1) Identificar entidades activas [EA] (instanciáveis como threads ou processos)

Cada EA tem a si associada um "programa" (uma função).

2) Comunicação entre EA (por partilha de memória):

Identificar a estrutura que é partilhada. Notar que a partilha de informação é muito mais de que simplemente a partilha de uma variável. As operações que definem a estrutura partilhada é que são essenciais (não esquecer que a noção de monitor foi inspirada para programação OO).

Por exemplo, uma mesma variável inteira partilhada, pode ter a si associada operações atómicas bastante diversas (dependendo da estrutura que de facto está-se a pretender partilhar). Assim pode ser simplesmente um repositório para um valor inteiro (operações get e set); pode ser um contador (operações: get, set, increment, decrement); uma data representada com uma variável inteira indicando o número de dias a partir de uma data referência (operações: set, day, month, year, increment, ...).
Assim uma mesma variável inteira pode requerer operações atómicas bastante diversas.

Dois problemas: atomicidade nas operações partilhadas; acesso condicional a operações parciais (aquelas que têm uma pré-condição).

2.1) Atomicidade (sincronização interna)

A implementação mais simples é tratar todas essas operações como sendo exclusivas entre si. Isto é, protegê-las por exclusão mútua (mutex).
Em todas as operações (normalmente são funções ou métodos):

```
lock(mtx)
... // operação
unlock(mtx)
```

NOTA: a variável mtx *tem* de ser a mesma! Não podemos usar dois mutexes diferentes para implementar *uma* região critica.

2.2.) Acesso condicional (sincronização condicional)

Associar a cada condição uma variável de condição. Cada condição diferente pode (e deve) ter uma variável de condição diferente (todas partilhando a mesma variável mutex).

Em todas as operações parciais (com pré-condição condition):

```
lock(mtx)
while(!condition)
   wait(cnd, mtx)
... // operação parcial
unlock(mtx)
```

Nas operações que podem tornar a condição verdadeira:

```
lock(mtx)
... // operação
broadcast(cnd)
unlock(mtx)
```