Artesanato de Aveiro*

Uma empresa de artesanato de Aveiro, dedicada à promoção de produtos da região, é formada por uma oficina de fabrico e por uma loja de exposição onde são unicamente vendidos os produtos produzidos pela empresa.

A dona da empresa encarrega-se pessoalmente da comercialização dos produtos, do seu transporte entre a oficina e a loja de exposição e da obtenção das matérias primas necessárias à produção. Na oficina, trabalham vários artesãos.

A empresa efectua o fabrico de um só tipo de produto. Cada unidade é manufacturada do princípio ao fim pelo mesmo artesão, socorrendo-se de diversos tipos de matéria-prima, em quantidade e variedade bem definidas. Quando a quantidade de produtos fabricados ultrapassa um certo limite, um dos artesãos contacta a dona da empresa para que ela os venha recolher e os coloque em exposição na loja para venda. Quando, por outro lado, a quantidade de alguma das matérias primas armazenadas baixa para além de um certo limite, um dos artesãos avisa igualmente a dona da empresa para que esta procure repor o stock.

Os clientes vão à loja para comprar os produtos expostos. Várias situações podem ocorrer: a loja está fechada – a dona ausentou-se a pedido de um dos artesãos para recolher os produtos já manufacturados ou para comprar novas matérias-primas – os clientes desistem momentaneamente e regressam mais tarde; a loja está aberta – os clientes entram e avaliam os produtos expostos, saindo sem comprar nada, ou seleccionando eventualmente produtos para compra. Neste último caso, formam uma fila junto ao balcão para efectuar a transacção, aguardando ser atendidos pela proprietária.

Assuma como parâmetros do problema que há três clientes, que a empresa emprega três artesãos e que as matérias-primas usadas no fabrico do produto são processadas em conjunto.

Construa uma simulação do ciclo de vida da dona da empresa, dos artesãos e dos clientes usando um dos modelos estudados de sincronização e de comunicação entre processos (*threads*): semáforos e memória partilhada, monitores ou passagem de mensagens.

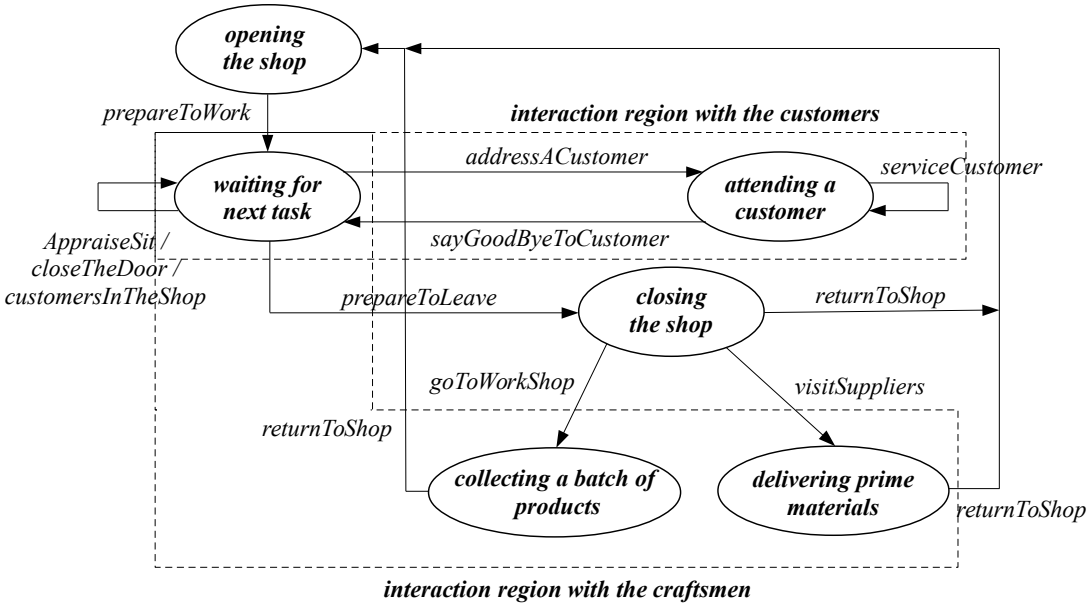
A solução deve ser implementada em Linguagem C e ser passível de execução em Linux.

Incorpore um ficheiro de *logging* que descreva de um modo conciso, mas claro, a evolução do estado interno das diversas entidades envolvidas.

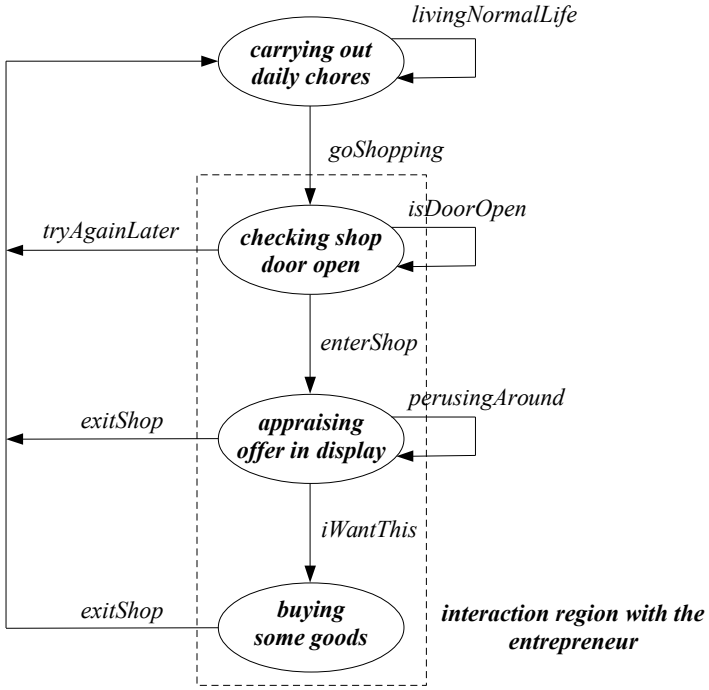
* Ideia original de Pedro Mariano

Sugestão de abordagem à solução

Ciclo de vida da dona da empresa

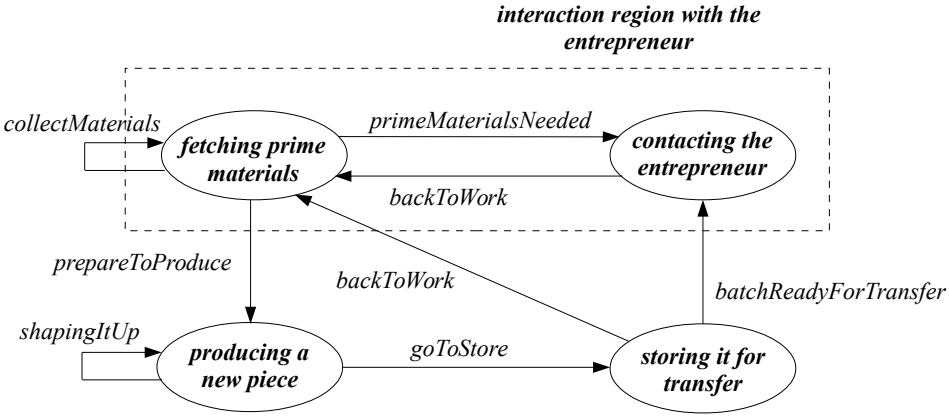


Ciclo de vida dos clientes



Nota – as combinações de operações *isDoorOpen-enterShop* e *isDoorOpen-TryAgainLater* dos clientes são percebidas como atómicas pela dona da empresa.

Ciclo de vida dos artesãos




```
/* simulation parameters */

#define  N       3                                    /* number of customers */
#define  M       3                                    /* number of craftsmen */

/* entrepreneur process (thread) */

void main (void)
{
  unsigned int c;                                              /* customer id */
  char nextTask;                                    /* task to be carried out */
  boolean busy;                                  /* in-the-shop signaling flag
                                        TRUE if the entrepreneur is in the shop */

  while (!endOperEntrep ())
  { prepareToWork ();                      /* the entrepreneur opens the shop
                                          and gets ready to perform her duties */
    busy = TRUE;                                             /* set busy flag */
    while (busy)
    { nextTask = appraiseSit ();                /* the entrepreneur waits for
                                                            service requests */
      if (nextTask == 'C')                     /* the entrepreneur checks if a
                                                 customer is needing attention */
         { c = addressACustomer (); /* the entrepreneur goes to the counter
                                                          to attend a customer */
           serviceCustomer ();    /* the entrepreneur services the customer */
           sayGoodByeToCustomer (c);          /* the entrepreneur completes
                                                            the transaction */
         }
         else { busy = FALSE;                                /* reset busy flag */
                /* the entrepreneur checks if some craftsman has phoned him */
                if ((nextTask == 'G') || (nextTask == 'P'))
                   { if (customersInTheShop ())
                        { closeTheDoor ();   /* the entrepreneur closes the
                                 door to prevent further customers to enter */
                          busy = TRUE;                          /* set busy flag */
                        }
                   }
              }
    }
    prepareToLeave ();                    /* the entrepreneur closes the shop */
    if (nextTask == 'G')
       goToWorkShop ();          /* the entrepreneur collects a batch of goods */
       else if (nextTask == 'P')
               visitSuppliers ();     /* the entrepreneur goes shopping for
                             prime materials and delivers them to the workshop */
    returnToShop ();                /* the entrepreneur goes back to the shop */
  }
}
```

```
/* customer processes (threads),   n = 0,1,...,N-1 */

void main (unsigned int n)
{
  unsigned int ng;                                /* number of selected goods */

  while (!endOperCustomer (n))
  { while (true)
    { livingNormalLife ();           /* the customer minds his own business */
      goShopping (n);                  /* the customer decides to visit the
                                                                 handicraft shop */
      if (isDoorOpen ())    /* the customer checks if the shop door is open */
         break;                                                      /* it is */
         else tryAgainLater (n);       /* it is not, the customer goes back
                                                      to perform his daily chores */
    }
    enterShop (n);                             /* the customer enters the shop */
    ng = perusingAround (n);  /* the customer inspects the offer in display
                                        and eventually picks up some goods */
    if (ng != 0)
       iWantThis (n, ng);             /* the customer queues by the counter
                                                 to pay for the selected goods */
    exitShop (n);                             /* the customer leaves the shop */
  }
}

/* craftsmen processes (threads),   m = 0,1,...,M-1 */

#define  MAX     4    /* storeroom nominal capacity (in number of products) */

void main (unsigned int m)
{
  unsigned int np;                             /* number of products in store */
  bool alert;                        /* low level of prime materials in store */

  while (!endOperCraftsman (m))
  { alert = collectMaterials (m);            /* the craftsman gets the prime
                                 materials he needs to manufacture a product */
    if (alert)
       { primeMaterialsNeeded (m); /* the craftsman phones the entrepreneur
                 to let her know the workshop requires more prime materials */
         backToWork (m);     /* the craftsman returns to his regular duties */
       }
    prepareToProduce (m);                     /* the craftsman sits down and
                          prepares things for the production of a new piece */
    shapingItUp ();                    /* the craftsman works on a new piece */
    np = goToStore (m);        /* the craftsman stores the finished product */
    if (np >= MAX)            /* the craftsman checks if it is transfer time */
       batchReadyForTransfer (n);              /* the craftsman phones the
                                    entrepreneur to let her know she should
                                   come and collect a new batch of products */
    backToWork (m);           /* the craftsman returns to his regular duties */
  }
}
```

```
/* generic parameters */

#define  NP          4      /* number of times prime materials are supplied */

/* state information data structures */

typedef struct
        { unsigned int stat,                                /* internal state */
                       boughtPieces;      /* amount of pieces so far bought */
          bool readyToWork;                /* activity flag required by the
                  simulation: true - customer is active / false - otherwise */
        } STAT_CUST;

typedef struct
        { unsigned int stat,                                /* internal state */
                       prodPieces;      /* amount of pieces so far produced */
          bool readyToWork;                /* activity flag required by the
                 simulation: true - craftsman is active / false - otherwise */
        } STAT_CRAFT;

typedef struct
        { unsigned int entrepStat;                    /* entrepreneur state */
          STAT_CUST custStat[N];                   /* customers state array */
          STAT_CRAFT craftStat[M];                 /* craftsman state array */
        } STAT;

/* data structures describing the shop */

typedef struct
        { unsigned int mem[N],           /* storage region for customer ids */
                       ii, ir;          /* insertion and retrieval pointers */
          bool full;                                    /* memory full flag */
        } QUEUE;

#define  SOPEN           0                              /* the shop is open */
#define  SDCLOSED        1                        /* the shop door is closed */
#define  SCLOSED         2                             /* the shop is closed */

typedef struct
        { unsigned int stat,              /* the shop status: either SOPEN,
                                                        or SDCLOSED, or SCLOSED */
                       nCustIn,          /* number of customers in the shop */
                       nProdIn;           /* number of products in the shop */
          bool prodTransfer,              /* flag signaling a craftsman has
                            phoned the entrepreneur requesting the transfer
                                                    of a new batch of products */
               primeMatReq;               /* flag signaling a craftsman has
                          phoned the entrepreneur asking for the deliver of
                                                        more prime materials */
          QUEUE queue;                /* queue by the counter formed by the
                                            customers which want to buy goods */
        } SHOPINFO;

typedef struct
        { unsigned int nPMatIn,             /* amount of prime materials in
                                                               the work shop */
                       nProdIn,           /* number of finished products in
                                                               the storeroom */
                       NSPMat,           /* number of times prime materials
                                                             have been supplied */
                       NTPMat,           /* total amount of prime materials
                                                                   supplied */
                       NTProd;           /* total number of pieces produced */
        } WORKSHOPINFO;
```

```
/* full state information */

typedef struct
        { STAT st;                    /* state of all intervening entities */
          SHOPINFO shop;                              /* state of the shop */
          WORKSHOPINFO workShop;                /* state of the work shop */
          unsigned int primeMaterials[NP];     /* amount of prime materials
                                                        supplied each time */
        } FULL_STAT;
```

Caracterização da interacção

Dona da empresa

OPENING_THE_SHOP – estado inicial / final (de transição)
WAITING_FOR_NEXT_TASK – estado de transição com espera eventual
a dona da empresa bloqueia se não houver pedidos de serviço e é acordada por um cliente que pretende comprar um ou mais ou mais produtos, *iWantThis*, ou que abandona a loja, *exitShop*, ou por um artesão que requer o envio de mais matérias primas, *primeMaterialsNeeded*, ou que pretende a transferência de produtos acabados da oficina para a loja, *batchReady ForTransfer*
ATTENDIND_A_CUSTOMER – estado independente
na simulação, a dona da empresa deve ser posta a dormir durante um intervalo de tempo aleatório
CLOSING_THE_SHOP – estado de transição
DELIVERING_PRIME_MATERIALS – estado de transição
COLLECTING_A_BATCH_OF_PRODUCTS – estado de transição

Clientes

CARRYING_OUT_DAILY_CHORES – estado independente
na simulação, o cliente deve ser posto a dormir durante um intervalo de tempo aleatório
CHECKING_DOOR_OPEN – estado de transição
APPRAISING_OFFER_IN_DISPLAY – estado de transição
BUYING_SOME_GOODS – estado de bloqueio
o cliente é acordado pela dona da empresa quando completa a transacção, *sayGoodByeToCustomer*

Artesãos

FETCHING_PRIME_MATERIALS – estado de transição com bloqueio eventual
o artesão bloqueia se não tiver matérias primas para produzir um novo produto e é acordado pela dona da empresa quando fornece mais matérias primas, *visitSuppliers*
PRODUCING_A_NEW_PIECE – estado independente
na simulação, o artesão deve ser posto a dormir durante um intervalo de tempo aleatório
STORING_IT_FOR_TRANSFER – estado de transição
CONTACTING_THE_ENTREPRENEUR – estado de transição

Comunicação e sincronização

Monitores

```
FULL_STAT fSt;                                  /* full state of the problem */
condition proceed;            /* entrepreneur appraise situation sync point */
condition waitForService[N];                /* customers waiting for service
                                          sync point array (one per customer) */
condition waitForMaterials;        /* craftsmen waiting for prime materials
                                                                 sync point */
unsigned int nWakeUp;        /* number of service requests generated by the
               customer or craftsman threads which are still pending to be
                                        processed by the entrepreneur thread */
unsigned int nCraftsmenBlk;             /* number of craftsman threads which
               are blocked waiting for the availability of prime materials */
bool aboutToEnter;            /* flag signaling the intention of a customer
                           to enter the shop (required to impose atomicity) */
unsigned int nCustAboutToEnter;              /* number of customers about to
                                       enter (required to impose atomicity) */
```

Semáforos e memória partilhada

```
shared FULL_STAT fSt;                           /* full state of the problem */
shared unsigned int access;             /* identification of critical region
                                             protection semaphore - val = 1 */
shared unsigned int proceed;               /* identification of entrepreneur
                                    appraise situation semaphore - val = 0 */
shared unsigned int waitForService[N];       /* identification of customers
           waiting for service semaphore array - val = 0 (one per customer) */
shared unsigned int waitForMaterials;        /* identification of craftsmen
                             waiting for prime materials semaphore - val = 0 */
shared unsigned int nCraftsmenBlk;     /* number of craftsman threads which
               are blocked waiting for the availability of prime materials */
```

Passagem de mensagens

```
message mfs;          /* full state of the problem plus number of craftsman
                  threads which are blocked waiting for the availability of
                                                    prime materials message */
message sync;                                     /* synchronization message */
mailbox access;                          /* main storage for the shared data */
mailbox accessAux;                 /* auxiliary storage for the shared data */
mailbox proceed;                                  /* storage for entrepreneur
                                          appraise situation synchronization */
mailbox waitForService[N];             /* storage for  customers waiting for
                         service a group synchronization (one per customer) */
mailbox waitForMaterials;                          /* storage for craftsmen
                                 waiting for prime materials synchronization */
```