RELAZIONE PROGETTO INFORMATICA 3B
AA 2019/2020:
PROGETTAZIONE E ALGORITMI

Exploring the concept of Abstract State Machines for the runtime enforcement of software systems

Daniel Hernan Altamirano [1035295]
Daniele Troiano [1035036]
Rebucini Federico [1035227]

Sommario

1. Introduzione

Per il progetto di informatica IIIB sono state sviluppate nuove funzionalità per il framework Asmeta, in particolare all'interno del simulatore di runtime per ASM asmeta.simulator@run.time, il quale consente l'utilizzo di modelli ASM come modelli di runtime, supporta la simulazione come servizio ed include funzionalità come il rollback del modello allo stato iniziale dopo un errore in esecuzione (ad es. violazioni invarianti, aggiornamenti incoerenti, input non validi, ecc.) durante l'elaborazione di un evento di input.
Il lavoro effettuato consiste nell'aggiunta di un'interfaccia grafica che permetta al cliente di caricare le simulazioni di uno o più modelli ASM e per ognuna è possibile visualizzare un catalogo di dichiarazioni di sicurezza, espresso sotto forma di invarianti ASM, che descrivono tutte le possibili situazioni che possono causare una violazione. Il catalogo può essere aggiornato dinamicamente in fase di esecuzione, aggiungendo invarianti, modificandole o cancellandole.
Inoltre, è stato aggiunto un interprete dei comandi che attraverso il CMD permette di eseguire tutte le funzioni presenti nell'asmeta.simulator@run.time.

2. Casi d’uso

In questa sezione verranno introdotti i casi d’uso mostrati in figura 1. Ogni caso d’uso verrà descritto tramite un modello uniforme che include una breve descrizione del caso d’uso stesso, il flusso del processo standard, le informazioni come precondizioni, postcondizioni e trigger. Infine, verrà dedicato uno spazio che descrive i tool necessari per l’esecuzione del progetto e quelli utilizzati per il suo sviluppo.

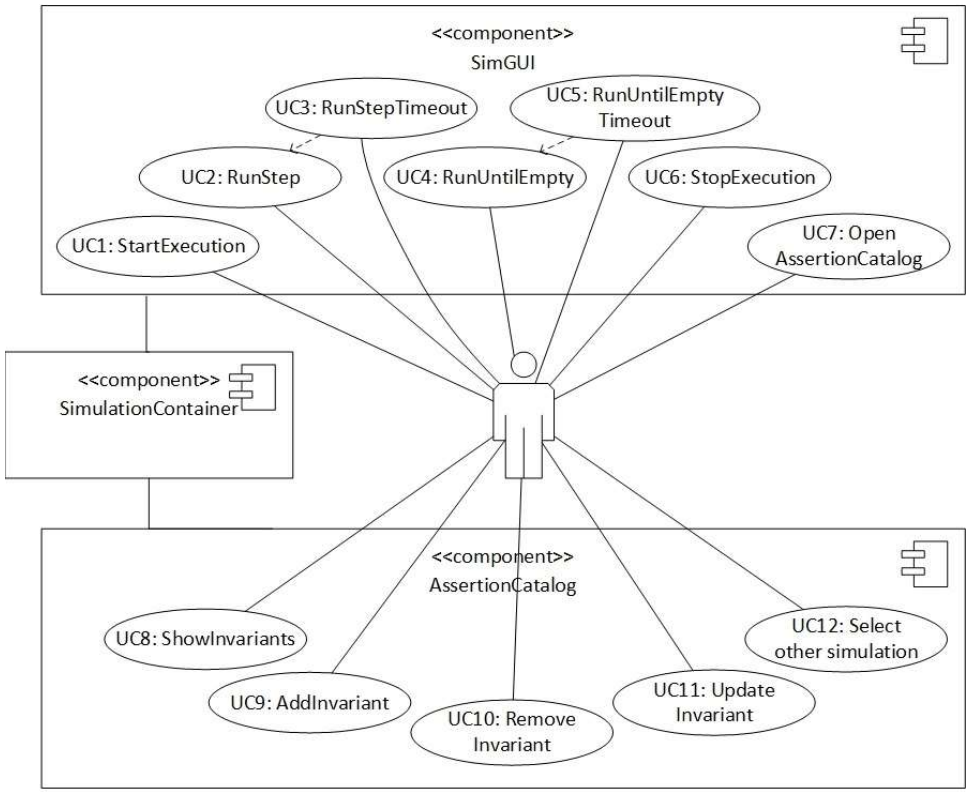

Figura 1 - Diagramma dei casi d'uso

2.1 StartExecution

Breve descrizione: prende come input il percorso di un modello ASM e restituisce un id corrispondente basato sulla correttezza del modello, l'id sarà maggiore di 0 se il modello non presenta problemi, altrimenti restituirà un valore minore di 0 che corrisponderà ad un'eccezione specifica.

Precondizioni: disponibilità del modello ad essere creato.

Trigger: l'utente vuole validare e verificare il modello scelto.

Postcondizioni: l'utente ha fornito un modello valido e ha ricevuto un id del simulatore memorizzato in una mappa.

Processo standard:

1. L'utente possiede un modello da simulare presente in locale sulla propria macchina.
2. L'utente invoca il metodo *start* fornendo come input il percorso del modello.
3. Il sistema crea il simulatore a cui viene associato un id tra quelli attualmente disponibili.
4. Il sistema salva il simulatore creato in una mappa usando come chiave l'id generato.
5. Il sistema fornisce all'utente l'id associato al simulatore.

Alternative:

- L'utente fornisce un modello non valido.
    1. Il sistema segnala all'utente l'impossibilità di caricare il modello fornito.
- Mappa simulazioni piena.
    1. Il sistema segnala all'utente l'impossibilità della richiesta.
    2. L'utente riprova più tardi.

2.2 RunStep

Breve descrizione: Questo metodo prende come input l'id restituito dalla StartExecution e la lista delle variabili monitorate del modello, viene eseguita la simulazione step per step in base all'input dell'utente ed ogni volta il sistema restituisce all'utente lo stato del modello (Figura 2).

Precondizioni: L'id fornito dalla StartExecution deve essere corretto e l'insieme delle variabili monitorate devono essere quelle richieste dal modello.

Trigger: L'utente vuole eseguire la simulazione step per step.

Postcondizioni: Il modello viene simulato e fornisce all'utente il nuovo stato.

Processo standard

1. L'utente possiede l'id del simulatore generato sul modello desiderato e l'insieme delle funzioni monitorate, qualora esse siano richieste dal modello
2. L'utente invoca il metodo *RunStep* fornendo come input il valore della variabile monitorata richiesta dal sistema
3. Il sistema controlla la correttezza dell'id fornito
4. Il simulatore viene estratto dalla mappa dal sistema
5. Il sistema aggiorna lo stato del modello
6. Il sistema fornisce all'utente il nuovo stato generato dalla simulazione

Alternative

Durante l'esecuzione del modello si verifica un errore di invariante oppure di update inconsistente all'interno del sistema.

1. Il sistema segnala all'utente il tipo di errore verificato.
2. Il sistema ritorna allo stato precedente, ovvero quello in cui era prima della richiesta di run.
3. L'utente riprova inserendo nuovi valori delle funzioni monitorate.

2.3 RunStepTimeout

Breve descrizione: funzionamento analogo alla RunStep, con la differenza che viene settato un timer entro il quale la simulazione deve terminare, altrimenti viene lanciata un'eccezione (Figura 2).
Ovviamente, l'esito della simulazione, in caso di timeout, è UNSAFE.

2.4 RunUntilEmpty

Breve descrizione: il metodo runUntilEmpty esegue la simulazione fino a che la Main rule del modello non risulta vuota (Figura 2). Questo metodo si occupa di richiedere dalla tabella il simulatore riferito al particolare id ricevuto in input.

Precondizioni: L'id fornito dalla StartExecution deve essere corretto e l'insieme delle variabili monitorate devono essere quelle richieste dal modello.

Trigger: L'utente vuole eseguire il modello fino a che la Main rule del modello non risulta vuota

Postcondizioni: Il modello viene simulato e fornisce all'utente lo stato finale.

Processo standard:

1. L'utente possiede l'id del simulatore generato sul modello desiderato e l'insieme delle funzioni monitorate, qualora esse siano richieste dal modello.
2. L'utente invoca il metodo *runUntilEmpty* fornendo come input il valore della variabile monitorata richiesta dal sistema
3. Il sistema controlla la correttezza dell'id fornito.
4. Il simulatore viene estratto dalla mappa dal sistema.
5. Il sistema memorizza lo stato corrente.
6. Il sistema esegue il run più volte fino a che la condizione sulla regola principale del modello non risulta verificata.
7. Il sistema fornisce all'utente lo stato finale generato dalla simulazione.

Alternative

1. Durante l'esecuzione del modello si verifica un errore di invariante oppure di update inconsistente all'interno del sistema.
2. Il sistema segnala all'utente il tipo di errore verificato.
3. Il sistema ritorna allo stato precedente, ovvero quello in cui era prima della richiesta di run.
4. L'utente riprova inserendo nuovi valori delle funzioni monitorate.

2.5 RunUntilEmptyTimeout

Funzionamento analogo alla RunUntilEmpty, con la differenza che viene settato un timer entro il quale la simulazione deve terminare, altrimenti viene lanciata un'eccezione (Figura 2).
Ovviamente, l'esito della simulazione, in caso di timeout, è UNSAFE.

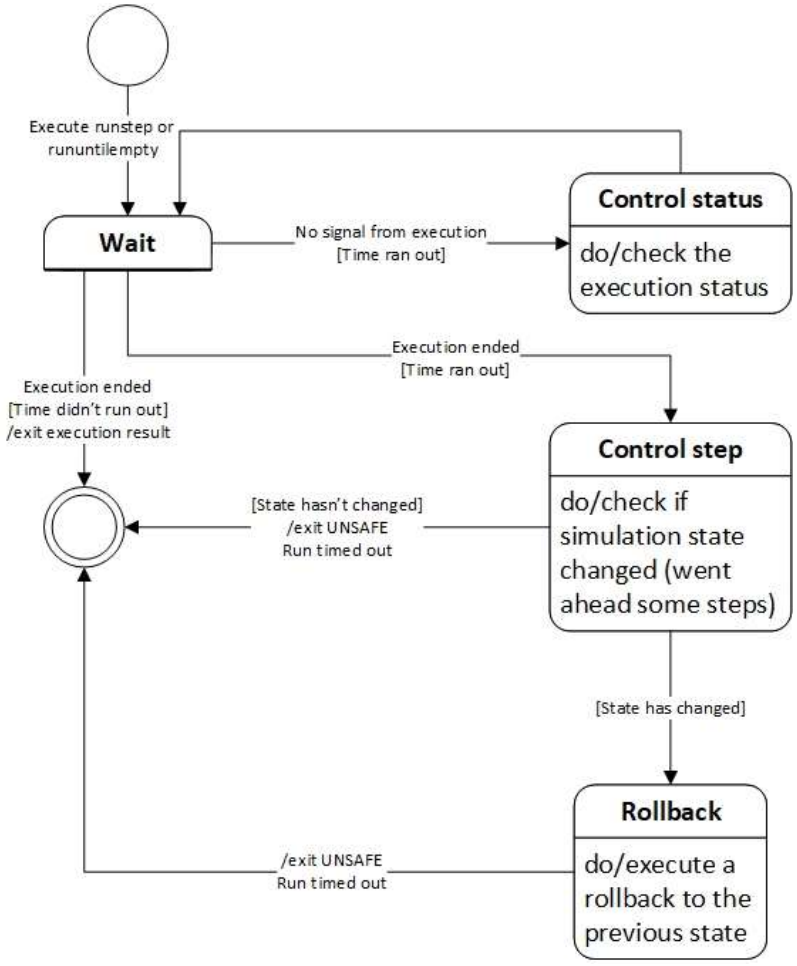

Figura 2 - RunStep e RunUntilEmpty

2.6 StopExecution

Breve descrizione: ferma una simulazione rimuovendo il suo id dalla tabella delle simulazioni.

Precondizioni: l'id fornito dall'utente deve essere corretto, ovvero deve essere associato alla simulazione che si vuole eliminare.

Trigger: l'utente vuole fermare la simulazione di un modello.

Postcondizioni: il simulatore viene cancellato dalla mappa interna.

Processo standard:

1. L'utente possiede l'id del simulatore da fermare
2. L'utente invoca il metodo *stop* fornendo come input un id del simulatore
3. Il sistema controlla la correttezza dell'id fornito
4. Il simulatore viene cancellato dalla mappa dal sistema

Alternative:

- L'utente fornisce un id non valido.
    1. Il sistema segnala all'utente l'impossibilità di trovare il simulatore fornito.
- L'utente fornisce un id valido ma riferito ad un altro simulatore.
    1. Il simulatore viene cancellato.

2.7 OpenAssertionCatalog

Breve descrizione: catalogo di dichiarazioni di sicurezza, espresso sotto forma di invarianti ASM, che descrivono tutte le possibili situazioni che possono causare una violazione. Il catalogo può essere aggiornato dinamicamente in fase di esecuzione, aggiungendo invarianti, modificandole o cancellandole.

Precondizioni: deve essere fornito un id valido della simulazione, il percorso del file .asm e l'istanza del container

Trigger: l'utente vuole gestire dinamicamente gli invarianti del modello scelto

Postcondizioni: viene mostrata una finestra contenente tutti gli invariant divisi per nome, over e corpo. È possibile, quindi, gestire gli stessi grazie a pulsanti presenti nella finestra.
Processo standard:

1. l'utente possiede l'id della simulazione scelta e i vari parametri
2. viene richiamata la classe InvariantGui
3. viene controllata la correttezza dei parametri
4. apertura dell'interfaccia grafica
5. caricamento automatico degli invariant

Alternative:

- non viene fornito un id valido di una simulazione di conseguenza l'interfaccia grafica non viene aperta

2.8 ShowInvariants

Breve descrizione: Questo metodo prende come input l'id della simulazione, permette di visualizzare tutti gli invarianti del modello corrispondente all'interno del catalogo (Figura 3).

Precondizioni: deve essere fornito un id valido della simulazione, il percorso del file .asm e l'istanza del container

Trigger: l'utente vuole vedere una versione aggiornata degli invarianti (un invariante può essere stato modificato staticamente senza utilizzare questo tool)

Postcondizioni: viene restituita la lista degli invarianti aggiornata all'ultima modifica

Processo standard:

1. viene passato in automatico l'id della simulazione
2. vengono caricati in una lista tutti gli invariant presenti nel file .asm
3. viene popolata una tabella suddividendo l'invariant nei 3 campi principali
4. viene mostrata a video la tabella

Alternative:

- se il file viene erroneamente cancellato durante l'esecuzione del programma, viene sollevata una eccezione e viene mostrato a video un messaggio di errore ("No file selected")
- se il percorso del file viene modificato oppure viene compromesso durante il passaggio da una classe all'altra, viene riportato un messaggio di errore del parser

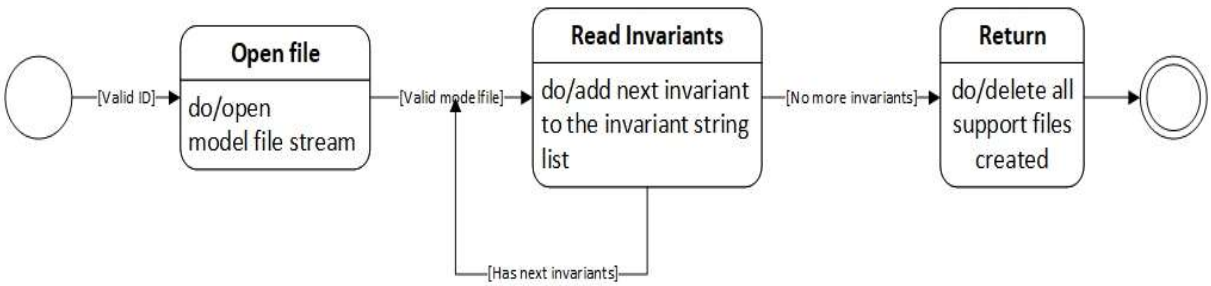

Figura 3 - ShowInvariants

2.9 AddInvariant

Breve descrizione: Questo metodo prende come input l'id della simulazione e il nuovo Invariant che si vuole aggiungere, verrà modificato il modello corrispondente aggiungendo il nuovo Invariant (Figura 4).

Precondizioni: deve essere fornito un id valido della simulazione, il percorso del file .asm e l'istanza del container

Trigger: l'utente vuole aggiungere un invariant con la possibilità di assegnargli un nome che non sia già presente

Postcondizioni: l'invariant viene aggiunto con successo

Processo standard:

1. viene passato automaticamente l'id della simulazione
2. compare una finestra di dialogo che permette all'utente di aggiungere un nuovo invariant con la possibilità di lasciare il campo nome vuoto
3. viene controllata la correttezza dell'invariant che si vuole inserire
4. terminato l'inserimento, viene richiamata la funzione addInvariant che scrive direttamente sul file il nuovo invariant
5. infine, viene richiamata in automatico la funzione viewListInvariant che aggiorna la tabella principale

Alternative:

- uno dei campi dell'invariant non viene compilato correttamente, viene restituito un messaggio di errore con la possibilità di reinserimento
    1. il nome è già stato usato: viene restituito un messaggio "the variable is already taken"
    2. vengono scelti due over identici: viene restituito un messaggio "double over selected"
    3. il corpo principale dell'invariant non è corretto: viene restituito un messaggio "invalid invariant"
    4. il corpo principale dell'invariant è vuoto: viene restituito un messaggio "the field 'content' cannot be empty"

2.10 RemoveInvariant

Breve descrizione: Questo metodo prende come input l'id della simulazione e l'Invariant che si vuole eliminare, verrà modificato il modello corrispondente rimuovendo l'Invariant corrispondente (Figura 4).

Precondizioni: deve essere fornito un id valido della simulazione, il percorso del file .asm e l'istanza del container

Trigger: l'utente vuole rimuovere un invariant

Postcondizioni: l'invariant viene cancellato con successo

Processo standard:

1. l'utente seleziona l'invariant da voler rimuovere
2. viene passato automaticamente l'id della simulazione
3. viene richiesta una conferma direttamente all'utente
4. in caso affermativo viene richiamata la funzione removeInvariant che rimuove l'invariant scelto
5. infine, viene richiamata in automatico la funzione viewListInvariant che aggiorna la tabella principale

Alternative:

- se l'utente non conferma la rimozione, il file rimane invariato

2.11 UpdateInvariant

Breve descrizione: Questo metodo prende come input l'id della simulazione, l'Invariant nuovo e l'Invariant vecchio, verrà modificato il modello corrispondente imuovendo l'Invariant vecchio e aggiungendo quello nuovo (Figura 4).

Precondizioni: deve essere fornito un id valido della simulazione, il percorso del file .asm e l'istanza del container

Trigger: l'utente vuole modificare alcuni campi dell'invariant (nome, over e corpo). Viene effettuato un controllo sul nome per evitare doppioni

Postcondizioni: l'invariant viene modificato con successo

Processo standard:

1. l'utente seleziona l'invariant da voler modificare
2. viene passato automaticamente l'id della simulazione
3. compare una finestra di dialogo con i campi precompilati che permette all'utente di modificare ciò che desidera
4. viene controllata la correttezza dell'invariant che si vuole modificare
5. terminate le modifiche, viene richiamata la funzione updateInvariant che sovrascrive il vecchio invariant con il nuovo
6. infine, viene richiamata in automatico la funzione viewListInvariant che aggiorna la tabella principale

Alternative:

- uno dei campi dell'invariant non viene compilato correttamente, viene restituito un messaggio di errore con la possibilità di reinserimento
    1. il nome è già stato usato non dallo stesso invariant: viene restituito un messaggio "the variable is already taken"
    2. vengono scelti due over identici: viene restituito un messaggio "double over selected"
    3. il corpo principale dell'invariant non è corretto: viene restituito un messaggio "invalid invariant"
    4. il corpo principale dell'invariant è vuoto: viene restituito un messaggio "the field 'content' cannot be empty"

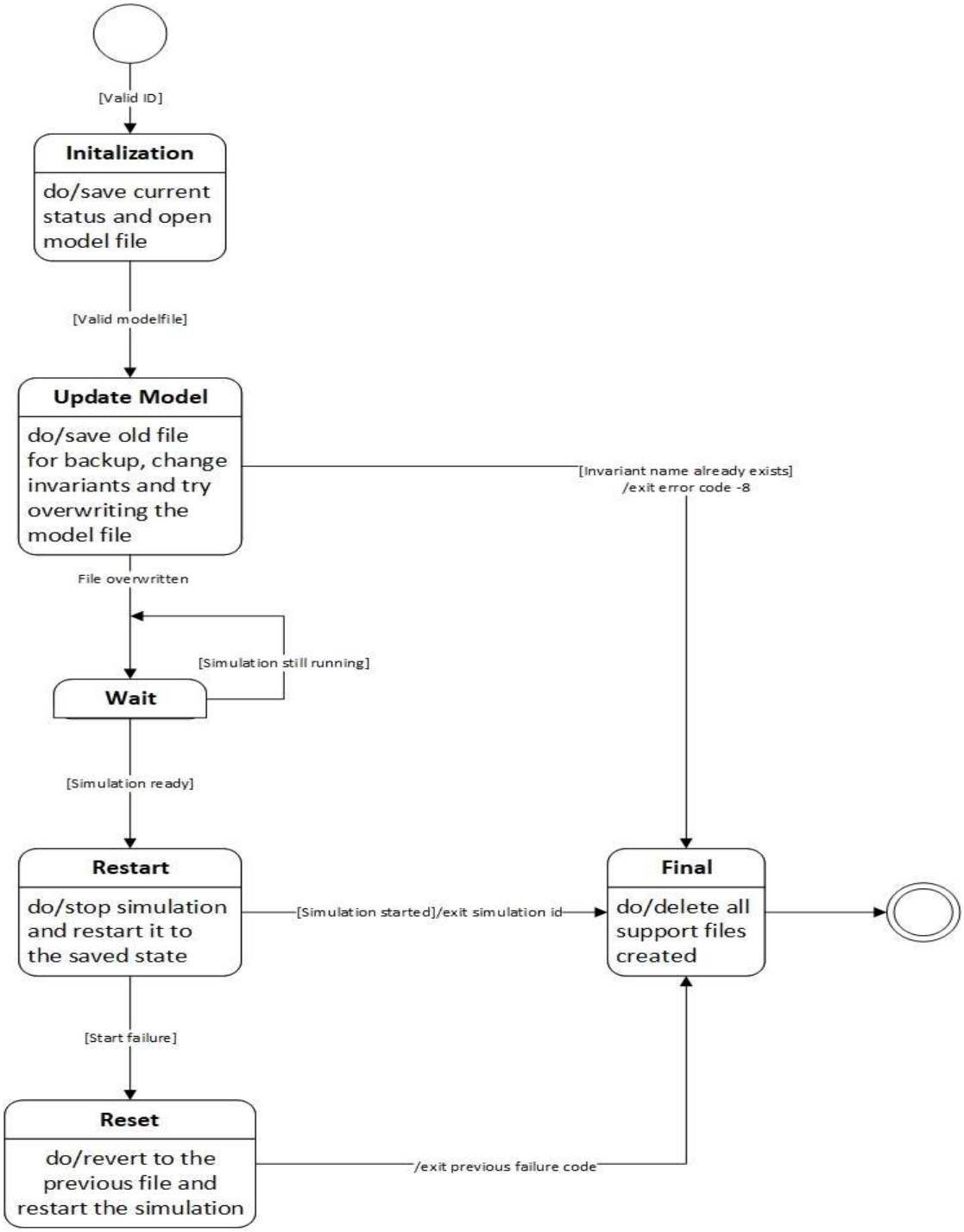

Figura 4 - Add, Update e Remove

2.12 Selectothersimulation

Breve descrizione: Permette di caricare una delle simulazioni che sono state istanziate.

Precondizioni: devono essere presenti simulazioni istanziate

Trigger: l'utente vuole caricare una delle simulazioni istanziate

Postcondizioni: viene restituita la lista degli invarianti del modello caricato

Processo standard:

1. viene passato in automatico l'id della simulazione
2. vengono caricati in una lista tutti gli invariant presenti nel file .asm
3. viene popolata una tabella suddividendo l'invariant nei 3 campi principali
4. viene mostrata a video la tabella

3. Architettura software e implementazione

In questa sezione, l'architettura del sistema è descritta nel dettaglio tramite UML 2.0 usando Microsoft Visio 2018.

3.1 Diagramma dei componenti

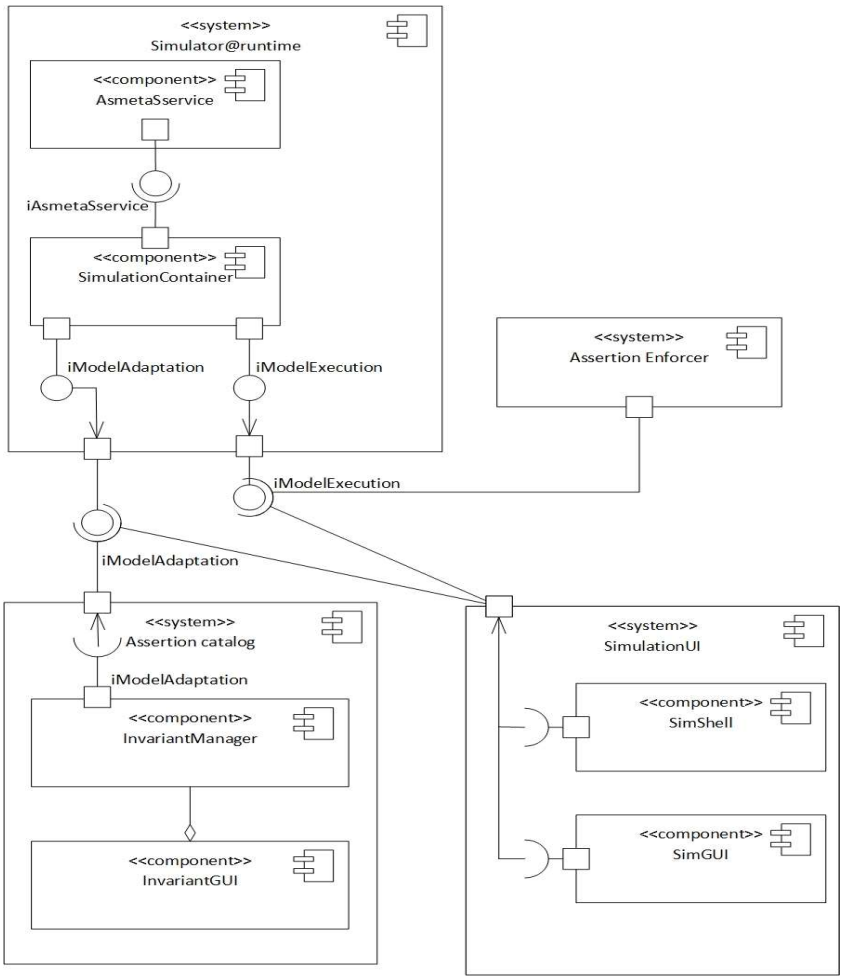

Figura 5 - Diagramma dei componenti

3.2 Commander

Tutto il sistema di simulazioni può essere comandato da linea di comando usando una serie di comandi a cui corrispondono le funzioni delle interfacce IModelExecution e IModelAdaptation.
Questo componente si occupa dell'interpretazione dei comandi inseribili da linea di comando dall'utente e della loro esecuzione.
Le funzioni utilizzabili da questo componente sono la maggior parte di quelle descritte nei casi d'uso (ad eccezione di Open AssertionCatalog e Select other simulation implementate dalle GUI di Simulation e AssertionCatalog) con l'aggiunta di init(per inizializzare il numero desiderato di slot per simulazioni), GetLoadedIDs per conoscere la lista di ID assegnati alle simulazioni caricate al momento e di un comando di help per conoscere i comandi e le loro opzioni.

3.2.1 Class Diagram

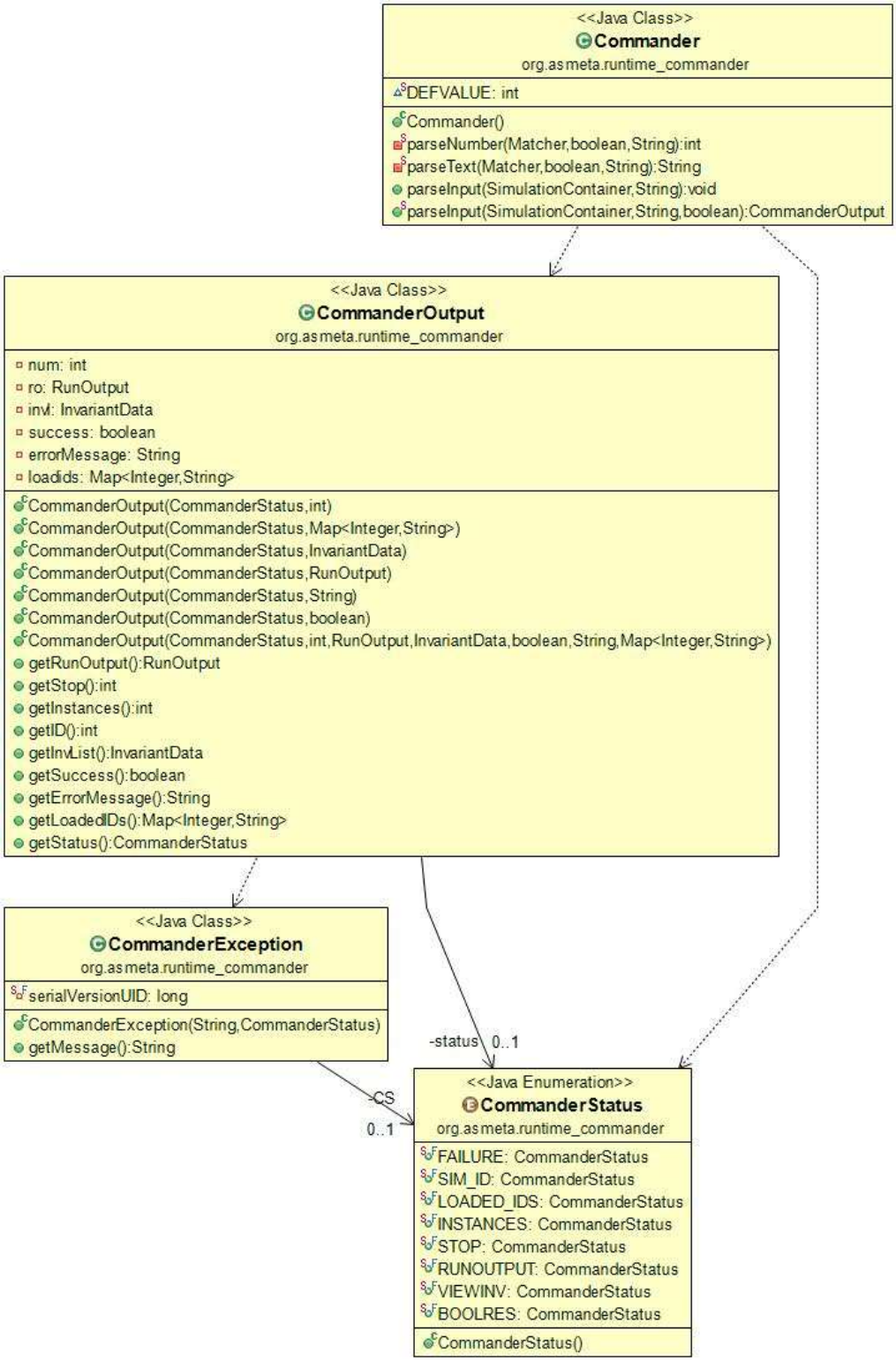

Figura 6 - Class Diagram Commander

3.2.2 Funzioni ed opzioni

Funzioni	Opzioni
Init	-n
StartExecution	-modelpath
StopExecution	-id
Runstep	-id -locationvalue <opzionale>
RunstepTimeout	-id -locationvalue <opzionale> -t
RununtilEmpty	-id -locationvalue <opzionale> -max
RununtilEmptyTimeout	-id -locationvalue <opzionale> -max <opzionale> -t
ViewListInvariant	-id
AddInvariant	-id -inv
RemoveInvariant	-id -inv
UpdateInvariant	-id -inv -invold
GetLoadedIDs	Nessuna
Help	Nessuna

-n:

Seguito da un numero intero, utilizzato solamente all'interno del comando init rappresenta gli slot di simulazione da instanziare.

-id:

Seguito da un numero intero, rappresenta l'ID assegnato alla simulazione (ottenuto dall'esecuzione di StartExecution).

-modelpath:

Seguito da una stringa alfanumerica delimitata da apici doppi (“), rappresenta il percorso completo della posizione del file di modello asm da cui far partire la simulazione.

-locationvalue:

Seguito da una lista chiave-valore nel formato *{chiave=valore,chiave=valore,ecc...}*, rappresenta la mappa di input da

mandare per l'esecuzione del modello, chiave è il nome della variabile monitorata.

-t:

Seguito da un numero intero, rappresenta il tempo in millisecondi da assegnare al timeout per le funzioni che lo supportano.

-max:

Seguito da un numero intero, rappresenta il massimo numero di cicli da effettuare se la main rule del modello non è ancora libera nei comandi RununtilEmpty.

-inv (-invold):

Seguito da una stringa alfanumerica delimitata da apici doppi (“), rappresenta il testo completo dell'invariante da aggiungere/rimuovere/aggiornare in base alla funzione chiamata (invold rappresenta il testo dell'invariant che si vuole aggiornare).

3.2.3 Sottocomponenti

Il componente principale è la classe Commander, implementato come oggetto static che offre la funzione *parseInput* che legge una stringa (comando inserito dall'utente), esegue il parsing del comando ed in caso questo fosse corretto viene eseguita la funzione associata sull'istanza del container specificato ritornando l'output attraverso l'oggetto *CommanderOutput*. Inoltre, è disponibile una modalità di debug (attivabile passando *true* nel parametro opzionale opzionale della funzione) che consente una visualizzazione nel dettaglio del processo di parsing e i valori trovati dei parametri in linea all'interno della console.
Accennato precedentemente, l'oggetto *CommanderOutput* è necessario per agglomerare tutti i diversi output possibili delle funzioni diverse, è composto da *CommanderStatus* che definisce la tipologia di output contenuto nell'oggetto e diversi get che permettono il suo recupero nel tipo corrispondente.
In caso venga richiesto un get di un tipo diverso da quello rappresentato nell'oggetto *CommanderOutput* viene lanciata l'eccezione *CommanderException* che contiene nel suo messaggio il *CommanderStatus* corretto.

3.2.4 Release

Per la release è stata predisposta la classe *SimShell* all'interno del package di user interface con un'implementazione di *ParseInput* con gestione corretta di *CommanderOutput* e un'istanza di Container dedicata. Tutti i risultati vengono stampati direttamente a console senza ulteriori elaborazioni. Il processo è arrestabile inserendo *qqq* come comando nella console.

3.3 AssertionCatalog

L'Assertion Catalog, ovvero la parte di programma che si occupa di fornire all'utente un'interfaccia grafica semplice per la modifica di invarianti di modelli Asm in runtime è stata implementata con un approccio model-view-controller. La classe frame InvariantGUI e le sue sottofinestre di dialogo di corredo servono al fine di offrire una vista degli invarianti contenuti nel modello e per l'aiuto alle funzionalità di aggiunta, modifica o cancellazione di questi. Tramite alcuni accorgimenti all'implementazione, questo componente è in grado di guidare l'utente per l'inserimento o modifica di un invariante, tramite la visualizzazione delle variabili del programma e l'inserimento schematico delle varie parti dell'invariante. La classe InvariantManager si occupa dell'esecuzione delle operazioni, funzionando da controller. All'interno di InvariantGUI è inoltre presente un riferimento di tipo IModelAdaptation, questo rappresenta il modello, ovvero la lista di operazioni possibili implementate dall'Assertion Catalog, su cui si sta eseguendo l'esecuzione dei modelli Asm.

3.3.1 Class Diagram

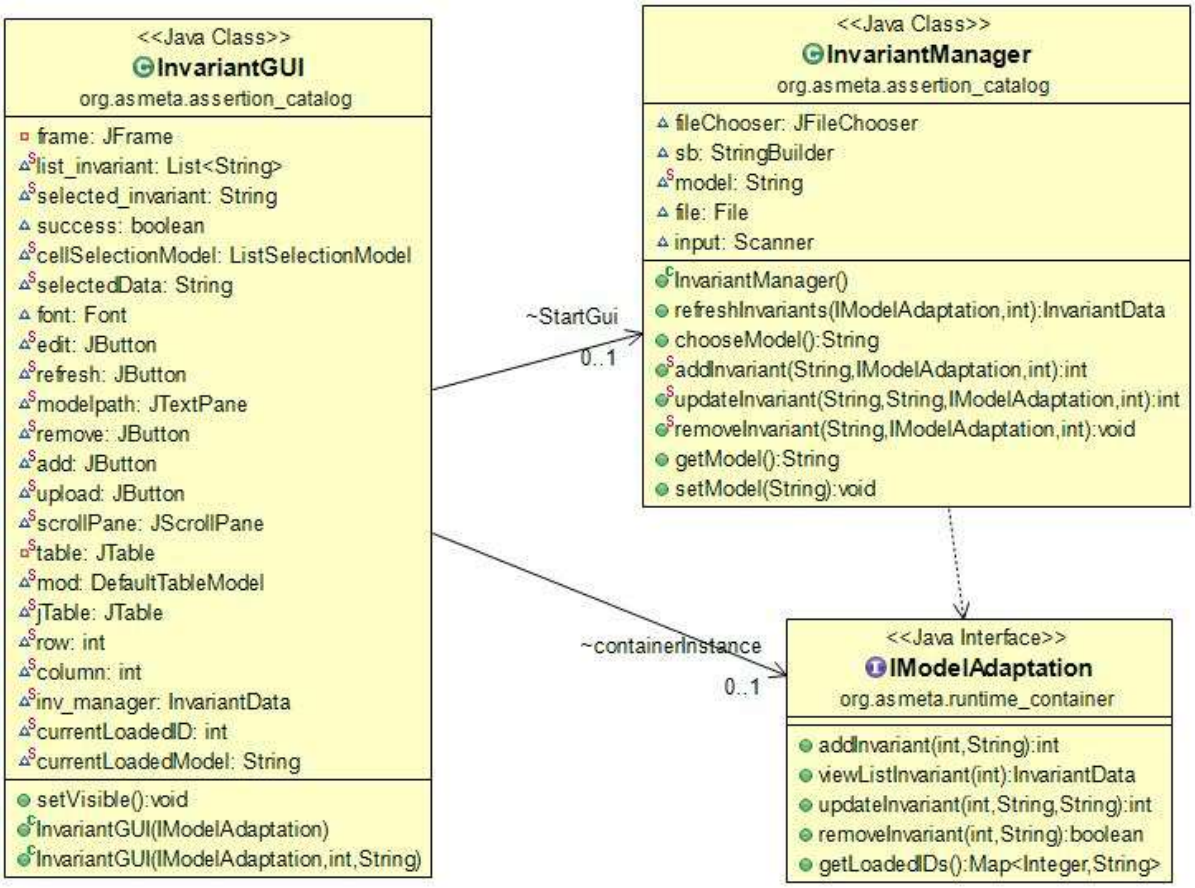

Figura 7 - Diagramma Assertion Catalog

3.3.2 Funzioni

Le funzionalità implementate dal componente sono tutte quelle relative alla gestione degli invarianti più la capacità di cambiare la simulazione caricata al momento senza dover riavviare.

Queste includono (all'interno dei UC):

-*ShowInvariants*

-*AddInvariant*

-*RemoveInvariant*

-*UpdateInvariant*

-*SelectOtherSimulation*

3.3.3 Release

Per una futura release standalone del componente si potrebbe ipotizzare l'esistenza di un servizio in background che esegue SimulationContainer a cui l'Assertion Catalog possa "agganciarsi", così da abilitare la gestione degli invarianti per le simulazioni in esecuzione.

3.4 SimGUI

SimGUI è l'ultimo componente che è stato implementato. Il componente offre tutte le operazioni offerte da SimulationContainer in forma di interfaccia grafica utente per facilitare il caricamento e l'esecuzione di steps.

3.4.1 Class Diagram

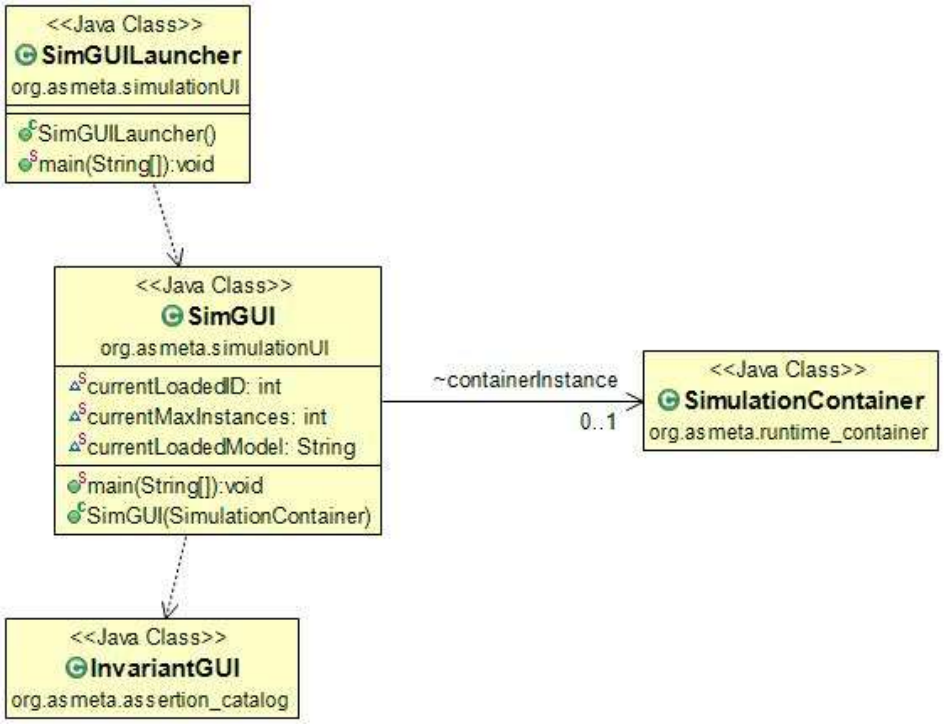

Figura 8 - Diagramma SimGui

3.4.2 Funzioni

SimGUI è in tutto e per tutto la rappresentazione su interfaccia grafica di SimulationContainer, permette l'inizializzazione degli slot di simulazione, il caricamento di un modello e il suo avvio, tutti i comandi di step (Runstep, RunUntilEmpty e le loro versioni con timeout), lo stop di una simulazione e, attraverso la possibilità di aprire l'AssertionCatalog, anche la gestione di invarianti.

Il caricamento o selezione di un modello è effettuato da una sottofinestra che mostra la lista delle simulazioni già caricate e permette il caricamento di ulteriori modelli attraverso un explorer di file integrato di Eclipse.

3.4.3 Release

All'interno del package è stata predisposta una classe che apre e carica un'istanza di SimulatorContainer nella GUI. Questo permette l'esportazione del progetto come file *.jar* in modo che sia eseguibile su una qualsiasi macchina che supporta java.

4. Aggiornamenti futuri

All'interno del codice di SimulationContainer sono state inoltre implementate le funzionalità per l'esecuzione corretta di transazioni. Una transazione consiste nell'esecuzione consecutiva di una serie di comandi (Runstep o RunUntilEmpty), in caso di esecuzione valida questa porterà ad un nuovo stato stabile. Nel caso venga raggiunto uno stato non valido durante l'esecuzione, la transazione deve essere annullata completamente ripristinando quindi lo stato della simulazione a come era prima di iniziare la sequenza di comandi. Queste funzioni non sono state finalizzate causa problemi di specifica non approfonditi.

Altre modifiche future riguardano:

- La gestione del ridimensionamento dell'interfaccia grafica
- La visualizzazione dello stato iniziale del modello che è appena stato caricato
- La gestione delle funzioni monitored parametrizzate n-arie

Elenco delle figure