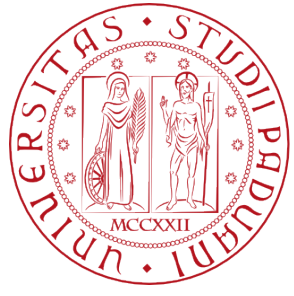
Linguaggi e strumenti per applicativi software di simulazione intensiva

Diego Casella, Fabio Marcuzzi
February 24, 2015



Università degli Studi di Padova

Contenuti

Introduzione

Appurate le complessitá di funzionamento della co-simulazione di un sistema controllato a microcontrollore, analizzeremo quest'oggi quali sono i linguaggi e gli strumenti idoneni per una co-simulazione intensiva ed amplieremo l'orizzonte delle applicazioni.

la GUI: Java, C++ o Python ?

Java:

- non si interfaccia direttamente a Python (c'é Jython ma non poteva importare i moduli scritti in C);
- non molto efficiente (sul server peró lo si potrebbe compilare!);
- librerie grafiche (Swing) e diverse altre: sará difficile tenerle aggiornate ?
- facile il reverse-engineering: con l'offuscamento ci sono problemi, ad es impossibile offuscare le chiamate a DLL, standard API (a meno di non shippare una JDK custom made), e l'interfaccia RMI tra server e client (deve essere comprensibile da entrambi)
- largamente insegnato nei corsi di laurea e facile da imparare;
- strumenti di sviluppo gratuiti e completi (Eclipse, NetBeans, ecc.)

la GUI: Java, C++ o Python ?

C++:

- riduzione di un layer di linguaggio, da *Java*→*C*→*Python* a *C++*→*Python*;
- efficienza
- esistenza di librerie opensource molto versatili come il framework Qt, multipiattaforma, che posseggono moduli per creare gui, comunicazioni IPC, networking, multithreading, parsing xml, gestione IO, oltre ad un core con containers e data structures ed una gestione facilitata della memoria
- reverse-engineering molto piú tedioso
- pochi studenti conoscono C++,
- strumenti di sviluppo gratuiti e completi come QtCreator/Eclipse/NetBeans/.., tool di debug/analisi consolidati come gdb/valgrind/callgrind

la GUI: Java, C++ o Python ?

Python:

- non molto efficiente, ma ben legato a C/C++;
- accessibilitá alle librerie C/C++ attraverso wrapper;
- facile il reverse-engineering;
- sempre piú diffuso tra gli studenti;
- strumenti di sviluppo gratuiti ma incompleti

la GUI: Java, C++ o Python ?

Questioni:

- Per usare C++ bisogna avere un'organizzazione piú grande, essere piú persone, di cui il gruppo "produzione" lavora in C++ e gli altri in Python/C (o Java/C/Python) ?
- la difficoltá del C++ rispetto a Java é un problema? Verrebbe da dire: meglio avere l'onere di JNA ma la facilitá del codice Java, piuttosto che dover scrivere sempre in C++ ... Oppure al giorno d'oggi non é cosí difficile, dato che gli smart pointers a che gestiscono la deallocazione della memoria in maniera automatica, paradigmi come *RAII*, i containers con una semantica simile a Java, e se poi passiamo ai container implementati in Qt, hanno un grado di affinitá ai containers Java molto alto ?

la GUI: Java, C++ o Python ?

(... segue) Questioni:

- Python é spesso visto adatto per fare script o applicazioni web-riented: la comunitá incontra difficoltá a mantenere la compatibilitá fra le varie versioni delle proprie release, e coi moduli sviluppati da terzi (vedi numpy) la situazione é ancora peggiore. Ad esempio, nella stessa distribuzione linux ci sono presenti piú versioni di Python installate, a seconda dell'applicazione che ne fa uso, e questo a causa della scarsa continuitá delle loro API;
- i costi dovuti all'aggiornamento alle API nuove (la community Python non sempre garantisce continuitá nelle loro release), oppure ai bachi di sicurezza che ci si tiene nel caso in cui non si fa l'aggiornamento alla nuova Python X.Y perché richiederebbe troppo tempo, potrebbero essere elevati.

Il modello di simulazione

- il modello di simulazione: dato che il kernel é ragionevolmente in C (o event. CUDA), il modello va bene in Java, Python, o cosa ?
- Come fatto positivo della GUI in C++ si puó spostare facilmente classi da C++ a Python; ad es., il mulabModel andrebbe bene in Python, ma con la GUI in Java non é proprio il caso ... oppure é meglio tenere Python relegato alla parte di componentistica poiché non "maturo" come Java né tantomeno come C++ che ormai sono trent'anni che esiste; per sviluppare modelli di simulazione complessi é necessario essere sicuri che il linguaggio sia solido e stabile ?

La concorrenza

- Ci sono vari motivi per cui una simulazione debba essere concorrente: o perché si vuole sfruttare un'accelerazione hardware, o perché é necessario far dialogare tra loro dei software che non possono essere fusi in unico processo.
- Allora, se muLab fosse in C++, come dialogherebbe con il plugin di mplabx ? Con dbus, esiste il modulo QtDbus pronto all'uso. Usando Java, ogni chiamata RMI implica una de/serializzazione di oggetti Java, ed un successivo passaggio fra due processi. Tale aspetto avverrebbe anche in dbus ovviamente, solo che in Java c'é la jvm di mezzo come aggravante.

La concorrenza

- Con dbus, il plugin μLab per MPLAB-X diventa un ibrido Java/C++, nel quale solo le parti che chiamano l'mdbcore rimarrebbero in Java mentre il resto, tipo adapters, ecc., in C++. Peró cosí buttiamo via il layer JNA da una parte e lo ricreiamo dall'altra: vale la pena di toccare una parte molto delicata per le prestazioni (molte chiamate di tipo *StepInstr()*) ?

Sviluppare piú codici diversi

Creare un applicativo specifico per ogni applicazione o evolvere muLab in un “framework” ? Si dovrebbe poter:

- costruire agevolmente modelli di simulazione: l’uso di Python e del linguaggio grafico ad elementi interconnessi va bene;
- poter includere librerie numeriche esterne per la soluzione dei modelli: Python va bene, basta fare i dovuti `import` (con i quali per un utente disinvolto pu fare danni);
- definire agevolmente gli esperimenti: il modello completamente parametrico e le *TestSequence* vanno bene;

Sviluppare piú codici diversi

Creare un applicativo specifico per ogni applicazione o evolvere muLab in un “framework” ? Si dovrebbe poter:

- poter fare facilmente il post-processing dei dati della simulazione: l’uso di Python e l’integrazione grafica tra dati della simulazione e dati del post-processing vanno bene;
- poter costruire una GUI personalizzata: la mulab-PythonComponentLibrary ed il fatto di costruire componenti da LaTeX costituiscono di fatto una GUI virtuale piú generale.

Sviluppare piú codici diversi

- caso di studio: software di analisi della corrosione nascosta mediante termografie all'infrarosso (controllo non-distruttivo).

Q&A