VI. ZAMIENNOŚĆ JĘZYKÓW I PROGRAMÓW

Zamienność języków i programów (napisanych w różnych wersjach tego samego języka) stanowi kluczowy problem rozwoju programowania. Chodzi o to, by nie dopuścić do sytuacji, iż będzie tyle języków, ile jest (i było) maszyn. Pozytywne rozwiązanie problemu umożliwi wykorzystanie dotychczasowego dorobku programowania i kwalifikacji posiadanej kadry kilkuset tysięcy (w skali światowej) programistów.

Stąd usiłowania standaryzacji języków prowadzone przez organizacje międzynarodowe (ISO, ECMA) i amerykańskie (CODASYL, USASI) oraz wysiłki niektórych rządów (np.USA w przypadku COBOLu). Działalność ta, z takich czy innych powodów, nie jest jeszcze należycie uwzględniana przez producentów, posiadających swoje własne ambicje i dla dobra walki konkurencyjnej (a niekiedy i postępu), stale wprowadzających innowacje, nie zawsze do końca przemyślane. Inaczej nie mielibyśmy paru tysięcy języków wyższego rzędu, z których wiele stawia sobie takie same zadania, częściowo dubluje składnię oraz .... błędy poprzedników. Ponadto "oficjalne" (teoretyczne) wersje niektórych języków tak abstrahują od technicznej i organizacyjnej strony przetwarzania (np. ALGOL w zakresie wejścia-wyjścia), że ich wdrożenie jest możliwe jedynie poprzez dopracowanie przez producentów i instytucje software'owe.

Najpoważniejszym czynnikiem różnicującym wersje są różnice w wyposażaniu komputerów. W przypadku bardziej złożonych języków (jak COBOL, PL/I), wymagających pojemnych pamięci operacyjnych, często następuje "obcinanie" pełnych wersji, przystosowując je do aktualnych możliwości sprzętu.

W warunkach dużej dynamiki rozwoju konstrukcji i programowania standaryzacja jest bardzo utrudniona. Można powiedzieć, że jeśli stosuje się ją przedwcześnie, to "konserwuje" ona elementy błędne i mało rozwojowe, hamując tym samym dalszy rozwój dziedziny.

Na pewno zamienność absolutna jest nie do osiągnięcia. Potrzebny jest kompromis ukształtowany pod wpływem kosztu przedsięwzięcia, efektywności kompilatorów itp.

Wydaje się, że standaryzacja programowania powinna być poprzedzona standaryzacją pewnych podstawowych elementów hardware'owych jak np. standaryzacja taśm magnetycznych, rodzajów kodów znaków (łącznie z problemem starszeństwa) itp. Ponadto więcej uwagi należałoby poświęcić rozwojowi technik reprogramowania z maszyny na maszynę (emulacja, symulacja itp.).

1. Koncepcja uniwersalnego języka pośredniego

Wzrost liczby języków programowania oraz typów maszyn powoduje znaczne zwiększenie pracochłonności opracowania translatorów. Jest ona proporcjonalna do iloczynu tych dwóch elementów (tj. liczby języków przez liczbę typów maszyn).

Gdyby udało się we wszystkich maszynach zastosować jeden uniwersalny język pośredni, to liczba translatorów tego języka równałaby się liczbie maszyn (LM). Z kolei wymagane byłoby przetłumaczenie każdego języka użytkowego na uniwersalny język pośredni i do tego celu potrzeba byłoby tyle translatorów, ile jest języków (LJ). Ogólna liczba translatorów równa się więc sumie LM + LJ, podczas gdy w pierwszym przypadku występował iloczyn. Różnica jest więc oczywista (nie tylko arytmetycznie), nawet jeśli uwzględnimy, że to liczbowe ujęcie jest niedokładne,

ponieważ nie uwzględnia różnic w pracochłonności opracowania translatora języka uniwersalnego i translatora języka specjalistycznego (w iloczynie oba argumenty przedstawiają ilości języków specjalistycznych, podczas gdy w sumie argumenty są mieszane).

Koncepcja języka uniwersalnego UNCOL (Universal Computer Oriented Language) pojawiła się pod koniec lat pięćdziesiątych (lata 1958-1961) w artykułach Convey'a M.E., Steel'a T.B., Stronga J. Niestety, języka takiego nie opracowano, a na przeszkodzie stanął zakres pracy, jaki należało wykonać: uzgodnienie, jakie elementy mają być uniwersalne, wybór poziomu programowania oraz aktualizacja pierwszych wersji w związku ze zmianami konstrukcji maszyn.

Koncepcja języka pośredniego nie została jednak całkowicie zarzucona. W charakterze takiego języka w ramach poszczególnych firm występuje zwykle język symboliczny (np. język PLAN dla serii 1900). Niekiedy językiem pośrednim jest język wyższego rzędu, np. FORTRAN stosowany jest jako język pośredni dla APT. Programy APT są za pomocą specjalnego podzbioru APT redagowane wstępnie (zmiana składni) na formę akceptowaną przez kompilator FORTRANu (G - 3).

2. Emulacja

Problemy zamienności języków i programów są równocześnie problemami zamienności komputerów. Stąd też są rozwiązywane m.in. metodą opartą zarówno o środki programowe jak i hardware'owe, zwaną emulacją. Środkiem programowym jest specjalny program, zaś hardware'owym - specjalna pamięć i dodatkowe rejestry. W "emulator" wyposaża się zwykle maszynę nowszą (zakupioną na

miejsce poprzedniej), po to, by mogła akceptować dotychczasowe programy opracowane dla maszyny wycofywanej (starszej). Emulacja nie zapewnia idealnej zamienności: doprowadza do zamienności kody operacji itp., a więc w przypadku zastosowania różnych urządzeń peryferyjnych i systemów operacyjnych pewne przepracowanie programów będzie konieczne.

Oto "zamienne" maszyny firmy IBM w stosunku do serii 360 (wg X - 13):

1620	model 30
1401, 1440, 1460, 1410, 7010	40
7070/7074	50
705/7080 709/7090	65

Hardware'owym elementem emulacji dla serii 360 jest pamięć pasywna ROM-Read Only Memory (zwana też ROS - Read Only Storage), zawierająca mikroprogramy. W miejsce klasycznego wykonywania instrukcji, pamięć ta powoduje otwieranie i zamykanie szeregu elektronicznych "bramek" w systemie obiegu impulsów. Elementem software'owym jest program emulatora, umieszczony w pamięci ferrytowej, służący do wykonywania programu obcej maszyny (również znajdującego się w tej pamięci) w konwencji serii 360. "Obcy" program znajduje się pod kontrolą ROM dopóki nie zostanie napotkana instrukcja, która nie może być hardware'owo interpretowana. Wtedy sterowanie przekazywane jest do programu emulatora.

Pod względem funkcjonalnym emulacja podobna jest do symulacji. Wykonywana jest jednak efektywniej dzięki mikroprogramom, które wykonują funkcje: interpretacji instrukcji, dekodowania adresu, wybierania zawartości adresu

i wykonania wskazanej operacji. Czynności te są najbardziej pracochłonne i pochłaniają dużo czasu przy symulacji.

Wykonanie programu techniką emulacji wymaga większej pamięci operacyjnej (do przechowywania specjalnego programu), zaś efektywność programu adaptowanego na serię 360 powinna być nie mniejsza, niż na maszynie pierwotnej (mimo, iż możliwości serii 360 nie są w pełni wykorzystywane).

Pamięć ROM zbudowana być może z różnych elementów. Model 30 serii 360 w charakterze ROM posiada specjalne miedziane karty perforowane. W innych maszynach (np.firmy Burroughs) ROM zbudowana jest na układach scalonych.

3. Symulacja

Słownik IFIP (V - 1) pod hasłem A2 podaje następującą definicję symulacji: "Symulacja. Reprezentowanie pewnych właściwości zachowania się jednego systemu poprzez działanie innego systemu". Symulacja działa podobnie jak emulacja lecz bez środków hardware'owych. W pamięci przechowywany jest program obcy i program symulatora. Każdy rozkaz może być interpretowany i wykonywany, i w tym przypadku nie trzeba w pamięci przechowywać całego programu obcego.

Firma ICL opracowała następujące symulatory (dla serii 1900):

- Elliott 803 Simulator (xME4, xME8).
  Przeznaczony jest do symulacji emc Elliott 803 z 4 K PAO (lub 8K). Na maszynie 1905 programy Elliott 803 są wykonywane przeciętnie dwukrotmie szybciej (przy czym zapotrzebowanie na PAO wynosi 9600 słów lub 17792).
- Pegasus Simulator (xMP4 dla 4K Pegasus i 12K 1900
  xMP7 7K " 18K ").

Symulator ten staje się niepraktyczny w przypadku używania ręcznych kluczy (przełączników na pulpicie) przez programy maszyny Pegasus.

4. Translacja z języka na język

Jest to najmniej efektywny sposób przerabiania programu z języka na język, ponieważ polega na "mechanicznym" przekształcaniu instrukcji, bez uwzględniania specyfiki nowej maszyny. Wymaga też większej pamięci, ponieważ najpierw tłumaczony jest cały program, a dopiero później wykonywany.

Jako przykłady translatorów wymienić można następujące prace firmy ICL:

- ATLAS ALGOL na 1900 ALGOL (xAC2),
- MPL (1300) na PLAN 1900 (xJP3).

  W xJP3 rozkład danych na bębnie jest "symulowany" w pamięci ferrytowej. Nie podlega tłumaczeniu E funkcja oraz zmodyfikowane instrukcje I. Zakłada się, że program pisany w MPL jest bezbłędny (nie ma kontroli diagnostycznej).

Z publikacji (0 - 1) znana jest koncepcja systemu XACT (X - Automatic Code Translation, przy czym X oznacza dowolną maszynę), opracowanego od 1961 roku przez Celestron Associates, Inc. na bazie pary maszyn 7090 - 1604. Maszyna 7090 jest maszyną źródłową (source), zaś 1604 - docelową (target). Problem polega na tym, by tak przekształcić programy 7090, by mogły być realizowane na 1604. Ocenia się, że opracowanie translatora dla tej pary wymaga 10-15 osobolat pracy programistów. Ciekawostką jest to, że translacja przebiega (przynajmniej częściowo) nie na maszynie docelowej, lecz na maszynie źródłowej (!), do której wprowadza się poza programem również opis

danych (liczb). W pierwszej fazie kompilacji tworzony jest maszynowo niezorientowany opis funkcji do wykonania, zaś w drugiej powstaje program w kodzie maszyny docelowej.

5. Uwagi ogólne

Można wyróżnić następujące typy tłumaczeń z języka na język:

1/ języka maszynowego jednej maszyny na maszynowy język innej maszyny (szczególne trudności występują tutaj m.in. przy tłumaczeniu programów wejścia-wyjścia),

2/ języka maszynowego na język zewnętrzny (mamy tutaj do czynienia z dekompilacją, czyli procesem odwrotnym do kompilacji)

3/ języka zewnętrznego na wewnętrzny,

4/ języka zewnętrznego na język zewnętrzny,

5/ translatora na translator (być może ten rodzaj tłumaczenia ma większe perspektywy rozwojowe, niż tłumaczenie konkretnych programów użytkowych).

Automatyczne tłumaczenie wymaga wstępnej selekcji materiału, odrzucającej zwroty nieprzetłumaczalne i błędne. Dlatego też systemy tłumaczące powinny posiadać organizację umożliwiającą programoście ingerencję w proces tłumaczenia oraz zapewniającą sygnalizację elementów błędnych lub do ręcznej konwersji.

Istnieje szereg sposobów postępowania w przypadku napotkania trudności w tłumaczeniu. W systemie tłumaczenia z języka symbolicznego IBM 7090 na język maszynowy IBM 7040, nieprzetłumaczalnym bezpośrednio zwrotom przydziela się "fikcyjne" makrorozkazy (ekstrakody), które są rozszyfrowywane dopiero w następnych przebiegach. Oto inne rodzaje trudności (G-4):

a/ cały program lub bloki programów są całkowicie nieprzetłumaczalne - ma to miejsce wtedy, gdy konstrukcja maszyny

jest częścią algorytmu (np. program testowania pamięci operacyjnej maszyny CDC 1604),

b/ samomodyfikacja programu (dynamiczna struktura programu),

c/ wyrażenia idiomatyczne (gdy niektóre rozkazy lub ich sekwencje mają sens jedynie wtedy, gdy rozpatrywane są na tle całego programu),

d/ różnice konstrukcyjne pomiędzy maszynami (np. słowo 24-bitowe i 36-bitowe).

6. Zamienność programów pisanych w COBOLu

Wersje oficjalne (np. ogłoszone przez CODASYL) COBOLu są co prawda maszynowo niezależne, lecz stanowią one dla twórców kompilatorów jedynie bazę koncepcyjną, do której wprowadzają własne pomysły, utrudniające zamienną eksploatację programów.

Pomysły te stanowią odbicie specyfiki hardware'owej lub też są po prostu innowacją formalną w składni języka (np. w wersji COBOLu na maszynę ZAM41 opracowanej przez Instytut Maszyn Matematycznych w Warszawie).

Rozważając problemy zamienności programów, trzeba wspomnieć o przypadkach obiektywnej niezamienności, np. gdy program napisany w wersji dyskowej (z zastosowaniem indeksowania lub randomizacji) chcemy eksploatować na maszynie wyposażonej tylko w taśmy magnetyczne.

Już od samego początku podejmowano starania, by COBOLowi zapewnić wysoki stopień zamienności. Tak np. w grudniu 1960 roku demonstrowano zamienność programów COBOLu pomiędzy maszynami RCA 501 i UNIVAC II (B-10). Później nadzór nad składnią tego języka przejęły takie organizacje międzynarodowe /poza

CODASYLem), jak ASA/USASI, ANSI/ISO, ECMA.

Zakres wersji kompilatora COBOLu i jego efektywność zależą przede wszystkim od wielkości dostępnej (po odjęciu obszaru dla systemu operacyjnego itp.) pamięci operacyjnej.

W celu sklasyfikowania poszczególnych wersji firmy IBM i Honeywell wprowadziły pojęcie poziomu (level) języka: Honeywell - B, D, H, I, F, zaś IBM - E, F. Każdy poziom języka różni się zakresem słownika dopuszczalnych zwrotów.

Firma ICL (seria 1900) usiłowała natomiast stosować jedną wersję języka, zmniejszając efektywność kompilacji przy mniej dogodnych konfiguracjach (użycie wielokrotnego wzywania segmentów wymiennych).

Dla programisty najważniejszym problemem jest niewątpliwie zakres czynności, jakie ma wykonać, chcąc zaadoptować program COBOLu pisany dla innej maszyny. Brakuje tego rodzaju informacji w publikacjach krajowych i obcych. Poniżej podamy pewne zasady oparte o własne doświadczenia w programowaniu na maszyny ICL 1900, ICL System 4-50 oraz ZAM-41:

1/ porównanie konfiguracji maszyn (w szczególności pojemności pamięci, rodzajów urządzeń wejścia-wyjścia i pamięci masowej),

2/ porównanie listy wyrażeń obu kompilatorów oraz sprawdzenie zakresów działania takich samych zwrotów (m.in. dotyczy to deklaracji COMPUTATIONAL, DISPLAY-n ściśle związanych z postacią liczb w maszynach - postać binarna, pojedyncza lub podwójna precyzja, stały lub zmienny przecinek, obliczenia w angielskim dziesiętnym systemie walutowym itp. W COBOLu i Honeywell I deklaracje COMP, COMP-1 posiadają identyczne znaczenie),

3/ sprawdzenie, czy program jest segmentowany i ile miejsca zajmuje w PAO w dotychczasowej posegmentowanej postaci,

4/ sprawdzenie, czy program posiada wstawki napisane w innych językach (w języku symbolicznym lub wyższego rzędu np. w FORTRANie lub języku symbolicznym dla COBOLu Honeywell). Jeśli takie wstawki wystąpują, to należy zapewnić odpowiednik wstawki w "macierzystym" języku wstawek i wprowadzić odpowiedni aparat formalny związany z użyciem instrukcji ENTER, CALL itp. Można oczywiście zamiast wstawki napisać fragment programu w języku COBOL.

5/ porównanie sposobów wejścia danych; należy ustalić, czy występuje zapis pozycyjny i niepozycyjny. Jeśli stosowany jest zapis niepozycyjny, to należy sprawdzić, jakie znaki są stosowane w charakterze standardowych ograniczników.

6/ porównanie treści metryk standardowych (w szczególności dotyczy to zbiorów na dyskach i taśmach magnetycznych),

7/ porównanie parametrów drukarek wierszowych (długość wiersza, znaki specjalne, funkcje poszczególnych kanałów taśmy sterującej itp.).

Jako przykład firmowych odrębności można podać zwroty występujące w COBOLu Honeywell:

- STOP "JOB" (zakończenie ciągu zadań) obok STOP RUN (np. w celu przejścia do podprogramu),
- CANCEL (zwolnienie obszaru pamięci operacyjnej, zajętego przez podprogram),
- LOAD (załadowanie podprogramu do pamięci operacyjnej).

Można powiedzieć, że ani jednego programu (z wyjątkiem trywialnych) COBOLu nie można mechanicznie przenieść na inny typ maszyny bez dokonania pewnych zmian.

Najbardziej zmienną częścią COBOLu jest ENVIRONMENT DIVISION, a więc rozdział opisujący konfigurację maszyny, najmniej - co nie oznacza, że w ogóle nie ulega zmianom - PROCEDURE DIVISION. DATA DIVISION jest z omawianego punktu widzenia czymś pośrednim: mogą wystąpić różnice w opisie metryk zbiorów, w klauzulach typów danych, w długości wiersza na tabulogramie itp.

Jeśli chodzi o część proceduralną, to nie we wszystkich wersjach występują bardziej złożone warianty instrukcji PERFORM (z AFTER), zwrot DEPENDING ON (w połączeniu z GO TO), MOVE CORRESPONDING itp. Pewien kłopot sprawić może uzyskanie takiej samej dokładności wyniku przy operacjach arytmetycznych. Do innych rezultatów doprowadzić mogą operacje porównywania pól znaków alfanumerycznych w sytuacji, gdy porównywane maszyny posiadają inną sekwencję starszeństwa znaków (np. w niektórych maszynach litery "są" "większe" od cyfr, w innych - mniejsze). Odrębne zagadnienie stanowi wpływ systemu operacyjnego maszyny, na którą opracowano program, stosowanie pakietów REPORT WRITER, procedur diagnostycznych typu READY TRACE, RESET TRACE, EXIBIT....

Pomocą w ocenie zamienności kompilatorów mogą być specjalistyczne generatory programów, np. CCVS - Cobol Compiler Validation System opracowany przez Air Force USA (H - 7), dostarczające odpowiedzi na najistotniejsze pytania:

a/ czy kompilator spełnia wymogi poziomu, do którego został zaliczony, tzn. czy rzeczywiście wykonywane są wszystkie funkcje przewidziane przez standard poziomu,

b/ czy program wewnętrzny otrzymany po kompilacji dostarcza wyniki odpowiadające określonym standardom.

Analiza zamienności różnych kompilatorów stanowi intersujący

Przykłady różnic pomiędzy wersjami języka COBOL

Lp.	Element	ICL seria 1900	ICL System 4-50	IMM ZAM 41	U w a g i
1.	Nazwy danych i nazwy paragrafów	1-szy znak musi być literą	w nazwie danych musi być co najmniej jedna litera, ale nie musi być 1-szy znak, nazwa paragrafu może być liczbą	1-szy znak musi być literą	
2.	Nazwy urządzeń np. czytnika kart	CARD-READER	UNIT - RECORD C-READER	INPUT /nr/	IBM 360: UNIT-RECORD 1402R
3.	Deklaracje np. COMPUTATIONAL	liczba dziesiętna kodowana dwójkowo /binary decimal/ 1/	postać binarna	—	1/ nie jest to typowa postać liczby dziesiętnej kodowanej dwójkowo, ponieważ co prawda 1 znak zajmuje 4 bity, lecz 2 znaki - 7b, 3 znaki - 10 bitów itp.
4.	COMPUTATIONAL-1	liczba binarna	liczba zmienno-przecinkowa	—	
	FIXED	—	—	liczba całkowita binarna	
	FLOAT	—	—	liczba zm. przec. w postaci binarnej	

i obszerny problem, kwalifikujący się do odrębnego opracowania. Ze względu na ograniczoną objętość niniejszej pracy, nie możemy poświęcić temu więcej miejsca.

7. Zamienność programów pisanych w FORTRANie

Podstawową wskazówką zamienności jest rodzaj podstawowej wersji: FORTRAN, FORTRAN II, FORTRAN IV. Ponadto w ramach tych samych wersji mogą występować znaczne różnice, np. kompilatory FORTRANu dla IBM 1620, IBM 650 nie zawierały deklaracji FORMAT i nie dawały możliwości wezwania podprogramów.
Następujące zwroty FORTRANu IV nie występowały w FORTRANIE II (G - 2): DATA, BLOCK DATA, NAMELIST, logiczne IF, numery zdań w charakterze argumentów podprogramów, wielokrotne wejścia do podprogramów.

Oprócz tego, istnieją następujące różnice w instrukcjach wejścia-wyjścia:

FORTRAN II	FORTRAN IV
READ INPUT TAPE j,i,parametry	READ (j,i) parametry
WRITE OUTPUT TAPE j,i, parametry	WRITE (j,i) parametry
READ TAPE j, param.	READ (j) param.
READ DRUM i,j, param.	READ (k) param.
WRITE DRUM i,j, param.	WRITE (k) param.

Opracowany został translator SIFT (Share Internal FORTRAN Translator) do modyfikacji programów FORTRANu II na programy FORTRANu IV.

8. Zamienność programów pisanych w ALGOLu

Kłopoty z zamiennością programów dotyczą przede wszystkim procedur wejść-wyjść, opisywanych nie przez wersje teoretyczne, lecz przez translatory. Pogląd ten zilustrujemy w poniższym zestawieniu.

Lp.	Procedury	ICL 1900	ICL system 4-50	ODRA 1204	GIER ALGOL	Ural 2
1.	Przydział urządzeń	select input/n/ select output/n/	OPEN/n/ SET /n,m/	set input/n/ set output /n/		
2.	Zwolnienie urządzeń	free input free outpur	CLOSE /n/			
3.	Operacje czytania, pisania itp.	print /E,m,n/ output /E/ write boolean/E/ write text /.../ copy text /.../ N: = read	WRITE /n,FORMAT .....n-d/ n-d : =READ/n/	print /$n_1 \ldots n_n$/ ininteger inreal inchar print/$n_1 .. n_n$/ outchar line /n/ read /$n_1 \ldots n_n$/	write writetext write cr write char output outtext outchar outsum input inone inchar setchar lyn typein typechar	writetexst outtext outer writeer outchar outelear input inone typein