UN ASSETTO MOODLE PER L'ESAME ONLINE DI UN CORSO DI PROGRAMMAZIONE

Felice Cardone, Sergio Rabellino, Luca Roversi

Dipartimento di Informatica, Università di Torino
felice.cardone@unito.it, sergio.rabellino@unito.it, luca.roversi@unito.it

— FULL PAPER —

***ARGOMENTO**: Istruzione - Valutazione dell'apprendimento a distanza*

Abstract

La recente emergenza pandemica ha costretto i corsi universitari ad una repentina transizione alla modalità on-line. In questo contributo presentiamo la trasformazione on-line di un insegnamento di programmazione per il corso di laurea triennale in Informatica. Per l'erogazione dell'insegnamento, la nostra soluzione utilizza Moodle, come già accadeva quando l'insegnamento era erogato in presenza ma, nella nuova veste completamente on-line, trae vantaggio dall'integrazione del plugin CodeRunner per la valutazione delle prove di esame.

Keywords – Didattica on-line, Progetti di ricerca, Esami informatizzati.

1 IL CONTESTO DIDATTICO

Il corso di Programmazione 1, di cui descriviamo ora brevemente il contesto e le finalità didattiche, è collocato al primo semestre del primo anno del Corso di Laurea Triennale in Informatica presso il Dipartimento di Informatica dell'Università di Torino.

Si tratta del primo insegnamento di settore informatico, unico nel primo semestre, e si svolge in parallelo a corsi di Matematica Discreta e Logica e di Calcolo Matriciale e Ricerca Operativa. Questi ultimi insegnamenti riepilogano ed espandono le basi formali, parzialmente acquisite dalle matricole durante l'insegnamento superiore, con la significativa eccezione della logica che è sistematicamente trascurata dai programmi della scuola secondaria. Per quanto riguarda l'interazione di questi insegnamenti con quello della programmazione, durante il semestre gli studenti dovrebbero acquisire familiarità, in particolare, con le basi della combinatoria, la manipolazione dei connettivi logici classici e dei quantificatori, con il principio di induzione nelle sue varie forme, le principali operazioni su matrici e le relative tecniche di manipolazione.

Anche se una significativa percentuale di matricole proviene da studi tecnici ed è quindi stata già esposta abbondantemente alla programmazione, l'insegnamento di Programmazione 1, per una esplicita decisione in fase di progettazione del corso, non presuppone conoscenze specifiche relative a linguaggi di programmazione o a tecniche di programmazione. L'effetto di questa decisione è di rendere necessaria una rivisitazione dei contenuti di programmazione di base in modo che le prime settimane del corso siano utili sia a chi non ha mai programmato, sia alle matricole che invece hanno familiarità con linguaggi le cui tecniche di programmazione siano immediatamente trasferibili al linguaggio Java, che è il linguaggio di riferimento per il corso.

A questo proposito, occorre precisare che la convergenza su Java è stata accompagnata da molte riflessioni e dubbi in fase di progettazione del corso. Da un lato Java ha una struttura orientata agli oggetti che ne pervade ogni aspetto. In particolare, questa struttura è difficilmente separabile dalla parte in cui ci si limita ad utilizzare metodi statici (anche ricorsivi), strutture di controllo iterative e *array*. Questa difficoltà è apparente anche solo considerando la scarsità di libri di testo che seguono un percorso "dal basso" introducendo la struttura orientata ad oggetti soltanto in un secondo tempo (il testo scelto per l'insegnamento di Programmazione 1 è il manuale di Walter Savitch [2], che viene utilizzato anche nell'insegnamento di Programmazione 2, al secondo semestre, in cui vengono sviluppate le tecniche di programmazione orientata agli oggetti).

D'altro lato, Java ha un supporto universale in termini di sistemi operativi e modelli di computer sui quali può essere gratuitamente installato, favorendo la sperimentazione individuale da parte degli studenti.

2 CONTENUTI

In un arco di tempo che ormai si approssima al decennio, il corso ha avuto relativamente poche modifiche relativamente ai contenuti, tranne forse per la prima parte, discorsiva e generale, sul pensiero computazionale e le tecniche attraverso cui esso si manifesta, che è stata progressivamente ridotta a pochi esempi iniziali di attività di programmazione tratti dalla vita reale, spesso espressi in forma ludica o come rompicapo. Questi esempi permettono di affrontare, in una fase precoce, problematiche relative alla complessità dei programmi, alla nozione di specifica e di correttezza (parziale e totale) di un programma (sebbene espresso ancora informalmente) rispetto ad una specifica. Occorre notare che la durata del corso è stata ridotta da 60 a 48 ore a seguito di una ristrutturazione del Corso di Laurea Triennale avvenuta pochi anni or sono; questo ha reso necessario economizzare tempo sugli aspetti più generali e non direttamente pertinenti agli argomenti di esame.

L'esposizione del linguaggio Java che segue immediatamente questa parte introduttiva segue le tappe tradizionali: struttura della memoria e assegnamento; composizione in sequenza di istruzioni; iterazione, con una importante scelta metodologica che consiste nell'utilizzo di dimostrazione di correttezza di base, in particolare gli invarianti di ciclo, esemplificati attraverso semplici programmi che operano — in questa fase del corso — su tipi primitivi.

È a questo punto che diventa importante la sincronizzazione con il corso di Matematica Discreta e Logica, dove si assume che sia già stato introdotto il Principio di Induzione, di cui la dimostrazione di invarianza di asserzioni è un'applicazione privilegiata, e fornisce esercizi utili anche per il programma di quest'ultimo insegnamento.

Segue l'introduzione dei metodi, della ricorsione con altre occasioni di applicazione del Principio di Induzione alla correttezza di metodi numerici con parametri di tipo intero e una condizione di ingresso che li assume ≥ 0.

Una terza fase del corso riguarda la programmazione su array, ai quali vengono estese le tecniche di programmazione iterative e ricorsive, e degli *array* multipli, che forniscono l'occasione di interazioni con il corso di Calcolo Matriciale.

Al corso di Programmazione 1 è affiancato un Laboratorio di Programmazione, della durata di 30 ore autovalutazioni ed esame finale

3 GLI ESAMI IN PRESENZA

Poste queste premesse didattiche, quali sono i contenuti di cui verificare l'apprendimento? Nel syllabus del corso troviamo i seguenti risultati dell'apprendimento, espressi in termini di capacità:

- formalizzare la soluzione a problemi computazionali di base per mezzo di costrutti linguistici iterativi e ricorsivi;
- esprimere questa formalizzazione nel linguaggio Java;
- valutare correttezza parziale e terminazione di procedure per mezzo di semplici dimostrazioni formali;
- valutare l'efficienza di una procedura;
- rappresentare istantanee dello stato in cui si trovi la memoria di un calcolatore che interpreti programmi ragionevolmente complessi nel linguaggio Java.

L'esame finale prevede le seguenti quattro grandi tipologie di esercizi:

- *"Programmazione iterativa"*. Richiede la scrittura di un programma iterativo, tipicamente centrato su varianti di algoritmi di riferimento in grado di verificare una proprietà di array o matrici, formulata in termini di alternanza di quantificatori logici. Le alternanze sono riassunte come esiste/esiste, per-ogni/esiste esiste/per-ogni e per-ogni/per-ogni. La difficoltà principale è la corretta gestione delle variabili booleane che determinano l'alternarsi del soddisfacimento

delle proprietà universali o esistenziali che il programma proposto come soluzione deve soddisfare;

- *"Programmazione ricorsiva".* Richiede la scrittura di un programma ricorsivo su array. La ricorsione si realizza attraverso l'uso di parametri numerici che individuano porzioni di array su cui risolvere istanze più semplici del problema iniziale da risolvere. La difficoltà principale è la corretta gestione dei parametri numerici che guidano la ricorsione, compreso il loro valore iniziale, determinato da un opportuno metodo "involucro", il quale richiama quello principale che costituisce la soluzione all'esercizio;
- *"Correttezza (parziale)"*. Lo scopo è applicare il "Principio di Induzione" al predicato che descrive una proprietà data di un algoritmo noto, con lo scopo di provare che l'algoritmo produce sempre il risultato atteso. Tenuto conto della intrinseca difficoltà formale della prova, l'applicazione del Principio di Induzione è guidata, pur mantenendo l'obiettivo di richiedere la corretta identificazione del caso base, del passo induttivo, e delle relative giustificazioni;
- *"Modello di gestione della memoria"*. Rispettando convenzioni notazionali opportune, occorre saper ripercorrere passo passo quel che una *Java Virtual Machine* (JVM) esegue durante l'interpretazione di un dato programma Java, composto da uno o più metodi, iterativi o ricorsivi. Tipicamente, lo scopo è saper descrivere l'organizzazione delle zone di memoria *frame stack* e *heap* in un preciso istante dell'interpretazione, specificato dal testo dell'esercizio; l'attenzione va posta sullo stato dei riferimenti alle strutture *array*, sui punti di rientro dei metodi e sui valori restituiti in seguito alla disallocazione dei *frame* dei metodi dal *frame stack*.

Prima della riorganizzazione degli esami in modalità on-line, un appello si svolgeva in presenza in laboratori informatizzati debitamente attrezzati e configurati. Sulle macchine erano disponibili un SDK Java, cioè uno *Software Developer Kit* con compilatore Java e JVM, accompagnato da un elaboratore testi minimale, adatto alla programmazione in modo che esaminande ed esaminandi potessero fornire soluzioni agli esercizi delle prime due tipologie. Riguardo alle due seconde tipologie, invece, era prevista la consegna di soluzioni scritte a mano.

4 GLI ESAMI ON-LINE

Nella recente emergenza sanitaria la prova di esame è stata convertita in una prova svolta in remoto direttamente su una istanza della piattaforma Moodle. Gli esercizi delle due ultime tipologie non hanno richiesto una significativa rimodulazione dei contenuti nella trasformazione in domanda a risposta multipla.

L'esercizio sulla gestione della memoria ad un certo istante nell'esecuzione di un programma, da disegno su carta (peraltro spesso di difficile interpretazione), ha potuto facilmente essere riconvertito in una domanda a risposte multiple rivolte ai dettagli della gestione dei riferimenti, dei punti di rientro e dei valori comunicati tra i vari *record* di attivazione sul *frame stack*.

Questi dettagli non sono normalmente rappresentati esplicitamente dai visualizzatori disponibili, quindi l'esito di questo esercizio è sufficientemente attendibile per accertare l'abilità, da parte dello studente, di manipolare *frame stack* e *heap*, simulando astrattamente quello che di fatto è il comportamento della Java Virtual Machine. Come esempio, assumendo che il programma (secondo la terminologia Java: classe) da interpretare sia:

```
class E40 {
  static int m(int[] a, int i){
    if (i < a.length)
      return a[i] + m(a,i+1);
    else
      return 0;
  }

  public static void main (String[] args){
    int[][] a = {{3,6},{1,2,3}};
    int x = m(a[0],0); // (*)
    int y = m(a[1],0); // (**)
  }
}
```

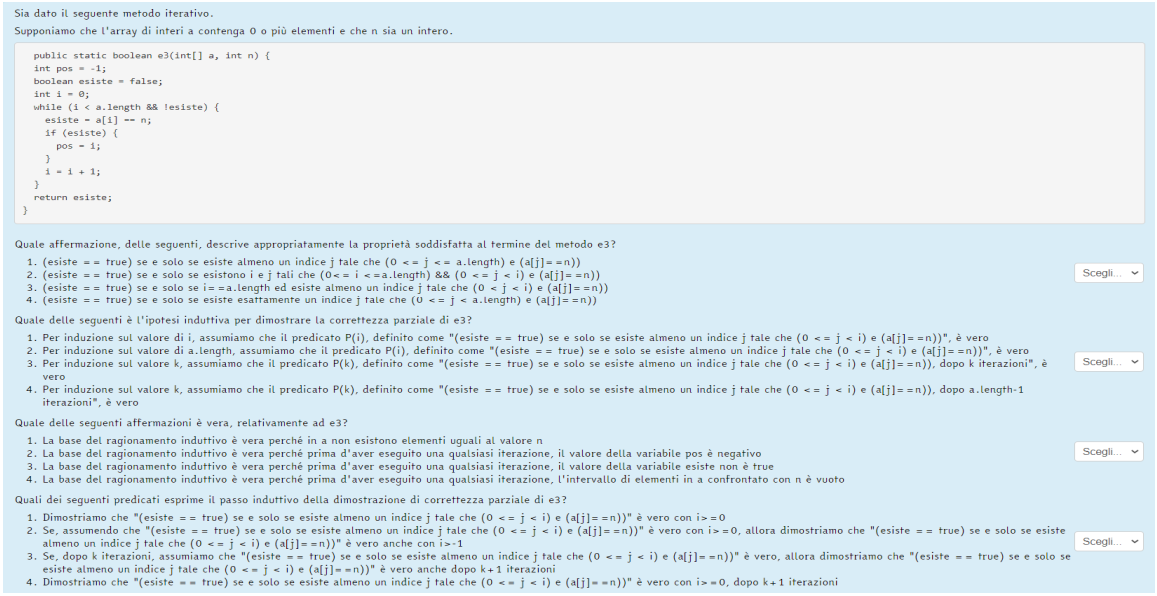
Fig 1: Esempio di programma Java da interpretare

la soluzione on-line si presta ad una valutazione completamente automatica. Essa consiste nel rispondere a domande per cui sono offerte risposte multiple. Le domande vertono su aspetti quantitativi e qualitativi dell'istantanea che si vuole descrivere. La natura non ambigua delle convenzioni illustrate a lezione, con cui rappresentare la configurazione della memoria, assicurano l'univocità della risposta. A titolo di esempio, in relazione al programma Java riportato, alla domanda "Quanti puntatori ad array esistono nei *frame* allocati sul *frame stack* quando nel *frame* del metodo **m** la variabile locale **i** vale 2?" si può rispondere scegliendo "Essi sono 2"

Per quanto riguarda l'esercizio sulla dimostrazione di correttezza, svolto su carta richiedeva un lavoro di "decifratura" della dimostrazione fornita come risposta, per diversi motivi:

- la difficoltà intrinseca dell'esercizio che, per sua natura, richiede una discreta capacità formale, sulla quale, tuttavia, il criterio di correzione è sempre stato ragionevolmente tollerante;
- la presenza di studenti non madrelingua;
- la necessità di dover leggere dipendenze causali e conclusioni, scritte con calligrafie non facilmente decifrabili.

Il tutto, unito alla possibile alta quantità di prove da valutare, rendeva il processo di correzione gravoso ed indubbiamente prono ad errori di valutazione involontari. La soluzione è stata fornita dall'utilizzo di menu a tendina che offrono una scelta tra diverse opzioni di risposta, una sola delle quali è corretta. Un esempio di testo in versione on-line è:


Sia dato il seguente metodo iterativo.

Supponiamo che l'array di interi a contenga 0 o più elementi e che n sia un intero.

```
public static boolean e3(int[] a, int n) {
  int pos = -1;
  boolean esiste = false;
  int i = 0;
  while (i < a.length && !esiste) {
    esiste = a[i] == n;
    if (esiste) {
      pos = i;
    }
    i = i + 1;
  }
  return esiste;
}
```

Quale affermazione, delle seguenti, descrive appropriatamente la proprietà soddisfatta al termine del metodo e3?

1. (esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j <= a.length) e (a[j]==n))
2. (esiste == true) se e solo se esistono i e j tali che (0<= i <=a.length) && (0 <= j < i) e (a[j]==n))
3. (esiste == true) se e solo se i==a.length ed esiste almeno un indice j tale che (0 < j < i) e (a[j]==n))
4. (esiste == true) se e solo se esiste esattamente un indice j tale che (0 <= j < a.length) e (a[j]==n))

Scegli...

Quale delle seguenti è l'ipotesi induttiva per dimostrare la correttezza parziale di e3?

1. Per induzione sul valore di i, assumiamo che il predicato P(i), definito come "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))", è vero
2. Per induzione sul valore di a.length, assumiamo che il predicato P(i), definito come "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))", è vero
3. Per induzione sul valore k, assumiamo che il predicato P(k), definito come "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n)), dopo k iterazioni", è vero
4. Per induzione sul valore k, assumiamo che il predicato P(k), definito come "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n)), dopo a.length-1 iterazioni", è vero

Scegli...

Quale delle seguenti affermazioni è vera, relativamente ad e3?

1. La base del ragionamento induttivo è vera perché in a non esistono elementi uguali al valore n
2. La base del ragionamento induttivo è vera perché prima d'aver eseguito una qualsiasi iterazione, il valore della variabile pos è negativo
3. La base del ragionamento induttivo è vera perché prima d'aver eseguito una qualsiasi iterazione, il valore della variabile esiste non è true
4. La base del ragionamento induttivo è vera perché prima d'aver eseguito una qualsiasi iterazione, l'intervallo di elementi in a confrontato con n è vuoto

Scegli...

Quali dei seguenti predicati esprime il passo induttivo della dimostrazione di correttezza parziale di e3?

1. Dimostriamo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero con i>=0
2. Se, assumendo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero con i>=0, allora dimostriamo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero anche con i>-1
3. Se, dopo k iterazioni, assumiamo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero, allora dimostriamo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero anche dopo k+1 iterazioni
4. Dimostriamo che "(esiste == true) se e solo se esiste almeno un indice j tale che (0 <= j < i) e (a[j]==n))" è vero con i>=0, dopo k+1 iterazioni

Scegli...

Fig. 2: esempio di quiz per le dimostrazioni di correttezza.

La formulazione delle domande e le alternative proposte sono pensate per educare gli studenti di Programmazione 1:

- a leggere un semplice programma, in questo caso iterativo, ma che può essere anche ricorsivo, per riassumerne funzione calcolata o risultato ottenuto;
- a saper individuare passi base ed induttivo della dimostrazione di correttezza (parziale), in funzione del codice dato (ma sviluppate in dettaglio durante le lezioni).

Il messaggio della nuova formulazione è che occorre saper seguire, ma non necessariamente saper sviluppare in piena autonomia, con l'alto livello di dettaglio che sarebbe necessario, le innumerevoli dimostrazioni di correttezza parziale viste a lezione.

Questo non ha impedito che un non numeroso, ma tutt'altro che trascurabile, drappello di studenti al termine del corso fosse in grado di sviluppare le dimostrazioni padroneggiando un livello formale soddisfacente.

Rimane la descrizione delle tipologie di esercizio *"Programmazione iterativa"* e *"Programmazione ricorsiva"*. Ne illustreremo una; l'altra sfrutta i medesimi criteri progettuali di correzione e utilizzo dello strumento on-line di cui parleremo nella sezione successiva.

Partiamo col ricordare il loro svolgimento in presenza, usando il seguente testo di esercizio:

Scrivere un metodo fMag con le seguenti caratteristiche:

- fMag ha un primo parametro formale a di tipo array di interi ed un secondo parametro formale l di tipo intero;
- fMag restituisce un array che contiene tutti e soli gli interi inizialmente in a e strettamente maggiori del valore in l, rispettando l'ordine originale degli elementi;
- fMag *non* puo' contare il numero di elementi in a che soddisfano la proprieta' indicata per dimensionare opportunamente l'array da produrre come risultato;
- fMag è wrapper di un solo metodo ricorsivo co-variante che risolve effettivamente il problema.
- Per ipotesi, l'array a passato al metodo fD come parametro attuale non puo' essere null.

Per facilitare il lavoro e aiutare nella comprensione del testo, la consegna è sempre corredata con esempi e controesempi come:

```
public static void main(String[] args) {
  System.out.println(uguali(new int[] {}      , fMag(new int[] {}       , 0)));
  System.out.println(uguali(new int[] {}      , fMag(new int[] {0}      , 0)));
  System.out.println(uguali(new int[] {1}     , fMag(new int[] {1}      , 0)));
  System.out.println(uguali(new int[] {1}     , fMag(new int[] {0,1}    , 0)));
  System.out.println(uguali(new int[] {1}     , fMag(new int[] {1,0}    , 0)));
  System.out.println(uguali(new int[] {}      , fMag(new int[] {0,0,0}  , 0)));
  System.out.println(uguali(new int[] {1}     , fMag(new int[] {0,1,0}  , 0)));
  System.out.println(uguali(new int[] {1,1,1}, fMag(new int[] {1,1,1}  , 0)));
  System.out.println(uguali(new int[] {1,2}   , fMag(new int[] {0,1,0,2}, 0)));
  System.out.println(uguali(new int[] {1,1}   , fMag(new int[] {1,0,1,0}, 0)));
}
```

Fig. 3 Esempi e controesempi di codice Java

in cui `uguali` è il nome di un programma disponibile per verificare l'uguaglianza tra array. Esempi e controesempi permettono la così detta fase di *test*: se su di essi il programma che verrà proposto come soluzione si comporta secondo le aspettative, è ragionevole pensare che esso sia corretto, ovvero che funzioni come atteso in ogni caso, anche se l'insieme di esempi e controesempi, per loro natura, non può assicurarlo.

Sempre in presenza, dopo il *test*, soprattutto se ogni caso forniva la risposta attesa, il file con il programma soluzione veniva consegnato elettronicamente, memorizzandolo in zone disco opportunamente configurate per un accesso controllato al fine di garantire sicurezza ed affidabilità necessarie.

Riguardo alla correzione, nel tempo, anche grazie alle competenze relative alla programmazione, i vari docenti coinvolti in Programmazione 1 hanno instaurato procedure semi automatizzate, per accelerarla, senza abbassare la qualità. I passi fondamentali della correzione degli esercizi "*Programmazione*", relativi ad esami in presenza, quindi, erano:

- **Fase 1**. assegnare un primo giudizio di qualità della soluzione fornita, sommando la votazione di ogni test, tra quelli noti, passati con successo;
- **Fase 2**. leggere il codice per fornire un giudizio oggettivo sulla possibilità che esso implementi effettivamente un algoritmo corretto, in accordo con la consegna, eventualmente, confrontandolo con una soluzione ufficiale, prodotta dal gruppo di docenti.

La differenza con lo svolgimento sia dell'esame, sia della correzione, tra modalità on-line e in presenza è ridotta al minimo, come confermato da studenti che hanno partecipato ad appelli in entrambe le modalità, ma con il significativo elemento di sgravio delle attività di correzione.

5 IL PLUGIN CODERUNNER

Grazie alla disponibilità del plugin CodeRunner [1] è stato possibile sperimentare con successo modalità di verifica che consentono agli esaminandi di scrivere e verificare nell'immediato il codice prodotto senza mai uscire dall'ambiente di esame.

CodeRunner è un plugin per Moodle che aggiunge un tipo di domanda «*adaptive*» la quale richiede a studentesse e studenti di rispondere, scrivendo un programma ed eseguendolo.

La valutazione della domanda può essere automatizzata sulla base del risultato dell'esecuzione. Dal lato studente, esso fornisce un ambiente di lavoro paragonabile a quello nei laboratori e disponibile agli esami in presenza:

- elaboratore testi adatto alla programmazione;
- compilatore Java e JVM;
- un meccanismo interattivo "*Testing*".

All'interno di Moodle, CodeRunner offre, del tutto integrato, il meccanismo identificato in precedenza come Fase 1. Ovviamente, la Fase 2 rimane, ma il docente è completamente sgravato dalla fase di preparazione del meccanismo semiautomatico di valutazione quantitativa dei *test* passati con ragguardevole risparmio di tempo.

Il *Plugin* si avvale di un server dedicato in cui viene realizzato in tempo reale un ambiente protetto e sicuro (chiamato *sandbox*) entro cui ciascuna studentessa e ciascuno studente esegue – sempre attraverso Moodle – il proprio codice. Con opportune configurazioni, è possibile aggregare diversi *server* in modalità *cluster* (*round-robin*) al fine di supportare numeri significativi di domande/esami contemporanei.

Nella nostra configurazione, sono utilizzati 4 server che nel contesto di CodeRunner sono identificati come Jobe Server, che consentono l'erogazione di remote-sandbox a supporto di CodeRunner; nella nostra esperienza sono stati erogati fino a 400 esami in contemporanea in modo efficace.

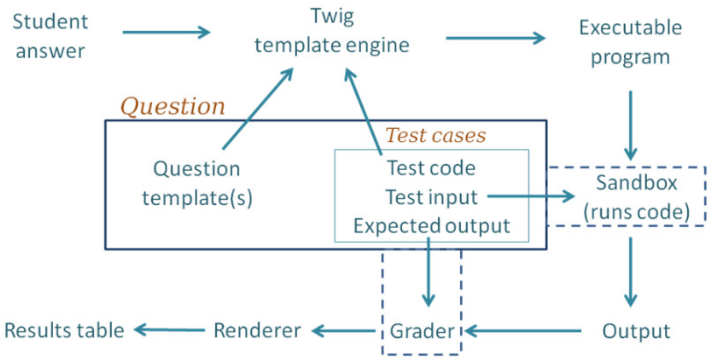


Figura 1 – Architettura logica di CodeRunner

Nella nostra architettura, abbiamo utilizzato 3 server per l'erogazione di sandbox Jobe e 1 server che realizza il meccanismo di round-robin tramite il software HAProxy [5] la cui parte saliente di configurazione è la seguente:

```
#---------------------------------
# frontend JobeServers
#---------------------------------
frontend jobecluster
    bind 130.192.157.38:80
    mode http
    option tcplog
    default_backend jobecluster
```

```
#--------------------------------------
# backend Jobe Servers
#--------------------------------------
backend jobecluster
    balance hdr(X-CodeRunner-Job-Id)
    mode http
    server node2 192.168.2.2:4000 check
    server node3 192.168.3.3:4000 check
    server node4 192.168.4.4:4000 check
```

In cui è possibile individuare come l'interfaccia pubblica del nodo proxy esporta una connessione http standard, veicolando successivamente le connessioni ai singoli server, utilizzando come modello di balancing un round-robin vincolato all'id del job di CodeRunner; questo è indispensabile per evitare che le attività di un esame in corso vengano sparpagliate in modo casuale sui vari server del cluster.

Ogni nodo è poi connesso su una rete privata interna al cluster per consentire di gestire con un singolo firewall sul proxy le connessioni in uscita dalle sandbox. Questo è un passo che abbiamo ritenuto fondamentale, perché all'interno della sandbox uno studente può effettuare diverse attività potenzialmente pericolose, come ad esempio l'apertura di una shell tramite cui effettuare anche connessioni ad altri server su internet. Pertanto, per eliminare questo rischio, l'accesso a internet dei nodi e conseguentemente delle sandbox, viene bloccata sul server che controlla il cluster.

Abbiamo così la sicurezza che le attività realizzate tramite CodeRunner siano completamente isolate sia tra di loro (strumento sandbox) che con l'esterno (strumento firewall) pur mantenendo la finalità per cui sono state installate, ovvero l'erogazione di esami basati sulla scrittura di programmi e la loro esecuzione.

L'installazione dei nodi Jobe risulta particolarmente semplice in quanto basata su container docker mantenuti e aggiornati costantemente dai mantainer del plugin CodeRunner.

L'hardware utilizzato per erogare 400 esami in contemporanea comprende:

- 1 server 8 core Intel(R) Xeon(R) CPU E5410 @ 2.33GHz con 16GB di RAM, 256GB HDD, 4xNIC 1GB
- 3 server 8 core Intel(R) Xeon(R) CPU E5410 @ 2.33GHz con 16GB di RAM, 128GB HDD, 1xNIC 1GB

collegati in cluster con cavi diretti host-to-host; i server sono stati recuperati da altri servizi dismessi e, benchè hardware non recente, nella nostra esperienza diretta sono risultati più che sufficienti a garantire lo svolgimento delle prove senza alcun rallentamento o intoppo.

6 AUTOVALUTAZIONE

È stato naturale estendere l'impiego di tutti gli strumenti adottati per lo svolgimento degli esami on-line anche all'autovalutazione da parte di studentesse e studenti, in virtù dell'automazione della valutazione da essi offerti. Le attività di autovalutazione sono state rese disponibili nella seconda parte del corso, quando il materiale per confezionarle era ormai abbondante. L'autovalutazione è un'attività asincrona rispetto agli orari di svolgimento delle lezioni, da svolgersi individualmente o in gruppo (on-line).

7 CONCLUSIONI ED INSEGNAMENTI

Globalmente, l'adattamento alla modalità d'esame on-line descritta sembra essersi svolto senza eccessive difficoltà da parte degli studenti, anche grazie all'offerta di facsimili di testi d'esame disponibili in tempo ragionevole prima della sessione estiva, la prima, come ricordato, svoltasi a distanza.

La progettazione dell'esame on-line ed il suo raffinamento sono stati molto impegnativi, durati per non meno di un paio di mesi. La raccolta metodica dei testi d'esame degli anni accademici antecedenti il 19/20, con relative soluzioni, da cui sono stati ricavate ulteriori varianti, si è rivelata fondamentale per costituire la base dati con cui CodeRunner, pescando a caso un esercizio per ciascuna delle quattro tipologie, è in grado di fornire un compito che, vista la quantità di esercizi esistenti, può essere descritto come unico per ogni esaminando. Forse, anche per questa caratteristica di aleatorietà nella composizione dei testi d'esame, tutti i docenti non hanno avuto l'impressione di "copiature a tappeto", essendo i numeri dei promossi e dei bocciati essenzialmente in linea con quelli degli esami in presenza.

La raccomandazione, per chi voglia accingersi a prendere spunto per una gestione di esami on-line simile a quella descritta, è di dedicare una cura spasmodica alla redazione dei testi. In presenza, l'eliminazione di un'ambiguità o, peggio, la correzione di un errore nel testo di un esercizio proposto, o negli esempi o controesempi che costituiscono i *test* degli esercizi della tipologia "*Programmazione*", sono recuperabili, anche solo grazie ad un suggerimento a voce, od alla possibilità di decidere una variante, scritta alla lavagna. On-line la relazione diretta con gruppi di studentesse o studenti è praticamente impossibile, ed il rischio di mettere inutilmente, anche se oggettivamente involontariamente, in difficoltà una esaminanda o un esaminando è alto e poco tollerato.

Riferimenti bibliografici

[1] Moodle plugins directory: CodeRunner, https://moodle.org/plugins/qtype_coderunner.

[2] Walter Savitch, Programmazione di base e avanzata con JAVA, Pearson, 2014.

[3] Barana, A., Marchisio, M., & Sacchet, M. (2021). Interactive Feedback for Learning Mathematics in a Digital Learning Environment. *Education Sciences*, *11*(6), 279. https://doi.org/10.3390/educsci11060279

[4] Barana, A., Fissore, C., & Marchisio, M. (2020). From Standardized Assessment to Automatic Formative Assessment for Adaptive Teaching: Proceedings of the 12th International Conference on Computer Supported Education, 285–296. https://doi.org/10.5220/0009577302850296

[5] HAProxy The Reliable, High Performance TCP/HTTP Load Balancer, https://www.haproxy.org