

Progetto Ingegneria del Software

Sistema “Prodigit”

A.Y. 2020-2021

Francesco Mauto

System Description	2
Operational scenario	3
System architecture	4
System requirements	5
Experimental results	6

System Description

Questo progetto modella, ad alto livello, un sistema di prenotazione di posti in aule di università. Ci permette di simulare l'interazione tra l'environment, ossia gli attori del sistema, quali sono le aule e gli studenti, con il sistema stesso, formato dal gomp, sistema esterno della Sapienza, il quale gestisce le informazioni relative alle aule, e prodigit, il sistema che permette agli studenti di effettuare prenotazioni o di cancellare prenotazioni già effettuate, mantenendo tutte le informazioni necessarie affinché i dati non vadano persi.

L'obiettivo principale è quello di far sì che la comunicazione tra i sistemi e gli attori sia ottimizzata a tal punto da non poter creare errori di sincronizzazione e quindi disagi per gli studenti.

Alcune premesse:

- Il sistema viene simulato come se fosse presente una sola aula contenente 60 posti alla quale prenotarsi. Approfondire il sistema è possibile, basta aggiungere nuove aule, ho però ritenuto sufficiente l'utilizzo di una singola aula per verificare il comportamento del sistema.
- L'unità di misura adattata per la simulazione del sistema corrisponde a $T=1$, equivalente ad 1 ora reale.
- Il sistema viene simulato per 5 giorni, da lunedì a venerdì, come richiesto nelle specifiche del progetto. In totale sono 120 ore, che equivalgono a $T = 120$.

Operational scenario

I casi d'uso del sistema sono i seguenti:

- **Studente**
 - Per utilizzare Prodigit, la matricola dello studente deve essere abilitata per la settimana successiva. Nel caso in cui sia abilitato, allora può utilizzare il sistema e decidere se prenotare oppure cancellare una prenotazione precedentemente effettuata.
- **Aule**
 - Forniscono al Gomp la loro agibilità.
- **Gomp**
 - Contiene informazioni sulle aule e le fornisce a Prodigit.
 - Può essere attivo o inattivo. Questo può influenzare Prodigit.
- **Prodigit**
 - Può essere attivo o inattivo ma solo in base allo stato del Gomp.
 - Gestisce le operazioni effettuate dagli studenti, mantenendo i dati delle prenotazioni per le aule, così da evitare casi di overbooking ed altri errori simili.

Il sistema è completo e corretto, il tutto verificabile tramite gli script python *verify* e *synth*. Attraverso di essi possiamo verificare come, modificando ad ogni iterazione i valori della probabilità che lo studente prenoti e la probabilità che il Gomp sia down, il sistema risulta essere stabile e rispetta i vincoli imposti.

System architecture

Di seguito vengono descritte le componenti del sistema, e come interagiscono tra di loro.

- **Aula**
 - Definita nel blocco *aule.mo* e rappresenta l'aula. L'unico scopo dell'aula è quello di variare la sua agibilità. Essa viene aggiornata ogni 0.5 secondi (Equivalenti a 30 minuti) ed ha una probabilità dell'80% che sia agibile.
- **Studente**
 - Definito nel blocco *studenti.mo* e rappresenta lo studente. Il suo scopo è quello di simulare lo studente e, per farlo, restituisce in output due variabili fondamentali: la possibilità di utilizzo di prodigit, scelta casualmente con una probabilità del 50%; L'operazione da eseguire, ossia la prenotazione o la cancellazione. Lo studente ha il 70% di probabilità di effettuare una prenotazione, e il 30% di effettuare una cancellazione (Probabilità variabile tramite gli script python).
- **Gomp**
 - Definito nel blocco *gomp.mo* e rappresenta il sistema esterno dell'università. Il suo scopo è quello di fornire a Prodigit informazioni riguardanti l'aula (Agibilità, posti,...). Inoltre, il Gomp può risultare alle volte inagibile (Con probabilità del 10%, ma variabile tramite gli script python).
- **Prodigit**
 - Definito nel blocco *prodigit.mo* e rappresenta il sistema Prodigit. Gestisce l'usabilità del sistema stesso, influenzata però dallo stato del Gomp. Inoltre, se la situazione totale del sistema risulta adeguata (Ad esempio: aula agibile, studente abilitato, ...) a far sì che lo studente possa eseguire operazioni, allora le esegue. Inoltre mantiene il numero di studenti prenotati per ogni aula. Quest'ultima caratteristica permette a Prodigit di essere agibile anche se il Gomp non lo è.
- **Monitor (Funzionali e non)**
 - Verificano se i requisiti del sistema sono rispettati.
 - *MonitorSafety.mo (Funzionale)*
 - Verifica se il sistema va in overbooking
 - *MonitorLiveness.mo (Funzionale)*
 - Verifica che, se ci sono ancora posti disponibili e tutte le condizioni sono rispettate affinché uno studente possa prenotare, allora prenota.

- *MonitorDown.mo (Non funzionale)*
 - Verifica che Prodigit sia down all'80% rispetto a quante volte il Gomp è down.

System requirements

Requisiti funzionali:

- 1) **Safety**: il sistema non permetterà mai di fare overbooking, ossia di prenotare quando non ci sono più posti disponibili.
- 2) **Liveness**: Il sistema non rifiuterà mai una prenotazione se ci sono posti disponibili e se tutte le condizioni tali per effettuarla sono rispettate.

Questi requisiti vengono gestiti all'interno di Prodigit, attraverso vari sistemi di controllo. Vengono inoltre monitorati all'interno degli appositi monitor, *MonitorSafety.mo* e *MonitorLiveness.mo*., i quali restituiscono un valore booleano *true* se i requisiti non vengono rispettati.

Requisiti non funzionali

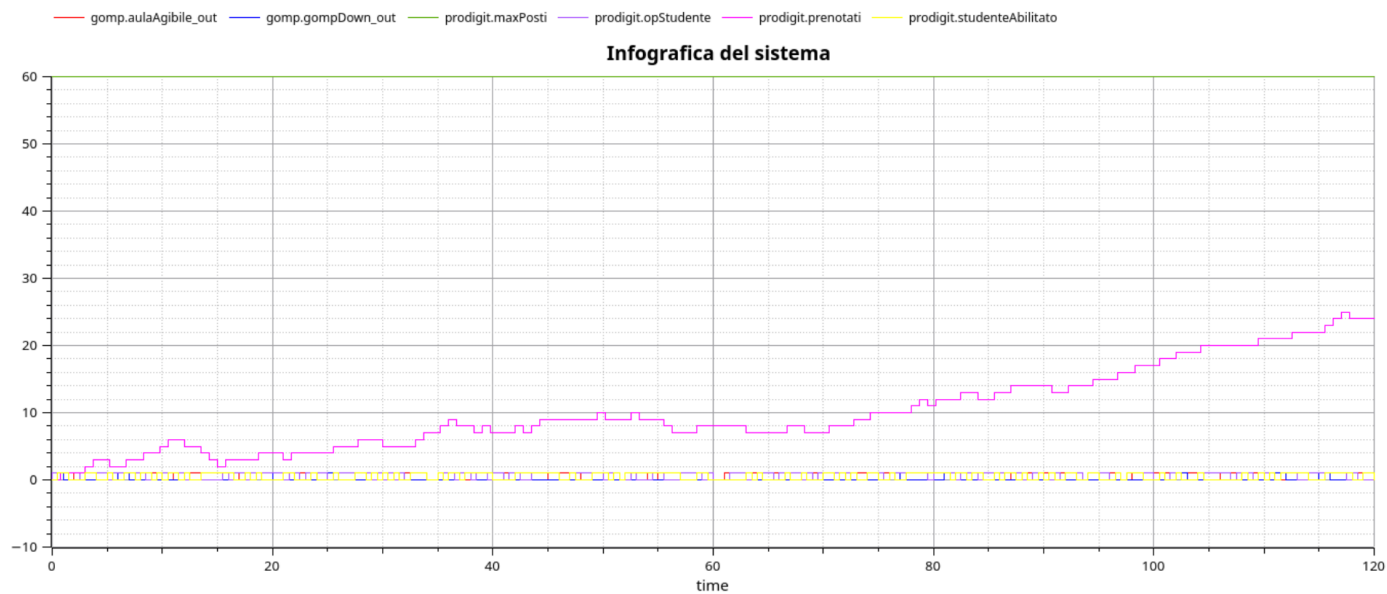
- 1) Si vuole che prodigit sia down al più per l'80% del tempo per cui il GOMP è down. Viene verificato all'interno del monitor apposito, *MonitorDown.mo*, il quale controlla, attraverso una semplice operazione, se il requisito viene rispettato. Nel caso in cui fosse violato restituirà una variabile booleana settata a *true*.

Experimental results

Il sistema risulta funzionante e simula perfettamente quello che potrebbe accadere se il sistema fosse usato da veri studenti.

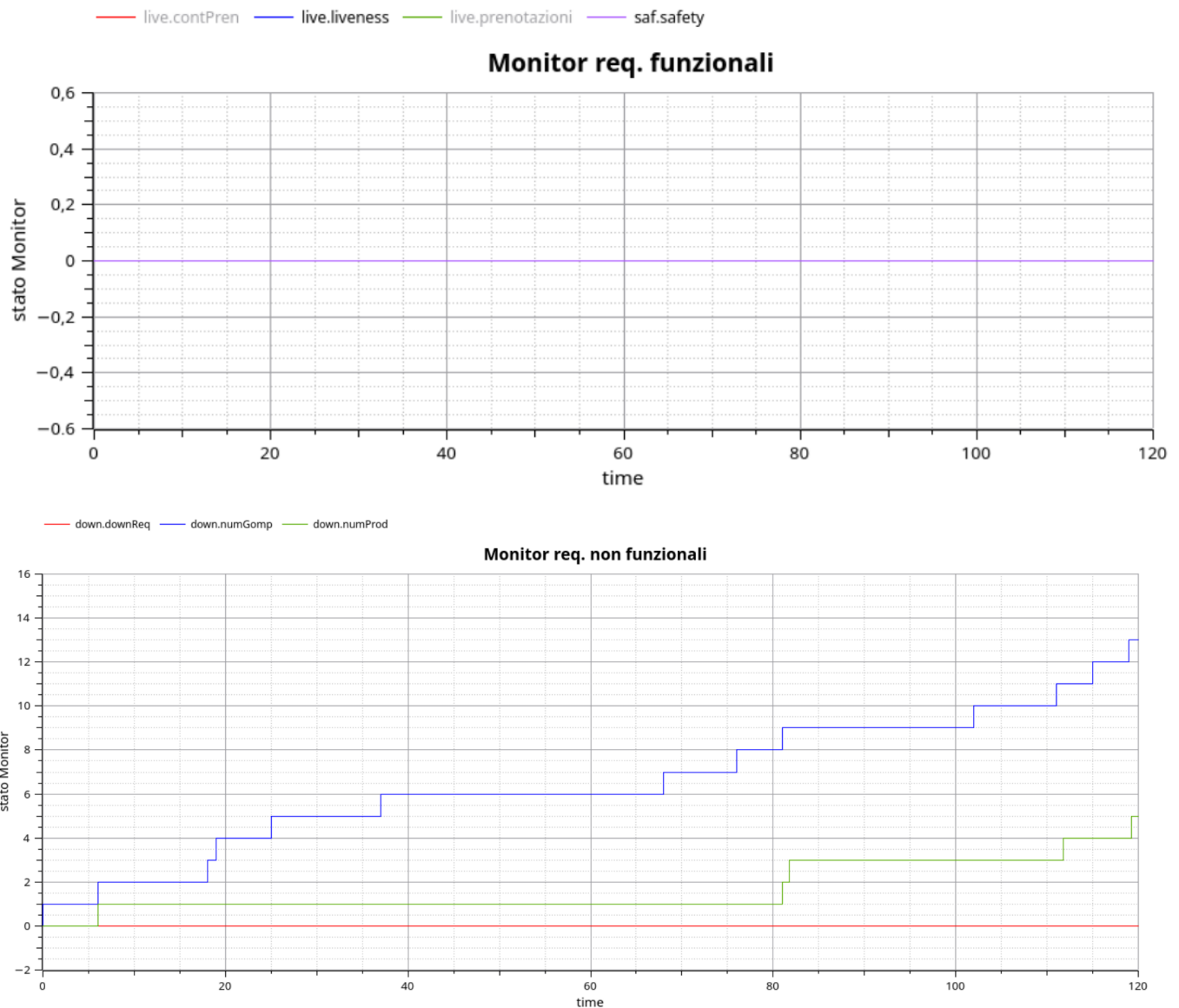
```
record SimulationResult
  resultFile = '/home/openmodelica/Desktop/ProdigitModelica/prj/Models/System_res.mat',
  simulationOptions = 'startTime = 0.0, stopTime = 120.0, numberOfIntervals = 500, tolerance = 1e-06, method = 'dassl', fileNamePrefix = 'System', options = '', outputFormat = 'mat', variableFilter = '.*', cflags = '', simflags = ''',
  messages = "LOG SUCCESS | info | The initialization finished successfully without homotopy method.",
  LOG_SUCCESS | info | The simulation finished successfully.
end SimulationResult;

timeFrontend = 0.376881456,
timeBackend = 0.288802748,
timeSimCode = 0.166302282,
timeTemplates = 0.189778817,
timeCompile = 13.408915188,
timeSimulation = 0.161775357,
timeTotal = 14.594105335
end SimulationResult;
```



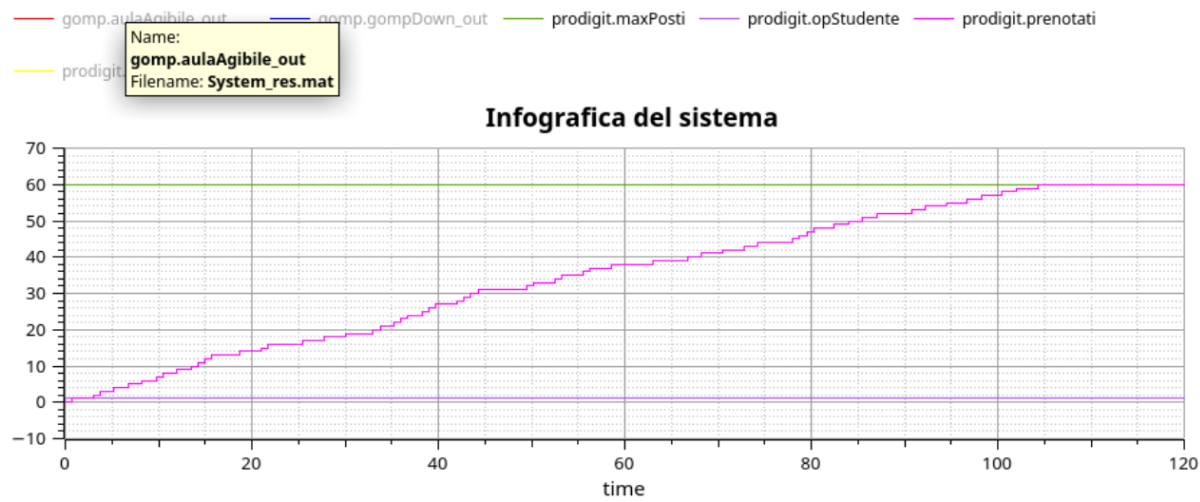
Su questo grafico possiamo notare come, in base allo stato dell'aula, allo stato dello studente e allo stato di prodigit, ci sono alcuni periodi di tempo nei quali non vengono effettuate prenotazioni o cancellazioni.

I plot dei monitor, accompagnati da variabili utili al riconoscimento del rispetto del requisito, mostrano che il funzionamento del sistema è corretto e solido, e i requisiti non vengono mai violati.

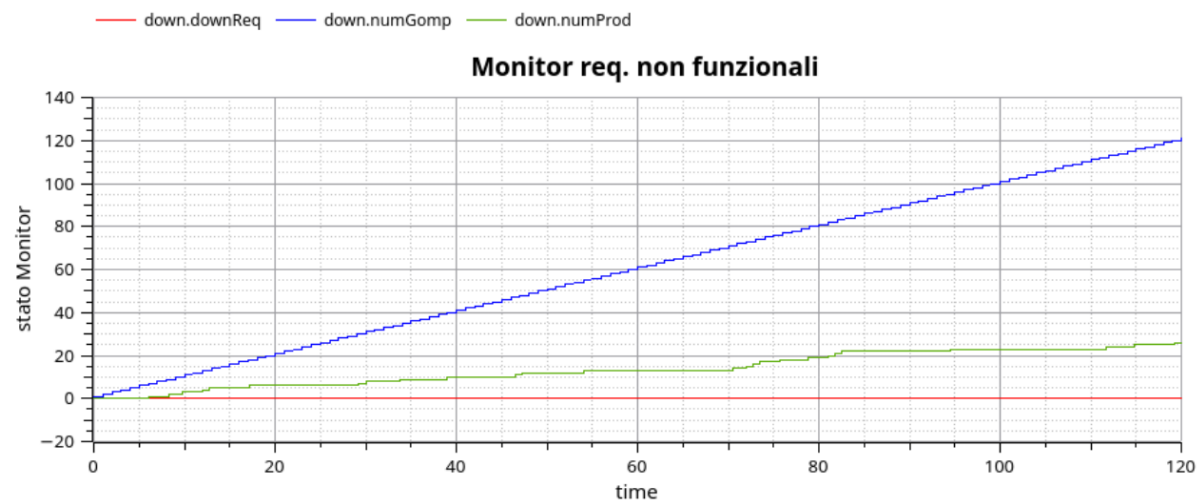


Possiamo inoltre vedere il comportamento del sistema spinto al limite dei vincoli imposti:

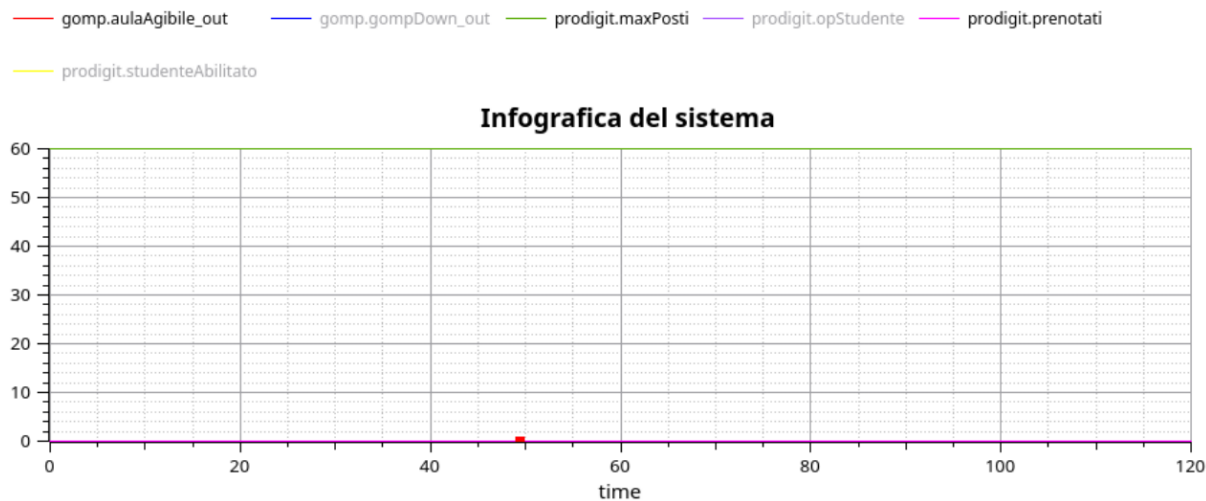
- Solo prenotazioni



- Gomp down



- Aula sempre inagibile (la linea rossa combacia con quella viola del numero di prenotazioni)



Inoltre, attraverso l'utilizzo dei 2 script python, *verify.py* (Eseguito 100,1000,10000 volte) e *synth.py* (Eseguito 100,1000 volte), possiamo solidificare ancora più la robustezza del sistema. Questo perchè attraverso di essi si vanno a modificare randomicamente alcuni parametri.

Il primo si occupa di controllare i 2 requisiti funzionali. Per farlo, randomizza la probabilità che lo studente si prenoti, in modo tale da prendere in considerazione anche i casi estremi.

Il secondo si occupa di controllare il requisito non funzionale, andando a modificare casualmente la probabilità che il Comp sia down.

Entrambi gli script, in ogni loro simulazione, non hanno mai riscontrato violazioni dei requisiti.