Facultatea de Matematică
și Informatică

Modelarea Formală a Proceselor Concurente

SePi: un limbaj concurent bazat pe tipuri de sesiune rafinate

Selegean Victor

Departamentul de Informatică, Universitatea Babeș-Bolyai
Strada M. Kogălniceanu 1, 400084, Cluj-Napoca, România
E-mail: victor.v.selegean@gmail.com,
victor.selegean@stud.ubbcluj.ro

Rezumat

Lucrarea aceasta prezintă ideile principale ale unei lucrări publicate în 2014 de către Juliana Franco și Vasco Thudichum Vasconcelos, afiliați Universității din Lisabona, Facultatea de Științe, Portugalia. Articolul, denumit "A concurrent programming language with refined session types", face prezentarea unui limbaj de programare numit SePi, creat cu scop demonstrativ pentru a ilustra beneficiile tipurilor de sesiune pentru programarea concurentă.

1. Introducere

Calculul concurent este o paradigmă de programare ce reprezintă o piatră de temelie pentru proiectarea sistemelor informatice moderne. Aceasta se bazează pe conceptul de proces, o entitate capabilă să execute instrucțiuni și să comunice cu alte procese. Paradigma de calcul concurent descrie și guvernează interacțiunea mai multor procese computaționale care cooperează sau interferează unul cu altul, concurând pentru resursele comune (memorie, timpul de procesor, etc.).

Interacțiunea dintre procese este modelată, în general, printr-o zonă comună de memorie sau transmițând mesaje prin canale de comunicare [5]. Atât modelul memoriei partajate, cât și cel al transmiterii de mesaje implică mecanisme de coordonare și, uneori, de sincronizare, între procesele care iau parte la ele. Lipsa acestora sau implementarea lor eronată este sursa unor erori dificil de identificat și replicat. De aceea, programarea concurentă și implementarea mecanismelor de coordonare între procese sunt recunoscute drept fiind dificile. Pornind [1] cu articolul lui Edsger Dijkstra din 1965, *"Solution of a Problem in Concurrent Programming Control"*, analiza sistematică a unor probleme precum excluderea mutuală, condițiile de cursă și interblocarea a luat amploare. În urma acestui fapt, au fost dezvoltate atât soluții practice, cât și modele formale pentru înțelegerea comportamentului sistemelor concurente.

Deoarece multe defecte de concurență depind de intercalări specifice ale execuției, ele sunt dificil de identificat prin testare și pot rămâne nedetectate până în producție. În domeniile de activitate care mizează pe stabilitatea și corectitudinea sistemelor informatice sunt depuse eforturi considerabile pentru identificarea și tratarea erorilor cât

mai devreme posibil. Acest fapt a motivat dezvoltarea unor mecanisme de verificare statică, capabile să identifice erori de comunicare și sincronizare încă din etapa de compilare.

În cele ce urmează, se va prezenta conținutul articolului *"A concurrent programming language with refined session types"*[2], publicat în 2014, care prezintă limbajul de programare SePi. Acesta este construit specific pentru calcul concurent, bazat pe comunicarea prin schimb de mesaje între procese. Această paradigmă este prezentă și în spatele altor limbaje mai populare, așa cum e Go. În mod particular, SePi aduce nou o implementare a conceptului de *"tipuri de sesiune"*, o metodă de a asigura corectitudinea comunicării prin mesaje încă din etapa de compilare.

2. Bază teoretică

2.1. Calculul-π

Limbajul SePi este bazat pe calculul-π [8], un sistem formal pentru modelarea proceselor concurente. Deși înrudit conceptual cu calculul-λ [4], calculul-π este orientat spre modelarea comunicării și mobilității canalelor, nu a funcțiilor și aplicării acestora. Procesele sunt notate prin litere mari (P, Q, R), iar numele canalelor și variabilele prin litere mici (a, b, c, x, y, z). Procesul inactiv, finalizat, este notat cu simbolul 0.

Procesele pot fi compuse pentru a forma sisteme mai complexe, utilizând următoarele construcții de bază:

- **Compunerea paralelă ($P \mid Q$):** procesele P, Q sunt active simultan și pot comunica între ele.
- **Restricția ($(\nu a)\, P$):** creează un canal sau o variabilă privată a accesibilă doar în procesul P.
- **Replicarea ($!P$):** asigură disponibilitatea constantă a procesului P.
- **Comunicarea:** un proces poate emite o valoare pe un canal ($\overline{a}\langle x\rangle$) sau o poate recepționa ($a(x)$).

Un exemplu simplu preluat de pe articolul despre calculul-π de pe website-ul nLab [8] este reprezentarea unui proces de tip server care acceptă numere de la mai mulți clienți și le transmite înapoi numărul incrementat și un client care trimite numărul și citește rezultatul. În calculul-π, procesul de server ar fi reprezentat astfel:

$$S = !incr(a, x).\overline{a}\langle x + 1\rangle \tag{1}$$

Serverul S va citi de pe canalul *incr* două valori, a și x. Prima reprezintă un canal de comunicare, iar a doua o variabilă. După citirea lor, S va transmite pe canalul a valoarea $x + 1$. Simbolul ! indică faptul că S este capabil să răspundă mai multor clienți simultan.

$$C = (\nu a)(\overline{incr}\langle a, 4\rangle.a(y)) \tag{2}$$

Clientul C va crea un propriu canal de comunicare a pe care îl va transmite alături de valoarea 4 canalului *incr*. După aceea, C va citi răspunsul de pe canalul a în variabila y.

Întregul sistem poate fi reprezentat cu expresia $S|C$, indicând execuția celor două procese în paralel.

2.2. Tipuri de Sesiune

2.2.1. Intuiție informală

Această secțiune va prezenta, mai întâi, o intuiție informală pentru conceptul tipurilor de sesiune.

În forma sa inițială, calculul-π nu este tipizat. Alternative tipizate ale acestuia există, dar discuția lor depășește scopul acestei lucrări.

Exemplele 1 și 2 presupun că mesajele transmise pe canalul *incr* vor avea tipul informal $(canal, nr)$. La rândul său, canalul a va accepta mesaje de tip (nr). Canalul a va avea, deci, două capete. Unul va citi valori de tip (nr), iar altul va scrie valori de tip (nr). Este posibilă construcția unui tip mai restrictiv pentru canalul a. Acest tip se va asigura că valorile transmise prin a se conformează așteptărilor proceselor care îl folosesc. Acest tip poate fi notat $canal(nr)$, iar capetele sale, cel de citit și cel de scris, pot avea tipul $read(nr)$ și, respectiv, $write(nr)$. La rândul lui, canalul *incr* ar putea fi descris prin tipul $canal(canal(nr), nr)$, sau, dacă se dorește o restrângere asupra capetelor canalului a prin care S poate comunica, $canal(write(nr), nr)$. Privit din perspectiva procesului S, *incr* va avea tipul $read(write(nr), nr)$, iar, din perspectiva lui C, $write(write(nr), nr)$.

Tipurile construite până acum sunt utile, dar nu pot fi folosite pentru cazurile în care interacțiunea dintre două procese presupune date mai complexe. Putem presupune un proces de tip server S' care primește un șir x de lungime arbitrară n de octeți și returnează numărul de biți egali cu 1 din x. Asemenea exemplului anterior, S' ar trebui să primească din partea posibililor clienți, C', un capăt de canal de comunicare pe care să posteze rezultatul.

$$S' = !one\_count(a, n).one\_count(x).\overline{a}\langle one\_count\_x\rangle \tag{3}$$
$$C' = (va)(\overline{one\_count}\langle a, 5\rangle.\overline{one\_count}\langle 11011_2\rangle.a(y)) \tag{4}$$
$$S'|C' \tag{5}$$

Exemplul anterior ilustrează o problemă a acestui model: canalul de comunicare *one_count* trebuie să transmită informațiile de la C' la S' în două tranșe distincte, mai întâi lungimea mesajului, n, apoi mesajul în sine, x. Pentru tipizarea acestui protocol, este necesară introducerea unui tip de canal care acceptă, mai întâi un număr, apoi un mesaj. În aceeași manieră informală ca înainte, acest tip va fi notat cu "$canal(canal(nr), nr) \rightarrow canal(str)$", unde "$\rightarrow$" reprezintă o relație de precondiție necesară între operanzi.

2.2.2. Definire formală

Secțiunea următoare este bazată pe cursul "CS 242: Session Types" al Universității Stanford [12].

Un tip de sesiune descrie un protocol de comunicare sub forma unei succesiuni tipizate de operații. Sintaxa tipurilor de sesiune poate fi exprimată prin următoarea gramatică formală:

$$\begin{aligned}
T := {} & !\tau.T && \text{\# transmiterea unei valori de tip } \tau \text{ urmată de continuarea } T \\
| {} & ?\tau.T && \text{\# recepționarea unei valori de tip } \tau \text{ urmată de continuarea } T \\
| {} & +\{\ell_i : T_i\}_{i\in I} && \text{\# o alegere internă (procesul alege o cale de execuție)} \\
| {} & \&\{\ell_i : T_i\}_{i\in I} && \text{\# o ofertă externă (procesul așteaptă alegerea)} \\
| {} & end && \text{\# încheierea sesiunii}
\end{aligned}$$

Protocolul serverului *one_count* poate fi exprimat prin tipul de sesiune "$T_S = ?nr.?str.!nr.end$", iar clientul va avea tipul dual "$T_C = !nr.!str.?nr.end$".

2.2.3. Rafinarea tipurilor de sesiune

Tipizarea protocoalelor prin tipuri de sesiune poate fi extinsă și mai mult prin asocierea unor predicate tipurilor componente. În exemplul 2.2.2 poate fi impusă condiția ca primul număr primit să fie unul pozitiv. În mod similar, clientul poate presupune că rezultatul returnat de server este, la rândul lui, un întreg pozitiv. Tipurile de sesiune rafinate pentru protocolul *one_count* pot fi notate astfel:

$$T_S = ?nr\ \{x \in \mathbb{N}^+\}\ .?str\ \{s \in BinarySequence \mid length(s) = x\}$$
$$.!nr\ \{y \in \mathbb{N}\}.end$$

$$T_C = !nr\ \{x \in \mathbb{N}^+\}\ .!str\ \{s \in BinarySequence \mid length(s) = x\}$$
$$.?nr\ \{y \in \mathbb{N}\}.end$$

3. SePi

SePi este un limbaj de programare specializat pentru calculul concurent. Pornind de la conceptele de bază asociate calculului-π și tipurilor de sesiune rafinate, o serie de noi concepte derivate sunt introduse ca parte a limbajului.

3.1. Implementarea tipruilor de sesiune

Exemplul următor prezintă traducerea implementarea proceselor S' și C' din exemplul 5.

```
type OneCountServer = ?integer.?string.!integer.end
new S C = OneCountServer
// C este automat de tipul !integer.!string.?integer.end
// S and C sunt executate concurent:
S?length.S?bitSequence.S!count | C!5.C!"01010".C?count
```

3.2. Rafinarea tipurilor de sesiune

Rafinarea tipurilor extinde sistemul de tipuri cu predicate logice declarate prin cuvântul-cheie `assume` asociate valorilor transmise. Predicatele sunt apoi consumate prin cuvântul-cheie `assert`. Pentru a ilustra acest concept se va folosi un exemplu din articolul original [2].

Fie un proces de tip server care citește un nume de card `ccard` și un număr `amount`, după care execută o plată de valoarea `amount` din contul asociat cardului `ccard`. În același timp, un proces de tip client trimite serverului parametrii `ccard` și `amount`.

```
new B, C = !string.!integer.end
B?ccard.B?amount.charge(ccard, amount) | C!"12345".C!500
```

Codul din exemplul anterior nu garantează faptul că procesul băncii va trata corect pasul de execuție a plății, `charge(ccard, amount)`. Astfel, actori rău-voitori ar putea înlocui `charge(ccard, amount)` cu `charge(ccard, amount + 100)`, sau chiar `charge(ccard, amount).charge(ccard, amount)`. Folosind tipurile de sesiune discutate anterior, procesul de client poate acorda permisiunea explicită pentru ca această operație să fie executată o singură dată folosind exact valorile oferite de client.

```
B?ccard.B?amount.assert can_charge(ccard, amount).charge(ccard, amount)
| (assume can_charge("12345", 500) | C! "12345".C!500.end)
```

Atunci când procesul server ajunge la secvența `assert can_charge(ccard, amount)`, predicatul generat de procesul client este consumat. Pentru a repeta operațiunea, serverul trebuie să primească o nouă permisiune. Mecanismul de aserțiuni și asumpții conferă limbajului siguranță fără cost operațional în timpul execuției. Aceste predicate sunt asociate sistemului de tipuri și verificate de typechecker la pasul de compilare. Un server care omite aserțiunea `assert can_charge(ccard, amount)`, atunci când clientul a declarat asumpția duală, `assume can_charge(ccard, amount)`, nu va putea trece de typechecker.

3.3. Operatorul de alegere

Ramificarea căilor de execuție are loc în SePi prin intermediul operatorilor + și &. Acești operatori vor fi ilustrați folosind un exemplu din documentația oficială[10]. Exemplul constă într-un proces de tip server care poate executa două funcționalități. Prima este citirea a două numere întregi și returnarea celui mai mare (funcționalitatea `max`), iar a doua este citirea unui singur număr și returnarea unui boolean care spune procesului client dacă numărul este sau nu egal cu 0.

```
new server  client : &{
            max: ?integer.integer.!integer.end,
            isZero: ?integer.boolean.end
}
```

Implementarea serverului constă într-o tratare a fiecărui caz posibil, iar cea a clientului în alegerea uneia dintre opțiuni:

```
case server of      max -> server?x. server?y.
                           if x>y then server!x else server!y
                    isZero -> server?x.server!(x==0)
| client select     isZero.client!5.client?b.prinBooleanLn!b.
// Sau (exclusiv)
| client select     max.client!5.client!6.client?n.prinIntegerLn!n.
```

3.4. Procese recursive

Până în acest punct, procesele de tip server prezentate nu au avut capacitatea de a răspune mai multor cerințe. Serverul din secțiunea anterioară ajunge la `end` după o singură interacțiune cu clientul. Un server ar trebui să aibă capacitatea de a lua parte la o nouă interacțiune în urma rezolvării celei curente. Pentru acest scop, SePi folosește recursivitatea, introdusă prin cuvântul-cheie `rec`. Exemplul următor reprezintă extinderea tipruilor din secțiunea 3.3 prin utilizarea recursiei.

```
new client : rec   a. &{ // "a" este numele prin care procesul se va referi la sine
                   max: !integer.!integer.?integer.a,
                   isZero: !integer.?boolean.a,
                   quit: end
}
```

Acest tip se poate prescurta cu ajutorul cuvântului-cheie `type`:

```
type Client =   &{
                max: !integer.!integer.?integer.Client,
                isZero: !integer.?boolean.Client
                quit: end }
```

Mecanismul de verificare a tipurilor se asigură de faptul că fiecare iteraţie respectă protocolul, împiedicând erori de ambele părți.

Recusrivitatea trebuie să apară atât în definirea tipului, cât și în definirea procesului. Pentru a defini un proces recursiv este folosit cuvântul-cheie `def`. Exemplul următor se bazează pe tipul `Server` dualul tipului `Client` definit anterior.

```
def serverLoop s: Server =  case s of
                              max ->
                                s?x. s?y.if x > y then s!x else s!y.serverLoop!s
                              isZero -> s?x.s!(x==0).serverLoop!s
                              quit -> end
new server client : Server
serverLoop!server       |   client select max.
                              client!5.client!6.client?n.prinIntegerLn!n.
                            client select isZero.
                              client!9.client?b.prinBooleanLn!b
                            client select quit
```

3.5. Sincronizarea

Adesea, este nevoie de servere care răspund mai multor clienți simultan. Pentru a obține asemenea comportament, este necesară introducerea unei noua pereche de termeni, `lin` și `un`. Aplicat unui canal de comunicare, `lin` va îl marca drept liniar. Canalele liniare permit un singur cititor și un singur scriitor în orice moment. În practică, majoritatea canalelor sunt folosite pentru interacțiuni liniare, deci orice canal este, din oficiu, liniar, iar cuvântul-cheie `lin` este opțional. Canalele `lin.!integer.?boolean.end` și `!integer.?boolean.end` sunt complet echivalente din punctul de vedere al limbajului.

Spre deosebire de `lin`, `un`, scurt de la `unrestricted`, trebuie menționat explicit mereu. Tipurile nerestricționate iau mereu forma "`rec a.un!Body.a`" sau "`rec a.un?T.a`", unde `rec` poate fi dedus din definiția recursivă și, deci, nu este obligatoriu. Acest șablon este de ajuns de comun încât creatorii SePi au introdus o notație prescurtată, anume "`*!Body`", respectiv "`*?Body`".

Cuvintele-cheie `lin` și `un` pot fi combinate pentru a asigura transmiterea corectă a tipurilor compuse de date chiar și în contextul unui canal nerestricționat. Acest mecanism funcționează prin crearea unui canal nerestricționat și folosirea acestuia pentru a transmite un canal liniar. În sintaxa standard SePi, acest comportament se poate exprima într-un mod compact folosind sintaxa de mai jos:

```
type MathSession = + {
  max: !integer.!integer.?integer.end,
  isZero: !integer.?boolean.end
}
type MathServer = *?MathSession
new client server : MathServer
// "server" crează două canale, îl păstrează pe primul sub numele
// de "s" și îl trimite pe celălalt, anonim, clientului:
def serverProc () = server!(new s : dualof MathSession).{
     serverProc!() | // apel recursiv
     case s of
       max -> s?x. s?y. if x > y then s!x else s!y
       isZero -> s?x. s!x==0
}

// Lansarea serverului și a 3 clienți paraleli
serverProc!() |
// Fiecare client primește un capăt de tip MathSession de la server și
// îl denumește "c":
client?c. c select isZero. c!-7. c?b. printBooleanLn!b |
client?c. c select max. c!3. c!9. c?n. printIntegerLn!n |
client?c. c select isZero. c!0. c?b. printBooleanLn!b
```

4. Concluzii

În raportul de față a fost prezentată o vedere de ansamblu asupra unei metode de modelare pentru procese concurente, calculul-π și implementarea unor concepte din acesta într-un limbaj de programare modern, SePi. Au fost discutate metodele idiomatice pentru crearea unor procese simple în acest limbaj.

Din 2014, anul publicării articolului care a stat la baza raportului acesta, proiectul SePi pare că nu este întreținut. Documentația oficială, care a stat la baza exemplelor de cod din raport, obișnuia să ofere un interpretor online pentru a vedea exemplele de cod în execuție. Acum, în decembrie 2025, acest interpretor nu mai funcționează. Un interpretor pentru SePi există în continuare ca un plug-in pentru Eclipse [10].

Totuși, există încă un interes pentru SePi ca resursă didactică, fiind discutat în universități precum Imperial College London [6] și University of Edinburgh [11].

Tipurile de sesiune sunt încă văzute drept un subiect abrupt de care mulți programatori se feresc în practică. Există astăzi, cel puțin două crate-uri pentru limbajul Rust care le implementează, `par` [7] și `session_types` [3]. Un vot informal propus de creatorul crate-ului `par` pe platoforma Reddit[9] a avut ca rezultat 58% dintre cei peste 350 de respondenți declarând că nu folosesc tipurile de sesiune pentru că nu le înțeleg în destulă măsură.

Interesul constant (deși restrâns), atât din direcția academică, precum și din cea practică, sugerează că tipurile de sesiune reprezintă o idee de viitor care nu a reușit încă să ocupe un loc în conștiința colectivă a programatorilor.

Referințe

[1] 2002 PODS Influential Paper Award, . URL: https://www.podc.org/influential/2002-influential-paper/. noiembrie 2025.

[2] Franco, J., Vasconcelos, V.T., 2014. A concurrent programming language with refined session types, in: Counsell, S., Núñez, M. (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham. pp. 15–28. URL: https://www.di.fc.ul.pt/~vv/papers/franco.vasconcelos_concurrent-language-refined-session-types.pdf.

[3] Jespersen, T.B.L., Munksgaard, P., Larsen, K.F., 2015. Session types for rust, in: Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming, Association for Computing Machinery, New York, NY, USA. p. 13–22. URL: https://doi.org/10.1145/2808098.2808100, doi:10.1145/2808098.2808100.

[4] λ-Calculus on nLab, . URL: https://ncatlab.org/nlab/show/lambda-calculus. noiembrie 2025.

[5] MIT: Concurrency, . URL: https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/. noiembrie 2025.

[6] Mobility Reading Group: SePI, . URL: http://mrg.doc.ic.ac.uk/tools/sepi/. decembrie 2025.

[7] Par Crate, . URL: https://docs.rs/par/latest/par/. decembrie 2025.

[8] π-Calculus on nLab, . URL: https://ncatlab.org/nlab/show/pi-calculus. noiembrie 2025.

[9] [Poll] Why are you not using session types for your concurrent projects?, . URL: https://www.reddit.com/r/rust/comments/1gwjeid/poll_why_are_you_not_using_session_types_for_your/. decembrie 2025.

[10] SePi Documentation, . URL: http://gloss.di.fc.ul.pt/tryit/tools/SePi. decembrie 2025.

[11] Session Types in Programming Languages: A Collection of Implementations, . URL: https://groups.inf.ed.ac.uk/abcd/session-implementations.html. decembrie 2025.

[12] Stanford CS242: Session types, . URL: https://stanford-cs242.github.io/f18/lectures/07-2-session-types.html. noiembrie 2025.