Astrazione automatica tramite lo standard FMI per la costruzione di piattaforme virtuali

Simone Girardi - VR421147
Dipartimento di Informatica
Universitá Degli Studi Di Verona
Email: simone.girardi@studenti.univr.it

***Sommario*—Lo standard FMI 2.0 [1] pubblicato a Luglio 2014, fornisce gli strumenti che permettono, attraverso la definizione di un'interfaccia univoca, l'integrazione di modelli provenienti da strumenti di modellazione diversi. L'obbiettivo di questo progetto é comprendere e dimostrare le potenzialitá di questo standard, partendo da un insieme di modelli eterogenei precostruiti con diversi tool, integrandoli, ed ottenendo cosí una "virtual ciberfisical platform" simulabile.**

I. Introduzione

Lo standard FMI (Functional Mock-Up Interface) é un'interfaccia standard progettata per la modellizzazione e la simulazione di sistemi complessi, dove gli elementi individuali possono appartenere a un insieme eterogeneo di discipline dell'ingegneria, potendo anche questi essere stati sviluppati con degli strumenti indipendenti. L'obiettivo di questa interfaccia é quello di riuscire a rappresentare interamente i modelli (fisici, meccanici, elettrici, ecc.) usati per la simulazione tramite delle funzioni e strutture dati predefinite, fornendo una maggiore facilitá di interazione con gli strumenti di simulazione dei modelli, nonché tra diversi modelli in una co-simulazione.

Il numero di tool di simulazione che si basano su FMI sta crescendo rapidamente, grazie ad alcune loro caratteristiche, come la facilitá di utilizzo che conferiscono alle piattaforme di simulazioni, la facoltá di proteggere la proprietá intellettuale e di garantire licenze in modo molto facilitato ai simulatori sviluppati. Questo report vuole riassumere il lavoro svolto nell'integrazione di una serie di componenti eterogenei al fine di costruire una piattforma virtuale che comprende tutte le caratteristiche sopracitate.

II. Background

Lo standard FMI é composto di due parti principali: **FMI for Model Exchange** e **FMI for Co-Simulation**.

La prima parte dello standard FMI permette di creare modelli eseguibili che rappresentino interamente modelli semplici di discipline diverse, oppure sistemi composti dinamici e direttamente simulabili. I modelli cosí generati sono chiamati FMU (Functional Mock-Up Unit). Gli FMU implementano l'interfaccia definita in FMI for Model Exchange e possono essere eseguiti da qualsiasi ambiente di modellazione e simulazione; quest'ultimo puó, implementare a sua volta l'interfaccia, in quanto i modelli non contengono informazioni o file di configurazione relativi a un simulatore specifico.

L'interfaccia consiste delle seguenti parti:

- **Interfaccia del modello**: tutte le funzioni necessarie per la descrizione del modello vengono elaborate tramite l'esecuzione di funzioni standardizzate nel linguaggio C.
- **Schema di descrizione del modello**: illustra la struttura e il contenuto di un file XML (eXtensible Markup Language) generato dagli strumenti di modellazione.

La seconda parte dello standard (FMI for Co-Simulation) serve per riuscire ad accoppiare (coupling) due o piú modelli in un ambiente di co-simulazione. Lo standard definisce che in un ambiente di co-simulazione lo scambio di dati tra due sottosistemi avviene solamente in alcuni punti discreti di comunicazione (communication point). Nel periodo in mezzo a due punti di comunicazione i sottosistemi sono elaborati in modo indipendente da altri, ciascuno dal proprio risolutore, chiamato solver. L'algoritmo di controllo dello scambio di dati tra i sottosistemi viene chiamato Master, e si preoccupa della sincronizzazione della comunicazione tra i solver delle simulazioni, meglio noti come Slaves. Linterfaccia passa attraverso tutte le fasi del processo di simulazione, partendo da quella di impostazione e inizializzazione dei modelli, nella quale il master analizza le connessioni tra i vari modelli, carica in memoria gli FMU necessari, crea tutte le istanze e assegna i valori iniziali per ciascuna di esse. In seguito si procede con l'integrazione temporale, in cui ciascun modello viene simulato individualmente in intervalli di tempo e lo scambio di dati viene orchestrato dal master nei communication point definiti prima, accedendo a delle variabili o assegnandoli dei valori. In fine, vengono elaborati a posteriori i dati e visualizzati i risultati, ciascuno in modo indipendente per qualsiasi sottosistema e viene liberata la memoria usata dai FMU.

Nella versione FMI 2.0, entrambe le interfacce (FMI for Model Exchange e FMI for Co-Simulation) sono descritte nello stesso documento e sono state unificate in una. Con questa unione, un FMU implementa entrambe le interfacce simultaneamente, e nella descrizione del modello (in XML) esistono nuovi elementi ModelExchange e Co-Simulation che indicano quale interfaccia é stata implementata. In seguito é l'ambiente di simulazione a stabilire quale interfaccia usare, chiamando la rispettiva funzione di inizializzazione, *fmi2InstantiateModel* o *fmi2InstantiateSlave*. La figura 1 mostra lo schema di descrizione del modello per la versione 2.0.

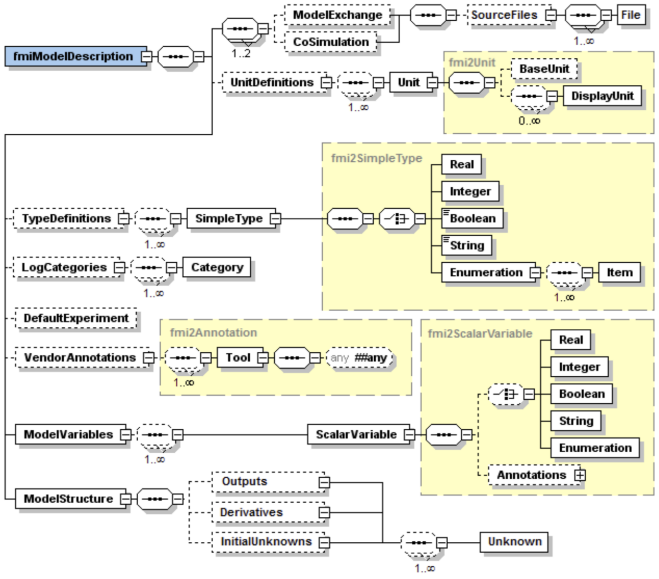

Figura 1: Schema di descrizione in FMI 2.0

III. Metodologia applicata

Il primo passo era volto a comprendere meglio gli aspetti coinvolti durante la creazione di un FMU, che in questo caso era dedicato a svolgere la sola funzionalitá di guadagno di un segnale. In secondo luogo sono state fatte delle operazioni di completamento sul master PyFMI al fine di rendere il tutto simulabile in un unica virtual platform.

A. FMU: GAIN

A partire dal file di descrizione XML del gain, esso risultava provvisto delle seguenti porte: *data_rdy*, *result_rdy* e *data*. É stata quindi aggiunta una porta di uscita *result* nel seguente modo:

```
1 <ScalarVariable causality="output" description="
      result" name="result" valueReference="1"
      variability="discrete">
2     <Integer />
3 </ScalarVariable>
```

In questo modo diventa possibile, una volta istanziato il modello, dereferenziare tale porta e salvare il risultato ottenuto dopo aver svolto la funzionalitá del gain. Facendo riferimento all'implementazione C++ del gain, invece, é stata aggiunta la rispettiva porta di result nel file header "fmuInterface.h" ed infine sono state completate le seguenti funzioni: *fmi2GetInteger*, *fmi2SetInteger* e *fmi2DoStep* presenti nel sorgente "fmuInterface.cc",
Nella prima e nella seconda sono stati aggiunti i seguenti *case* rispettivamente:

```
1 case 1L:
2     value[i] = comp->result;
3     break;
```

```
1 case 1L:
2     comp->result = value[i];
3     break;
```

in cui il valore del "case" rappresenta il value reference della porta in esame. Mentre per quanto riguarda la funzione *fmi2DoStep* é stata definita quella che rappresenta la funzionalitá che il gain dovrá svolgere durante lo step di simulazione:

```
1 // If data are ready, gain the data.
2 if(comp->data_rdy == fmi2True) {
3     comp->result = comp->data * GAIN_VALUE;
4     comp->result_rdy = fmi2True;
5 }
```

dove GAIN_VALUE = 10 implica che il segnale di uscita sará amplificato di 10 volte rispetto al valore di ingresso. Una volta compilati, tramite gli apposti makefile, sia i sorgenti C++ che il file xml, risulta quindi generato automaticamente il file gain.fmu. Infine é stato fatto un controllo di warning/errori lanciando il comando fmu check. simulando quindi il file .fmu con dei dati randomici. L'operazione di generazione degli FMU é stata ripetuta per tutti i moduli presenti nella cartella modules, e una volta generati tutti gli FMU essi sono stati spostati nella cartella fmus in cui é presente il nostro master (coordinator.py).

B. FMI: PyFMI

PyFMi é un package per Python 2.7 che permette di utilizzare gli FMU (Model Exchagne e Co-simulation). Esso, attraverso delle chiamate a funzione, é in grado di caricare, settare, leggere gli FMU e simularli. Nello specifico caso in esame, l'ambiente di PyFMI risulta giá pronto per la simulazione all'interno del file coordinator.py, a meno di alcune operazione di completamento di seguito descritte.

Aggiunto il caricamento del gain.fmu:

```
1 gain = load_fmu('./fmus/gain.fmu', log_level=
      NOTHING)
```

Aggiunta l'inizializzazione del gain:

```
1 gain.initialize()
```

Aggiunta l'invocazione dello step di simulazione del gain:

```
1 gain.do_step(current_t=time, step_size=
      digital_step)
```

IV. Risultati

Una volta che tutto risulta correttamente collegato all'interno dell'ambiente di PyFMI, i risultati ottenuti lanciando la simulazione tramite lo script coordinator.py sono rappresentati in figura 2 e 3. Dai risultati ottenuti é chiaramente visibile l'amplificazione del segnale data dal gain, che altrimenti arriverebbe ad un valore intorno a 30-35. Notiamo inoltre che il gain ha un comportamento molto fitto, perché va a computare solamente quando ci sono dei dati di input pronti, quando i dati non sono pronti il gain manda in uscita un guadagno pari a zero perché i dati non sono pronti.

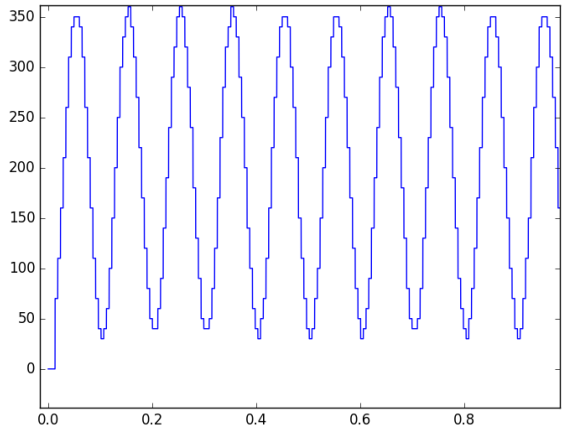

Figura 2: gain output signal

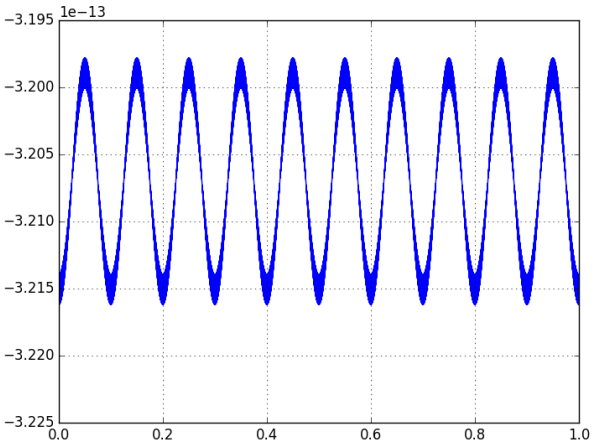

Figura 3: accelerometer output signal

V. Conclusioni

Grazie allo sviluppo di questo progetto si ha ora un'idea generale di come sia possibile aggregare modelli provenienti da tool di progettazione di diversa natura, e simulare il tutto all'interno di un unica piattaforma virtuale. Lo standard FMI risulta quindi essere un ottimo strumento per mettere in comunicazione progettisti che lavorano in diverse discipline con diversi strumenti all'interno del campo dei sistemi embedded e dei sistemi ciberfisici, e questo risulta particolarmente utile quando si vuole poter avere a disposizione una piattaforma composta da componenti a analogici e digitali simulabile prima di effettuare gli opportuni raffinamenti e sintetizzare quindi i vari componenti sintetizzabili.

Riferimenti bibliografici

[1] Daimler AG *et al.*, "Fmi 2.0," *Online, July*, 2014.