

Din experiențele cu asigurarea portabilității s-au născut o serie de articole dedicate portabilității, scrise împreună cu colegii mei. Printre ele prezint următoarele cinci:

1. I. Athanasiu, M. Necula, L. D. Serbanati, *Program Portability. Automatics, Metrology, Computers, Vol. 20B, pp. 201–206, Technical Publishing House, Bucharest, 1975 (în română).*

În acest prim articol despre portabilitate am încercat, împreună cu Athanasiu și Necula, să punem ordine într-un domeniu care, în practică, era tratat doar empiric. Plecam de la întrebări simple, dar presante:

- ce înseamnă exact ca un program să fie „portabil”?
- față de ce anume trebuie să fie el independent: hardware, sistem de operare, limbaj, mediu de execuție?
- care sunt costurile reale ale portabilității?

Articolul a reprezentat o încercare de a construi un vocabular comun: am discutat tipuri de portabilitate (orizontală – între mașini similare, verticală – între generații diferite de mașini), bariere tipice (instrucțiuni specifice, convenții de apel, reprezentări interne), precum și strategii de atenuare a acestor bariere. Ideile proveneau direct din experiențele noastre cu compilatoarele și instrumentele existente în laborator, dar în articol au fost abstractizate într-un cadru conceptual care să poată fi reutilizat.

Privind în urmă, a fost mai puțin un articol „spectaculos” și mai mult unul de poziționare: definea, într-un limbaj cât de cât sistematic, problema pe care îmi propusesem s-o atac în anii următori.

2. L. D. Serbanati, M. Cosma, C. Popescu, M. Necula, *BCPL – a Language for Program Writing. Automatics, Metrology, Computers, Vol. 24A-B, pp. 345–360, Technical Publishing House, Bucharest, 1976 (în română).*

Dacă articolul despre „Program Portability” definea scena, lucrarea despre BCPL introducea actorul principal al acelor ani: limbajul BCPL, gândit aproape ideal pentru a demonstra în practică portabilitatea. În articol am prezentat BCPL pentru comunitatea românească:

- ca limbaj procedural, structurat, precursor al lui B și C;
- ca limbaj în care compilatorul este scris în el însuși (self-hosting);
- ca limbaj asociat explicit unei mașini abstracte și unui cod intermediar (OCODE, respectiv INTCODE), tocmai pentru a separa partea independentă de cea dependentă de mașină.
- În spatele descrierii de limbaj stătea însă experiența noastră foarte concretă cu portarea compilatorului BCPL folosind interpretorul INTCODE pe FELIX. Articolul nu era doar un manual condensat de BCPL; era și un manifest indirect pentru ideea de a baza portabilitatea pe un strat intermediar bine definit – un cod pentru o mașină abstractă, care poate fi interpretat sau tradus ulterior pe orice platformă.

Aici se vede deja cum conceptul de „mașină abstractă”, care va deveni central în teza de doctorat și în interformele, își face loc, încă de atunci, în discursul meu tehnic, dar într-o formă încă strict legată de implementarea compilatoarelor.

3. L. D. Serbanati, *Metode și tehnici de asigurare a portabilității programelor. Automatics, Metrology, Computers, Vol. 25, pp. 69–82, Technical Publishing House, Bucharest, 1977 (în română).*

Acest articol a fost, într-un fel, prima mea încercare de „mini-tratat” în domeniu. După experiențele acumulate cu BCPL și după discuțiile din laborator, am simțit nevoia să sintetizez într-un mod mai sistematic:

- nivelurile la care poate fi abordată portabilitatea (limbaj, compilator, mediu de execuție, set de utilitare);

- tehnicile concrete: utilizarea codului intermediar, macroprocesarea, standardizarea interfețelor, discipline de programare;
- compromisurile inevitabile între portabilitate, eficiență și simplitate.

Articolul propunea câteva „metode” – în sensul larg al termenului – prin care dezvoltatorii puteau proiecta, încă de la început, programe mai puțin dependente de mașina pe care urmau să ruleze. În fundal, se vedea deja tentația de a privi aceste metode nu doar ca rețete locale, ci ca părți ale unei viitoare Inginerii software: o disciplină în care portabilitatea nu este un accident fericit, ci un obiectiv asumat de la început.

4. I. Athanasiu, M. Necula, L. D. Serbanati, M. Cosma, C. Popescu, *Comparative Study of some BCPL Compilers. Automatics, Metrology, Computers*, Vol. 27, Technical Publishing House, Bucharest, 1978 (în română).

După ce am înțeles conceptual ce înseamnă portabilitatea și am prezentat BCPL ca limbaj-vehicul, a venit firesc un studiu comparativ: cum se comportă, în practică, diverse implementări ale aceluiași limbaj pe platforme diferite?

În acest articol am analizat mai multe compilatoare BCPL – unele obținute direct de la Richards, altele adaptate sau dezvoltate local. Ne interesa:

- cât din specificația limbajului era respectată uniform;
- ce diferențe de performanță și de resurse apăreau între implementări;
- în ce măsură arhitectura compilatorului (în special rolul codului intermediar) facilita sau frâna portabilitatea.

Studiul nostru avea o dublă miză. Pe de o parte, era o contribuție tehnică: oferă date și observații concrete despre o familie de compilatoare reale, în condițiile infrastructurii românești. Pe de altă parte, era și un exercițiu metodologic: arăta cum poate fi „măsurată” portabilitatea, nu doar invocată ca ideal. Această atitudine – de a pune în paralel implementări și de a extrage din ele reguli și modele – o voi relua, mai târziu, în contexte legate de procesul software și de interformele.

Articol privind experiența dobândită din cercetarea portabilității cu macroprocesorul STAGE -2:

5. Specialized Macroprocessor for Software Development – L. D. Șerbănați, R. D. Berbaru, D. F. Mândușianu, *Automatics, Metrology, Computers*, Vol. 27, pp. 103–115, Technical Publishing House, Bucharest, 1978 (în română).

Plecând de la experiența cu STAGE-2, am propus aici un macroprocesor specializat pentru dezvoltarea de software. Ideea era să construim, deasupra unui macroprocesor general, un set de facilități dedicate: șabloane pentru structurile tipice ale compilatoarelor, descrieri de tabele de simboluri, generatoare de cod parametrizabile. Scopul era să obținem o infrastructură care să facă mai ușoară și mai sistematică dezvoltarea de unelte portabile.

Dacă STAGE-2 reprezenta, în viziunea lui Waite, un macroprocesor universal, acest articol marchează o derivare spre un instrument specializat pentru dezvoltarea de software. Experiența cu STAGE-2, cu tehnicile de *bootstrap* și cu ideea de a construi software independent de mașină m-a condus la întrebarea: putem avea un macroprocesor adaptat explicit nevoilor procesului de dezvoltare, nu doar un instrument general de prelucrare de text?

În lucrare am descris proiectarea și implementarea unui macroprocesor specializat care:

- accepta descrieri orientate pe structuri tipice ale programelor (fragmente de cod, șabloane de modul, convenții de apel);
- integra, în măsura posibilului, experiența acumulată în laborator — de exemplu, patternuri uzuale de generare de cod portabil;
- putea fi folosit ca infrastructură pentru alte unelte, de la generatoare de cod până la instrumente de configurare a mediului de dezvoltare.

În fundal, se simte deja ecoul ideilor lui Strachey despre macrogeneratorul universal GPM și al visului, mai vechi, UNCOL — un limbaj intermediar universal pentru compilatoare. Articolul nostru aducea această linie de gândire în contextul concret al informaticii românești, cu resurse limitate, dar cu ambiția de a lucra în același orizont conceptual.