

Software Testing and Validation

A.A. 2023/2024

Testo del Progetto

Igor Melatti

1 Come si consegna

Il presente documento descrive le specifiche per il progetto d'esame. La consegna deve consistere in un singolo file **STV_2023_2024_matricole.zip** (se il progetto è fatto in gruppo, scrivere tutte le matricole separate dall'underscore `_`), che contenga un'unica directory **STV_2023_2024_matricole**, la quale a sua volta deve includere:

- un file PDF **relazione.pdf** con le seguenti caratteristiche:
 - deve indicare nome, cognome e matricola di ogni studente/studentessa del gruppo;
 - deve descrivere come il progetto sia stato svolto;
- un file PDF **presentazione.pdf** che da usare come slide per illustrare lo svolgimento del progetto;
- una directory **progetto** contenente tutti i file che fanno parte del progetto, con un'opportuna organizzazione in sottodirectory.

Il suddetto file **STV_2023_2024_matricole.zip** andrà poi inviato per email al docente **igor.melatti@univaq.it**.

È possibile consultarsi con i compagni. Tuttavia, occorre che ciascun gruppo presenti una propria soluzione personale.

2 Esercizio per Gruppi da 3 Studenti/Studentesse

Si consideri un sistema che modella un'organizzazione che vende oggetti tramite un sito Web; occorre considerare sia i clienti, che i fornitori, che i consegnatori. Ci sono 2 versioni protipali di tale sistema: della prima versione c'è solo il modello, specificato nella Sezione 2.1, mentre della seconda c'è sia un modello (specificato nella Sezione 2.2) che un'implementazione, fornita separatamente nel file `code.tgz`.

2.1 Modello del Primo Sistema

È presente un DB con le seguenti tabelle:

- ACTOR(id, type, descr)
- ITEM(id, descr, producer_ids)
- ORDER(id, customer_id, date_placed, date_due, date_shipped, date_arrived, item_id, how_many, processed)
- STORAGE(id, order_id, how_many, producer_id)

Il sistema vero e proprio è costituito da 3 sottosistemi: uno per gestire i clienti (Algorithm 7), uno per gestire i produttori (Algorithm 1–Algorithm 4) e uno per gestire i consegnatori (Algorithm 9). Il sistema comunica con l'ambiente esterno tramite siti Web, che qui vengono visti come generatori di messaggi: Algorithm 8 mostra come vengono generate le richieste da parte dei $c \geq 1$ clienti, Algorithm 5 e Algorithm 6 quelle dei $p \geq 1$ produttori e Algorithm 10 quelle dei $s \geq 1$ consegnatori.

Sono presenti i seguenti canali di comunicazione (per il nome completo in REDIS, basta aggiungere all'inizio `work:queue:`, quindi ad esempio il nome completo del canale CP è `work:queue:CP`):

CP, PC : dal sistema di gestione clienti al sistema di gestione produttori e viceversa

PII_p, POI_p : dal sistema di gestione produttori all'ambiente del produttore p -esimo e viceversa; usato per chiedere/ottenere la disponibilità di magazzino che un produttore ha per un certo prodotto

PIP_p, POP_p : dal sistema di gestione produttori all'ambiente del produttore p -esimo e viceversa; usato per chiedere di riservare un certo numero di istanza di un certo prodotto per soddisfare un certo ordine, ed ottenere la comunicazione dell'avvenuta produzione; tali unità potrebbero essere già in magazzino oppure necessitare una produzione ad-hoc

PPI_p, PPO_p : dall'ambiente del produttore p -esimo all'ambiente del “vero” produttore p -esimo e viceversa; usato per chiedere di produrre un certo prodotto per il quale la quantità di magazzino non era sufficiente, ed ottenere la comunicazione dell'avvenuta produzione

CI_c, CO_c : dal sistema di gestione clienti all'ambiente del cliente c -esimo e viceversa; usato per chiedere/ottenere la disponibilità di magazzino per un certo prodotto o per piazzare un nuovo ordine/ottenere risposta

SI_s, SO_s : dal sistema di gestione consegnatori all'ambiente del consegnatore s -esimo e viceversa; usato per chiedere di consegnare un prodotto pronto per la consegna/ottenere una risposta di presa in carico della consegna stessa

SOA_s : dall'ambiente del consegnatore s -esimo al sistema di gestione dei consegnatori; usato per indicare che l'ordine è stato consegnato

```

1 function ProducerAPI()
2   while True do
3     AnswerItemAvailReq();
4     SendProd();
5     CollectFromProd();

```

Algorithm 1: Sistema: gestione produttori

```

1 function AnswerItemAvailability()
2   if  $CP \neq \emptyset$  then
3      $i \leftarrow$  (blocking) read from CP;
4      $P \leftarrow$  read producers for item  $i$  from ITEM;
5      $m \leftarrow 0$ ;
6     foreach  $p \in P$  do
7       send  $i$  to PIIp;
8        $m \leftarrow m + \text{BlockingRead}(\text{POI}_p)$ ;
9     send  $m$  to PC;

```

Algorithm 2: Sistema: gestione produttori, funzione ausiliaria

2.2 Modello del Secondo Sistema

Le premesse sono le stesse di cui sopra per quanto riguarda il DB, i canali e lo scopo generale del sistema.

I pseudo-codici sono dati nel seguito, da [Algorithm 11](#) a [Algorithm 21](#).

Per quanto riguarda l'implementazione, presuppone Linux con i seguenti pacchetti: PostgreSQL, REDIS, Python3 con pycopg-2. Per eseguire i processi

```

1 function SendProd()
2    $O \leftarrow$  read orders from ORDER s.t. processed is false;
3   foreach  $(o, c, d_1, d_2, i, n, \text{false}) \in O$  do
4     UPDATE ORDER set processed to true where id =  $o$ ;
5      $P \leftarrow$  read producers for item  $i$  from ITEM;
6      $m \leftarrow 0$ ;
7     foreach  $p \in P$  do
8       send  $(i, n - m)$  to PII $p$ ;
9        $m' \leftarrow$  BlockingRead(POIp);
10      if  $m' > 0$  then
11        INSERT INTO STORAGE  $(o, m', p, \text{false})$ ;
12         $m \leftarrow m + m'$ ;
13      if  $m \geq n$  then break;
14      if  $n > m$  then
15         $(a, b) \leftarrow (\frac{n-m}{|P|}, (n-m) \bmod |P|)$ ;
16        if  $a \neq 0$  then
17          foreach  $\ell \in P$  do
18            send  $(i, o, a)$  to PIP $\ell$ ;
19        foreach  $\ell \in \{1, \dots, b\}$  do
20          send  $(i, o, 1)$  to PIP $\ell$ ;

```

Algorithm 3: Sistema: gestione produttori, funzione ausiliaria

```

1 function CollectFromProd()
2   // If a producer produced something as an answer to a processed
   // order, record it on the DB
3    $P \leftarrow$  read all producers from ACTOR;
4   foreach  $\ell \in P$  do
5     if  $POP\ell \neq \emptyset$  then
6        $(o, n) \leftarrow$  (blocking) read from POP $\ell$ ;
7       INSERT INTO STORAGE  $(i, o, n, \ell, \text{false})$ ;

```

Algorithm 4: Sistema: gestione produttori, funzione ausiliaria

```

1 function Producer( $\ell, \tilde{p}, l, I$ )
2    $p, p' \leftarrow \tilde{p}, 0$ ;
3   while True do
4     // Manage incoming messages from Producer API, if any
5     if  $PII\ell \neq \emptyset$  then
6        $X \leftarrow$  (blocking) read from  $PII\ell$ ;
7       if  $X = (i, n)$  then
8         // check availability and book it
9         send  $\min\{p_i, n\}$  to  $POI\ell$ ;
10         $p_i \leftarrow p_i - \min\{p_i, n\}$ ;
11      else
12        //  $X = i$ : simply check availability for item  $i$ 
13        send  $p_i$  on  $POI\ell$ ;
14      if  $PIPl \neq \emptyset$  then
15         $(i, o, n) \leftarrow$  (blocking) read from  $PIPl$ ;
16        // produce  $n$  objects of item  $i$  because of an order  $o$ 
17        if  $n - p_i > 0$  then
18          send  $(i, n - p_i, o)$  to  $PPI\ell$ ;
19           $p_i \leftarrow 0$ ;
20        else
21           $p_i \leftarrow p_i - n$ ;
22          send  $(o, n)$  on  $POP\ell$ ;
23      foreach  $i \in I$  do
24        if  $l_i > p_i + p'_i$  then
25          send  $(i, l_i - p_i, \perp, T_i)$  to  $PPI\ell$ ;
26           $p'_i \leftarrow l_i - p_i$ ;
27      if  $PPO\ell \neq \emptyset$  then
28         $i, m, o \leftarrow$  (blocking) read from  $PPO\ell$ ;
29        if  $o \neq \perp$  then
30          send  $(o, m)$  on  $POP\ell$ ;
31        else
32           $p_i, p'_i \leftarrow m, 0$ ;

```

Algorithm 5: Ambiente: generazione richieste produttori

```

1 function ProducerPhys( $\ell$ )
2   while True do
3     if  $PPI\ell \neq \emptyset$  then
4        $i, m, o \leftarrow$  read from  $PPI\ell$ ;
5       sleep(random);
6       send  $i, m, o$  to  $PPO\ell$ ;

```

Algorithm 6: Ambiente: gestione richieste dei “veri” produttori

```

1 function ClientAPI()
2   while True do
3      $C \leftarrow$  read all clients from ACTOR;
4     foreach  $\ell \in C$  do
5       if  $CO\ell \neq \emptyset$  then
6          $X \leftarrow$  (blocking) read from  $CO\ell$ ;
7         if  $X = (i, n, d)$  then
8           INSERT INTO ORDER ( $\ell$ , now(), now() +  $d$ ,  $\perp$ ,  $\perp$ ,  $i$ ,  $n$ , false) ;
9           send OK to  $CI\ell$ ;
10        else
11          //  $X = i$ 
12          send  $i$  to CP;
13           $r \leftarrow$  BlockingRead(PC);
14          send  $r$  to  $CI\ell$ ;

```

Algorithm 7: Sistema: gestione dei clienti

```

1 function Client( $\ell, I, N, f, D$ )
2   while True do
3      $J \leftarrow$  random( $2^I$ );
4     foreach  $j \in J$  do
5       Order( $\ell, j$ , random( $N_j$ ),  $f$ , random( $D_j$ ));
6   //
7   // Function which decides what and how much to order
8   //
9   function Order( $\ell, j, n, f, d$ )
10    send  $j$  on  $CO\ell$ ;
11     $m \leftarrow$  BlockingRead( $CI\ell$ );
12    if  $n \leq m \vee$  random( $0, 1$ ) >  $f$  then
13      send ( $j, n, d$ ) on  $CO\ell$ ;
14       $\perp \leftarrow$  BlockingRead( $CI\ell$ );

```

Algorithm 8: Ambiente: generazione richieste clienti

```

1 function ShipperAPI()
2   while True do
3      $O \leftarrow$  read orders from ORDER s.t. processed is true;
4      $S \leftarrow$  read all shippers from ACTOR;
5     foreach  $(o, c, d_1, d_2, i, n, \text{true}) \in O$  do
6        $S \leftarrow$  read from STORAGE s.t. order_id is  $o$  and  $o.\text{date\_shipped}$  is
         null;
7        $m \leftarrow$  sum all how_many from  $S$ ;
8       if  $m \geq n$  then
9         foreach  $\ell \in S$  do
10           // better to maintain an array of busy shippers...
11           send  $(i, o, n)$  to  $SI\ell$ ;
12            $d \leftarrow$  blocking read with timeout from  $SO\ell$ ;
13           if  $d \neq \perp$  then
14             send OK to  $SI\ell$ ;
15             UPDATE ORDER set date_shipped to  $d$  on  $o$ ;
16             break;
17           send NO to  $SI\ell$ ;
18         foreach  $\ell \in S$  do
19           if  $SOA\ell \neq \emptyset$  then
20              $X \leftarrow$  (blocking) read from  $SOA\ell$ ;
21             if  $X = (o, d)$  then
22               UPDATE ORDER set date_arrived to  $d$  on  $o$ ;

```

Algorithm 9: Sistema: gestione dei consegnatori

```

1 function Shipper( $\ell, \tilde{T}, T$ )
2   while True do
3     if  $SI\ell \neq \emptyset$  then
4        $X \leftarrow$  (blocking) read from  $SI\ell$ ;
5       if  $X = NO$  then continue;
6       // Otherwise,  $X = (i, o, n)$ 
7       send now() to  $SO\ell$ ;
8        $x \leftarrow$  BlockingRead( $SI\ell$ );
9       if  $x = OK$  then
10        sleep(random);
11        send  $(o, \text{now}())$  to  $SOA\ell$ ;

```

Algorithm 10: Ambiente: generazione richieste dei consegnatori

che sono nel sistema, occorre prima creare 2 database PostgreSQL; per fare ciò, usare gli script `src/system/psql/db.sql` e `src/environment/psql/db.sql`, che creano dei database di nome `main_db_website` e `env_db_website`, rispettivamente. Qualora dei database con questi nomi esistano già (come accade dopo una esecuzione del sistema...), occorre prima eseguire un `drop database` per entrambi.

È necessario inoltre che sia attivo un server Redis, ad esempio sulla porta di default 6379. Anche qui, occorre prestare attenzione alle esecuzioni successive alla prima: dato che potrebbero essere rimasti messaggi nelle code REDIS, bisogna azzerare ogni coda c con il comando `DEL c` dato all'interno del client di redis (ad es.: `redis-cli <<< "DEL work:queue:PC"`)

Infine, affinché si possa eseguire il sistema, siano N_c, N_p, N_s , rispettivamente, il numero di clienti, di produttori e di consegnatori nell'ambiente (per esempio, dei tipici valori potrebbero essere rispettivamente 5, 1 e 1). È quindi necessario che i database vengano (parzialmente) caricati con dei valori in alcune tabelle. Più in dettaglio:

- Per il database `main_db_website`, occorre caricare le seguenti tabelle nel seguente modo:
 - la tabella `customer` deve contenere N_c righe, ognuna con `descr` qualsiasi (ad es., settato a NULL) e `id = channel_code = i`, per $i = 1, \dots, N_c$;
 - lo stesso per le tabelle `producer` e `shipper`, con N_p, N_s rispettivamente;
 - la tabella `item` deve contenere k righe (con k parametro della fase di verifica), dove `descr` può essere qualsiasi ed `id` è crescente da 1 a k ;
 - la tabella `item_producer` deve contenere, per ogni riga con `id = i` della tabella `item`, almeno una riga con `item_id = j` e `producer_id = p`, dove p è tra i valori dell'attributo `id` di `producer`.
- Per il database `env_db_website`, occorre caricare le seguenti tabelle nel seguente modo:
 - la tabella `function_info` deve contenere $N = 5 + N_c + 2N_p + N_s$ righe, ognuna con `queue_in_mon_sl`, `queue_out_mon_sl` qualsiasi, `id` crescente da 1 a N e `fname` di volta in volta uguale a:
 - * `customer`
 - * `shipper`
 - * `producer`
 - * `producer_prod_ans`
 - * `ext_monitor`
 - * `customer_envi`, per $i = 1, \dots, N_c$
 - * `shipper_envi`, per $i = 1, \dots, N_s$
 - * `producer_envi`, per $i = 1, \dots, N_p$

- * `producer_phys_envi`, per $i = 1, \dots, N_p$
- la tabella `function_time` deve contenere, per ogni valore di `id= i` della tabella `function_info`, almeno una riga con `function_id= i`, `single_time` settato ad un qualche valore positivo (attenzione: vengono tradotte in `sleep`, quindi valori troppo alti implicano maggiori attese nell'esecuzione), e `statm_type` settato ad una delle seguenti stringhe: `insert`, `poll`, `recv`, `select`, `send`, `recv`, `update`, `min_sleep`, `main_sleep`. L'attributo `id` può essere semplicemente crescente da 1 al numero di righe totale. Per ogni coppia (`function_id`, `statm_type`), ci possono essere più valori di `single_time`;
- la tabella `item_customer_other_info` deve contenere, per ogni riga con `id= i` della tabella `item` e `id= c` della tabella `customer`, almeno una riga con `item_id= i`, `customer_id= c`, `order_num` e `order_due_date` settati a numeri interi positivi;
- la tabella `item_producer_other_info` deve contenere, per ogni riga con `id= i` della tabella `item` e `id= p` della tabella `producer`, almeno una riga con `item_id= i`, `producer_id= p`, `prod_starting`, `prod_min` e `prod_time` settati a numeri interi positivi (attenzione, l'ultimo viene tradotto in una `sleep`);
- la tabella `item_shipper_other_info` deve contenere, per ogni riga con `id= i` della tabella `item` e `id= s` della tabella `shipper`, almeno una riga con `item_id= i`, `shipper_id= s`, e `ship_time` settato ad un numero intero positivo (attenzione, viene tradotto in una `sleep`).

Una volta fatto ciò, un modo di eseguire i diversi processi è il seguente (R è un qualsiasi intero random):

- `system/customer/bin/main` 127.0.0.1 6379 `main_db_website`
`main_db_website_user` `main_db_website_user_password` 127.0.0.1
5432 R
- `system/ext_monitor/bin/main` 127.0.0.1 6379 `main_db_website`
`main_db_website_user` `main_db_website_user_password` 127.0.0.1
5432 `monitor_log.csv` R
- `system/producer/bin/main` 127.0.0.1 6379 `main_db_website`
`main_db_website_user` `main_db_website_user_password` 127.0.0.1
5432 R
- `system/producer/bin/main.prod_ans` 127.0.0.1 6379 `main_db_website`
`main_db_website_user` `main_db_website_user_password` 127.0.0.1
5432 R
- `system/shipper/bin/main` 127.0.0.1 6379 `main_db_website`
`main_db_website_user` `main_db_website_user_password` 127.0.0.1
5432 R

- per $i = 1, \dots, N_c$, python3 environment/customer/src/main.py -
i i -r R 127.0.0.1 6379 main_db_website main_db_website_user
main_db_website_user_password 127.0.0.1 5432 env_db_website
env_db_website_user env_db_website_user_password 127.0.0.1 5432
- per $i = 1, \dots, N_p$, python3 environment/producer/src/main.py -
i i -r R 127.0.0.1 6379 main_db_website main_db_website_user
main_db_website_user_password 127.0.0.1 5432 env_db_website
env_db_website_user env_db_website_user_password 127.0.0.1 5432
- per $i = 1, \dots, N_p$, python3 environment/producer_phys/src/main.py -
i i -r R 127.0.0.1 6379 env_db_website env_db_website_user
env_db_website_user_password 127.0.0.1 5432
- per $i = 1, \dots, N_s$, python3 environment/shipper/src/main.py -
i i -r R 127.0.0.1 6379 env_db_website env_db_website_user
env_db_website_user_password 127.0.0.1 5432

```

1 function ProducerAPI1()
2   while True do
3     AnswerItemAvailReq();
4     SendProd();
  //
  // In parallel
  //
5 function ProducerAPI2( $t_s$ )
6   while True do
7     CollectFromProd();
8     MonSleep( $t_s$ );

```

Algorithm 11: Sistema: gestione produttori 1 e 2

```

1 function AnswerItemAvailReq()
2   if  $CP \neq \emptyset$  then
3      $i \leftarrow$  (blocking) read from CP;
4      $P \leftarrow$  read producers for item  $i$  from ITEM;
5      $m \leftarrow 0$ ;
6     foreach  $p \in P$  do
7       send  $i$  to PII $p$ ;
8        $m \leftarrow m +$  BlockingRead(POI $p$ );
9     send  $m$  to PC;

```

Algorithm 12: Sistema: gestione produttori, funzione ausiliaria

```

1 function SendProd()
2    $O \leftarrow$  read orders from ORDER s.t. processed is false;
3   foreach  $(o, c, d_1, d_2, i, n, \text{false}) \in O$  do
4     UPDATE ORDER set processed to true where id =  $o$ ;
5      $P \leftarrow$  read producers for item  $i$  from ITEM;
6      $m \leftarrow 0$ ;
7     foreach  $p \in P$  do
8       send  $(i, n - m)$  to  $\text{PII}p$ ;
9        $m' \leftarrow \text{BlockingRead}(\text{POI}p)$ ;
10      if  $m' > 0$  then
11        INSERT INTO STORAGE  $(o, m', p, \text{false})$ ;
12         $m \leftarrow m + m'$ ;
13      if  $m \geq n$  then break;
14      if  $n > m$  then
15         $(a, b) \leftarrow (\frac{n-m}{|P|}, (n-m) \bmod |P|)$ ;
16        if  $a \neq 0$  then
17          foreach  $\ell \in P$  do
18            send  $(i, o, a)$  to  $\text{PIP}\ell$ ;
19        foreach  $\ell \in \{1, \dots, b\}$  do
20          send  $(i, o, 1)$  to  $\text{PIP}\ell$ ;

```

Algorithm 13: Sistema: gestione produttori, funzione ausiliaria

```

1 function CollectFromProd()
2   // If a producer produced something as an answer to a processed
   // order, record it on the DB
3    $P \leftarrow$  read all producers from ACTOR;
4   foreach  $\ell \in P$  do
5     if  $\text{POP}\ell \neq \emptyset$  then
6        $(o, n) \leftarrow$  (blocking) read from  $\text{POP}\ell$ ;
7       INSERT INTO STORAGE  $(i, o, n, \ell, \text{false})$ ;

```

Algorithm 14: Sistema: gestione produttori, funzione ausiliaria

```

1 function Producer( $\ell, T, \tilde{p}, l, I$ )
2    $p, p' \leftarrow \tilde{p}, 0$ ;
3   while True do
4     // Manage incoming messages from Producer API, if any
5     if  $PII\ell \neq \emptyset$  then
6        $X \leftarrow$  (blocking) read from  $PII\ell$ ;
7       if  $X = (i, n)$  then
8         // check availability and book it
9         send  $\min\{p_i, n\}$  to  $POI\ell$ ;
10         $p_i \leftarrow p_i - \min\{p_i, n\}$ ;
11      else
12        //  $X = i$ : simply check availability for item  $i$ 
13        send  $p_i$  on  $POI\ell$ ;
14      if  $PIPl \neq \emptyset$  then
15         $(i, o, n) \leftarrow$  (blocking) read from  $PIPl$ ;
16        // produce  $n$  objects of item  $i$  because of an order  $o$ 
17        if  $n - p_i > 0$  then
18          send  $(i, n - p_i, o, T_i)$  to  $PPI\ell$ ;
19           $p_i \leftarrow 0$ ;
20        else
21           $p_i \leftarrow p_i - n$ ;
22          send  $(o, n)$  on  $POP\ell$ ;
23      foreach  $i \in I$  do
24        if  $l_i > p_i + p'_i$  then
25          send  $(i, l_i - p_i, \perp, T_i)$  to  $PPI\ell$ ;
26           $p'_i \leftarrow l_i - p_i$ ;
27      if  $PPO\ell \neq \emptyset$  then
28         $i, m, o \leftarrow$  (blocking) read from  $PPO\ell$ ;
29        if  $o \neq \perp$  then
30          send  $(o, m)$  on  $POP\ell$ ;
31        else
32           $p_i, p'_i \leftarrow m, 0$ ;

```

Algorithm 15: Ambiente: generazione richieste produttori

```

1 function ProducerPhys( $\ell$ )
2   while True do
3     if  $PPI\ell \neq \emptyset$  then
4        $i, m, o, T \leftarrow$  read from  $PPI\ell$ ;
5       if  $i, m, o, T = \perp, \perp, \perp, \perp$  then
6         // another ProducerPhys got this message
7         continue;
8       MonSleep( $\{\sum_{j=1}^m \text{random}(T)\}$ );
9       send  $i, m, o$  to  $PPO\ell$ ;

```

Algorithm 16: Ambiente: gestione richieste dei “veri” produttori

```

1 function ClientAPI( $t_s, T$ )
2   while True do
3      $C \leftarrow$  read all clients from ACTOR;
4     foreach  $\ell \in C$  do
5       if  $CO\ell \neq \emptyset$  then
6          $X \leftarrow$  (blocking) read from  $CO\ell$ ;
7         if  $X = (i, n, d)$  then
8           INSERT INTO ORDER ( $\ell, \text{now}(), \text{now}() + d, \perp, \perp, i, n, \text{false}$ ) ;
9           send OK to  $CI\ell$ ;
10        else
11          //  $X = i$ 
12          send  $i$  to CP;
13           $r \leftarrow$  BlockingRead(PC);
14          send  $r$  to  $CI\ell$ ;

```

Algorithm 17: Sistema: gestione dei clienti

```

1 function Client( $\ell, T, I, N, f, D$ )
2   while True do
3      $J \leftarrow \text{random}(2^I)$ ;
4     foreach  $j \in J$  do
5       Order( $\ell, j, \text{random}(N_j), f, \text{random}(D_j)$ );
6     MonSleep( $T$ );
//
// Function which decides what and how much to order
//
7 function Order( $\ell, j, n, f, d$ )
8   send  $j$  on  $\text{CO}\ell$ ;
9    $m \leftarrow \text{BlockingRead}(\text{CI}\ell)$ ;
10  if  $n \leq m \vee \text{random}(0, 1) > f$  then
11    send  $(j, n, d)$  on  $\text{CO}\ell$ ;
12     $\perp \leftarrow \text{BlockingRead}(\text{CI}\ell)$ ;

```

Algorithm 18: Ambiente: generazione richieste clienti

```

1 function ShipperAPI( $T$ )
2    $S \leftarrow \text{read all shippers from ACTOR}$ ;
3    $s \leftarrow 0^{|S|}$ ;
4   while True do
5      $O \leftarrow \text{read orders from ORDER s.t. processed is true}$ ;
6     foreach  $(o, c, d_1, d_2, i, n, \text{true}) \in O$  do
7        $S \leftarrow \text{read from STORAGE s.t. order\_id is } o \text{ and } o.\text{date\_shipped is}$ 
         null;
8        $m \leftarrow \text{sum all how\_many from } S$ ;
9       if  $m \geq n \wedge \{\ell \in S \mid s_\ell = 0\} \neq \emptyset$  then
10         $\ell \leftarrow \min\{\ell \in S \mid s_\ell = 0\}$ ;
11        send  $(i, o, n)$  to  $\text{SI}\ell$ ;
12         $s_\ell \leftarrow 1$ ;
13        BlockingRead( $\text{SO}\ell$ );
14        UPDATE ORDER set date\_shipped to now() on  $o$ ;
15    foreach  $\ell \in S$  s.t.  $s_\ell = 1$  do
16      if  $\text{SOA}\ell \neq \emptyset$  then
17         $X \leftarrow (\text{blocking}) \text{ read from SOA}\ell$ ;
18         $s_\ell \leftarrow 0$ ;
19        UPDATE ORDER set date\_arrived to now() on  $o$ ;

```

Algorithm 19: Sistema: gestione dei consegnatori

```

1 function Shipper( $\ell, T$ )
2   while True do
3     if  $SI\ell \neq \emptyset$  then
4        $X \leftarrow$  (blocking) read from  $SI\ell$ ;
5       send “sent” to  $SO\ell$ ;
6       MonSleep( $T_i$ );
7       send ( $o$ , “arrived”) to  $SOA\ell$ ;

```

Algorithm 20: Ambiente: generazione richieste dei consegnatori

```

1 function ExtMonitor( $t_s$ )
2   while True do
3     MonSleep( $t_s$ );
4      $O_1 \leftarrow$  read from ORDER s.t.  $\text{date\_arrived} > \text{date\_due}$ ;
5      $O_2 \leftarrow$  read from ORDER s.t.  $\text{now}() > \text{date\_due}$ ;
6     report as errors all not-yet-reported lines in  $O_1 \cup O_2$ ;

```

Algorithm 21: External Monitor

2.3 Cosa Fare

Occorre soddisfare le seguenti richieste:

1. Scegliere un opportuno model checker e modellare il sistema di cui sopra, sia nella prima che nella seconda versione. Verificare se esistono problemi di comunicazione (in sostanza, deadlock). Attenzione a non modellare troppo nel dettaglio.
 - se vengono scoperti degli errori, spiegare come potrebbero essere corretti
 - almeno un errore è presente nella gestione delle risposte su SI/SO della prima versione.
2. Testing: testare l’implementazione prototipale data insieme a queste specifiche. In tale implementazione, le comunicazioni sono realizzate tramite REDIS; ogni parte del sistema ha un’implementazione in C++ ed ogni parte dell’ambiente ne ha una in Python. Realizzare quindi:
 - unit testing: selezionare 6 unità (spiegare perché sono tali) e testarle con le tecniche che si ritengono opportune
 - integration testing: scegliere 3 insiemi di sottounità da testare insieme (possono includere anche unità non considerate nel punto precedente)
 - system testing: testare l’intero sistema.
3. Selezionare una unità e verificare la coverage relativa a MC/DC e loop boundary, instrumentando opportunamente il codice.

4. Scrivere la relazione chiarendo quali tecniche di testing e di model checking sono state usate e perché

3 Esercizio per Gruppi da 2 Studenti/Studentesse

Come sopra, con le seguenti facilitazioni:

- nella parte di model checking, modellare solo la prima o solo la seconda versione
- nella parte di unit testing, selezionare 4 unità
- nella parte di integration testing, selezionare 2 insiemi di unità

4 Esercizio per Singoli Studenti/Studentesse

Come sopra per i gruppi da 2, con le seguenti ulteriori facilitazioni:

- nella parte di model checking, tralasciare i produttori (da modellare quindi non-deterministicamente)
- nella parte di unit testing, selezionare 2 unità
- nella parte di integration testing, selezionare 1 insieme di unità