Progetto "Prodigit"

Software Engineering
Module: Model Based Software Engineering

William De Vena

A.A. 2020-2021

1 Descrizione Sistema

Il Progetto consiste nella modellazione ad alto livello di un sistema di prenotazione di aule. Il Sistema si inserisce nel contesto universitario durante la pandemia Covid-19 per permettere agli studenti di prenotare un posto in aula per le lezioni in presenza, in quanto la capienza di quest' ultima è limitata dalle regole di distanziamento sociale e in quanto la presenza degli studenti deve essere tracciabile dalle autorità. Le componenti principali del sistema sono:

- **Studente**: può *prenotare* un posto in aula oppure *cancellare* una prenotazione fatta in precedenza.
- **Aula**: ha una capienza massima, al raggiungimento della quale le prenotazioni non vengono più permesse (ovviamente le prenotazioni vengono riaperte quando uno studente cancella una prenotazione).
- **Gomp**: un sistema esterno attraverso il quale *Prodigit* ottiene diverse informazioni fondamentali sulle aule (agibilità, capienza massima, ...) senza le quali il nostro sistema non può operare.
- **Prodigit**: il sistema attraverso il quale gli studenti prenotano un posto in aula o cancellano una prenotazione. Inoltre in base alle informazioni,che ottiene attraverso il Gomp, e al numero di prenotazioni apre, chiude o azzera le prenotazioni.
- **Monitor**: verifica il corretto funzionamento del sistema. In particolare controlla che i *requisiti funzionali e non funzionali* siano rispettati.

Infine il progetto contiene due categorie di programmi Python:

- *Verify*: servono a testare il sistema eseguendolo un numero elevato di volte, cambiando randomicamente ad ogni esecuzione i parametri randomici del sistema e controllando i valori dei *Monitor*.
- *Synth*: servono a testare il sistema eseguendolo un numero elevato di volte, incrementando ad ogni esecuzione i valori di parametri non randomici del sistema che vogliamo massimizzare e controllando i valori del *Monitor*.

2 Scenari operativi

I possibili scenari operativi sono i seguenti:

- **Aula**: un aula può essere *Agibile* o *Inagibile*, nel primo caso le prenotazioni sono aperte mentre nel secondo vengono chiuse e le prenotazioni effettuate fino a quel momento vengono cancellate. In Modelica questi due stati sono rappresentati dal valore della variabile *state* della classe *Aula*, che può assumere due valori: 1 (*Agibile*) e 2 (*Inagibile*). In particolare, per generare randomicamente i due valori viene usata un *Catena di Markov* implementata con una matrice di dimensione N=2.

- **Studente**: l' input dello studente ha tre possibili valori: -1 che corrisponde alla cancellazione di una prenotazione, 0 che corrisponde al non uso del sistema e +1 che corrisponde alla prenotazione di un posto. In Modelica questo input è rappresentato dalla variabile x della classe *Student*, che può assumere i tre valori sopra elencati. Per generare randomicamente il valore della variabile viene usato il generatore di numeri casuali *Xorshift1024*.

- **Gomp**: ha due stati possibili: *Up* o *Down*, nel primo caso il sistema è operativo e di conseguenza Prodigit può ottenere attraverso esso le informazioni necessarie, nel secondo caso invece non è operativo e Prodigit non riesce ad ottenere le informazioni necessarie e di conseguenza diventa non operativo anche esso. In Modelica questi due stati sono rappresentati dal valore della variabile *state* della classe *Gomp*, che può assumere due valori: 1 (*Up*) e 2 *(Down)*. Anche qui viene usata una *Catena di Markov* simile a quella usata per la classe *Aula*.

- **Prodigit**: come menzionato sopra anche esso può essere *Up* o *Down* in base allo stato del Gomp. Ovviamente quando non è operativo (Down) non può fornire nè i servizi di prenotazione nè di cancellazione. In Modelica questi due stati sono rappresentati dal valore della variabile *state* delle classi *Controller1* e *Controller2*.

Il modello sviluppato può essere considerato completo in quanto tutte le entità che interagiscono con il sistema (*Gomp, Aula e Studente*) e il sistema stesso sono stati modellati rappresentando tutti i loro possibili input/output e stati. Inoltre si può osservarne la correttezza usando i *Monitor* ed effettuando una serie di test variando i parametri del sistema. In particolare, i *Monitor* controllano durante tutta l' esecuzione del sistema che esso si comporti nel modo corretto, ovvero che rispetti i requisiti funzionali e non funzionali. Questa correttezza in Modelica viene rappresentata dal valore della variabile booleana y delle classi *Monitor1 e Monitor2*, che assumono il valore *True* quando un requisito non viene rispettato.

3 Architettura del Sistema

Di seguito vengono elencate e descritte le componenti del sistema, come interagiscono tra di loro e i loro input e output:

- **Aula**: viene modellata dalla classe *Aula* nel file *aula.mo* e rappresenta l' oggeto aula. Il suo scopo è quello di variare randomicamente il suo stato (viene usata una *Catena di Markov*) e tenere conto del numero dei posti prenotati. Questi valori, rappresentati rispettivamente dalle variabili *aula.state* e *aula.num_pre*, vengono letti, il primo da *Gomp* e il secondo da *Controller2*. Il numero di posti prenotati varia in base all' input del *Controller2* (*ctr2.prenotazione*) che apre o chiude le prenotazioni. Inoltre anche il *Controller1* gli fornisce due input (*ctr1.u1_1 e ctr1.u1_2*) in base ai quali le prenotazioni vengono chiuse (se Prodigit entra nello stato *Down* e quindi *ctr1.u1_1=false*) oppure azzerate (se l' aula diventa *Inagibile* e quindi *ctr1.u1_2=false*).

- **Gomp**: viene modellato dalla classe *Gomp* nel file *gomp_mc.mo* e rappresenta il sistema esterno Gomp. Il suo scopo è quello di raccogliere e fornire a Prodigit (*Controller1*) le informazioni di cui necessita: la capienza massima dell' aula rappresentata dalla variabile *gomp.max* e lo stato dell' aula (che prende in input dalla classe *Aula*) rappresentato dalla variabile *gomp.agibilita*. Inoltre gli fornisce anche il suo stato, rappresentato dalla variabile *gomp.state*, che viene generato randomicamente attraverso una *Catena di Markov*.

- **Studente**: viene modellato dalla classe *Student* nel file *student.mo* e rappresenta l' utente del sistema. Il suo scopo è quello di generare randomicamente (viene usato il generatore *Xorshift1024*) l' input di Prodigit, che rappresenta il tentativo di prenotare un posto o cancellare una prenotazione. Questo tentativo è rappresentato dalla variabile *student.x* e può assumere tre valori: -1 che rappresenta il tentativo di cancellare una prenotazione; 0 che rappresenta il fatto che non c'è input da parte dello studente; +1 che rappresenta il tentativo di prenotare un posto in aula.

- **Prodigit**: viene modellato dalle due classi *Controller1* e *Controller2* nei file *controller1.mo* e *controller2.mo* e rappresenta il sistema con cui interagisce l' utente e che gestisce le prenotazioni. Di seguito si descrivono le due classi che lo modellano:

  - **Controller1**: ha lo scopo di leggere le informazioni sullo stato dell' aula dal *Gomp* e in base ad esse tenere aperte le prenotazioni oppure azzerarle nel caso in cui l' aula sia diventata *Inagibile*.
  - **Controller2**: ha lo scopo di prendere in input, l' output dello studente (*ctr2.input_studente*), il numero di prenotazioni dall' aula (*ctr2.num_pre*) e la capienza massima di quest' ultima dal Gomp (*ctr2.gomp_max*) e in base a questi variare l' output da mandare all' aula (*ctr2.prenotazione*).

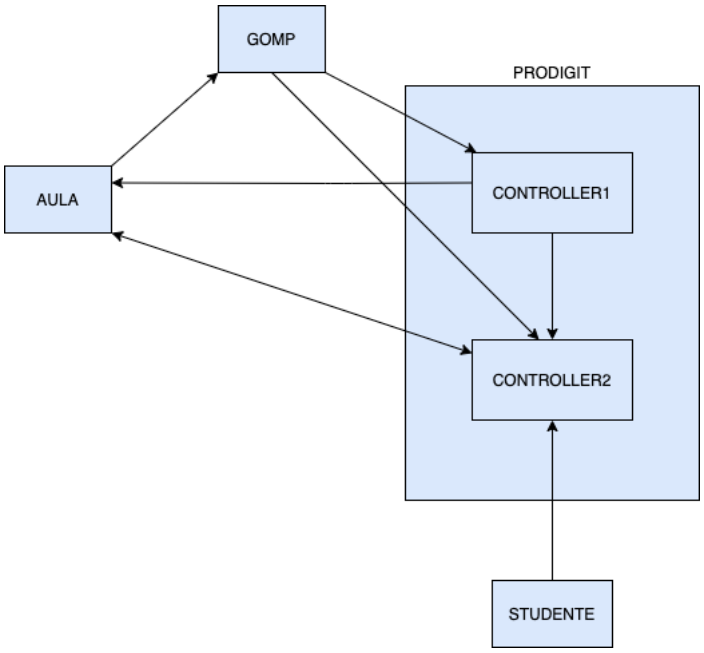


Figure 1: Diagramma di alto livello dell' architettura del sistema

Inoltre il *Controller1* prende in input lo stato del Gomp (*ctr1.gomp_state*) e lo ritrasmette anche al *Controller2* (*ctr2.ctr1_state*), in quanto quando Gomp è *Down* Prodigit non potendo ottenere le informazioni necessarie entra anche esso nello stesso stato (*ctr1.state e ctr2.state*).

4 Requisiti del sistema

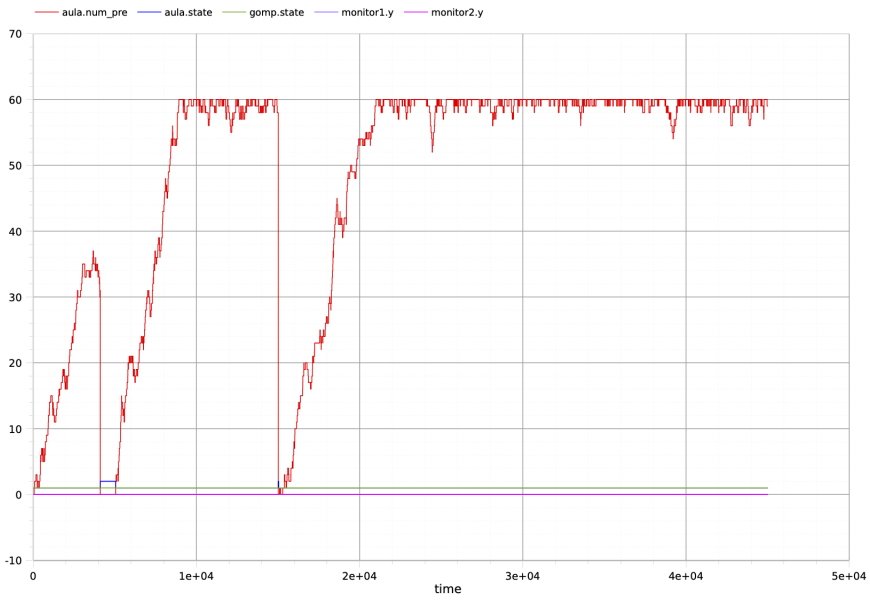
Di seguito vengono elencati e descritti i requisiti funzionali e non del sistema, che sono stati presi in considerazione durante la modellazione e che vengono verificati dai *Monitor*:

- **Requisiti funzionali**:
  - **R_F_Safety_1**: *Non fare mai overbooking (cioè, prenotare per un aula un numero di studenti superiore alla capienza covid dell'aula).* Questo requisito viene rispettato grazie alla presenza del *Controller2*, che ricevendo la richiesta di prenotazione dallo *Studente* e il numero di prenotazioni attuale dall' *Aula*, decide se "inoltrare" la prenotazione all' aula (mettendo *ctr2.prenotazione=1*) o bloccarla (mettendo *ctr2.prenotazione=0*) nel caso in cui il numero di prenotazioni abbia raggiunto la capienza massima.
  - **R_F_Safety_2**: *Non accettare mai prenotazioni se l' aula è inagibile.* Questo requisito viene rispettato grazie al *Controller1* che ottenendo informazioni sullo stato dell' aula dal *Gomp* blocca le prenotazioni nel caso in cui l' *Aula* diventi *Inagibile*.
- **Requisiti non funzionali**:
  - **R_NF_Availability_1**: *Quando un' aula torna agibile, riaprire le prenotazioni al massimo dopo un' ora (60 min).* Questo requisito viene rispettato grazie al *Controller1* che leggendo le informazioni sullo stato dell' aula dal Gomp ogni $T <= 60$ min., garantisce che l' intervallo di tempo tra quando l' aula ritorna *Agibile* e quando vengono riaccettate prenotazioni sia inferiore o uguale a un' ora.

5 Risultati sperimentali

Di seguito vengono riportati una serie di test effettuati, con annessi i grafici rispettivi:

- **Run_1**: il sistema è stato eseguito per 45000 unità di tempo (circa un mese). Il grafico mostra come variano i seguenti valori:
  - aula.num_pre (il numero di prenotazioni)
  - monitor1.y (requisiti funzionali)
  - monitor2.y (requisiti non funzionali)
  - gomp.state
  - aula.state (agibilità)



- **Run_2**: il sistema è stato eseguito per 10000 unità di tempo (circa una settimana). Il grafico mostra come variano i seguenti valori:
  - aula.num_pre
  - controller1.u1_2

  L' esperimento serve a vedere come il controller1 interviene in caso di inagibilità dell' aula (intorno al tempo 4000)

- **Run_3**: il sistema è stato eseguito per 10000 unità di tempo (circa una settimana). Il grafico mostra come variano i seguenti valori:
  - aula.num_pre
  - controller1.u1_1
  - gomp.state

  L' esperimento serve a vedere come il controller1 interviene nel caso in cui il Gomp entra nello stato *Down* (intorno al tempo 5500)

- **Verify**: Lo script è stato eseguito con 100, 1000 e poi 10000 iterazioni. Ad ogni iterazione sono stati cambiati randomicamente i parametri randomici del sistema e testato, attraverso il controllo dei valori dei monitor, il suo corretto funzionamento.
  Tutti e tre i test hanno dato un esito positivo, ovvero il 100% di test passati. Di seguito si mostrano i tempi di esecuzione dei tre test.
  - Test con 100 iterazioni: 106.904 sec.
  - Test con 1000 iterazioni: 1105.3577 sec. (circa 18 min.)
  - Test con 10000 iterazioni: 12000.6979 sec. (circa 200 min. ovvero quasi 3.5 ore)
- **Synth**: Per quanto riguarda il *Controller1* è stato eseguito prima un *synth* con il T che è variato da 1 a 10 compiendo ogni volta 100 test. Dopodichè sono stati presi i valori che hanno passato tutti e 100 i test ed è stato rieseguito il synth con solo questi valori compiendo 1000 test ognuno. Di seguito si mostrano i risultati.
  - **T = 1.0** : 1000/1000 test passati (100%)
  - **T = 1.3** : 996/1000 test passati (99.6%)
  - **T = 1.38** : 995/1000 test passati (99.5%)
  - **T = 1.76** : 995/1000 test passati (99.5%)
  - **T = 1.9** : 997/1000 test passati (99.7%)
  - **T = 2.14** : 989/1000 test passati (98.9%)

- **T = 2.52** : 986/1000 test passati (98.6%)
- **T = 2.8** : 993/1000 test passati (99.3%)
- **T = 3.66** : 988/1000 test passati (98.8%)
- **T = 4.0** : 991/1000 test passati (99.1%)
- **T = 4.8** : 978/1000 test passati (97.8%)
- **T = 5.5** : 985/1000 test passati (98.5%)
- **T = 6.39** : 978/1000 test passati (97.8%)
- **T = 6.7** : 977/1000 test passati (97.7%)

Da come si può leggere, l' unica iterazione a passare tutti i test è stata quella con $T = 1$. Questo però non toglie che tutti i valori hanno raggiunto un affidabilità superiore al 97.6% (la più bassa con il 97.7%) e 6 valori su 14 superiore al 99% con un picco di 99.7% (con $T = 1.9$).

Di seguito si mostrano i tempi di esecuzione dei test:

- Test con 100 iterazioni per ogni valore (60 valori): 9370.7418 sec. (circa 2.5 ore)
- Test con 1000 iterazioni per ogni valore (14 valori): 20519.7177 sec. (circa 5 ore e 42 min.)

Per quanto riguarda il *Controller2*, l' unica iterazione a ritornare il 100% dei test passati è stato quella con $T = 1.0$. Inoltre, a differenza del *Controller1*, altri valori del T non hanno raggiunto una percentuale di test passati alta. Questo è causato dal fatto che, spesso arrivano due o più prenotazioni di fila e quando questo succede in momenti in cui i posti disponibili sono quasi terminati, con un controller con $T > 1.0$ spesso viene fatto *overbooking*.