JACK: Un framework pentru algeba proceselor implementat în Java

Mali Imre Gergely

November 2019

1 Algebra proceselor și Ingineria Sistemelor Soft

Algebra proceselor e o abordare de formalizare a sistemelor concurente, și se definește printr-un set de nume care reprezintă la nivel abstract canelele de comunicare. În afară de asta, pentru a defini o alegbră de procese, mai avem nevoie de operatori, pentru a forma procese noi din procese existente. Operatorii de bază definesc compunerea paralelă sau secvențială a proceselor, comunicarea între procese , semantica de reducere, recursivitate și replicare a proceselor.

Framework-ul JACK(*Java Arhitecture for CSP kernel*) este o abordare de a construi procese concurente prin intermediul unui framework orientat obiect, folosind concepte preluate direct din algebra proceselor. Limbajul Java a fost ales de către autor pentru că pe de o parte, este un limbaj popular în industrie, pe de altă parte pune la dispoziție un suport simplu dar extensiv de concurență.

Deși mai multe abordări similare există, JACK se consideră inedit pentru că implementează un set de operatori din algebra proceselor mult mai larg, decât orice alt framework existent, pe de altă parte pune un accent la fel de mare și pe proiectare, extensibilitate și alte concepte importante de ingineria sistemelor soft.

De exemplu, frameworkul JCSP este o librărie care reprezintă o implementare a limbajului CSP. Introdus de *Tony Hoare*, CSP(*Communicating Sequential Processes*) e un limbaj formal pentru a descrie procese concurente. JCSP modelează procese concurente prin thread-uri Java, și reflectă implementarea de CSP din limbajul occam. Un alt exemplu este CTJ(*Concurrent Threads in Java*) care la rândul lui este tot o implenentare care se bazează pe occam. CTJ pune un accent mai mare pe procese care rulează în paralel, și împărtășesc date și care trebuie planificate, adică aduce noi abstracții, precum critical section, context switcher, scheduler, etc. Aceste două librării implementează numai cei mai importanți operatori din algebra CSP, și nu sunt menite să fie extensibile. În schimb, JACK e o implementare de CSP care e construită pe principiul extensibilității și dispune o colecție mai largă de operatori. Datorită extensibilității, JACK e ușor de modificat în așa fel încât să fie utilizat cu alte algebre de procese.

Pentru a construi un framework ca și JACK, următoarele cerințe trebuie respectate:

- rigoare și formalitate
- separarea conceptelor
- modularitate
- abstracție
- anticiparea schimbărilor
- generalitate
- incrementalitate (?)

Aceste cerințe au fost enumerate pentru că fiecare servește un scop în proiectarea frameworkului JACK, și deciziile de implementare au fost luate bazându-se pe acestea. Arhitectura JACK este unul de nivele multiple(layered arhitecture). Decizia de arhitectură, și perspectiva cerințelor enumerate determină numărul de nivele, rolul și complexitatea acestora, tehnicile de modelare, șabloanele de proiectare care sunt în concordanță cu cerințele funcționale și non-funcționale a aplicației. Această afirmație este adevărată pentru sisteme soft în general, însă, implementarea JACK pune un accent

deosebit pe acestea, cerințele funcționale ale acesteia fiind diferite de cele des-întâlnite în industrie. Astfel, aspectele funcționale sunt semantica operatorilor, care trebuie separate de aspectele non-funcționale(cum se implementează de fapt procesele în limbajul respectiv, threading, locking, etc.).

2 Arhitectura JACK

Cel mai de jos nivel din arhitectura JACK folosește un framework numit DASCO(*Development of Distributed Application with Separation and Composition of Concerns*), framework construit pe aceleași principii spre care și JACK tinde. Deși, DASCO nu a fost scris în Java, JACK folosește o variantă redefinită și reimplementată în Java, numită JDASCO. Acest nivel tratează problemele clasice în programarea orientată obiect și programarea paralelă, precum lock inheritance. La rândul lui, nivelul JDASCO are trei subnivele. Cel mai de jos dintrea acestea este nivelul *Concern Layer* care se ocupă de primitivele de sincronizare, precum lockuri, mutexuri, tranzacții, threaduri, etc. Cel de al doilea nivel combină aceste aspecte și oferă concurență cu sincronizare, mecanisme de recuperare, etc. Ultimul nivel de JDASCO combină aceste concepte pentru a oferi funcționalitățile necesare de a construi semantica operatorilor CSP.

Nivelul menționat anterior oferă posibilitatea de a construi semantica operatorilor dintr-o algebră de procese. Aceasta poate fi orice algebră(punctul unde se vede cel mai bine extensibilitatea lui JACK), însă în cazul acesta, algebra aleasă a fost CSP, și nivelul următor servește definirea operatorilor din CSP. Acest layer este numit JASC(*Java Architecture with CSP Semantics*) și folosește direct layerul JDASCO. Acest layer iară are două layeruri, anume, layerul de integrare și layerul de procese. Pe lângă descrierile de procese date de către CSP(sau algebra de procese folosită) este important posibilitatea de a defini procese de către utilizator, implementând interfața Behavior. Pe de altă parte, JACK oferă o abordare de execuție simbolică pentru execuția proceselor, abordare prin care tratarea tipurilor de date infinite devine posibil.

Ultimul layer este layerul care interacționează direct cu utilizatorul. În afară de acesta, nu oferă nicio funcționalitate în plus.

3 Exemple

Pentru a demonstra utilizabilitatea frameworkului, autorul exemplifică protocolul *stop-and-waint* între două procese. În primul rând, protocolul e formalizat în termeni CSP.Canalele principale de comunicare sunt reprezentate prin *in*, *out* și *mid*. Operatorii principali în CSP reprezintă comunicarea paralelă între procesele S, și R, care la rândul or sunt procesele emițător și receptor(producer/consumer). Implementarea în Java trebuie să țină cont de ordinea topologică a entităților formale enumerate. Astfel, în primul rând construiește canalele de comunicare, după care construiește procesele, S și R. Pentru posibilitatea apelurilor recursive se mai construiesc subprocese auxiliare.

4 Concluzii

JACK poate fi privit ca un client de JDASCO, care la rândul lui este un framework populare pentru modelarea sistemelor concurente, iar abstracțiile definide de către JACK folosesc principiile de proiectare implementate de JDASCO, care sunt concurența, sincronizarea, recuperarea și combinarea acestora.

Algebrele de procese sunt abstracții care prin natura lor, necesită modele recursive și compuse. Modelarea formală a frameworkului JACK folosește tehnici de OOP care rezultă printr-un framework care pe de o parte respectă cerințele implicite ale algebrelor de procese, pe de altă parte totuși rămâne robust, extensibil. Datorită proiectării extensibile, JACK oferă și evaluare de structuri de date de tip *on-demand*, caracteristică datorită căruia JACK poate să interpreteze și structuri de date infinite.

În concluzie, JACK se poate considera o contribuție semnificativă în cadrul modelelării sistemelor concurente, pentru că pe de o parte respectă toate cerințele necesare care vin din partea abstracțiilor de algebre de procese, pe de altă parte cerințele acestea sunt respectate în așa fel încât frameworkul rămâne extensibil, robust, relativ ușor de schimbat și nu în ultimul rând oferă și posibilitatea tratării structurilor de date infinite.