Software Testing and Validation
A.A. 2024/2025
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_2024_2025_matricole.zip` (se il progetto è fatto in gruppo, scrivere tutte le matricole separate dall'underscore _), che contenga un'unica directory `STV_2024_2025_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 da spiegare in `relazione.pdf`.

Il suddetto file `STV_2024_2025_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. Di tale sistema è disponibile sia un modello a pseudocodice (specificato nella Sezione 2.1) che un'implementazione, fornita separatemente nel file `prg.tgz`.

2.1 Modello del Sistema

È presente un DB con le seguenti tabelle:

- `actor(id, type, descr)`
- `item(id, descr, producer_ids, how_many, how_many_min, producing)`
- `order(id, customer_id, item_id, how_many, date_placed, date_processed, date_ready, date_shipped, date_arrived)`

Il sistema vero e proprio è costituito da 6 processi in esecuzione concorrente:

- uno per gestire le richieste dei clienti che arrivano sul canale `CO`; si attiva non appena c'è un messaggio da gestire e può o mandare informazioni su quali prodotti sono attualmente disponibili su canali dedicati `CI`c o inserire nel DB un ordine (Algorithm 3);
- uno per trovare sul DB gli ordini ancora da considerare; si attiva ogni t secondi e può o direttamente mettere un ordine in consegna (se c'è la disponibilità di magazzino) oppure chiedere al produttore p di produrre nuovi oggetti tramite il canale `PI`p (Algorithm 4);
- uno per trovare sul DB gli oggetti con disponibilità più bassa della minima consentita; si attiva ogni t secondi e può chiedere al produttore p di produrre nuovi oggetti tramite il canale `PI`p (Algorithm 6);
- uno per trovare sul DB gli ordini da consegnare; si attiva ogni t secondi e chiede al consegnatore s con meno carico di lavoro, tramite il canale `SI`s, di consegnare l'opportuno insieme di oggetti (Algorithm 9);
- uno per gestire le risposte dei produttori sul canale `PO`; si attiva non appena c'è un messaggio da gestire e aggiorna opportunamente le disponibilità di magazzino sul DB (Algorithm 7);
- uno per gestire le risposte dei consegnatori sul canale `SO`; si attiva non appena c'è un messaggio da gestire e aggiorna opportunamente gli ordini come consegnati sul DB (Algorithm 10);
- uno che monitora eventuali anormalità (Algorithm 12).

L'ambiente esterno è costituito da 3 entità:

- clienti: possono chiedere cosa c'è da comprare e decidere in modo casuale cosa prendere tra gli oggetti disponibili; chiedono sul canale comune `CO` e ricevono risposte su canali dedicati `CI`c (Algorithm 2);
- produttori: accettano richieste di produzione e comunicano l'avvenuta produzione; rispondono sul canale comune `PO` e ricevono richieste su canali dedicati `PI`p (Algorithm 8);
- consegnatori: accettano richieste di consegna e comunicano l'avvenuta consegna; rispondono sul canale comune `SO` e ricevono richieste su canali dedicati `SI`s (Algorithm 11).

Le 3 entità dell'ambiente vengono create e possibilmente terminate dal processo descritto in Algorithm 1.

```
1  function EnvGen(p, m, M, μ, σ, f_d, f_a)
2      InitTime();
3      n ← 0;
4      while True do
5          if p = customer ∧ a customer terminated then
6              n ← n − 1;
7          else if m < n ≤ M ∧ random(0, 1) > f_d then
8              let p be a random process of type t;
9              RedisCmd(send, Ttp, ⊥);
10             n ← n − 1;
11         if n < m ∨ (n < M ∧ random(0, 1) > f_a) then
12             create max{1, m − n} new processes of type p, let I be their pids;
13             n ← n + 1;
14         SleepWrap(μ, σ);
```

Algorithm 1: Environment: create all environment processes. Three instances of this process must be created with p in {customer, producer, shipper}

2.2 Esecuzione del Sistema

Per eseguire il sistema è necessario Linux (è possibile usare una macchina virtuale). I pacchetti che occorre avere installati sono: `python3, redis, redis-server, redis-tools, python3-redis, libhiredis-dev, libhiredis0.14, time, postgresql, postgresql-client, postgresql-client-common, postgresql-common, libpq-dev, libpq5, libpqxx-6.4, libpqxx-dev, python3-psycopg2, python3-more-itertools, python3-numpy, bc, psmisc, gawk, flex, bison, libreadline-dev, zlib1g-dev`.

```
1  function Customer(ℓ, μ_m, σ_m)
2  |  InitTime();
3  |  RedisCmd(send, CO, (Init, ℓ));
4  |  (I, d, h) ← RedisCmd(BlockingRecv, CIℓ);
5  |  J, S ← random(2^I), ∅;
6  |  foreach j ∈ J do
7  |  |  if h(j) > 0 then  S ← S ∪ {(j, random(1, h(j)), ℓ)} ;
8  |  if S ≠ ∅ then
9  |  |  SleepWrap(μ_m, σ_m);
10 |  |  RedisCmd(send, CO, S);
11 |  |  ⊥ ← RedisCmd(BlockingRecv, CIℓ);
12 |  Terminate();
```

Algorithm 2: Environment: single customer issuing requests

```
1  function CollectFromCustomer()
2  |  InitTime();
3  |  while True do
4  |  |  M ← RedisCmd(BlockingRecv, CO);
5  |  |  foreach X ∈ M do
6  |  |  |  if X = S then
7  |  |  |  |  S′ ← {(c, i, n, GetTime()) | (c, i, n) ∈ S};
8  |  |  |  |  DBCmd(INSERT order S′);
9  |  |  |  |  RedisCmd(send, CIc, OK);
10 |  |  |  else
   |  |  |  |  // here, X = (Init, c)
11 |  |  |  |  (I, d, h) ← DBCmd(SELECT id, descr, how_many FROM item);
12 |  |  |  |  RedisCmd(send, CIc, (I, d, h));
13 |  |  |  |  DBCmd(INSERT actor c);
```

Algorithm 3: System: Customer API

```
1  function ManageOrders(t)
2      InitTime();
3      while True do
4          R ← DBCmd(SELECT item_id, o.id, o.how_many, it.how_many
             FROM order o INNER JOIN item it ON item_id = it.id WHERE
             (o.date_processed IS NULL OR o.date_ready IS NULL) ORDER
             BY item_id, o.id);
5          if R ≠ ∅ then
6              Q ← {o | ∃(i, o, h, h′) ∈ R};
7              DBCmd(UPDATE order SET date_processed=GetTime()
                 WHERE id in Q AND date_processed IS NULL);
8              S, O ← ∅, ∅;
9              J ← {j | (j, o, h, h′) ∈ R ∧ h ≤ h′};
10             foreach j ∈ J do
11                 T ← {(j, o, h, h′) ∈ R ∧ h ≤ h′};
12                 foreach d ∈ {|T|, . . . , 0 do
13                     T′ ← first d elements of T;
14                     if h′ ≥ ∑_{(j,o,h,h′)∈T′} h then break;
15                 if T′ ≠ ∅ then
16                     S ← S ∪ (j, h′ − ∑_{(j,o,h,h′)∈T′} h);
17                     O ← O ∪ {o | (j, o, h, h′) ∈ T′};
18             if O ≠ ∅ then
19                 foreach (j, h) ∈ S do DBCmd(UPDATE item SET how_many
                     = h WHERE id = j) ;
20                 DBCmd(UPDATE order SET date_ready=GetTime() WHERE
                     id in O);
21                 Q ← DBCmd(SELECT item_id, o.how_many, it.how_many
                     FROM order o INNER JOIN item it ON it.id = o.item_id
                     WHERE o.id in Q \ O);
22                 S ←
                     {(i, n) | ∃(i, h, h′) ∈ Q ∧ n = ∑_{(i,h,h′)∈Q} h − vs.t.∃(i, h, v) ∈ Q};
23                 SendProd(S);
24         SleepWrap(t);
```

Algorithm 4: System: Producer API function 1

```
1  function SendProd(R)
2  |  S ← DBCmd(UPDATE item SET producing=true WHERE
   |    NOT(producing) AND id in R.id RETURNING id);
3  |  P ← DBCmd(SELECT (id, producer_ids) FROM item WHERE id in
   |    S);
4  |  Q ← ∪_{P | (i,P)∈P} P;
5  |  foreach (i, m) ∈ S do
6  |  |  P ← P s.t. (i, P) ∈ P;
7  |  |  (a, b) ← (m/|P|, m mod |P|, 0);
8  |  |  if a ≠ 0 then
9  |  |  |  foreach ℓ ∈ P do  M_ℓ ← M_ℓ ∪ {(i, a)} ;
10 |  |  foreach ℓ ∈ {1, ..., b} do  M_ℓ ← M_ℓ ∪ {(i, 1)} ;
11 |  foreach ℓ ∈ Q | M_ℓ ≠ ∅ do  RedisCmd(send, PIℓ, M_ℓ) ;
```

Algorithm 5: System: Producer API auxiliary function

```
1  function ManageMinStorage(t)
2  |  InitTime();
3  |  while True do
4  |  |  W ← DBCmd(SELECT id, how_many_min-how_many FROM
   |  |    item WHERE how_many<how_many_min);
5  |  |  P ← DBCmd(SELECT id FROM actor WHERE type=prod AND
   |  |    descr=term);
6  |  |  SendProd(W ∪ {(j, r) | ∃p ∈ P : (j, r) ∈PIp});
7  |  |  SleepWrap(t);
```

Algorithm 6: System: Producer API function 2

```
1  function CollectFromProd()
2      InitTime();
3      while True do
4          X ← RedisCmd(BlockingRecv, PO);
5          foreach X ∈ 𝒳 do
6              if X = (Init, ℓ) then
                   // only once per iteration
7                  I ← DBCmd(SELECT id FROM item);
8                  RedisCmd(send, PIℓ, I);
9                  DBCmd(INSERT actor ℓ);
10             else if X = (Term, p) then
11                 DBCmd(UPDATE actor SET descr=term WHERE id= p);
12             else if X = (i, n) then
13                 DBCmd(UPDATE item SET producing=false WHERE id= i);
14                 DBCmd(UPDATE item SET how_many=how_many+n
                     WHERE id= i);
15             else
                   // X is the subset of items produced by ℓ
16                 DBCmd(UPDATE actor SET producer_ids=producer_ids∪ℓ
                     WHERE id IN X);
```

Algorithm 7: System: Producer API function 3

```
1  function Producer(ℓ, μ_m, σ_m, μ_p, σ_p)
2      InitTime();
3      RedisCmd(send, PO, (Init, ℓ));
4      I ← RedisCmd(BlockingRecv, PIℓ);
5      RedisCmd(send, PO, random subset of I);
6      while True do
7          X ← RedisCmd(recv, TPℓ);
8          if X ≠⊥ then
9              RedisCmd(send, PO, (Term, ℓ));
10             Terminate();
11             exit;
12         X ← RedisCmd(recv, PIℓ);
13         foreach (i, m) ∈ X do
14             SleepWrap(μ_p, σ_p);
15             RedisCmd(send, PO, (i, m));
16         SleepWrap(μ_m, σ_m);
```

Algorithm 8: Environment: producer

```
1  function SendOrders(t)
2      InitTime();
3      while True do
4          W ← DBCmd(SELECT item_id, id, how_many FROM order
             WHERE date_ready IS NOT NULL AND date_shipped IS
             NULL);
5          T ← DBCmd(SELECT id FROM actor WHERE type=ship AND
             descr=term);
6          S, U ← ∅, ∪_{s∈T} SIs;
7          while |S| < |W| + |U| do
8              attach DBCmd(UPDATE actor SET descr=descr+1 WHERE
                 type=shipper AND descr IS MINIMUM) to S;
9          foreach j = 1, ..., |S| do
10             if j < |W| then ((o, i, n), s) ← W_j, S_j ;
11             else ((o, i, n), s) ← U_{j−|W|}, S_j ;
12             M_s ← M_s ∪ {(i, o, n)};
13         DBCmd(UPDATE order SET date_shipped=GetTime() WHERE id
             in W.id);
14         foreach s ∈ S do
15             RedisCmd(send, SIs, M_s);
16         SleepWrap(t);
```

Algorithm 9: System: shipper API function 1 when no work balance is needed

```
1  function CollectFromShip()
2      InitTime();
3      while True do
4          X ← RedisCmd(BlockingRecv, SO);
5          foreach X ∈ X do
6              if X = (Init, s) then
7                  DBCmd(INSERT actor (s, 0));
8              else if X = (Term, s) then
9                  DBCmd(UPDATE actor SET descr=term WHERE id= s);
10             else
11                 DBCmd(UPDATE actor SET descr=descr-1 WHERE id= s);
12                 DBCmd(UPDATE order SET date_arrived=GetTime() WHERE
                     id= o);
```

Algorithm 10: System: shipper API function 2

```
1  function Shipper(ℓ, μ_m, σ_m, μ_p, σ_p)
2  |  InitTime();
3  |  RedisCmd(send, SO, (Init, ℓ));
4  |  while True do
5  |  |  X ← RedisCmd(recv, TSℓ);
6  |  |  if X ≠⊥ then
7  |  |  |  RedisCmd(send, SO, (Term, ℓ));
8  |  |  |  Terminate();
9  |  |  |  exit;
10 |  |  X ← RedisCmd(recv, SIℓ);
11 |  |  foreach (i, o, n) ∈ X do
   |  |  |  // here, only o is needed, but in real life...
12 |  |  |  SleepWrap(μ_p, σ_p);
13 |  |  |  RedisCmd(send, SO, (o, ℓ));
14 |  |  SleepWrap(μ_m, σ_m);
```

Algorithm 11: Environment: shipper

```
1  function IntMonitor(θ, t)
2  |  InitTime();
3  |  while True do
4  |  |  O_1 ← DBCmd(SELECT id FROM order WHERE
   |  |    date_arrived-date_placed> θ);
5  |  |  O_2 ← DBCmd(SELECT id FROM order WHERE
   |  |    GetTime()-date_placed> θ;
6  |  |  report as errors all not-yet-reported lines in O_1 ∪ O_2;
7  |  |  SleepWrap(t);
```

Algorithm 12: Internal Monitor

Per l'esecuzione dell'intero sistema, è possibile usare lo script Bash `run.sh`; i suoi input sono elencati all'inizio del file stesso, con in più il file `init_tables.txt` che inizializza le tabelle del DB. Alla fine dello standard output di `run.sh` c'è un riga del tipo `kill -9 PID`, utilizzando la quale è possibile mettere fine all'esecuzione dell'intero sistema.

2.3 Cosa Fare

Occorre soddisfare le seguenti richieste:

1. Scegliere un opportuno model checker e modellare il sistema descritto in Section 2.1. Verificare se esistono problemi di comunicazione che portino a deadlock. Formulare una formula LTL o CTL da verificare. Attenzione a non modellare troppo nel dettaglio.

2. Testare l'implementazione prototipale data insieme a queste specifiche. In tale implementazione, le comunicazioni sono realizzate tramite REDIS, su canali il cui nome è sempre prefissato da `work:queue:` (quindi, pre esempio, il canale PI1 in realtà si chiama `work:queue:PI1`). Come DBMS, invece, viene usato PostgreSQL. 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 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