The Need for Software Migration Systems

Robert ENYEDI
Catedra de Informatică Economică, ASE Bucureşti

Software translation activities tend to decrease in efficiency as the number of supported source and target languages increase. This paper presents an improved approach by organizing translators into software migration systems. It also analyzes the impact of this approach on the efficiency in the dynamics of such systems.
Keywords: *software, migration systems, dynamics.*

1 Introducere

Translatoarele software realizează transformarea codului sursă al aplicațiilor software către noi limbaje de programare şi tehnologii. Translatoarele software sunt specializate pe perechi de limbaje sursă şi destinație [ENY205]. Atunci când există nevoia pentru a realiza translatări automate între mai multe limbaje, abordarea translatoarelor pe perechi de limbaje devine ineficientă.

2. Analiza translatoarelor clasice

Se consideră două translatoare mixte: translator de la limbajul A la limbajul B şi un translator de la limbajul A la limbajul C. Aceste două translatoare au structura internă din figura 1.

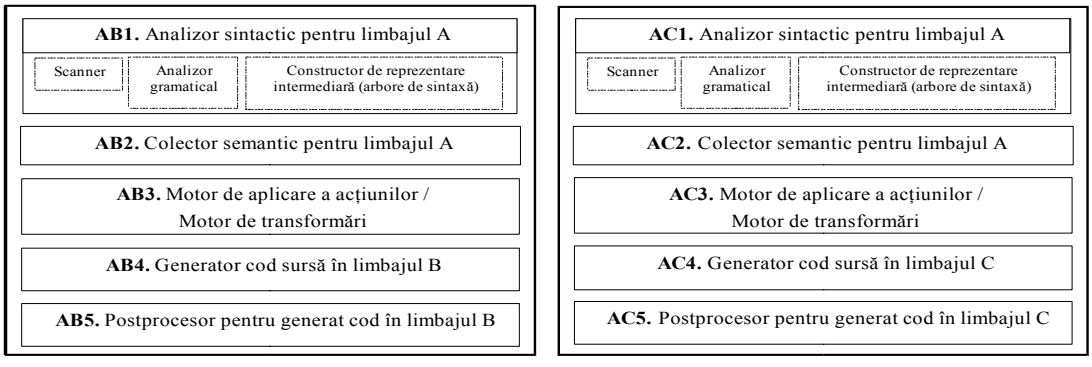

Fig.1. Analiză comparată a două translatoare mixte cu acelaşi limbaj sursă:
a) translator de la limbajul A la limbajul B
b) translator de la limbajul A la limbajul C

Pentru două translatoare mixte care realizează translația din limbaje sursă diferite în acelaşi limbaj țintă: translator de la limbajul A la limbajul C şi translator de la limbajul B la limbajul C.

În acest caz, elementele comune sunt AC4 şi BC4 respectiv AC5 şi BC5 şi parțial AC3 şi BC3. Generatorul de cod sursă în limbajul C precum şi postprocesorul de cod C îndeplinesc aceeaşi funcționalitate dar efortul dezvoltării acestora este dublat. De asemenea, acțiunile şi transformările necesare pentru a genera cod sursă C sunt în mare parte dublate.

Există mai multe elemente comune între cele două translatoare: AB1 şi AC1 respectiv AB2 şi AC2 şi parțial AB3 şi AC3. Aceste elemente îndeplinesc acelaşi scop, dar nu sunt identice. Aceasta înseamnă că în procesul de dezvoltare a celor două translatoare, efortul necesar dezvoltării analizorului sintactic şi colectorului semantic pentru limbajul A a fost duplicat. Mai mult, în cadrul componentelor AB3 şi AC3 există de asemenea elemente comune:

- modulul de traversare a arborelui de sintaxă al motorului de aplicare a acțiunilor,
- modulul de identificare de subarbori din cadrul motorului de transformări.

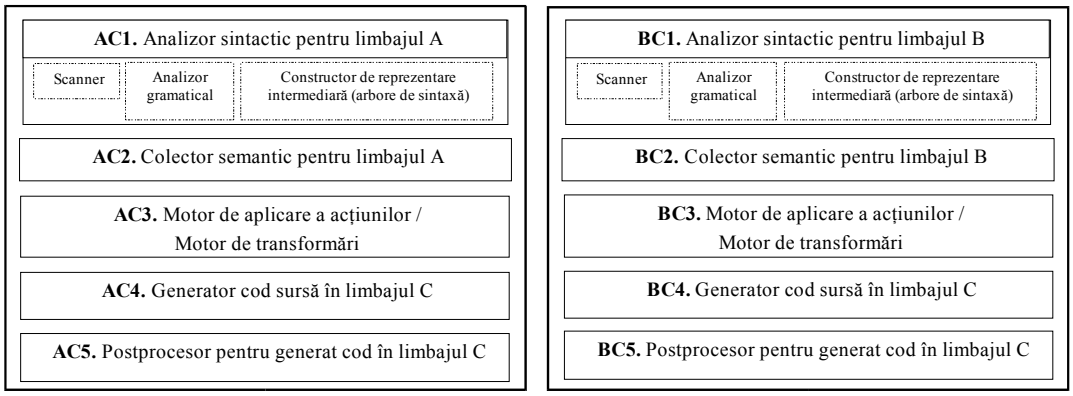

Fig.2. Analiză comparată a două translatoare mixte care au acelaşi limbaj ţintă
a) translator de la limbajul A la limbajul C
b) translator de la limbajul B la limbajul C

Duplicarea funcționalității modulelor este exemplul clasic de duplicare de cod. Experiența a demonstrat că efortul necesar adaptării unui modul software pentru nevoile unei alte aplicații decât cea inițială este mai mic decât efortul dezvoltării de la zero. Mai mult, există multiple avantaje ale codului sursă reutilizat:

- din evoluția în timp a modulului beneficiază toate aplicațiile care îl folosesc,
- din punct de vedere statistic există o înjumătățire a erorilor de programare față de situația în care modulul ar fi dezvoltat de două ori.

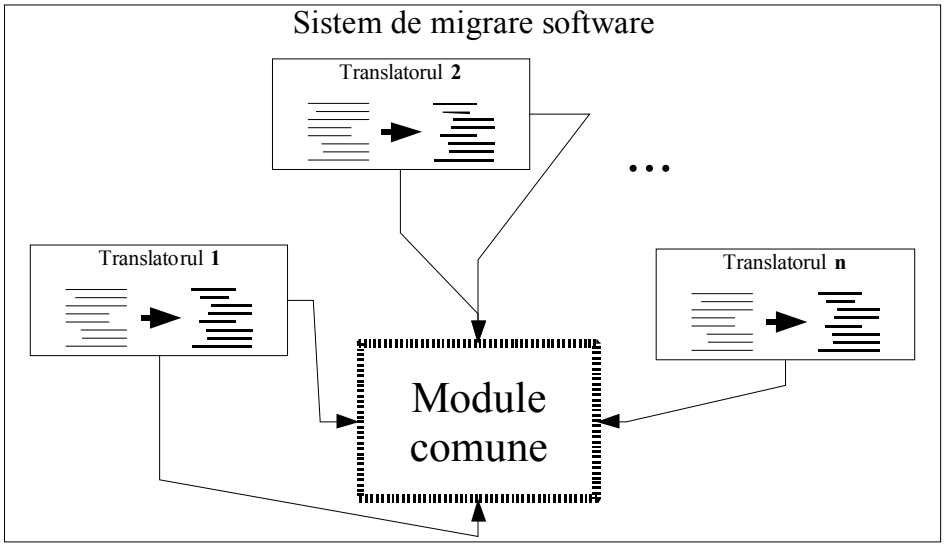

Fig.3. Structura simplificată a unui sistem de migrare software

Reutilizarea modulelor de translatoare este un prim deziderat pentru dezvoltarea unui sistem de migrare software. Un sistem de migrare software include două sau mai multe translatoare software care utilizează module în comun şi care au ca scop realizarea cât mai eficientă a migrării de software, figura 3.

3. Structura sistemelor de migrare software

Reutilizarea componentelor în cadrul sistemelor de migrare software este un prim pas în direcția eficientizării dezvoltării translatoare software. Totuşi, această optimizare este una generică aplicabilă oricăror aplicații software. Însă restricționând domeniul abordat la specificul translatoare se identifică noi opțiuni în ceea ce priveşte dezvoltarea şi organizarea unui sistem de migrare software.

În prezent tehnologia compilatoarelor este cea mai elaborată din ansamblul teoriei translatoarelor software. De aceea în cazul abor-

dării problematicii translatoare software este indicat a se construi pe ansamblul de cunoştinţe din teoria compilatoarelor. În acest sens ca un prim pas a fost analizată structura compilatorului GCC [GGCC05].

Se consideră un sistem de migrare software care constă în două translatoare mixte cu limbaje sursă şi destinaţie complet diferite:

- translatorul M1 care translatează din limbajul A în limbajul B,
- translatorul M2 care translatează din limbajul C în limbajul D.

Conform tehnicii de reutilizare a modulelor translatoare prezentat în secţiunea precedentă nu este posibilă o reutilizare a modulelor celor două translatoare. Prin inventarierea modulelor translatoarelor se obţine mulţimea din figura 4.

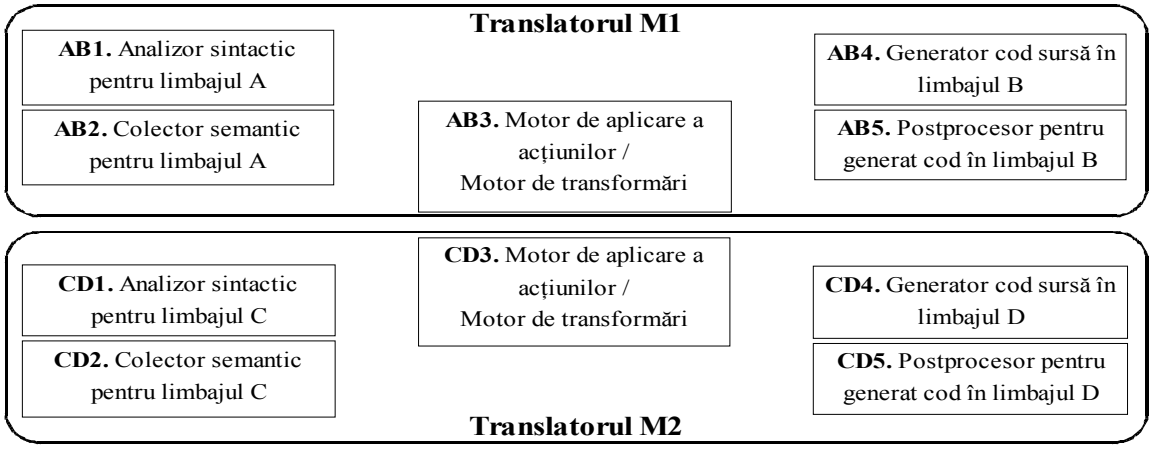

Fig.4. Inventarul componentelor a translatoarelor software M1 şi M2

Analizând inventarul se observă că există de analizoare sintactice şi semantice pentru limbajele A şi C, de generatoare şi postprocesoare de cod pentru limbajele B şi D. De asemenea avem două motoare de acţiuni şi transformări care mapează conceptele limbajelor A şi respectiv C la conceptele limbajelor B şi respectiv D.

Pe studiul compilatorului GCC ştim că acesta compilează un set de limbaje de intrare în oricare din arhitecturile ţintă suportate. Arhitectura acestui compilator este astfel realizată încât suportul pentru N limbaje sursă şi M arhitecturi destinaţie generează M r N variante de transformări.

Acelaşi concept dacă ar fi pus în practică în cadrul sistemului de translatoare prezentat mai sus ar însemna următoarea mulţime de variante translatare:

$$\{A \to B, C \to D, A \to D, C \to B\}$$

Se observă două variante suplimentare de translatare: de la limbajul A la limbajul D şi de la limbajul C la limbajul B.

Pentru a obţine această funcţionalitate a sistemului de migrare soluţia adoptată este cea a unui limbaj intermediar de translatare LIT, figura 5. Dificultatea acestei abordări provine însă din faptul că limbajele sursă şi destinaţie ale sistemului de migrare sunt de nivel apropiat.

Astfel limbajul intermediar este necesar să fie de un nivel suficient de ridicat pentru a nu se pierde informaţii esenţiale de structură din limbajele sursă, de exemplu informaţii despre clase şi obiecte, şi a permite o transformare fără dificultăţi în limbajele destinaţie. Pe de altă parte, limbajul intermediar trebuie să fie de un nivel suficient de scăzut pentru a elimina artificii de construcţie din limbajele sursă.

În multe limbaje de programare există 3 tipuri de cicluri: ciclu cu test iniţial while, ciclu cu test final do-while şi ciclu cu test iniţial şi final for. Dintre aceste trei tipuri, ciclul for este cel mai cuprinzător deoarece celelalte două tipuri de cicluri sunt simulate folosind cicluri for.

De asemenea majoritatea limbajelor de programare au construcţii distincte pentru decizie simplă if-else şi decizie multiplă switch. Orice decizie simplă se mapează pe o decizie multiplă şi, invers, orice decizie multiplă se exprimă printr-o înlănţuire de decizii multiple.

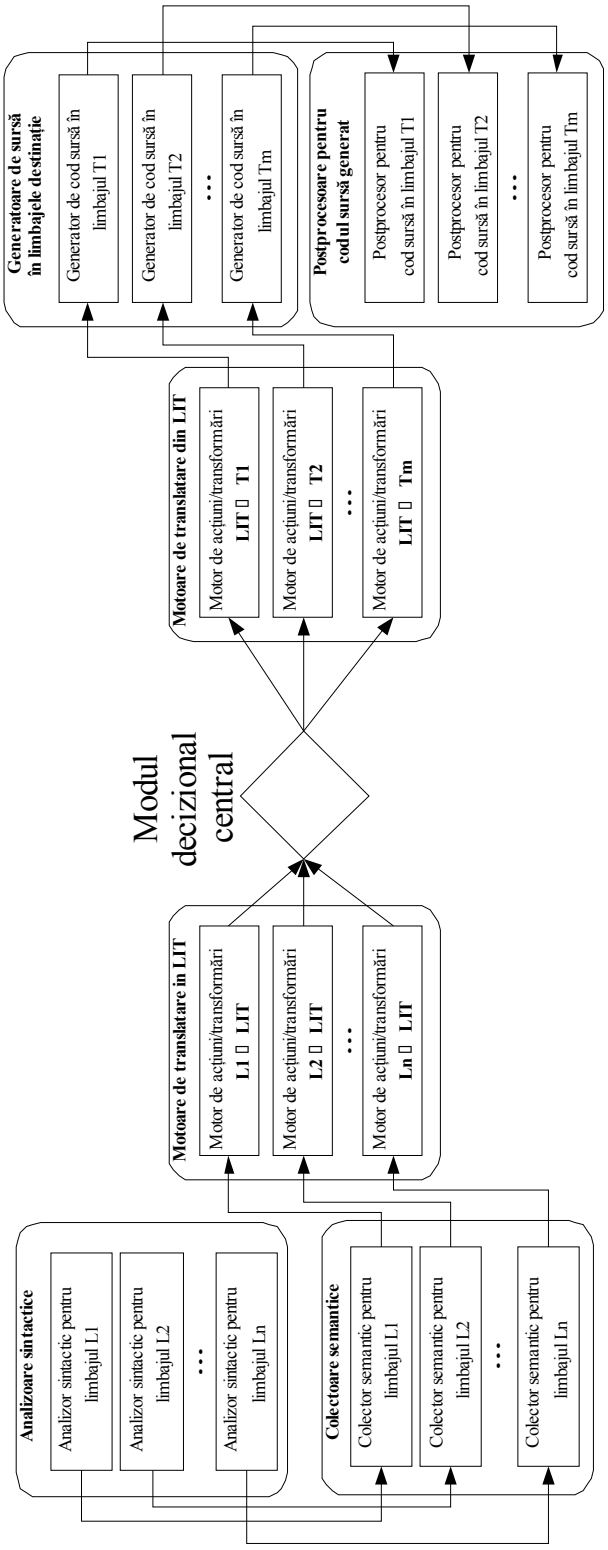

Fig.5. Structura unui sistem de migrare software cu limbaj intermediar

Gradul de generalitate a unui sistem de migrare cu limbaj intermediar este dat de numărul maxim al combinațiilor de limbaje sursă şi destinație suportate. Această generalitate este însă limitată de expresivitatea limbajului intermediar. Expresivitatea limbajului intermediar înseamnă capacitatea de a reprezenta codul sursă pentru limbaje de diferite tipuri.

În proiectarea limbajului intermediar este recomandată stabilirea tipurilor de limbaje suportate. Este mult mai dificil de a proiecta un limbaj intermediar care să fie potrivit simul-

tan pentru limbaje de intrare şi ieşire procedurale, orientate obiect, funcționale şi declarative.

Pe de altă parte este mult mai convenabilă situația în care sistemul de migrare se adresează doar limbajelor orientate obiect şi cel mult procedurale. Acest caz este simplificat şi din motiv că limbajele orientate obiect sunt o evoluție a celor procedurale. Decizia potrivită referitor la nivelul limbajului intermediar este cel mai probabil de a reprezenta conceptele de clasă şi obiect împreună cu conceptele pe care le introduc: moştenire, încapsulare, abstracție şi polimorfism.

Structura unui sistem de migrare software cu limbaj intermediar, figura 5, este o generalizare a structurii unui translator mixt. Datorită utilizării unui limbaj intermediar în procesul de translatare, există trei module suplimentare:

- motoarele de translatare din limbajele sursă în limbajul intermediar,
- motoare de translatare din limbajul intermediar în limbajele destinație,
- un modul decizional central.

Motoarele de translatare din limbajele sursă în limbajul intermediar fac trecerea de la codul sursă original la reprezentarea acestuia în limbajul intermediar al sistemului de migrare. Strategia concretă de realizare a translației la acest nivel este stabilită la nivelul fiecărui motor de translatare în parte. Pentru un limbaj sursă cu o structură apropiată de limbajul intermediar este de preferat un translator monofazic în timp ce pentru limbaje sursă mai complicate un translator multifazic este alegerea corectă.

Motoarele de translatare din limbajul intermediar în limbajele destinație realizează ultima fază a transformării codului sursă inițial în codul sursă dorit. Aici se transformă reprezentarea în limbaj intermediar în codul sursă în limbajul țintă. În funcție de diferențele dintre limbajul intermediar şi limbajul destinație se optează pentru un translator monofazic sau multifazic.

Modulul decizional central coordonează întreg procesul de translatare. Acesta alege modulele specifice de limbaj necesare pentru a realiza translatarea dorită din limbajul sursă în limbajul destinație.

4. Concluzii

Privit în ansamblu, un sistem de migrare software cu limbaj intermediar este o colecție de translatoare din limbaje sursă în limbajul intermediar şi respectiv din limbajul intermediar în limbajele destinație suportate. Dezvoltarea unei perechi complet noi de limbaj sursă – limbaj destinație necesită implementarea a două translatoare în loc de abordarea construirii unui translator mixt clasic. Acesta este un efort sporit față de abordarea tradițională, însă beneficiile obținute sunt superioare: orice limbaj sursă suportat în momentul respectiv va avea un nou limbaj ca destinație a translatării. Limbajul sursă nou introdus va fi translatabil automat în toate limbajele destinațiile suportate în acel moment.

Din acest motiv o structura mai potrivită pentru dezvoltarea unui sistem de migrare software este structura cu limbaj intermediar. În cazul unui astfel de sistem dezvoltarea de module comune adiționale este mult mai simplu de realizat. Aceasta deoarece acest proces de translatare este mult mai bine structurat şi modularizat în spiritul strategiei divide et impera. În acelaşi timp devine mai simplă şi includerea unor module de translatare dezvoltate în alte limbaje de programare decât limbajul nativ al sistemului de migrare.

Bibliografie

[AHOS86] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman – Compilers: Principles, Techniques, and Tools – Addison-Wesley, 1986

[ENY205] Robert Enyedi – *The Structure of Language Translators* – "THE IMPACT OF EUROPEAN INTEGRATION ON THE NATIONAL ECONOMY" Conference, Babeş-Bolyai University, Cluj Napoca, România, 2005

[GGCC05] The GCC Wiki – http://gcc.gnu.org/wiki

[Watt93] David A. Watt – Programming Language Processors: Compilers and Interpreters – Prentice Hall International, 1993