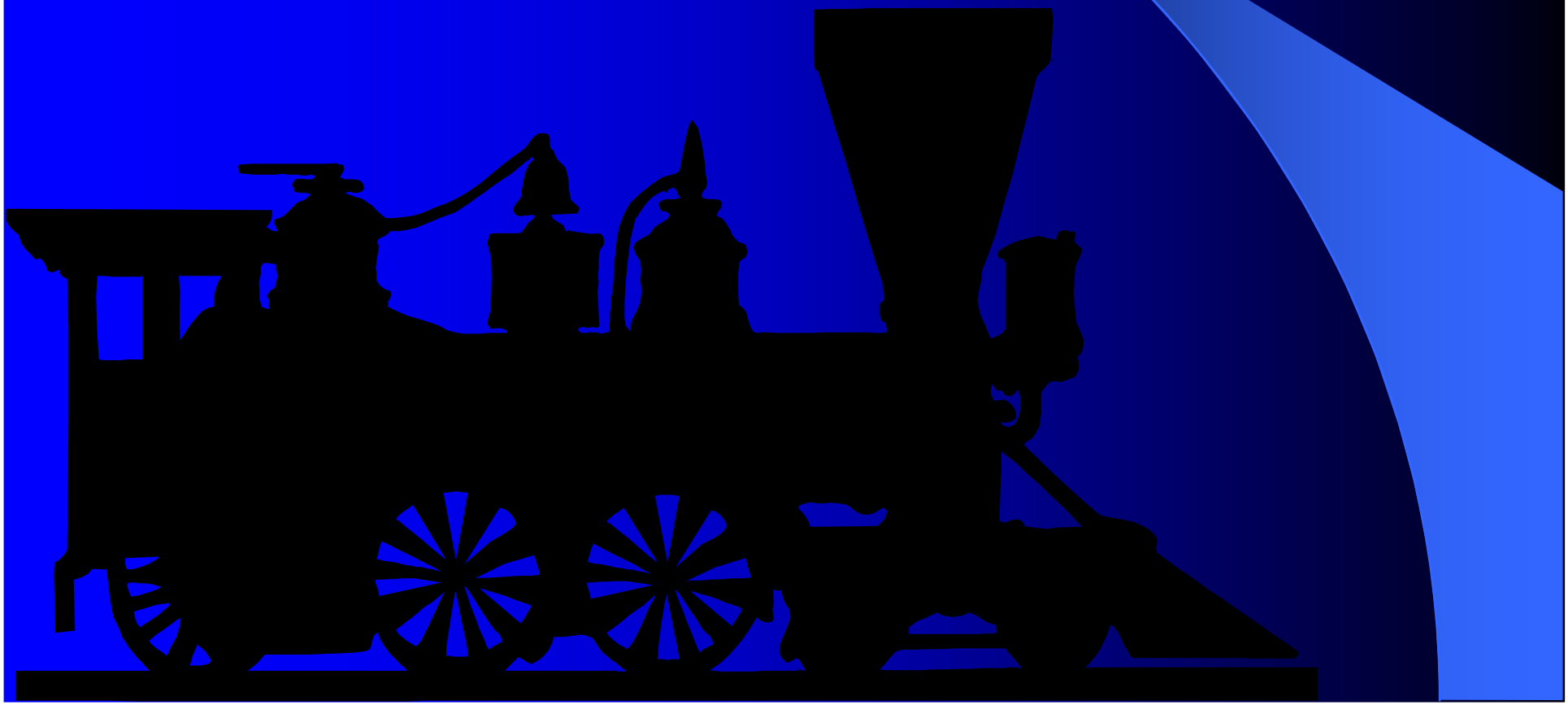
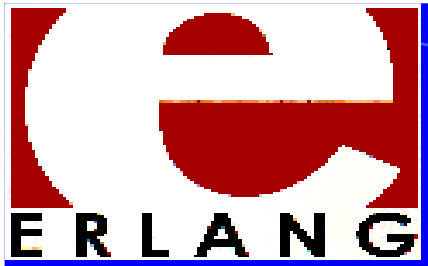
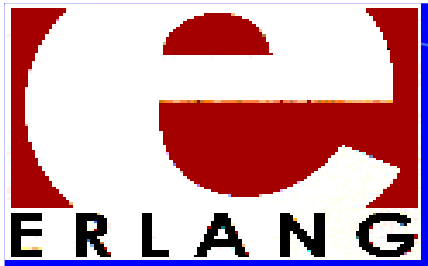
TCC – 2000
Aplicação em tempo real utilizando a linguagem Erlang





Aplicação em tempo real utilizando a linguagem Erlang

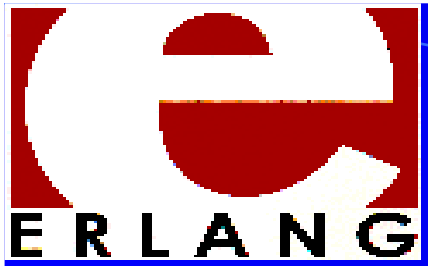
- **Título: Aplicação em tempo real utilizando a linguagem Erlang**
- **Acadêmico: Amilton Cesar Schmidt**
- **Orientador: José Roque Voltolini da Silva**



Aplicação em tempo real utilizando a linguagem Erlang

Apresentação:

- Introdução
- Alguns comentários sobre processos concorrentes e tempo real
- A linguagem de programação Erlang
- Problema e especificação
- Implementação e apresentação do protótipo
- Conclusão



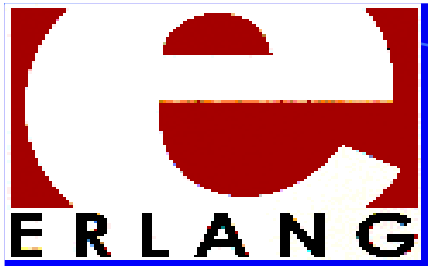
INTRODUÇÃO

Motivação:

- Disciplina de Linguagens de Programação
- Características da linguagem

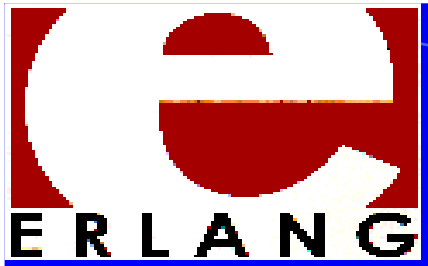
Objetivo:

- Desenvolver um estudo da Linguagem de Programação Erlang e uma aplicação, através desta, utilizando conceitos de processos concorrentes e tempo real .



PROCESSOS CONCORRENTES

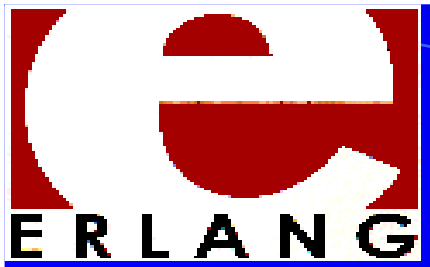
Uma das principais características na implementação de processos concorrentes é o acesso mutuamente exclusivo a recursos compartilhados. Para a proteção destas regiões críticas, podem ser utilizados alguns mecanismos conhecidos, como: semáforos, monitores e passagem de mensagens.



TEMPO REAL

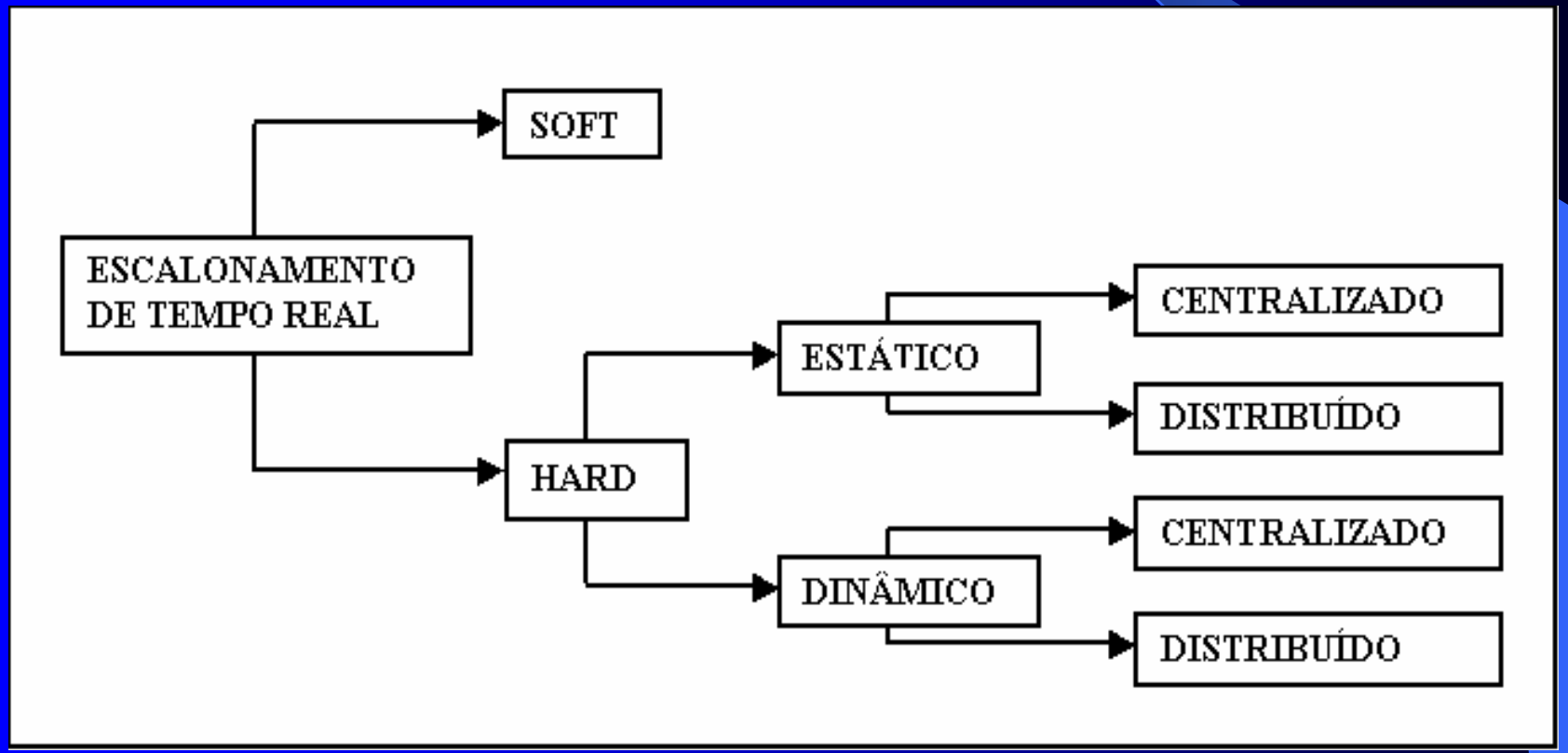
"Um Sistema de Tempo Real é um sistema que produz reações a estímulos oriundos do ambiente dentro de intervalos de tempos impostos pelo ambiente." ([FAR2000]).

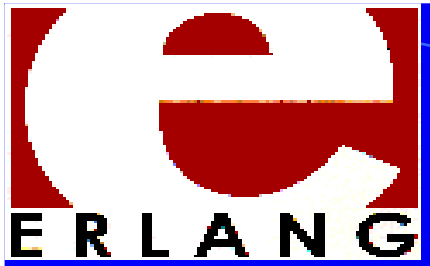
- Tempo na atualidade
- Tempo real hoje
- Restrições temporais



TEMPO REAL

Classificações de Sistemas de Tempo Real:

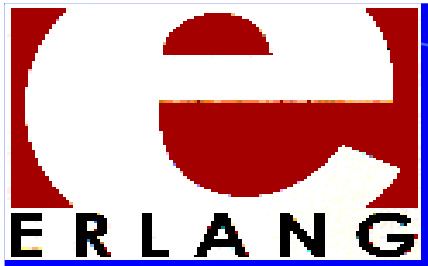




A LINGUAGEM ERLANG

História:

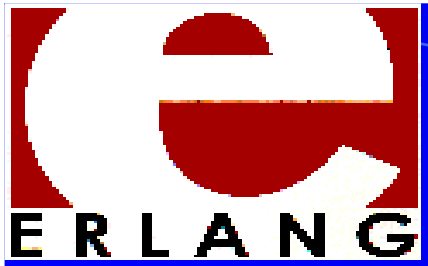
- Erlang e sua origem
- Necessidades da Ericsson
- Linguagem declarativa, estilo Prolog



A LINGUAGEM ERLANG

Principais características:

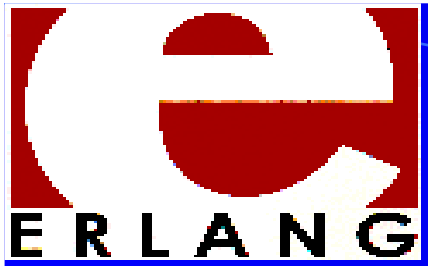
- redução do tempo de desenvolvimento e de correção de erros;
- código mais simples e de fácil entendimento;
- soluções completas para problemas básicos;
- explora conceitos de processos concorrentes e tempo real.



A LINGUAGEM ERLANG

Erlang suporta cinco tipos simples de dados:

- ***integer***: um número inteiro positivo ou negativo;
- ***float***: um número com parte fracionária;
- ***atom***: um nome constante;
- ***pid***: um identificador de processos;
- ***reference***: um único valor que pode ser copiado ou passado mas não pode ser gerado novamente.



A LINGUAGEM ERLANG

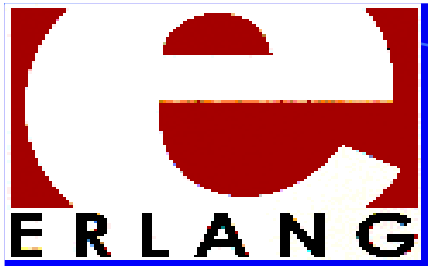
Também suporta dois tipos de dados compostos:

- ***tuple***: um conjunto de elementos de tamanho fixo.

```
{1, 2, {3, 4}, {a, {b, c}}}
```

- ***list***: um conjunto de elementos de tamanho variável.

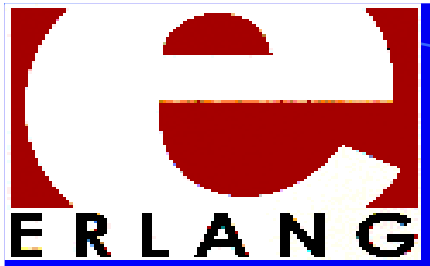
```
[1, abc, [12], ‘hello’]
```



Principais mecanismos da linguagem Erlang

- Pattern Matching (Padrão Emparelhado)
- Module

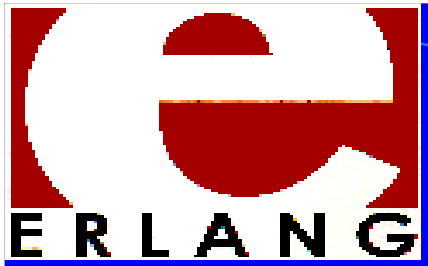
```
-module(datas).
-export([classifica_dias/1]).

classifica_dias(sábado)   -> “fim de semana”;
classifica_dias(domingo)  -> “fim de semana”;
classifica_dias(_)        -> “dia de semana”.
```



Principais mecanismos da linguagem Erlang

- Cláusula Guard
- Primitiva Case

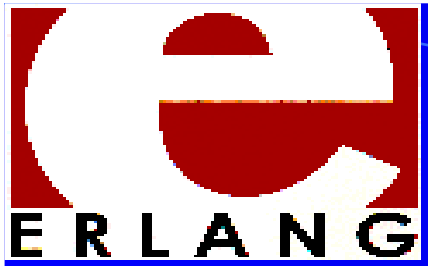
```
case Expr of
Pattern1 [when Guard1] -> Seq1;
Pattern2 [when Guard2] -> Seq2;
...
PatternN [when GuardN] -> SeqN;
True -> Seq99
end.
```



Principais mecanismos da linguagem Erlang

- Primitiva If

```
if
  Guard1 ->
          Sequence1;
  Guard2 ->
          Sequence2;
     ...
  GuardN ->
       SequenceN;
  True -> Sequence99
end.
```



Concorrência em Erlang

- Criação de processos

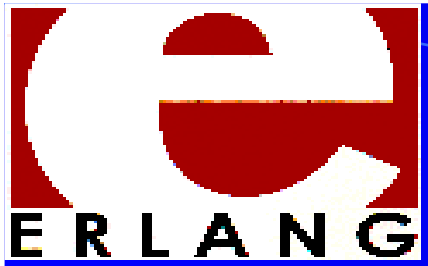
```
Pid = spawn(Module, FunctionName, ArgumentList)
```

- Envio de mensagem

```
Pid ! mensagem
```

```
foo(12) ! bar(baz)
```

O envio de mensagens é assíncrono. O processo envia a mensagem e continua sua execução, sem aguardar que a mensagem chegue ao seu destino ou que esta seja recebida.

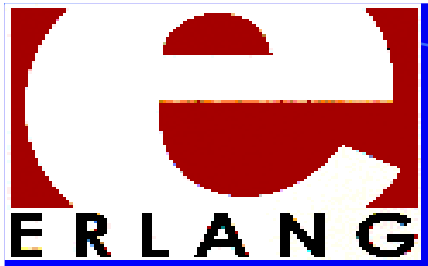


Concorrência em Erlang

- Primitiva receive

```
Receive
  Message1 [when Guard1]  ->
        ações1;
  Message2 [when Guard2]  ->
        ações2;
   ...
end.
```

Quando há o casamento de uma mensagem, e a execução da ***guard*** correspondente tem sucesso, a mensagem é selecionada, removida da caixa postal e então a ação correspondente é avaliada.

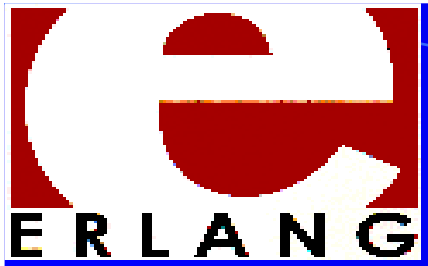


Tempo Real em Erlang

Uma das exigências da linguagem Erlang era que seu uso fosse satisfatório para aplicações de ***soft real-time***, onde o tempo de resposta estivesse na ordem de milissegundos.

- Primitiva receive com timeout

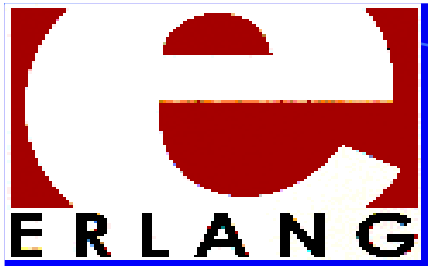
```
Receive
     Message1 [When Guard1]
         Actions1;
     Message2 [When Guard2]
         Actions2;
     ...
     after
         TimeOutExpr -> ActionsT
end.
```



Tempo Real em Erlang

- Timeout independente

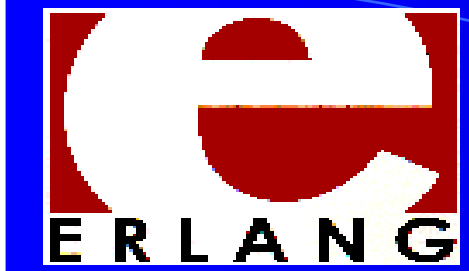
```
-module(timer).
-export([timeout/2, cancel/1, timer/1]).

timeout(Time, Alarm) ->
    spawn(timer, timer, [self(), Time, Alarm]).
cancel(Timer) ->
    Timer ! {self(), cancel}.

timer(Pid, Time, Alarm) ->
    receive
        {Pid, cancel} ->
            true
    after Time ->
        Pid ! Alarm
   end.
```



ESPECIFICAÇÃO

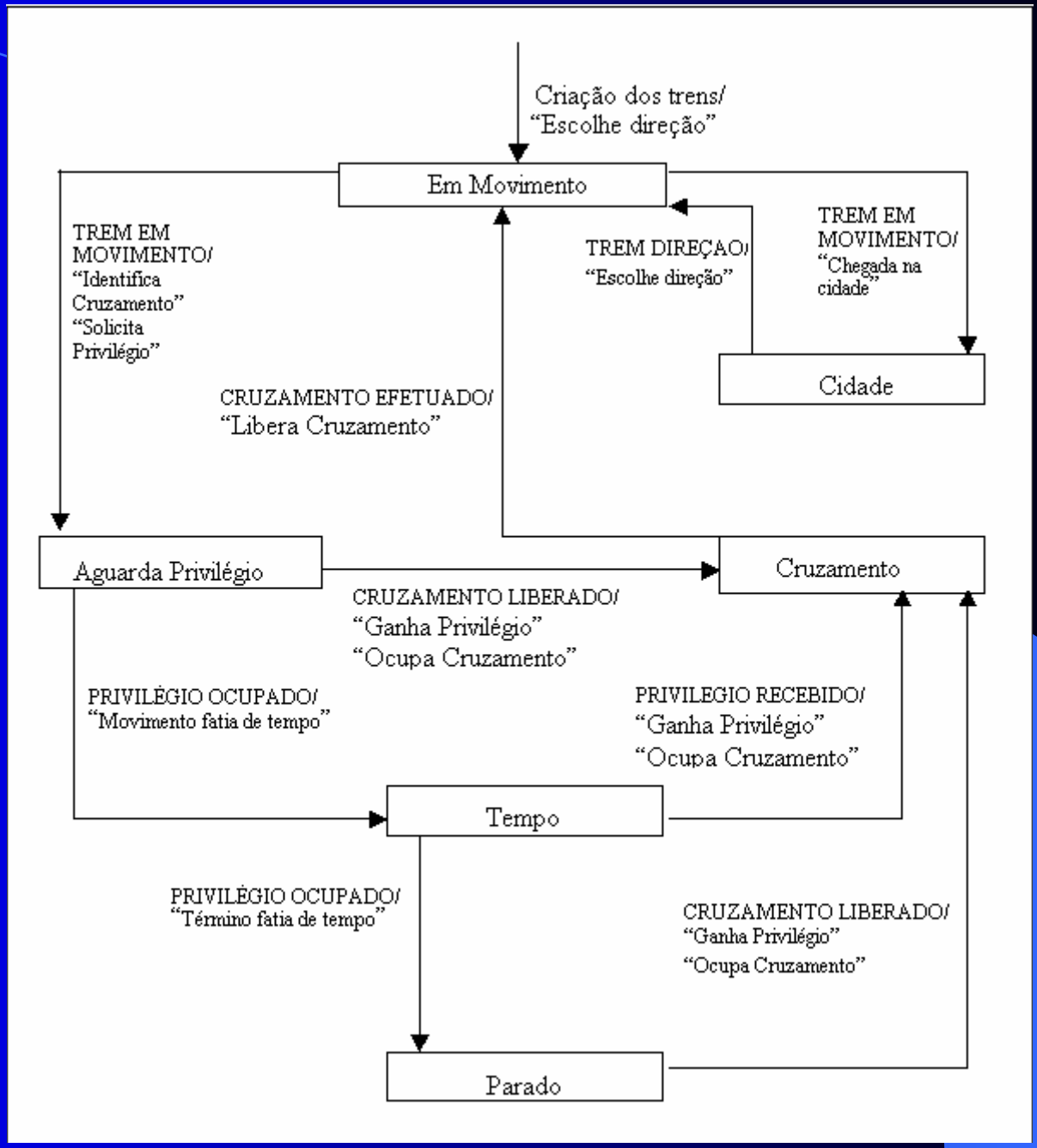
Problema:

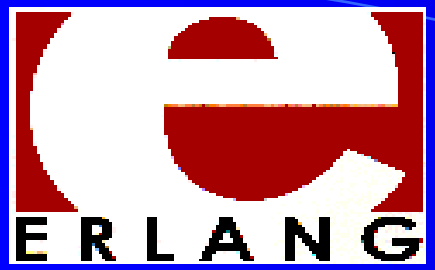
- Implementar o controle de uma malha ferroviária, especialmente o controle dos cruzamentos, utilizando a linguagem Erlang e os conceitos de processos concorrentes e tempo real.



Especificação

Diagrama de Transição de Estados (DTEs)



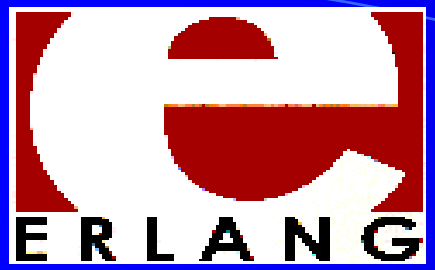


Implementação do semáforo

- **Semáforo binário encontrado em [CAS1998]**

```
%%----------------------------------------------------------------------
%% P(s): se s > 0, então s = s - 1; senão coloca na fila de espera
%%----------------------------------------------------------------------

p(S) ->
        S ! {semaforo, p, self()},        % cont exige o nome do processo
        % espera pela mensagem prosseguir indicando para fila existente
        receive
                {semaforo, cont} ->
                 true
        end.

%%----------------------------------------------------------------------
%% V(s): se a fila de espera não estiver vazia, desperta um; senão s = s + 1
%%----------------------------------------------------------------------

v(S) ->
        S ! {semaforo, v}.
```

```
%%--------------------------------------------------------------------
%% A função mensagem
%%--------------------------------------------------------------------

mensagem(S, L) ->
          io:format("INICIO MENSAGEM S: ~w L: ~w~n", [S,L]),
          {NewS, NewL} =
                    receive
                    {semaforo, p, Pid} ->
                              io:format("MENSAGEM no P - L: ~w PID: ~w~n", [L, Pid]),
                              if
                                        % P(s): se s > 0, então s = s - 1;
                                        S > 0 ->
                                                  Pid ! {semaforo, cont},
                                                  {S - 1, L};
                                        true ->
                                                  % senão coloca na fila de espera
                                                  {S, [Pid | L]}
                                        end;

                    % V(s): se a fila não estiver vazia
                    {semaforo, v} when length(L) =/= 0 ->
                              [H|T] = L,
                              % desperta um;
                              H ! {semaforo, cont},
                              {S, T};
                    % se a lista está vazia
                    {semaforo, v} ->
                              io:format("LIBEROU S= ~w L= ~w~n", [S,L]),
                              % se s = s + 1
                              {S + 1, L}
                    end,
          io:format("MENSAGEM L: ~w NewS: ~w, NewL: ~w~n", [L,NewS,NewL]),
          mensagem(NewS, NewL).
```



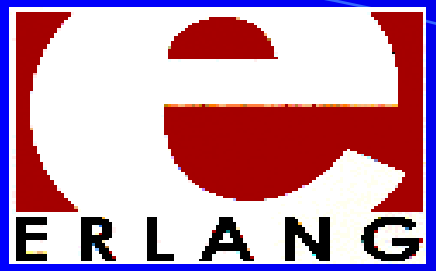
Implementação do tempo

- Para a implementação do tempo real, foi determinado um tempo máximo de 1.500 milissegundos para o trem se aproximar do cruzamento mesmo sem receber a mensagem de liberação do mesmo. Se este tempo terminar, o trem fica parado aguardando a liberação, porém, se a liberação for recebida antes do tempo terminar, o trem continua o seu movimento normalmente, sem perder tempo.

```
tempo1(C, X, Y, W, PC1, PC2, PC3) ->
        io:format("X tempo: ~w  Y: ~w~n", [X,Y]),
        PidTimer1 = spawn (tcc,timer, [self(),1500,alarm]),
        PidMandaP1 = spawn (tcc, mandaP1,[self(),C, X, Y, W, PC1, PC2, PC3]),
        Xatual_a = andaAteCruzamento1 (PidTimer1,C, X, Y, W, PC1, PC2, PC3),
        esvazia_fila (),
        Xatual_a.
```

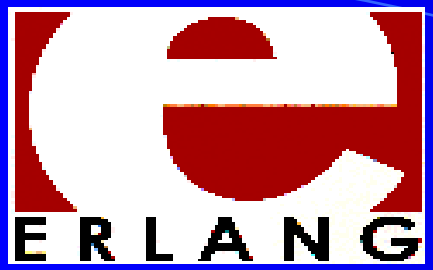
```
mandaP1(Pid,C, X, Y, W, PC1, PC2, PC3) ->
        io:format("X: ~w Y: ~w Pid do MandaP: ~w~n", [X, Y, self()]),
        if X == 185, Y == 220 ->
                p(PC1),
                Pid ! liberouPrivilegio1;
        X == 445, Y == 320 ->
                p(PC2),
                Pid ! liberouPrivilegio1;
        X == 185, Y == 420 ->
                p(PC3),
                Pid ! liberouPrivilegio1
        end.
andaAteCruzamento1 (PidTimer1,C, X, Y, W, PC1, PC2, PC3) ->
        receive
                liberouPrivilegio1 ->
                        cancel (PidTimer1),
                        Xatual_a = X-5;
                alarm ->
                        receive
                                liberouPrivilegio1 -> true,
                                Xatual_a = X-5
                        end
                after 0 ->
                        deleta1 (C, X-5, Y),
                        cria1 (C, X, Y, W),
                        Xatual_a = andaAteCruzamento1 (PidTimer1,C, X+5, Y, W,
                                PC1, PC2, PC3)
        end,
        Xatual_a.
```


ERLANG
Apresentação
do
Protótipo
TCC - Aplicação em Tempo Real utilizando a Linguagem de Programação Erlang - Controle Ferroviário - Amilton Cesar Schmidt
PARE
Quantidade de trens inicializados (máximo 50 trens): 12
Click nos botões da malha ferroviária para inicializar os trens!
ERLANG
A
B
C
D
E
F
G
H
I
J
K
L
Cruzamento São José
Cruzamento São Francisco
Cruzamento Sto Antônio
Cruzamento São José:
Liberado!!!
Cruzamento São Francisco:
Ocupado!!!
Cruzamento Sto Antônio:
Ocupado!!!
Iniciar
Erlang
TCC - Aplicação em ...
20:59



CONCLUSÃO

- **Pesquisa**
- **Linguagem Erlang**
- **Objetivo**



Sugestões para possíveis extensões

- **pesquisar e utilizar o modelo *Specification and Description Language* (SDL), sugerido em [ARM1996];**
- **implementar o protótipo com exclusividade em relação às ferrovias;**
- **desenvolver uma avaliação do desempenho da linguagem Erlang, fazendo uma comparação com outras linguagens;**
- **desenvolver um algoritmo mais genérico para a implementação do protótipo da malha ferroviária;**
- **desenvolver uma aplicação na área de telecomunicações utilizando Erlang, visto que a finalidade primeira quando da criação da linguagem, foi para esta área.**

FIM...

OBRIGADO!!!