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

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

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

Identificar a estrutura que é partilhada. Notar que a partilha de informação é muito mais de que simplesmente 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 diferentes.

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

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

Uma condição é concorrente se o seu valor (puder) depender de outra EA (por exemplo, a pré-condição not_empty da função que retira um elemento de uma fila partilhada).

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)
```

(Numa implementação com memória partilhada e semáforos, ver o documento que mostra como se pode usar o padrão de programação interna e condicional aqui apresentado, com estes mecanismos.)

(3) Comunicação directa entre EA (por mensagens):

Identificar os pontos de comunicação: emissor e receptor (E -> R).

A comunicação pode envolver uma mensagem binária simples (on/off, sim/não, true/false); ou mensagens mais complexas (inteiro, estrutura de dados, ...).
A mensagem pode residir em memória partilhada, sendo que o emissor coloca a mensagem e notifica o receptor. O receptor, espera a notificação, confirma a recepção da mensagem, e depois vai buscá-la.

No caso particular de mensagens binárias, a comunicação pode recorrer tão só a um semáforo binário (inicializado a zero).