(goal-focus monitorar1)
(spp monitorar :p .50)

```

Explorar

```

(defun inicio ()
  (defvar *posx1* nil)
  (defvar *posy1* nil)
  (defvar *posx2* nil)
  (defvar *posy2* nil)
  (defvar *posx3* nil)
  (defvar *posy3* nil)
  (defvar *posx4* nil)
  (defvar *posy4* nil)
  (let* ((letters (permute-list '("Objeto1"))))

    (primeiro (first letters))
    (window (open-exp-window "Experimento Explorar"
                             :visible t
                             :width 200
                             :height 200))

    (text1 primeiro))
  (reset)
  (pm-install-device window)

```

```

      (add-text-to-exp-window :text text1 :x 95 :y 55)
      (pm-proc-display)
      (pm-run 10)
    )
  )
  (defun setVars (obj par1 par2)
    (case obj
      (1 (setf *posx1* par1)
          (setf *posy1* par2))
      (2 (setf *posx2* par1)
          (setf *posy2* par2))
      (3 (setf *posx3* par1)
          (setf *posy3* par2))
      (4 (setf *posx4* par1)
          (setf *posy4* par2))
    )
  )
  (defun printResults()
    (format t "~%Experimento Explorar Concluido !!!!!")
    (format t "~%A posição do objeto1 é .: X.: ~S e Y.:
~S.~%" *posx1* *posy1*)
    (format t "~%A posição do objeto2 é .: X.: ~S e Y.:
~S.~%" *posx2* *posy2*)
    (format t "~%A posição do objeto3 é .: X.: ~S e Y.:
~S.~%" *posx3* *posy3*)
    (format t "~%A posição do objeto4 é .: X.: ~S e Y.:
~S.~%" *posx4* *posy4*)
  )
  (defun explorar (str1 posy)
    (setf *window* (open-exp-window "Experimento Explorar"
                                     :visible t
                                     :width 200
                                     :height 200))

    (pm-install-device *window*)
    (add-text-to-exp-window :text str1 :x 95 :y posy)
    (pm-proc-display)
  )

  (clear-all)
  (pm-reset)
  (sgp
    :LF 0.05
    :ESC t
    :ga 0
  )
  (sgp :v t)
  (pm-set-params :real-time t :show-focus t)
  (chunk-type explora objeto atributo x y)
  (add-dm
    (exploral isa explora objeto objeto1 atributo passo1))
  (p explorar_passo_1
    =goal>
      ISA                explora
      objeto              objeto1
      atributo             passo1
    =visual-state>
      ISA                module-state
      modality             free
  ==>

```

```

=goal>
  ISA      explora
  objeto   objeto1
  atributo passo2
+visual-location>
  ISA      visual-location
  attended nil
)
(p explorar_passo_2
=goal>
  ISA      explora
  objeto   objeto1
  atributo passo2
=visual-location>
  ISA      visual-location
=visual-state>
  ISA      module-state
  modality free
==>
+visual>
  ISA      visual-object
  screen-pos =visual-location
=goal>
  ISA      explora
  objeto   objeto1
  atributo passo3
)
(p explorar_passo_3
=goal>
  ISA      explora
  objeto   objeto1
  atributo passo3
=>visual-location>
  ISA      visual-location
  screen-x  =x
  screen-y  =y
  attended  t
  !eval!    (setVars 1 =x =y)
==>
=goal>
  ISA      explora
  objeto   objeto2
  atributo passol
  !eval!    (explorar "Objeto2" 90)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(p explorar_passo_4
=visual-state>
  ISA      module-state
  modality free
=goal>
  ISA      explora
  objeto   objeto2
  atributo passol
==>
=goal>
  ISA      explora
  objeto   objeto2
  atributo passo2

```

```

+visual-location>
  ISA      visual-location
  attended nil
)
(p explorar_passo_5
=goal>
  ISA      explora
  objeto   objeto2
  atributo passo2
=visual-location>
  ISA      visual-location
=visual-state>
  ISA      module-state
  modality free
==>
+visual>
  ISA      visual-object
  screen-pos =visual-location
=goal>
  ISA      explora
  objeto   objeto2
  atributo passo3
)
(p explorar_passo_6
=goal>
  ISA      explora
  objeto   objeto2
  atributo passo3
=>visual-location>
  ISA      visual-location
  screen-x  =x
  screen-y  =y
  attended  t
  !eval!    (setVars 2 =x =y)
==>
=goal>
  ISA      explora
  objeto   objeto3
  atributo passo1
  !eval!    (explorar "Objeto3" 120)
)
;;;;;;;;;;;;;
(p explorar_passo_7
=visual-state>
  ISA      module-state
  modality free
=goal>
  ISA      explora
  objeto   objeto3
  atributo passo1
==>
=goal>
  ISA      explora
  objeto   objeto3
  atributo passo2
+visual-location>
  ISA      visual-location
  attended nil
)

```

```

(p explorar_passo_8
=goal>
  ISA          explora
  objeto       objeto3
  atributo     passo2
=visual-location>
  ISA          visual-location
=visual-state>
  ISA          module-state
  modality     free
==>
+visual>
  ISA          visual-object
  screen-pos   =visual-location
=goal>
  ISA          explora
  objeto       objeto3
  atributo     passo3
)
(p explorar_passo_9
=goal>
  ISA          explora
  objeto       objeto3
  atributo     passo3
=>visual-location>
  ISA          visual-location
  screen-x     =x
  screen-y     =y
  attended     t
  !eval!       (setVars 3 =x =y)
==>
=goal>
  ISA          explora
  objeto       objeto4
  atributo     passo1
  !eval!       (explorar "Objeto4" 160)
)
;;;;;;;;;;;;;
(p explorar_passo_10
=visual-state>
  ISA          module-state
  modality     free
=goal>
  ISA          explora
  objeto       objeto4
  atributo     passo1
==>
=goal>
  ISA          explora
  objeto       objeto4
  atributo     passo2
+visual-location>
  ISA          visual-location
  attended     nil
)
(p explorar_passo_11
=goal>
  ISA          explora
  objeto       objeto4

```

```

        atributo      passo2
=visual-location>
    ISA              visual-location
=visual-state>
    ISA              module-state
    modality         free
==>
+visual>
    ISA              visual-object
    screen-pos       =visual-location
=goal>
    ISA              explora
    objeto           objeto4
    atributo         passo3
)
(p explorar_passo_12
=visual-state>
    ISA              module-state
    modality         free
=goal>
    ISA              explora
    objeto           objeto4
    atributo         passo3
=>visual-location>
    ISA              visual-location
    screen-x         =x
    screen-y         =y
    attended         t
    !eval!           (setVars 4 =x =y)
==>
    -goal>
        !eval!       (printResults)
)
(setf *actr-enabled-p* t)
(goal-focus explora1)

```

Detector

```

(defun inicio ()

  (let* ((letters (permute-list '("OBJETO")))
        (op1 (first letters))
        (window (open-exp-window "Experimento Detector"
                                :visible t
                                :width 200
                                :height 200))

        (var1 op1))
    (reset)
    (pm-install-device window)
    (case (random 5)
      (0 (add-text-to-exp-window :text var1 :x 85 :y 55 :color
'blue))
      (1 (add-text-to-exp-window :text var1 :x 85 :y 55 :color
'black))
      (2 (add-text-to-exp-window :text var1 :x 85 :y 55 :color
'red))
      (3 (add-text-to-exp-window :text var1 :x 85 :y 55 :color
'green))
    )
  )

```

```

        (4 (add-text-to-exp-window :text var1 :x 85 :y 55 :color
'yellow))
    )
    (pm-proc-display)
    (pm-run 10)
  )
)
(defun impResult (cor)
  (format t "~%A cor do objeto identificado é .: ~S.~%"cor)
)

(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type detectar atributo)
(add-dm
  (detectar isa detectar atributo ver_obj))
(p detectar_passo_1
  =goal>
    ISA          detectar
    atributo      ver_obj
  =visual-state>
    ISA          module-state
    modality      free
==>
  =goal>
    ISA          detectar
    atributo      prep_ver
  +visual-location>
    ISA          visual-location
    attended      nil
)
(p detectar_passo_2
  =goal>
    ISA          detectar
    atributo      prep_ver
  =visual-location>
    ISA          visual-location
  =visual-state>
    ISA          module-state
    modality      free
==>
  +visual>
    ISA          visual-object
    screen-pos    =visual-location
  =goal>
    ISA          detectar
    atributo      obj_ok
)
(p detectar_passo_3
  =goal>
    ISA          detectar
    atributo      obj_ok

```

```

=>visual>
    ISA          TEXT
    color        =cor
    !eval!      (impResult =cor)
==>
    -goal>
)
(setf *actr-enabled-p* t)
(goal-focus detectal)

```

Identificar

```

(defun inicio (posx posy)
  (defvar *x* nil)
  (defvar *y* nil)
  (setf *x* posx)
  (setf *y* posy)

  (let* ((letters (permute-list '("objeto"))))
    (opl (first letters))
    (window (open-exp-window "Experimento Identificar"
                             :visible t
                             :width 200
                             :height 200))

    (varl opl))
    (reset)
    (pm-install-device window)
    (add-text-to-exp-window :text varl :x 85 :y 55)
    (pm-proc-display)
    (pm-run 10)
  )
)
(defun compVars(corx cory)
  (if (= corx *x*)
    (if (= cory *y*)
      (impResult *x* *y*)
      NIL)
    NIL
  )
)
(defun impResult (x y)
  (format t "~~~Parametros corretos, Identificação
concluída!!!")
  (format t "~%0 objeto identificado esta em X.: ~S e Y.:
~S.~%" x y)
)

(clear-all)
(pm-reset)
(sgp
 :LF 0.05
 :ESC t
 :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type identificar atributo x y)

```



```

(add-dm
  (identifical isa identificar atributo ver_obj))
(p identificar_passo_1
  =goal>
    ISA      identificar
    atributo  ver_obj
  =visual-state>
    ISA      module-state
    modality  free
==>
  =goal>
    ISA      identificar
    atributo  prep_ver
  +visual-location>
    ISA      visual-location
    attended  nil
)
(p identificar_passo_2
  =goal>
    ISA      identificar
    atributo  prep_ver
  =visual-location>
    ISA      visual-location
  =visual-state>
    ISA      module-state
    modality  free
==>
  +visual>
    ISA      visual-object
    screen-pos =visual-location
  =goal>
    ISA      identificar
    atributo  obj_ok
)
(p identificar_passo_3
  =visual-state>
    ISA      module-state
    modality  free
  =goal>
    ISA      identificar
    atributo  obj_ok
  =>visual-location>
    ISA      visual-location
    screen-x  =x
    screen-y  =y
    !eval!    (compVars =x =y)
==>
  -goal>
)
(setf *actr-enabled-p* t)
(goal-focus identifical)

```

Localizar

```

(defun inicio ()

  (let* ((letters (permute-list '("OBJETO"))))
    (opl (first letters))
    (window (open-exp-window "Experimento Localizar"

```

```

:visible t
:width 200
:height 200))

    (var1 op1))
  (reset)
  (pm-install-device window)
  (add-text-to-exp-window :text var1 :x 85 :y 55)
  (pm-proc-display)
  (pm-run 10)
)
)
(defun impResult (posx posy)
  (format t "~%0 objeto identificado esta em X.: ~S e Y.:
~S.~%"posx posy)
)

(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type localizar atributo x y)
(add-dm
  (localizar isa localizar atributo ver_obj))
(p localizar_passo_1
  =goal>
    ISA      localizar
    atributo  ver_obj
  =visual-state>
    ISA      module-state
    modality free
==>
  =goal>
    ISA      localizar
    atributo  prep_ver
  +visual-location>
    ISA      visual-location
    attended nil
)
(p localizar_passo_2
  =goal>
    ISA      localizar
    atributo  prep_ver
  =visual-location>
    ISA      visual-location
  =visual-state>
    ISA      module-state
    modality free
==>
  +visual>
    ISA      visual-object
    screen-pos =visual-location
  =goal>
    ISA      localizar
    atributo  obj_ok

```

```

)
(p localizar_passo_3
  =visual-state>
    ISA          module-state
    modality      free
  =goal>
    ISA          localizar
    atributo      obj_ok
  =>visual-location>
    ISA          visual-location
    screen-x      =x
    screen-y      =y
    !eval!        (impResult =x =y)
==>
  -goal>
)
(setf *actr-enabled-p* t)
(goal-focus localizar)

```

Mover

```

(defun inicio ()

  (defvar *item* "Cursor")

  (let* (
    (window (open-exp-window "Experimento mover")))

    (reset)

    (pm-install-device window)

    (setf text1 *item*)

    (add-text-to-exp-window :text text1 :x 95 :y 55)

    (pm-proc-display)
    (pm-run 10)
  )
)

(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(pm-set-params :real-time t :show-focus t)
(chunk-type teclar tecla status)(add-dm
(chunk1 isa teclar status inicio)
(chunk2 isa visual-location screen-x 75 screen-y 25 attended T)
(chunk3 isa visual-object screen-pos =chunk2))
(p posiciona
  =goal>

```

```

        ISA      teclar
        status   inicio
    =visual-state>
        ISA      module-state
        modality  free
    ==>
    +visual-location>
        ISA      visual-location
        attended  nil
    =goal>
        status   ato
    )
    (p teclar
    =goal>
        ISA      teclar
        status   ato
    =visual-location>
        ISA      visual-location
    =visual-state>
        ISA      module-state
        modality  free
    ==>
    +visual>
        ISA      visual-object
        screen-pos =visual-location
    +goal>
        ISA      teclar
        status   fim
    )
    (p finalizar
    =visual>
        ISA      visual-object
    =visual-state>
        ISA      module-state
        modality  free
    =manual-state>
        ISA      module-state
        modality  free
    =goal>
        ISA      teclar
        status   fim
    ==>
    =goal>
        status   mover
    )
    (p mover
    =manual-state>
        ISA      module-state
        modality  free
    =goal>
        ISA      teclar
        status   mover
    ==>
    +manual>
        ISA      hand-to-mouse
    =goal>
        status   mover1
    )
    (p mover1

```

```

=manual-state>
  ISA      module-state
  modality  free
=goal>
  ISA      teclar
  status    mover1
==>
+manual>
  ISA      move-cursor
  device    mouse

=goal>
  status    mover2)
(setf *actr-enabled-p* t)
(pm-prepare-motor punch :hand right)
(pm-set-cursor-position 50 37)
(pm-set-hand-location right 1 4)
(goal-focus chunk1)

```

Teclar

```

(defun inicio()

  (defvar *input_k* nil)
  (setf *input_k* "B")

  (let* ((lis (permute-list '("T")))
         (window (open-exp-window "Teclar"))))

    (reset)
    (pm-install-device window)
    (pm-proc-display)

    (pm-run 10)))

(clear-all)
(pm-reset)
(sgp
  :LF 0.05
  :ESC t
  :ga 0
)
(sgp :v t)
(chunk-type teclar tecla status)(add-dm
  (chunk1 isa teclar status inicio))

(p posiciona
  =goal>
    ISA      teclar
    status    inicio
  =manual-state>
    ISA      module-state
    modality  free
==>
+manual>
  ISA      point-hand-at-key
  hand      right
  to-key    *input_k*
=goal>

```

```

        status          ato
    )
    (p teclar
      =goal>
        ISA      teclar
        status   ato
      =manual-state>
        ISA      module-state
        modality  free
    ==>
      +manual>
        ISA      punch
        hand      right
        finger    index
      =goal>
        status   fim
    )
    (p finalizar
      =goal>
        ISA      teclar
        status   fim
      =manual-state>
        ISA      module-state
        modality  free
    ==>
      -goal>
    )(pm-prepare-motor punch :hand right)
    (pm-set-hand-location right 1 4)
    (goal-focus chunk1)

```

Anexo B – Comportamentos elementares associados a erros

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 B.1 - Comportamento elementar Calcular do Processo Cognitivo

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 B.2 - Comportamento elementar Escolher do Processo Cognitivo

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 B.3 - Comportamento elementar Decidir do Processo Cognitivo

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 B.4 - Comportamento elementar Comparar do Processo Cognitivo

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 B.5 - Comportamento elementar Interpolar do Processo Cognitivo

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 B.6 - Comportamento elementar Verificar do Processo Cognitivo

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 B.7 - Comportamento elementar Lembrar do Processo Cognitivo

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

Tabela B.8 - Comportamento elementar Inspeccionar do Processo Perceptivo

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 B.9 - Comportamento elementar Observar do Processo Perceptivo

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 B.10 - Comportamento elementar Ler do Processo Perceptivo

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 B.11 - Comportamento elementar Monitorar do Processo Perceptivo

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 B.12 - Comportamento elementar Explorar do Processo Perceptivo

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

Tabela B.13 - Comportamento elementar Detectar do Processo Perceptivo

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

Tabela B.14 - Comportamento elementar Identificar do Processo Perceptivo

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 B.15 - Comportamento elementar Localizar do Processo Perceptivo

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 B.16 - Comportamento elementar Mover do Processo Motor

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

Tabela B.17 - Comportamento elementar Segurar do Processo Motor

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

Tabela B.18 - Comportamento elementar Empurrar/Puxar do Processo Motor

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

Tabela B.19 - Comportamento elementar Prender do Processo Motor

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

Tabela B.20 - Comportamento elementar Dar e Devolver do Processo Motor

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

Tabela B.21 - Comportamento elementar Remover e Descartar do Processo Motor

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 B.22 - Comportamento elementar Posicionar do Processo Motor

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 B.23 - Comportamento elementar Ajustar do Processo Motor

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 B.24 - Comportamento elementar Teclar do Processo Motor

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 B.25 - Comportamento elementar Instalar do Processo Motor

REFERÊNCIAS BIBLIOGRÁFICAS

- ACT-R. Pittsburgh. 2002. **Página do Grupo de Pesquisa da Arquitetura Cognitiva ACT-R**. Disponível em: <<http://act-r.psy.cmu.edu/tutorials/>>. Acesso em 13 de mar. 2004.
- ALART, J. et al. **Dependability Handbook**. Toulouse: Laboratory for Dependability Engineering, 1998.
- ALMEIDA JÚNIOR, J. R. **Segurança em Sistemas Críticos e em Sistemas de Informação** – Um estudo Comparativo. Tese de Livre Docência. São Paulo: Usp, 2003.
- ANDERSON, J.R. **Rules of the mind**. Hillsdale: Lea, 1993. 320 p.
- ANDERSON, J.R. et al. An integrated theory of the mind. **Psychological Review**, n.111, p.1036-1060, 2004.
- ANDERSON, J. R.; LEBIERE, C. **The Atomic Components of Thought**. Mahwah: Lea, 1998. 490 p.
- BEGOSSO, L. C. **Ciência Cognitiva: uma abordagem informacional da mente**. Dissertação de mestrado. Marília: Unesp, 1999.
- BEGOSSO, L. C.; FILGUEIRAS, L. V. L. S. PERERE – Simulation of Performance in Error and Recovery: uma proposta visando a confiabilidade humana. In: INTERNATIONAL INFORMATION AND TELECOMUNICATION TECHNOLOGIES SYMPOSIUM, 3, São Carlos, 2004. **I2TS'2004**. São Carlos: 2004. CD-ROM.
- _____ An Error Generator Model. In: INFORMATION RESOURCES MANAGEMENT ASSOCIATION, 16., San Diego, 2005. **IRMA'2005**. San Diego: 2005. p.1141-1142. CD-ROM.
- 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.

- BUDIÚ, R. Pittsburgh. **Página do Grupo de Pesquisa da Arquitetura Cognitiva ACT-R**. Disponível em: <<http://act-r.psy.cmu.edu>>. Acesso em: 12 de fev. 2004.
- 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.
- _____. Perception and action. In: **The atomic components of thought**. Mahwah: Lea, 1998. p.167-200.
- BYRNE, M. D. ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. **International Journal of Human-Computer Studies**, v.55, p.41-84, 2001.
- BUTTERWORTH, R. et al. Demonstrating the Cognitive Plausibility of Interactive System Specifications. **Bulletin of Computer Science**, v.12, p.237-259, 2000.
- CACCIABUE, P. C. A methodology of human factors analysis for systems engineering: theory and applications. **IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Human**, v. 27, n.3, p. 325-339, 1997.
- CALLANTINE, T. J. **Analysis of flight operational quality assurance data using model-based activity tracking**. Warrendale, PA: SAE International, 2001. (SAE Technical paper 2001-01-2640).
- _____. Air traffic controller agents. In: **International Joint Conference on Autonomous Agents and Multiagent Systems**, 2., Melbourne, 2003. Anais. Melbourne: ACM, 2003. p.952-953.
- CARD, S. K.; MORAN, T. P.; NEWELL, A. **The psychology of human-computer interaction**. Hillsdale: LEA, 1983.
- CUNHA, L. S.; GIRAFFA, L. M. M. SCORE: a proposal for the design and implementation of BDI Agent behavior applied to interactive computer game characters. In: Enia, 4., Campinas, 2003. **Anais do XXIII Congresso da Sociedade Brasileira de Computação**. Campinas: SBC, 2003. CD-ROM.
- DEKKER, S.; HOLLNAGEL, E. **Human factors and folk models**. Cognition, Technology & Work. 2003. Disponível em: <<http://www.springerlink.com>>. Acesso em: 02 de mar. 2004.

- DUPUY, J. P. **Nas origens das ciências cognitivas**. São Paulo: Ed. Unesp, 1996.
- FERREIRA, A. B. **Dicionário Aurélio Básico da Língua Portuguesa**. Rio de Janeiro: Nova Fronteira, 1988.
- FILGUEIRAS, L.V.L. **APIS: Método para Análise e Projeto de Interfaces Homem-Computador visando confiabilidade humana**. Tese de doutorado. São Paulo: Usp, 1996.
- _____. São Paulo, Universidade de São Paulo, 2004. /Anotações de aula da disciplina PCS-5006, São Paulo, 2004.
- FILGUEIRAS, L.V.L. MELNIKOFF, S.S.S. A Method for Analysis and Prototyping of Interaction Intense Software. **Proceedings of the 6th IFAC/IFIP/IFORS/IEA symposium on analysis and evaluation of man-machine systems**. Cambridge, EUA: Massachussets Institute of Technology, 1995. p. 511-516.
- FISCHHOFF, B.; SLOVIC, P.; LICHTENSTEIN, S. Fault trees: sensitivity of estimated failure probabilities to problem representation. **Journal of Experimental Psychology: human perception and performance**, n.4, p.330-334, 1978.
- GORE, B.F.; CORKER, K.M. A System Engineering Approach to Behavioral Predictions of an Advanced Air Traffic Management Concept. In: **DIGITAL AVIONICS SYSTEMS CONFERENCES**, 19., Philadelphia, 2000. **Proceedings 19th Digital Avionics Systems Conferences**. Philadelphia: 2000. p.4.B.3-1 – 4.B.3-8.
- HOLLAN, J. HUTCHINS, E.; KIRSH, D. **Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research**. ACM Transactions on Computer-Human Interaction, vol 7, n° 2, June 2000, pp 174-196.
- HOLLNAGEL, E. **Human Reliability Analysis**. Academic Press, p. 20-37, 1993.
- HOLLNAGEL, E. **Cognitive Reliability and Error Analysis Method**. Elsevier, 1998.
- ISAAC, A. et al. Short report on human performance models and taxonomies of human error in ATM (HERA). **European Air Traffic Management Programme – EATMP – Infocentre – Eurocontrol**, 2002. Disponível em:

- <[http://www.eurocontrol.int/humanfactors/gallery/content/public/docs/DELIVERABLES/HF27%20\(HRS-HSP-002-REP-02\)%20Released.pdf](http://www.eurocontrol.int/humanfactors/gallery/content/public/docs/DELIVERABLES/HF27%20(HRS-HSP-002-REP-02)%20Released.pdf)>. Acesso em 12 de ago. 2004.
- ITOH, K. et al. Risk Analysis of Ship Navigation by Use of Cognitive Simulation. **Cognition, Technology & Work**, n.3, p.4-21, 2001.
- JOHNSON, C. The causes of human error in medicine. **Cognition, Technology & Work**, n.4, p.65-70, 2002.
- JONES, G.; RITTER, F. E.; WOOD, D. J. **Using a Cognitive Architecture to Examine What Develops**. *Psychological Science*, vol 11, n.2, March 2000, pp 93-100.
- JOSHI, V.V.; KAUFMAN, L.M.; GIRAS, T.C. Human Behavior Modeling in Train Control Systems. In: **Proceedings Annual Reliability and Maintainability Symposium**. 2001. p. 183-188.
- KIERAS, D.E.; HORNOF A.J. **Building cognitive models with the Epic architecture for human cognition and performance**. Disponível em: <http://simon.lrdc.pitt.edu/~iccm/proceedings/tutorials/EPIC_Tutorial.pdf>. Acesso em 27 de ago.2004.
- KIERAS, D.E.; MEYER D.E. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. **Human-Computer Interaction**, vol 12, 1997, pp.391-438.
- KIERAS, D.E.; MEYER D.E. **EPIC: A cognitive architecture for computational modeling of human performance**. Disponível em: <<http://www.eecs.umich.edu/~kieras/epic.html>>. Acesso em 27 de ago.2004.
- LAIRD, J.E.; NEWELL A.; ROSEMBLOOM P.S. Soar: an architecture for general intelligence. **Artificial Intelligence**, vol 33, nº 1, 1987, pp.1-64.
- LAIRD, J. E. **Página do Grupo de Pesquisa da Arquitetura Cognitiva Soar**. Disponível em <http://www.eecs.umich.edu/~soar/tutorial/Soar_Part1.pdf>. Acesso em 04 de abr.2004.
- LASALA, K. P. **A Pratical Guide to Developing Reliable Human-Machine Systems and Processes**. Rome-NY: RAC, 2002.
- LAVERY, R.G. Artificial intelligence and simulation: an introduction. In: **Proceedings of Winter Simulation Conference**. 1986. p.448-452.

- LEHMAN, J. F.; LAIRD, J. E.; ROSENBLOOM, P. **A gentle introduction to Soar, an architecture for human cognition.** Disponível em <http://eecs.umich.edu/~soar/docs/Gentle.pdf>>. Acesso em 04 de abr.2004.
- 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/ih/hcsl/HPM_pubs/HumanErrorModels.pdf> Acesso em: 09 de abr.2004.
- 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.
- NEWELL, A. **Unified Theories of Cognition.** Cambridge: Harvard University Press, 1990.
- NIELSEN, F. Human behavior: another dimension of standards setting. **Standard View**, v.4, n.1, p.36-41, 1996.
- NORMAN, D. A. **Design Rules Based on Analyses of Human Error.** ACM Transactions on Mathematical Software, vol 26, nº 4, April 1983, pp. 254-258.
- NORMAN, D. A.; DRAPPER, S. W. **User Centered System Design: New Perspectives on Human-Computer Interaction.** Hillsdale: LEA, 1986.
- PEW, R. W.; MAVOR A. S. **Modeling human and organizational behavior: application to military simulations.** Washington: National Academy Press, 1998.
- PUTZER, H.; ONKEN, R. Cosa: A generic cognitive system architecture based on a cognitive model of human behavior. **Cognition, Technology & Work**, n.5, p.140-151, 2003.
- 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.; MADDUX, M.E. **Human Error.** <http://hfskyway.faa.gov/HFAMI>> Acesso em: 06 de abr. 2003.

- 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: 14 de set. 2003.
- SEARLE, J.R. **A redescoberta da mente**. São Paulo: Martins Fontes, 1997.
- SANDERS, M.S.; McCORMICK, E. J. **Human factors in engineering and design**. New York: McGraw-Hill, 1987.
- SHORROCK, S.T.; KIRWAN, B. Development and application of a human error identification tool for air traffic control. **Applied Ergonomics**, n.33, p.319-336, 2002.
- SWAIN, A.D.; GUTTMANN, H.E. **Handbook of Human Reliability Analysis with emphasis on Nuclear Power Plant Applications**: Final report. Albuquerque: U.S. Nuclear Regulatory Commission, 1983. (NUREG/CR-1278).
- TATE, A.; HENDLER, J.; DRUMMOND, M. Introduction. In: ALLEN, J.; HENDLER, J.; TATE, A. **Readings in Planning**. Palo Alto: Morgan Kaufmann, 1990. p.26-49.
- WICKENS, C. D. **Engineering Psychology and Human Performance**. New York: HarperCollins, 1992.
- WOOD, S. D. **Extending GOMS to Human Error and Applying it to Error-Tolerant Design**. Tese de doutorado. Michigan: University of Michigan, 2000.
- ZACHARY, W. et al. Intelligent Computer-Human Interaction in Real-Time, Multi-Tasking Process Control and Monitoring Systems. In: HELANDER, M; NAGAMACHI. **Human factors in design for manufacturability**. New York: Taylor and Francis, 1992. p.377-402.
- ZACHARY, W; LE MENTEC, J. C.; RYDER J. Interface agents in complex systems. In: NTUEN, C.; PARK, E. H. **Human interaction with complex systems**: conceptual principles and design practice. Norwell: Kluwer Academics Publishers, 1996.