3 1993

# informatyka

WYDAWNICTWO SIGMA NOT

Cena 25 000 zł

# informatyka

nr 3 1993
Marzec
Rok wydania XXVIII

Miesięcznik
ISSN 0542-9951
INDEKS 36124





WYDAWCA:
Wydawnictwo Czasopism i Książek
Technicznych SIGMA NOT
Spółka z o.o.
ul. Ratuszowa 11
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych
redakcja nie zwraca




## W numerze:



## W najbliższych numerach:

- Tomasz Kalinowski zajmuje się programowaniem systemów transputerowych.
- Lucyna Korbel, Elżbieta Miłosz i Marek Miłosz analizują możliwości wykorzystania nowoczesnych relacyjnych baz danych do budowy systemów informatycznych.
- Henryk Sroka omawia systemy wspomagania decyzji z bazą wiedzy.
- Zdzisław Szyjewski charakteryzuje klasyfikację narzędzi wspomagania CASE.
- Andrzej Matlak opisuje prawne środki ochrony programów komputerowych w USA.
- Leszek Kotzian zajmuje się integracją środowiska sieci NetWare ze światem UNIX.
- Dariusz Gaik i Andrzej Przybylski nawiązując do badań z zakresu zastosowań komputera opisują pakiety graficzne współpracujące z oprogramowaniem narzędziowym AWP.




Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 75/93

ANNA OSTASZEWSKA
PIOTR FUGLEWICZ
Computer Systems for Business International LTD.
Warszawa

# PROGRESS-em przez ocean danych

**Współczesny świat zalewa nas ogromna liczba różnorodnych informacji. Wszystkie te informacje o tysiącach połączeń lotniczych, wiarygodności finansowej milionów firm, tony bibliografii czy prognozy pogody dla każdego zakątka świata są po prostu DANYMI dla komputera, który możemy postawić sobie na biurku. Aby nie utonąć w ocenie danych, lecz żeglować po nim wytyczonym kursem, potrzebne są systemy zarządzania bazami danych. Przetwarzanie danych, to najbardziej rozpowszechnione ze współczesnych zastosowań informatyki, a liczba obecnych na rynku systemów zarządzania bazą danych wprowadza niejednokrtonie w zakłopotanie potencjalnego użytkownika.**

W poniższym artykule podjęliśmy próbę scharakteryzowania Systemu Zarządzania Relacyjną Bazą Danych PROGRESS, pragnąc ułatwić ocenę przydatności tego systemu dla konkretnych zastosowań, bądź też umożliwić porównanie PROGRESS-a z innymi bazami danych.

## Środowisko projektowe PROGRESS

PROGRESS jest nowoczesnym i wydajnym narzędziem służącym do tworzenia aplikacji obsługujących relacyjne bazy danych. Pierwsza wersja tego systemu powstała w 1984 r. w amerykańskiej firmie *Progress Software Corporation* (PSC). Od tamtego czasu system jest stale rozwijany i udoskonalany, a od czterech lat zajmuje pierwsze miejsce w rankingach przeprowadzanych przez niezależną firmę *Datapro Research*.

System PROGRESS zapewnia projektantowi i programiście zintegrowane środowisko projektowe, którego głównymi składowymi są: Słownik Bazy Danych *(Data Dictionary)*, edytor, kompilator języka PROGRESS 4GL, system zarządzania relacyjną bazą danych (RDBMS) oraz generator aplikacji. Zastosowany we wszystkich modułach jednolity tryb komunikacji z użytkownikiem ma zadanie ułatwić naukę posługiwania się systemem oraz jego późniejsze użytkowanie.

Spróbujmy prześledzić proces pisania aplikacji w środowisku stworzonym przez system PROGRESS. Pierwszym etapem będzie zbudowanie schematu bazy danych, czyli szczegółowego opisu wszystkich tablic i powiązań relacyjnych między nimi, więzów integralności, które chcemy nałożyć na dane, struktury indeksów itd. Słownik Bazy Danych umożliwia nam zdefiniowanie schematu metodą interakcyjną. W Słowniku są przechowywane także takie informacje pomocnicze, jak format wyświetlania danych, czy postać etykiet lub tekstów podpowiedzi związanych z poszczególnymi polami. W słowniku są przechowywane warunki poprawności danych dotyczące pól (mogą to być dowolnie skomplikowane formuły logiczne, również odwołujące się do zawartości innych pól w innych tablicach), warunki sprawdzane przy próbie usunięcia rekordu z danej tablicy, reguły narzucające obligatoryjność czy też unikalność danego pola. Aby zabezpieczyć zawartość bazy przed niepowołanym dostępem posługujemy się również Słownikiem Bazy Danych, gdzie zapisujemy informacje o prawach dostępu do poszczególnych tablic bądź pól dla poszczególnych użytkowników lub ich grup.

Jedną z opcji udostępnianych przez Słownik Bazy Danych jest sporządzanie raportów dotyczących struktury bazy danych, stworzonych struktur oraz ich wzajemnych powiązań. Raporty takie umożliwiają bieżące dokumentowanie stanu prac projektowych.

Mając zdefiniowaną strukturę bazy danych możemy przystąpić do pisania operujących na tej bazie procedur. Oczywiście zakładamy, że projekt tworzonego systemu, zawierający funkcjonalny opis modułów programowych i ich wzajemnych powiązań, jest już gotowy.

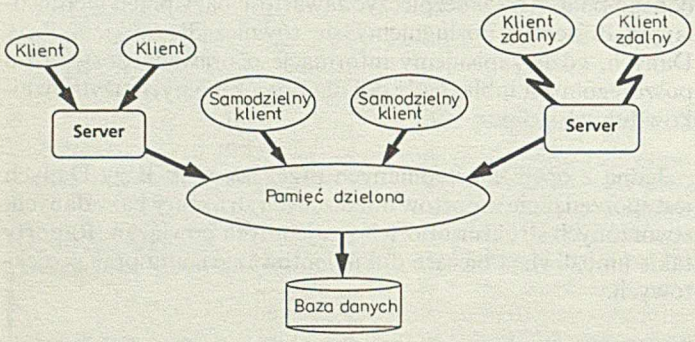
Do pisania procedur używamy pełnoekranowego edytora, stanowiącego integralną część systemu PROGRESS. Edytor ten jest sprzężony z kompilatorem, Słownikiem Bazy Danych i Systemem Zarządzania Bazą Danych. Jedno naciśnięcie klawisza powoduje kompilację, a następnie wykonanie znajdującej się w polu edytora procedury. Jeśli w trakcie kompilacji lub wykonania wystąpią błędy, wracamy do edytora, poprawiamy procedurę i powtórnie ją uruchamiamy. Gotowe procedury możemy przechowywać w postaci skompilowanej, również łącząc je w biblioteki. Dostępne są także (w postaci źródłowej) gotowe biblioteki procedur napisanych w PROGRESS-ie.

Ponieważ o samym języku PROGRESS będziemy jeszcze pisać poniżej, w tym miejscu zaznaczymy tylko, że wszystkie obsługujące daną bazę procedury korzystają z informacji zawartych w Słowniku. Tak więc programista może skoncentrować się na logice aplikacji, nie zajmując się takimi drobiazgami, jak format wyświetlania danych, podpowiedzi czy sprawdzanie poprawności wprowadzanych danych (rzecz jasna, jeśli wszystko to zostało dokładnie wyspecyfikowane w Słowniku).

Programy napisane w PROGRESS-ie są wykonywane pod kontorlą Systemu Zarządzania Bazą Danych (SZBD). Zajmuje się on m.in. fizyczną realizacją dostępu do danych, ochroną danych, realizacją przetwarzania rozproszonego oraz pracy z rozproszonymi bazami danych. Ochrona danych obejmuje zarówno zapewnienie prawidłowego wykonywania transakcji przez zabezpieczenie obrazu pierwotnego (ang. *before imaging*) i obrazu wtórnego (ang. *after imaging*), jak i wykonywanie archiwizacji, przy czym mogą być sporządzane kopie pełne lub przyrostowe. W przypadku systemów pracujących 24 godziny na dobę jest przydatna możliwość archiwizacji bez przerywania

pracy systemu (ang. *backup on-line*). SZBD dba również o przestrzeganie ustalonych przez administratora systemu praw dostępu do danych. Jeżeli z bazy danych korzysta równocześnie wielu użytkowników, dostęp do danych modyfikowanych przez innego użytkownika jest blokowany na poziomie rekordu.

W przypadku aplikacji napisanej w PROGRESS-ie, przetwarzanie rozproszone polega na rozdzieleniu funkcji spełnianych przez program aplikacji między maszyną zwaną serwerem bazy danych oraz stacje robocze, przy których pracują użytkownicy końcowi (tryb klient-serwer). PROGRESS pracuje w trybie wieloserwerowym i wielowątkowym (ang. multuserver & multithreaded), co oznacza, że dla jednej bazy może istnieć wiele aktywnych procesów serwera, przy czym każdy z nich może obsługiwać wielu klientów (rys. 1). Rozproszenie przetwarzania między maszynę serwera i komputery użytkowników zwiększa szybkość i efektywność przetwarzania.




**Rys. 1. Wieloserwerowe i wielowątkowe przetwarzanie w trybie klient-serwer**

Rozproszeniu może ulec również baza danych, a z różnych względów korzystne może być rozdzielenie danych między kilka serwerów. Zarówno w przypadku sieci jednorodnej, jak i heterogenicznej takie rozproszenie nie pociąga za sobą konieczności wprowadzania jakichkolwiek zmian w programach aplikacyjnych i jest dla tych programów całkowicie przezroczyste. Spójność bazy danych przy transakcjach dotyczących kilku baz danych jest zagwarantowaną przez dwufazowy protokół potwierdzeń (ang. *two-phasse commit*), polegający na tym, że transakcja zostaje uznana za pomyślnie wykonaną po nadejściu potwierdzeń od wszystkich biorących w niej udział serwerów baz.

## Czy możemy mówić „relacyjna"?

Termin „relacyjna baza danych" ma ściśle określone znaczenie, choć bywa czasem używany na wyrost. Aby pokazać, że baza danych PROGRESS jest istotnie bazą relacyjną, przedstawimy w formie tabeli dwanaście reguł Codda stanowiących o relacyjności bazy [2] oraz sposób spełnienia tych wymagań przez PROGRESS.

## PROGRESS jako język czwartej generacji

Nie istnieje precyzyjna definicja określająca, czym są języki czwartej generacji. Na podstawie porównania cech produktów określanych tym mianem można stwierdzić, że język czwartej generacji powinien spełniać określony zbiór postulatów. Do najważniejszych z nich należą:

- możliwość wyrażenia podstawowych struktur programowych, takich jak: iteracja, instrukcja warunku, funkcje i procedury parametryzowane;
- realizacja funkcji kontroli poprawności, spójności i bezpieczeństwa danych zapisanych w używanym przez język systemie zarządzania bazą danych;
- udostępnianie przez język funkcji umożliwiających formowanie i obsługę ekranu.

Dodatkowo systemy, których częścią są języki czwartej generacji, zwykle umożliwiają:

- pracę wielu użytkowników oraz synchronizację dostępu do danych;
- przetwarzanie rozproszone przez obsługę wielu protokołów sieciowych;

**Dwanaście reguł Codda i sposób ich spełnienia przez PROGRESS**

| Treść reguły | System PROGRESS |
|---|---|
| Wszystkie informacje w bazie są reprezentowane expicite na pozimie logicznym w dokładnie jeden sposób, tzn. przez wartości w tablicach | Wszystkie dane są przechowywane w tablicach |
| Każda niepodzielna dana jest dostępna logicznie przez nazwę tablicy, wartość klucza pierwotnego oraz nazwę kolumny | PROGRESS umożliwia wyszukiwanie danych w tablicy przez wartość klucza głównego |
| Jest możliwa reprezentacja na poziomie logicznym faktu, że dana informacja jest nieznana | PROGRESS umożliwia wprowadzenie jako zawartości pola tzw. *Unknown Value* |
| Opis bazy danych jest reprezentowany na pozimie logicznym tak samo, jak same dane i ten sam język umożliwiają dostęp zarówno do danych, jak i do schematu | Opis schematu jest przechowywany w bazie danych i jest dostępny przez ten sam język, przez który jest realizowany dostęp do danych |
| Relacyjny system zarządzania bazą danych musi wspierać przynajmniej jeden język:<br>– którego instrukcje są wyrażalne jako ciągi znaków o dobrze określonej składni;<br>– który umożliwia:<br>(1) opis danych,<br>(2) opis perspektyw,<br>(3) manipulowanie danymi (interakcyjne i programowe),<br>(4) określenie więzów integralności,<br>(5) autoryzację transakcji,<br>(6) określanie granic transakcji | Wbudowany SQL spełnia te wymagnia. PROGRESS 4GL spełnia je wszystkie oprócz (2) |
| System zapewnia możliwość aktualizacji tablic wirtualnych (perspektyw) | Wbudowany SQL spełnia te wymagania |
| Argumentami operacji wstawiania, modyfikowania, wyszukiwania i kasowania są całe tablice | Wbudowany SQL spełnia te wymagania |
| Zmiany metod dostępu lub wewnętrznych struktur pamięciowych nie pociągają za sobą konieczności zmian w programach aplikacyjnych | Oprócz spełnienia tego wymagania PROGRESS zapewnia pełną przenośność aplikacji między wszystkimi obsługiwanymi platformami |
| Zmiany w tablicach nie powodują konieczności zmian w programach aplikacyjnych | Zmiany w bazie danych nie mają wpływu na kod aplikacji w PROGRESS-ie |
| Więzy integralności, specyficzne dla konkretnej bazy, muszą być definiowane w języku manipulacji danymi oraz przechowywane wraz z opisem bazy, a nie w programie aplikacyjnym | Informacje o więzach integralności są przechowywane w Słowniku Bazy Danych PROGRESS |
| System relacyjny zapewnia obsługę baz rozproszonych, bez konieczności zmian w programach aplikacyjnych | PROGRESS obsługuje rozproszone bazy danych w sieciach heterogenicznych, a rozproszenie bazy nie wpływa na kod aplikacji |
| Jeśli system relacyjny ma język niskiego poziomu (operujący na pojedynczych rekordach), nie można na tym poziomie ominąć lub złamać reguł integralności wyrażonych w języku wyższego poziomu (operującym na wielu rekordach) | PROGRESS wspiera zachowanie reguł integralności przez Słownik Bazy Danych, a nie przez język używany do realizowania dostępu do danych |

- obsługę graficznego trybu komunikacji z użytkownikiem;
- możliwość pracy z wieloma bazami, zarówno w formacie podstawowym dla danego języka, jak i z bazami innych systemów obsługi baz danych.

Zapisany przy użyciu języka czwartej generacji kod programu jest bardziej zwarty i bardziej czytelny. W konsekwencji pisanie i utrzymanie programów jest prostsze niż w językach poprzednich generacji. Styl programowania w językach niższego poziomu polega na ogół na przekazywaniu wskaźników do obszarów zawierających dane. Stosowanie takiej techniki może prowadzić do niebezpiecznych i trudno wykrywalnych błędów działania programu. Kompilatory języków czwartej generacji sprawdzają składnię, co eliminuje w znacznym stopniu możliwość powstania takich błędów. Programowanie operacji ekranowych i raportów jest w językach czwartej genracji prostsze i szybsze. Za siłę języków czwartej generacji płaci się jednak degradacją wydajności programów stworzonych przy ich użyciu, w porównaniu z programami pisanymi w językach niższego poziomu. Wobec rosnącej wydajności sprzętu komputerowego jest to jednak cena niewielka.

PROGRESS spełnia wymagania stawiane językom czwartej generacji, będąc przy tym językiem kompletnym: programista pisze cały kod aplikacji w 4GL, nie sięgając do języków niższego poziomu. Przejrzysta, zbliżona do składni języka angielskiego składnia PROGRESS 4GL ułatwia naukę i późniejsze programowanie. Czas pisania aplikacji w PROGRESS-ie bywa nawet dziesięciokrotnie krótszy od czasu pisania takiej samej aplikacji w języku trzeciej generacji (np. C, Cobol).

## CASE a PROGRESS 4GL

Podobnie jak wiele innych rozwiązań w informatyce, języki czwartej generacji bywają reklamowane jako rozwiązanie generalne, ostateczne i zastępujące wszystkie inne. Twierdzi się czasem, że stosowanie narzędzi CASE jest zbędne, jeśli tylko korzysta się z 4GL. Pogląd taki nie wydaje się być słuszny. Oba elementy dobrze uzupełniają się, zwłaszcza, że obszarem ich działania są różne fazy cyklu życia oprogramowania. O ile CASE koncentruje się na fazach wstępnych, a zarazem najważniejszych dla poprawności przyszłej aplikacji, pokrywając swym zasięgiem analizę problemu i projekt, o tyle 4GL jest używany w fazach implementacji i utrzymania aplikacji.

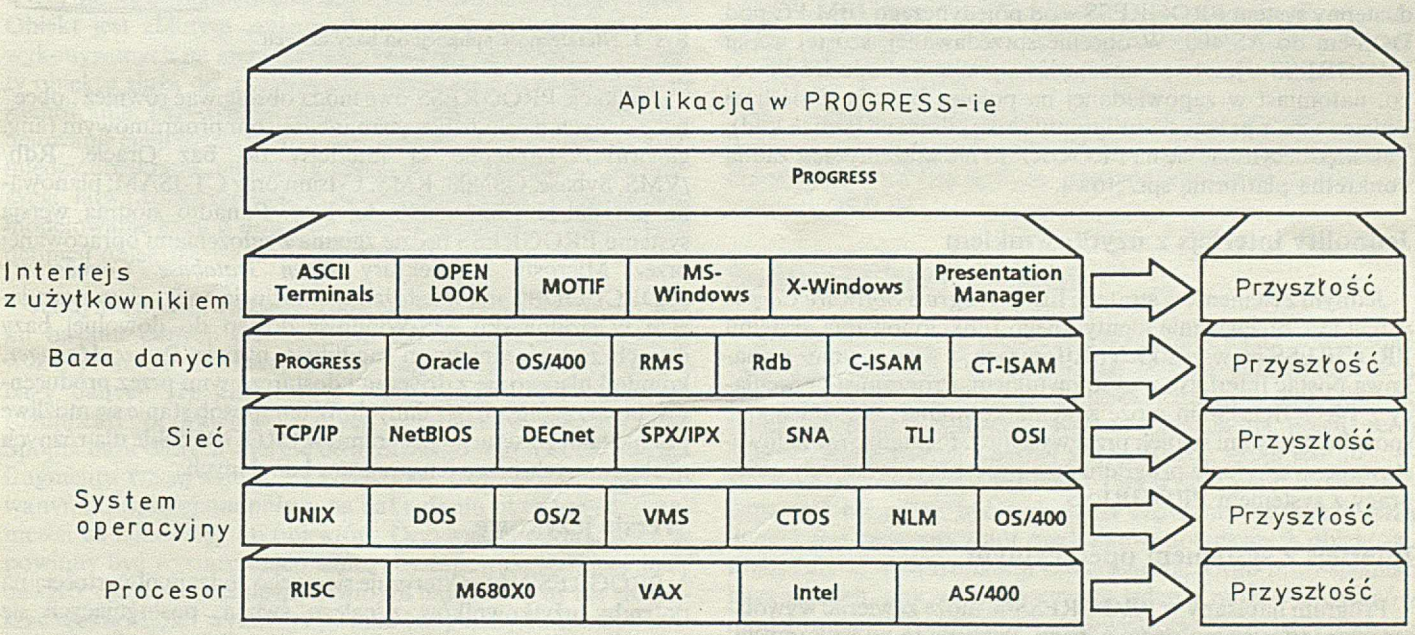
Firma PSC prowadzi otwartą politykę w zakresie stosowania narzędzi CASE. Nie jest oferowane żadne narzędzie CASE fabrycznie zgodne z PROGRESS-em. Istnieją pomosty do takich popularnych narzędzi, jak: ER-Modeler (*Chen & Associates, Inc.*), Excelerator (*Intersolv*), ADW i IEW (*KnowledgeWAre*), EasyCASE (*Evergreen CASE Tools*), System Architect (*Popkin Software & Systems Inc.*), XPONENT (*Software Solutions Unlimited, Inc.*). Użytkownik może wybrać taką metodę projektowania i takie narzędzie CASE, które najbardziej odpowiada jego potrzebom bądź upodobaniom, a PSC zapewnia mu możliwość integracji tego narzędzia z systemem PROGRESS.

Pomost między PROGRESS-em a programem EasyCASE został stworzony przez programistów firmy CSBI (polskiego dystrybutora systemu PROGRESS). Niebagatelną zaletą EasyCASE-a jest jego umiarkowana cena (ok. 1300 USD), a dotychczasowe doświadczenia CSBI wskazują na dużą przydatność tego pakietu. Możliwe jest zarówno przenoszenie do PROGRESS-owego Słownika Bazy Danych schematów relacji, zależności między obiektami czy założeń związanych z integralnością zaprojektowanych w środowisku CASE, jak i tzw. *reverse engineering*, czyli zrealizowanie za pomocą EasyCASE-a projektu technicznego wcześniej stworzonej aplikacji. Stosowanie narzędzia CASE znacznie zmniejsza ryzyko popełnienia błędów w fazie projektowej, a także umożliwia prawidłowe dokumentowanie wszystkich faz życia aplikacji.

## Otwartość systemu PROGRESS

System otwarty – to kolejny niejednoznaczny, ale za to modny termin, którego używa się ostatnio w bardzo różnorodnych kontekstach. Wielu producentów reklamuje swoje systemy jako otwarte lub wręcz umieszcza słowo open w nazwie produktu (np. *Open Look, Open Desktop*). Firma *Progress Software Corporation* również dąży do tego, by PROGRESS był postrzegany jako system otwarty. Jedno z haseł reklamowych używanych przez PSC brzmi: *Welcome to the Wide Open Spaces of PROGRESS*. Zanim zweryfikujemy zasadność mówienia o PROGRESS-ie jako o systemie otwartym, wymieńmy ważniejsze postulaty, wysuwane pod adresem takich systemów. Jest to suma (a nie iloczyn) warunków występujących w różnych definicjach systemu otwartego.

**A. Spójność** – możliwość łączenia komputerów i innych urządzeń, niezależnie od ich producenta.




Rys. 2. Niezależność aplikacji w PROGRESS-ie od środowiska

**B. Wymiana danych** – swobodny dostęp do danych znajdujących się na dowolnym komputerze w sieci (pracującym pod dowolnym systemem operacyjnym).
**C. Przenośność aplikacji** – możliwość uruchamiania tej samej aplikacji na różnych masznych.
**D. Jednolite umiejętności** – zredukowanie potrzeby nauki języków lub interfejsów specyficznych dla jednego komputera.
**E. Interfejsy programowe** – umożliwienie projektantom aplikacji dostępu do systemu operacyjnego dla zwiększenia efektywności programów.
**F. Skalowalność** – możliwość efektywnej pracy z tą samą aplikacją na różnych, zarówno małych jak i dużych komputerach.
**G. Współpraca różnych produktów programowych** – możliwość współdziałania z innymi aplikacjami działającymi w tym samym lub innym systemie.

Rzadziej używana, choć chyba bardziej adekwatna nazwa systemów otwartych, to systemy otwartych możliwości. Istotnie, system spełniający powyższe warunki otwiera przed użytkownikami nieograniczone możliwości wyboru sprzętu, systemu operacyjnego, rozwiązań sieciowych i oprogramowania (rys. 2). Popatrzmy teraz, na ile system PROGRESS spełnia wymagania stawiane systemom otwartym.

## Praca w sieciach heterogenicznych

PROGRESS pracuje zarówno w sieciach jednorodnych, jak i heterogenicznych, przy czym obsługiwane są protokoły sieciowe: TCP/IP, NetBios, SPX IPX oraz DECnet. Baza danych może być bazą rozproszoną, rezydującą na kilku różnych sprzętowo serwerach połączonych różnymi protokołami komunikacji. Każdy użytkownik sieci ma dostęp do wszystkich zainstalowanych w sieci baz, a rozproszenie jednej bazy między kilka serwerów nie ma wpływu na kod aplikacji. Bezpieczeństwo danych przy transakcjach dotyczących kilku baz zapewnia stosowanie dwufazowego protokołu potwierdzeń. Jeśli potrzeby użytkownika sieci będą powodowały konieczność jej rozbudowy lub modyfikacji, nie będzie to miało wpływu na kod pracującej w sieci aplikacji PROGRESS-owej.

## Przenośność aplikacji napisanych w PROGRESS-ie

Gotową, napisaną w PROGRESS-ie aplikacji można przenieść bez zmian na dowolną platformę sprzętową, na której jest dostępny system PROGRESS – od pojedynczego IBM PC pod DOS-em do AS/400. W obecnie sprzedawanej, szóstej wersji PROGRESS-a jest to przenośność na poziomie kodu źródłowego, natomiast w zapowiadanej na połowę br. wersji siódmej będzie można przenosić programy skompilowane (tzw. r-kod). Tak więc decydując się na PROGRESS nie wiążemy się z żadną konkretną platformą sprzętową.

## Jednolity interfejs z użytkownikiem

Jednym z elementów strategii firmy *Progress Software Corporation* jest zapewnienie identycznego funkcjonowania systemu PROGRESS na wszystkich platfomrach, a więc również jednakowa postać interfejsu z użytkownikiem. Programista pracujący z PROGRESS-em może zmienić komputer, nie zmieniając sposobu pracy ani swoich przyzwyczajeń. Pieniądze raz zainwestowane w szkolenie programistów procentują przez cały okres pracy z systemem PROGRESS.

## Interfejs z systemem operacyjnym

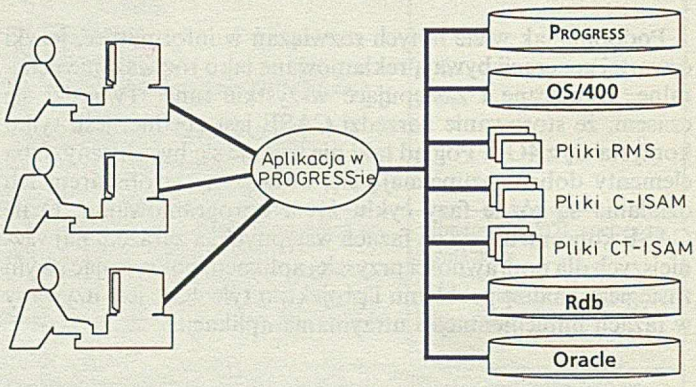
Program napisany w PROGRESS-ie może zawierać wywołania komend systemu operacyjnego. Pozwala to na wykorzystanie specjalnych cech danego systemu.

## Skalowalność systemu PROGRESS

Skalowolność systemu jest wskaźnikiem tego, jak system potrafi efektywnie wykorzystać dodatkowe zasoby sprzętowe. Aby zbadać skalowalność systemu PROGRESS przeprowadzono test TP1 Benchmark kolejno dla czterech, dziesięciu i szesnastu procesorów (test był wykonywany na komputerze Sequent Symmetry 81 z procesorami 80386), przy czym rozmiar bazy danych pozostawał niezmieniony [10]. Test TP1 jest ogólnie uznawanym sprawdzianem wydajności systemu obsługi baz danych (wydajność ta jest mierzona liczbą transakcji na sekundę – TPS). Test symuluje prostą aplikację bankową, przy czym transakcja jest zdefiniowana jako dokonanie wpłaty lub wypłaty z konta bankowego. Przy bazie danych zawierającej 170 mln kont PROGRESS osiągnął liczbę 170 TPS, wykorzystując 16 procesorów (przy 4. i 10. procesorach wydajność wyniosła odpowiednio 60 i 125 TPS). Wydajność aplikacji rosła wraz ze zwiększaniem liczby dostępnych procesorów, co świadczy o tym, że PROGRESS efektywnie wykorzystuje dodatkowe zasoby sprzętowe, a więc jest systemem skalowalnym.

## Współpraca z innymi produktami programowymi

Dążąc do stworzenia wokół systemu PROGRESS bogatego środowiska programów zintegrowanych z PROGRESS-em, firma PSC realizuje tzw. *Open Access Program*, czyli program współpracy z firmami piszącymi takie oprogramowanie. Wspominaliśmy już o pomostach do programów typu CASE; oprócz tego użytkownicy PROGRESS-a już obecnie mogą wybierać spośród wielu współpracujących z bazą PROGRES arkuszy kalkulacyjnych, programów do graficznej prezentacji danych itd.



**Rys. 3. Niezależność aplikacji od bazy danych**

Aplikacje PROGRESS-owe mogą obsługiwać również „obce" bazy danych dzięki istniejącym interfejsom programowym (ang. *gateways*). Dostępne są interfejsy do baz Oracle, Rdb/VMS, Sybase, OS/400, RMS, C-Isam oraz CT-ISAM; planowane jest dalsze rozszerzanie tej listy. Ponadto siódma wersja systemu PROGRESS będzie zgodna z założeniami opracowanej przez Microsft architektury *Open Database Connectivity* (ODBC). ODBC ma w założeniu umożliwić aplikacjom pracującym w środowisku MS-Windows dostęp do dowolnej bazy danych za pośrednictwem modułu o nazwie *Driver Manager*, komunikującego się z driverami dostarczonymi przez producentów poszczególnych baz danych. W ten sposób stanie się możliwe pisanie takiego samego programu w PROGRESS-ie dla różnych obsługiwanych przez ten program baz danych.

## Wersje językowe

PROGRESS charakteryzuje się swego rodzaju otwartością na potrzeby użytkowników z całego świata, posługujących się

**dokończenie na s. 26**

WALDEMAR WIECZERZYCKI
Francusko-Polska Wyższa Szkoła
Nowych Technik Informatyczno-Komunikacyjnych
Poznań

# Zarządzanie współbieżnością transakcji w obiektowych bazach danych

**Nowoczesne systemy informatyczne są budowane wokół systemów baz danych stanowiących ich jądro. Takie systemy dominują dzisiaj w bankowości, księgowości, w zastosowaniach magazynowych, ewidencyjnych, bibliotecznych itp., a ich jądrem są relacyjne bazy danych. Jednocześnie rozszerza się zakres zastosowań baz danych na takie dziedziny, jak: systemy komputerowego wspomagania projektowania i zarządzania (CAD/CAM), systemy informacji biurowej (OIS), systemy komputerowego wspomagania inżynierii oprogramowania itp. Dziedziny te wymagają złożonych struktur danych umożliwiających reprezentowanie schematów, rysunków, obrazów, dźwięków itp. oraz nowych typów związków między nimi, które wykraczają poza zakres modelu relacyjnego. Z myślą o tych dziedzinach gwałtownie rozwijają się w ostatnim okresie obiektowe bazy danych.**

**Obiektowe bazy danych** są połączeniem klasycznych mechanizmów systemów baz danych z obiektowym paradygmatem danych, stosowanym w nowoczesnych językach programowania. U podstaw tego paradygmatu leży pojęcie obiektu. **Obiekt** jest reprezentantem (modelem) stanu i zachownia wyróżnionego fragmentu świata rzeczywistego zwanego **encją.** Obiekt jest zbiorem zmiennych, zwanych **atrybutami,** oraz wykonywanych na nich operacji, zwanych **metodami.** Atrybuty obiektu służą do modelowania stanu encji, a metody do jej zachowania. Paradygmat obiektowy umożliwia elastyczne modelowanie encji dzięki wprowadzeniu związków semantycznych między obiektami, takich jak specjalizacja, agregacja i kompozycja, łatwą rozszerzalność o nowe typy (klasy) obiektów oraz modelowanie encji o dowolnej złożoności dzięki rekurencyjnej definicji obiektu.

Jednym z podstawowych problemów zarządzania obiektowymi bazami danych jest problem zarządzania współbieżnym wykonywaniem transakcji w celu zagwarantowania spójności bazy danych [1]. Intuicyjnie, przez spójność bazy danych rozumiemy poprawność danych i związków między nimi. Spójna baza danych wiernie odwzorowuje stan modelowanego fragmentu rzeczywistości. Zarządzanie współbieżnie wykonywanymi transakcjami polega na nakładaniu określonych ograniczeń na ich dostęp do obiektów. Oczywiście, ograniczenia te powinny być wystarczająco silne, aby zapewnić spójność bazy danych, ale jednocześnie możliwie jak najsłabsze, by nie zmniejszać efektywności wykorzystywania bazy danych. Zarządzanie współbieżnością transakcji w obiektowych bazach danych jest znacznie trudniejsze niż w relacyjnych bazach danych. Wynika to z istnienia powiązań semantycznych między obiektami, których nie ma w relacyjnych bazach danych.

Dotychczas zaproponowano kilka podejść do problemu zarządzania współbieżnym wykonywaniem transakcji. W praktyce, w większości baz danych jest stosowane podejście blokowania [2, 4]. W podejściu tym poprawność współbieżnego wykonania transakcji została uzyskana przez zastosowanie blokad, które najogólniej mówiąc są binarnymi semaforami, związanymi z poszczególnymi danymi przechowywanymi w bazie. Założenie blokady na danej przez określoną transakcję oznacza, że dokonuje ona dostępu do tej danej, a zatem dostęp do niej innej transakcji jest częściowo lub całkowicie ograniczony.

Najczęściej stosowaną metodą blokowania jest metoda hierarchiczna, w której rozwiązano problem właściwego doboru rozmiaru blokowanych jednostek danych [5, 8]. Problem ten wynika ze sprzecznych wymagań efektywnościowych transakcji żądających dostępu do małej lub dużej liczby danych.

Ze względu na specyfikę zarządzania współbieżnością w obiektowych bazach danych, klasyczna metoda blokowania hierarchicznego może być stosowana w tych bazach jedynie w ograniczonym zakresie. Dotychczasowe próby skonsturowania metody blokowania hierachicznego, dostosowanej do wymagań obiektowych baz danych, polegają na rozszerzeniu metody klasycznej o strategię blokowania danych, nie objętych blokowaną jednostką, lecz powiązaniach semantycznie z jej danymi.

W artykule przedstawiono klasyczne metody blokowania dotyczące relacyjnych baz danych oraz sposób ich rozszerzenia dla obiektowych baz danych, zaproponowany przez twórców systemu ORION.

## Spójność bazy danych

System zarządzania bazą danych realizuje elementarne operacje na obiektach. W relacyjnych bazach danych operacją elementarną jest operacja dostępu do jednej lub wielu krotek relacji. Z punktu widzenia systemu wykonania pojedynczej operacji elementarnej stanowi **akcję.** Każda akcja jest atomowa, czyli niepodzielna – może być ona wykonana lub niewykonana, ale nie może być wykonana częściowo. Z każdą bazą danych jest związany zbiór predykatów dotyczących obiektów. Predykaty te są warunkami integralności, które musi spełniać baza danych, aby odwzorowywała jeden z możliwych stanów modelowanego fragmentu rzeczywistości. Stan bazy danych nazywamy stanem **spójnym**, jeśli są spełnione wszystkie zdefiniowane warunki integralności.

Użytkownik manipuluje danymi z bazy za pośrednictwem programów użytkowych, odwołujących się do systemu zarządzania bazą danych. Wykonanie programu użytkowego odpowiada, na poziomie systemu zarządzania bazą danych, wystąpieniu określonej **transakcji.** Transakcję możemy traktować jako funkcję przejścia bazy danych ze stanu $S_1$ inny stan $S_2$. W przypadku transakcji, w czasie której następuje jedynie odczyt danych, $S_1 = S_2$. Zdefiniowane dla bazy danych warunki integralności określają warunki początkowe wykonywania programu użytkowego, a ponadto stanowią niezmiennik programu. Inaczej mówiąc, w wyniku transakcji baza danych przechodzi z jednego stanu spójnego w drugi.

Z danej bazy danych korzysta wielu użytkowników. Ze względu na efektywność systemu jest istotne, aby kilka transakcji było wykonywanych współbieżnie. System zarządzania bazą danych musi więc sprawować kontrolę nad przebiegiem transakcji, nie dopuszczając do wzajemnej blokady poszczególnych transakcji lub stanu niespójnego bazy danych.

Każda z transakcji wykonuje operację odczytu, zapisując w odpowiednich obszarach pamięci operacyjnej wartości pobrane z bazy danych. W przypadku operacji zapisu, określona wartość z pamięci operacyjnej jest przepisywana do bazy danych w miejsce nieaktualnej już wartości z bazy. Tak więc podczas wykonywania, każda z transakcji, dysponuje w każdej chwili określonym obrazem bazy różniącym się lub nie różniącym się od rzeczywistego stanu bazy. Sytuacje **konfliktowe** mogą wystąpić wtedy, gdy dwie transakcje są zainteresowane dostępem do tego samego obiektu.

Z analizy możliwych konfliktów wynika, że podstawowym problemem związanych ze współbieżnym wkonywaniem transakcji jest określenie takiego porządku wykonania akcji poszczególnych transakcji, przy którym jest zachowana spójność danych. Wiemy bowiem, że w przypadku sekwencyjnego wykonywania poszczególnych transakcji, baza danych będzie zawsze w stanie spójnym, bowiem będzie ona przechodziła kolejno, w wyniku wykonania każdej z transakcji, z jednego stanu spójnego w inny stan spójny.

Problem określania poprawnego uporządkowania akcji współbieżnie wykonywanych transakcji jest rozwiązywany za pomocą metod, które możemy podzielić na trzy grupy:
- porządkowania według etykiet czasowych,
- walidacji,
- blokowania.

Jak wspomniano na wstępie artykułu, w praktyce w zdecydowanej większości komercyjnych baz danych są stosowane wyłącznie metody blokowania.

## Podstawowe algorytmy zarządzania współbieżnością

W metodzie blokowania (ang. *locking method*), z każdą daną $x$ jest związana blokada (ang. *lock*). Ściśle biorąc, blokada może być związana z jednostką dostępu do danej, np. stroną pamięci wirtualnej, zawierającą całą daną lub tylko jej część. Zakłada się, że każda transakcja przed odczytem lub zapisem danej musi założyć odpowiednią blokadę na tej danej oraz, że wszystkie założone przez transakcję blokady zostaną zdjęte przed zakończeniem jej wykonywania. Zakładanie blokad danych, do których transakcja żąda dostępu, eliminuje do nich dostęp przez inne transakcje w czasie, gdy ograniczenia integralnościowe mogą być przejściowo naruszone.

Wyróżniamy dwa podstawowe typy blokad: **blokadę współdzieloną** (ang. *shared lock*), którą oznaczamy przez $S$ oraz blokadę wyłączną (ang. *exclusive lock*), którą oznaczamy przez $X$. Operacje na danej nie powodujące jej uaktualnienia (np. odczyt) powinny być poprzedzone założeniem na niej blokady współdzielonej. Operacje uaktualniające daną (np. zapis) powinny być poprzedzone założeniem na niej blokady wyłącznej.

| $L_S$ \ $L_R$ | S | X |
|---|---|---|
| S | *true* | *false* |
| X | *false* | *false* |

**Rys. 1. Tablica zgodności blokad dla klasycznej metody blokowania**

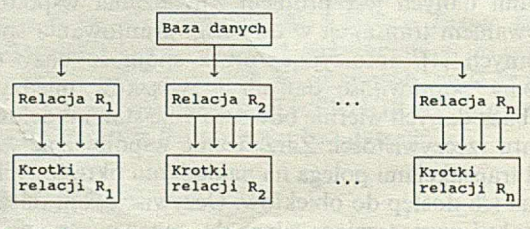
Mówimy, że dwie blokady są **zgodne** (ang. *compatible*), jeżeli mogą być jednocześnie założone na tę samą daną przez dwie różne transakcje. W przeciwnym razie mówimy o blokadach **niezgodnych.** Dla klasycznej metody blokowania tablicę zgodności blokad przedstawiono na rysunku 1.

## Dwufazowy algorytm blokowania

Najszerzej stosowanym w pryktyce algorytmem metody blokowania jest algorytm **blokowania dwufazowego** (ang. *two-phase locking*), oznaczony skrótem 2PL. Istotą tego algorytmu jest wymaganie, aby wykonywanie każdej transakcji przebiegało w dwóch fazach: blokowania oraz odblokowania. W fazie blokowania transakcja musi uzyskać blokady wszystkich danych, do których będzie wykonywać dostęp. Moment założenia wszystkich żądanych blokad, równoznaczny z zakończeniem fazy blokowania, jest nazywany **punktem akceptacji** (ang. *commit point*). Po nim następuje faza odblokowywania. W algorytmie 2PL odczyt danej jest możliwy natychmiast po założeniu blokady tej danej, a więc w fazie blokowania, natomiast zapis jest możliwy dopiero po osiągnięciu przez transakcję punktu akceptacji, a więc w fazie odblokowywania. Ściśle biorąc, w większości baz danych operacja zapisu jest wykonywana w następujący sposób. Założenie blokady wyłączonej jest równoznaczne z wykonaniem tzw. zapisu wstępnego (ang. *prewrite*) w związanym z zapisywaną daną obszarze roboczym. **Zapis właściwy** jest realizowany dopiero w fazie odblokowania, w momencie zdejmowania blokady tej danej na podstawie zawartości obszaru roboczego. Taka procedura gwarantuje atomowość transakcji.

### Hierachiczne algorytmy blokowania

Potrzeba konstruowania hierarchicznych algorytmów blokowania wynika z konieczności rozwiązania tzw. *problemu ziarnistości blokad* (ang. *granularity*). Problem ten polega na doborze rozmiaru blokowanej jednostki, a wynika ze sprzecznych wymagań efektywnościowych transakcji żądających dostępu do małej lub dużej liczby danych. Mówiąc obrazowo, wykonanie transakcji żądającej dostępu do trzech danych wymaga założenia i zdjęcia trzech blokad, a do tysiąca danych – tysiąca blokad.




**Rys. 2. Przykładowa hierarchia ziaren**

Stąd problem określenia optymalnego rozmiaru jednostki blokady. Maksymalizacja współbieżności zbioru małych transakcji wymaga drobnej ziarnistości blokad, natomiast minimalizacja

kosztu blokowania danych dla dużych transakcji wymaga grubej ziarnistości blokad. W pracach [4, 5] zaproponowano koncepcję hierarchicznego blokowania, w której przyjęto hierarchiczną strukturę jednostek blokowania. Zgodnie z tą koncepcją baza danych jest widziana jako hierarchia jednostek blokowania zwanych **ziarnami.** Hierarchię ziaren można przedstawić w postaci acyklicznego grafu skierowanego, którego wierzchołki odpowiadają ziarnom, natomiast łuki określające realcję zawierania się jednych ziaren w drugich. Przykładową hierarchią ziaren przedstawiono na rysunku 2. W hierarchicznych algorytmach blokowania wyróżnia się dwa rodzaje blokad: podstawowe i intencjonalne.

**Blokady podstawowe** są tymi blokadami, które rozważono omawiając niehierarchiczną metodę blokowania. Wyróżniamy zatem blokadę podstawową współdzieloną, oznaczoną przez *S*, oraz blokadę podstawową wyłączną, oznaczoną przez *X*. Różnica w porównaniu z blokadami rozpatrywanymi poprzednio polega na tym, że zakładając blokadę podstawową na określonym poziomie hierarchii ziaren blokujemy implicite wszystkie następniki tej jednostki w hierarchii. Na przykład, blokując relację blokujemy jednocześnie w taki sam sposób wszystkie jej krotki, a blokując bazę danych blokujemy implicite wszystkie relacje należące do niej. Jednakże stosując wyłącznie mechanizm blokad podstawowych i nienaruszalność warunku zgodności typów blokad, nie możemy zapewnić, że w przypadku współbieżnej realizacji transakcji spójność bazy nie zostanie naruszona. W celu zapewnienia poprawności hierarchicznego blokowania danych należy zatem zagwarantować, że po założeniu przez transakcję blokady podstawowej danego wierzchołka *W* w hierarchii ziaren, żadna inna transkcja nie uzyska zgodnie z nią blokady żadnego wierzchołka będącego poprzednikiem *W*.

Jednym z możliwych rozwiązań tego problemu jest rozwiązanie zaproponowane przez Gray'a i in. [5] polegające na wprowadzeniu **blokad intencjonalnych** (ang. *intention lock*). Założenie blokady intencjonalnej na danym poziomie hierarchii ziaren oznacza, że na niższym poziomie jest założona blokada podstawowa. Typy blokad intencjonalnych zależą bezpośrednio od typów blokad podstawowych. Problem określania blokad podstawowych i intencjonalnych oraz ich wzajmnej zgodności był rozważany szczegółowo w pracach Kortha [8].

W modelu Gray'a wyróżniono następujące trzy typy blokad:
- intencjonalną blokadę współdzieloną, oznaczoną przez *IS*,
- intencjonalną blokadę wyłączną, oznaczoną przez *IX*,
- blokadę mieszaną, która jest połączeniem podstawowej blokady współdzielonej i intencjonalnej blokady wyłącznej, oznaczoną przez *SIX*.

| $L_S$ \ $L_R$ | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | true | true | true | true | false |
| IX | true | true | false | false | false |
| S | true | false | true | false | false |
| SIX | true | false | false | false | false |
| X | false | false | false | false | false |

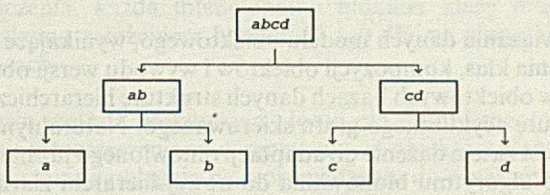
**Rys. 3. Tablica zgodności blokad dla hierarchicznej metody blokowania**

Blokada *IS* wierzchołka *W* sygnalizuje intencję odczytu co najmniej jednego z jego następników. Blokada ta pozwala na zakładnie blokad typu *IS* lub typu *S* na wszystkich następnikach wierzchołka *W* w hierarchii ziaren. Blokada *IX* wierzchołka sygnalizuje intencję zapisu co najmniej jednego z jego następników. Blokada ta pozwala na zakładanie blokad typu *IX, X, IS, S* i *SIX* na wszystkich następnikach wierzchołka *W* w hierarchii ziaren. Blokada mieszana *SIX* wierzchołka *W* pozwala na dostęp współdzielony do danych należących do tego wierzchołka i do wszystkich jego następników oraz pozwala na zakładanie na nich blokad wyłącznych.

Zgodność omówionych wyżej typów blokad przedstawiono na rysunku 3.

Obecnie zostanie przedstawiony hierarchiczny algorytm blokowania dla transakcji *T*. Jest on przeznaczony dla drzewiastej struktury ziaren.

1. Pierwszym wierzchołkiem, do którego *T* żąda dostępu, jest korzeń drzewa ziaren.
2. Transakcja *T* może założyć blokadę *IS* lub *S* wierzchołka nie będącego korzeniem wtedy i tylko wtedy, gdy blokuje jego poprzednika blokadą typu *IS* lub *IX*.
3. Transakcja *T* może założyć blokadę *IX, SIX* lub *X* wierzchołka nie będącego korzeniem wtedy i tylko wtedy, gdy blokuje jego poprzednika blokadą typu *IX* lub *SIX*.
4. Wszystkie blokady założone przez transakcję *T* muszą być zdjęte albo po zakończeniu jej wykonywania, albo w trakcie jej wykonywania w kolejności odwrotnej do kolejności zakładania blokad (tj. od liści do korzenia drzewa).




**Rys. 4. Przykładowa hierarchia ziaren**

Jako przykład zostanie rozważona hierarchia ziaren przedstawiona na rys. 4. Niech transakcja $T_1$ odczytuje daną *b* i zapisuje dane *c* i *d*. W tym celu, zgodnie z powyższym algorytmem. Transakcja $T_1$ zakłada następujące blokady:

*IS (abcd), IS (ab), S (b),*
*IX (abcd), X (cd).*

Załóżmy teraz, że współbieżnie z transakcją $T_1$ są wykonywane transakcje $T_2$ i $T_3$, z których pierwsza zamierza odczytać dane *a* i *b*, natomiast druga – daną *c*. Żądanie transakcji $T_2$ będzie spełnione ze względu na zgodność zakładanych przez nią blokad *IS (abcd)* oraz *S (ab)* z blokadami transakcji $T_1$. W przypadku transakcji $T_3$ jest konieczne założenie następujących blokad: *IS (abcd), IS (cd)* oraz *S (c)*.
Blokada *IS (cd)* jest niezgodna z założoną przez transkację $T_1$ blokadą *X (cd)*. Żądanie transakcji $T_3$ zostanie zatem odrzucone.

## Specyfika zarządzania współbieżności w obiektowych bazach danych

Przedstawione w poprzednim punkcie klasyczne algorytmy zarządzania współbieżnością nie mogą być bezpośrednio zastosowane w obiektowych bazach danych. W szczególności dotyczy to algorytmów blokowania hierarchicznego, które wymagają istotnych rozszerzeń i modyfikacji. Wynika to z faktu przyjęcia w obiektowych bazach danych nowego, bogatszego modelu danych i powiązań sementycznych między nimi. Obiektowy model danych opiera się na pojęciach obiektu i klasy, które nie są odpowiednikami krotki i relacji modelu relacyjnego. Odpowiednikiem krotki byłby obiekt pozbawiony hermetyczności i metod oraz zawierający wyłącznie atrybuty typu prostego. Odpowiednikiem relacji byłby zbiór wyróżnionych lub wszystkich wystąpień określonych poddrzewa hierarchii klas,

z których każde spełnia warunek odpowiedniości w stosunku do krotki. Takie odpowiedniki krotki i relacji nie istnieją w modelu obiektowym.

Powiązania semantyczne między danymi wynikają z dziedziczenia klas i kompozycji obiektów. W modelu relacyjnym krotki i relacje są wzajemnie niezależne. W modelu obiektowym obiekt jest powiązany z jednej strony ze swoimi komponentami i obiektami złożonymi, których jest komponentem, a z drugiej strony – z klasą, której jest wystąpieniem. Z kolei klasa jest powiązana ze swoimi nadklasami i podklasami. Ponadto, w przypadku wielowersyjnych baz danych, obiekt wielowersyjny jest powiązany ze swoimi wersjami, które są także powiązane między sobą, odzwierciedlając sposób ich wywodu.

Wskazane różnice między modelem relacyjnym i obiektowym mają bezpośredni wpływ na metody zarządzania współbieżnością transakcji. W modelu relacyjnym, dostęp transakcji do określonej krotki lub relacji nie wymaga ograniczenia dostępu odpowiednio do innych krotek i innych relacji bazy danych. W modelu obiektowym, dostęp transakcji do obiektu lub klasy w ogólności wymaga ograniczenia dostępu do wszystkich obiektów i klas i nimi powiązanych. Dotyczy to przy tym wszystkich typów powiązań.

Powiązania danych modelu obiektowego, wynikające z dziedziczenia klas, kompozycji obiektów i wywodu wersji obiektów mają w obiektowych bazach danych strukturę hierarchiczną lub strukturę acyklicznego grafu skierowanego. Naturalnym dążeniem jest zatem dążenie do adaptacji omówionego już hierarchicznego algorytmu blokowania do nowej hierarchi ziarnistości i jego rozszerzenie na hierarchię dziedziczenia klas i kompozycji obiektów.

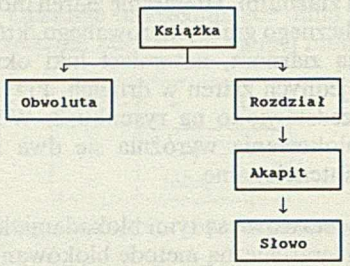
## Zarządzanie współbieżnością w obiektowej bazie danych ORION

W obiektowej bazie danych ORION zarządzanie współbieżnością transakcji odbywa się zgodnie z rozszerzoną metodą blokowania hierarchicznego [3, 6, 7]. W metodzie tej, poza hierarchią ziarnistości, uwzględniono hierarchie dziedziczenia klas i kompozycji obiektów. W odniesieniu do dwóch pierwszych hierarchii: ziarnistości i dziedziczenia, w metodzie stosuje się blokowanie implicite węzłów poddrzew hierarchii. W odniesieniu do hierarchii kompozycji stosuje się jedynie blokowanie explicite.

W hierarchii ziarnistości bazy danych ORION wyróżniono trzy ziarna: bazę danych, klasę i obiekt (wystąpienie klasy). Wersja obiektów nie mogą być niezależnie blokowane. Zatem w przypadku obiektu wielowersyjnego najmniejszym ziarnem blokowania jest cały obiekt wielowersyjny (nazywany obiektem uogólnionym, ang. *generic object*), traktowany jako zbiór wszystkich swoich wersji. W konsekwencji, w metodzie zarządzania współbieżnością nie uwzględniono powiązań między wersjami obiektu, wynikających ze sposobu ich wywodu.

W bazie danych ORION, w hierarchii dziedziczenia dopuszczono możliwość wielodziedziczenia, a zatem możliwość konstrukcji acyklicznego grafu skierowanego dziedziczenia klas. W metodzie blokowania zaproponowano dwa warianty blokowania w hierarchii dziedziczenia: wariant jednorodny, z użyciem wyłącznie blokad jawnych, oraz wariant mieszany z użyciem blokad jawnych i niejawnych.

Hierarchia kompozycji w bazie danych ORION jest powiązaniem dziedzin atrybutów, których wartościami są wskazania kompozycyjne. Dla przykładowego obiektu *Książka* hierarchię kompozycji w rozumieniu bazy danych ORION przedstawiono na rysunku 5. Założono, że obiekty klasy *Obwoluta* są obiektami prostymi (nie mają komponentów), a obiekty klasy *Rozdział* składają się z wystąpienień klasy *Akapit*, które z kolei składają się z wystąpień klasy *Słowo*.



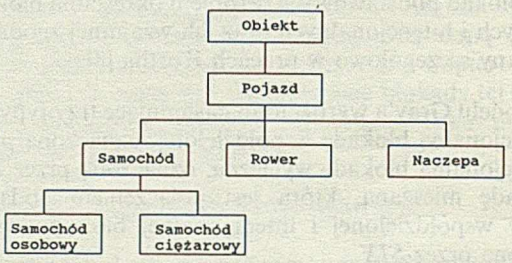

**Rys. 5. Hierarchia kompozycji obiektów klasy „Książka"**

Zamierzeniem projektantów bazy danych ORION było wykorzystanie powiązania kompozycyjnego klas do wprowadzenia blokad umożliwiających niejawne blokowanie wszytkich komponentów obiektu złożonego. Zamierzenie to nie zostało jednak zrealizowane. Zaproponowano cztery strategie blokowania, uwzględniające hierarchię kompozycji, nie oferujące jednak mechanizmu blokad niejawnych w stosunku do hierarchii kompozycji. Stosowane w strategiach nowe typy blokad, zakładanych na klasach obiektów – komponentów, wydatnie zmniejszają liczbę zakładanych blokad.

Poniżej zostaną szczegółowo przedstawione metody blokowania zastosowane w obiektowej bazie danych ORION, najpierw w odniesieniu do hierarchii ziarnistości i dziedziczenia, a następnie z uwzględnieniem hierarchii kompozycji.

### Blokowanie w hierarchii dziedziczenia

W bazie danych ORION zaproponowano dwie alternatywne metody blokowania w hierarchii dziedziczenia: niehierarchiczną i hierarchiczną.




**Rys. 6. Poddrzewo dziedziczenia klas**

Metoda niehierarchiczna polega na zakładaniu jawnych blokad na wszystkich klasach poddrzewa dziedziczenia. Dla zilustrowania tej metody zostanie rozważone poddrzewo dziedziczenia przedstawione na rysunku 6 oraz transakcja, której celem jest wyszukanie wszystkich zielonych pojazdów. Transakcja ta jest uogólnionym zapytaniem dotyczącym całego poddrzewa dziedziczenia. Algorytm zakładania blokad jest dwukrokowy:

1. Ustaw blokady typu *IS* na klasie *Pojazd* i wszystkich jej podklasach;
2. Ustaw blokadę typu *S* na każdym wyselekcjonowanym wystąpieniu.

Hierarchiczna metoda blokowania polega na wprowadzeniu nowych typów blokad. Blokady te umożliwiają jawne zablokowanie pojedynczej klasy oraz niejawne – wszystkich jej podklas. Wyróżniono dwie blokady podstawowe: *R* i *W* oraz dwie

blokady intencjonalne *IR* i *IW*. Blokady podstawowe *R* (ang. *read sublattice*) i *W* (ang. *wirte sublattice*) blokują podgraf dziedziczenia odpowiednio w trybie współdzielonym i wyłącznym. Blokady intencjonalne *IR* i *IW* oznaczają możliwość zablokowania podklasy w trybie podstawowym *R* lub *W*. Algorytm blokowania jest dwukrokowy.

W celu zablokowania klasy *C* w trybie *R* lub *W*:
1. Ustaw blokady, odpowiednie *IR* lub *IW* na wszystkich nadklasach *C;*
2. Zablokuj klasę *C*.

Dla zilustrowania metody hierarchicznej przyjmiemy, że celem transakcji *T* jest dodanie nowej metody do klasy *Samochód* (por. rys. 6). Transakcja *T* jest transakcją modyfikującą schemat bazy danych. Modyfikacja dotyczy obiektu klasowego *Samochód* i wszystkich jego następników w drzewie dziedziczenia, z jednej strony, oraz wszystkich wystąpień poddrzewa klas o korzeniu *Samochód,* z drugiej strony. Po wykonaniu pierwszego kroku algorytmu, klasa *Pojazd* i wszystkie jej nadklasy będą zablokowane w trybie *IW*. Po wykonaniu drugiego kroku klasa *Samochód* będzie zablokowana w trybie *W*.

Konsekwencją wprowadzenia blokad umożliwiających blokowanie poddrzew hierarchii dziedziczenia jest konieczność poprzedzania blokad podstawowych: *X, SIX* i *IX* oraz *S* i *IS* ustawieniem odpowiednio blokad intencjonalnych *IW* oraz *IR* na wszystkich nadklasach blokowanej klasy. Przykładowo, chcąc zablokować pojedyncze wystąpienie klasy *Samochód-osobowy* w trybie *X*, należy ustawić blokady *IW* na klasach *Pojazd* i *Samochód* oraz blokadę *IX* na klasie *Samochód-osobowy*. Zabezpiecza to przed ustawieniem blokady hierarchicznej *R* lub *W* przez inne transakcje na wyższych poziomach drzewa dziedziczenia.

Na rysunku 7 przedstawiono tabelę zgodności nowych typów blokad z blokadami klasycznymi.

| | IS | IX | S | SIX | X | IR | IW | R | W |
|---|---|---|---|---|---|---|---|---|---|
| IS | T | T | T | T | N | T | T | T | N |
| IX | T | T | N | N | N | T | T | N | N |
| S | T | N | T | N | N | T | T | T | N |
| SIX | T | N | N | N | N | T | T | N | N |
| X | N | N | N | N | N | T | T | N | N |
| IR | T | T | T | T | T | T | T | T | N |
| IW | T | T | T | T | T | T | T | N | N |
| R | T | N | T | N | N | T | N | T | N |
| W | N | N | N | N | N | N | N | N | N |

**Rys. 7. Tabela zgodności blokad zakładanych w hierarchiach ziarnistości i dziedziczenia**

Efektywność przedstawionych metod blokowania zależy od pozycji blokowanej klasy *C* w hierarchii dziedziczenia. Jeśli klasa *C* znajduje się blisko ilości grafu, bardziej celowe jest stosowanie metody niehierarchicznej. Unika się wówczas ustawiania blokad intencjonalnych na długiej ścieżce nadklas. Natomiast jeżeli klasa *C* znajduje się blisko korzenia hierarchii dziedziczenia, to bardziej celowe jest stosowanie metody hierarchicznej. Unika się wówczas jawnego blokowania w ogólności wielu podklas klasy *C*.

Niestety, obie metody nie mogą być mieszane. Ostateczny wybór jednej z nich powinien być uwarunkowany badaniami statystycznymi dotyczącymi charakterystyki większej transakcji w systemie.

## Blokowanie w hierarchii kompozycji

W bazie danych ORION obiekt złożony nie jest traktowany jako pojedyncze ziarno blokowania. Blokowanie obiektu złożonego nie jest blokowaniem hierarchicznym, to znaczy dotyczy wyłącznie obiektu będącego korzeniem hierarchii kompozycji, a nie jego komponentów. Analogicznie, blokowanie klasy obiektów złożonych dotyczy klasy obiektów będących korzeniem hierarchii kompozycji, a nie klas obiektów-komponentów.

W bazie danych ORION zaproponowano cztery metody blokowania obiektów złożonych.

Pierwsza metoda polega na zablokowaniu blokadą podstawową obiektu złożonego, poprzedzona założeniem odpowiedniej blokady intencjonalnej na jego klasie, a następnie zablokowaniu blokadą podstawową wszystkich klas, będących dziedzinami jego atrybutów kompozycyjnych, na wszystkich poziomach hierarchii kompozycji. Wadą tej metody jest istotnie ograniczenie stopnia współbieżności transakcji. Ograniczenie to dotyczy głównie klas-dziedzin atrybutów kompozycyjnych.

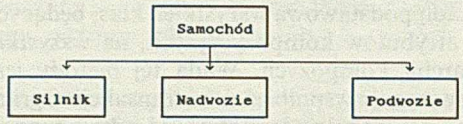
W drugiej metodzie są zakładane blokady podstawowe na obiekcie złożonym, a następnie na wszystkich jego komponentach. Klasy, których wystąpieniami są komponenty muszą być wcześniej zablokowane blokadą intencjonalną. Wadą tej metody jest duża liczba zakładanych blokad. Zauważmy bowiem, że przy równoległym stosowaniu blokad typu *R* i *W* w hierarchii dziedziczenia, każda intencjonalna blokada klasy musi być poprzedzona założeniem blokad *IR* lub *IW* na wszystkich jej nadklasach.

Trzecia metoda blokowania, dotąd nie zaimplementowana, wiąże się z rozszerzeniami w reprezentacji obiektów złożonych. Rozszerzenia te polegają na wprowadzeniu do obiektów-komponentów tzw. wskaźników zwrotnych (ang. *reverse pointers*). Wskaźnik zwrotny wskazuje bezpośrednio na obiekt, w którym dany obiekt jest kompozycyjnie zagnieżdżony. Wielokrotna iteracja po wskaźnikach zwrotnych umożliwia odszukanie obiektu-korzenia hierarchii kompozycji. Algorytm blokowania polega na wprowadzeniu blokad hierarchicznych w hierarchii kompozycji obiektów. Zablokowanie dowolnego obiektu *O*, rozumiane jako zablokowanie niejawne również wszystkich jego komponentów, jest poprzedzane sprawdzeniem, czy blokowany obiekt nie jest z kolei komponentem innego obiektu. Jeśli tak, to przy użyciu wskaźników zwrotnych jest odszukiwany obiekt będący korzeniem hierarchii kompozycji. Następnie są blokowane intencjonalnie wszystkie obiekty na ścieżce kompozycji, od korzenia do blokowanego obiektu *O*. Rozwiązanie to gwarantuje maksymalny stopień współbieżności transakcji. Ma ona następujące istotne wady:

- rozszerzenie reprezentacji każdego obiektu o wskaźnik zwrotny wprowadza istotny narzut implementacyjny;
- blokady typu hierarchicznego w hierarchii kompozycji dotyczą wyłącznie pojedynczych obiektów, tzn. nie są dostępne na poziomie klas obiektów złożonych. Wynika to z braku możliwości implementacji wskaźników na poziomie klas – dziedzin atrybutów;
- wskazania kompozycyjne są wskazaniami typu wyłącznego wprowadzenie wskazań współdzielonych połączone byłoby z koniecznością wiązania z reprezentacją obiektu list wskaźników.

Czwarta, ostatnia metoda blokowania jest kompromisem między dążeniem do minimalizacji liczby blokad, a przeciwstawnym mu dążeniem do maksymalizacji stopnia współzależności transakcji. W metodzie tej wprowadzono nowe typy blokad, zakładanych na dziedzinach atrybutów kompozycyjnych. Specyfika nowych blokad polega na tym, że nie ograniczają one dostępu do obiektów będących wystąpieniami klas-dziedzin atrybutów kompozycyjnych innym transakcjom, pod warunkiem, że dostęp ten jest realizowany pośrednio, tzn. przez wskazania kompozycyjne z innego obiektu. Przykładowo roz-

ważmy hierarchię klasy *Samochód* przedstawioną na rysunku 8. Transakcja $T_1$, zakładająca blokadę podstawową na pojedynczym „samochodzie" i nowe blokady na dziedzinach wszystkich jego atrybutów kompozycyjnych: *Silnik, Podwozie* i *Nadwozie*, nie wchodzi w konflikt z transakcją $T_2$ blokującą inny samochód. Konflikt występuje natomiast w przypadku transakcji $T_3$ realizującej bezpośredni dostęp do klas-dziedzin atrybutów kompozycyjnych oraz transakcji $T_4$, realizującej dostęp konkretnych wystąpień tych klas. Nie jest zatem możliwy bezpośrednio dostęp do klasy *Silnik*, jak również do pojedynczych wystąpień klasy *Podwozie*.




**Rys. 8. Hierarchia kompozycji klasy „Samochód"**

Prezentowana metoda jest satysfakcjonująca w przypadku blokowania klasy obiektów złożonych. Blokowanie pojedynczego obiektu złożonego wyraźnie ogranicza stopień współbieżności transakcji. Jest to widoczne w prezentowanej poniżej tabeli zgodności nowych blokad (rys. 9). Charakteryzują się one dużym stopniem restryktywności w porównaniu do blokad podstawowych.

| | IS | IX | S | SIX | X | ISO | IXO | SIXO |
|---|---|---|---|---|---|---|---|---|
| IS | T | T | T | T | N | T | N | N |
| IX | T | T | N | N | N | N | N | N |
| S | T | N | T | N | N | T | N | N |
| SIX | T | N | N | N | N | N | N | N |
| X | N | N | N | N | N | N | N | N |
| ISO | T | T | T | T | T | T | T | T |
| IXO | T | T | T | T | T | T | T | N |
| SIXO | T | N | T | N | N | T | N | N |

**Rys. 9. Tabela zgodności blokad zakładanych w hierarchii kompozycji**

Nowe blokady: *ISO, IXO, SIXO* są zakładane na wszystkich klasach-dziedzinach atrybutów kompozycyjnych. Założenie każdej z nich musi być poprzedzone założeniem odpowiednio blokady: *S, X* lub *SIX* na obiekcie złożonym. Przy równoległym stosowaniu blokad *R* i *W* w hierarchii dziedziczenia, każda z nowych blokad musi być ponadto poprzedzona założeniem odpowiednich blokad intencjonalnych na nadklasach klas będących dziedzinami atrybutów.

W przypadku, gdy wskazania kompozycyjne mogą być wskazaniami współdzielonymi, metoda ulega rozszerzeniu. Konieczne jest dodatkowe ograniczenie dostępu do klas będących dziedzinami atrybutów kompozycyjnych. Pośrednie odwołanie się transakcji $T_1$ przez wskazanie kompozycyjne może bowiem dotyczyć tego samego obiektu, co odwołanie pośrednie transakcji $T_2$. W bazie danych ORION dla klas będących dziedzinami współdzielonych atrybutów kompozycyjnych wprowadzono trzy kolejne blokady: *ISOS, IXOS. SIXOS*, będące odpowiednikami prezentowanych wcześniej blokad: *ISO, IXO, SIXO*. W stosunku do swoich odpowiedników, blokady te są znacznie bardziej restryktywne.

LITERATURA

[1] Cart M., Ferrie J.: Integrating Concurrency Control into an Object-Oriented Databasse System. EDBT Proc., Venice, Italy, 1990
[2] Cellary W., Gelenbe E., Morzy T.: Concurrency Control in Distributed Database Systems. North-Holland, Amsterdam, 1988
[3] Garza J., Kim W.: Transaction Management in an Object-Oriented Database System. ACM SIGMOD Conf., 1988
[4] Gray J.: Notes on Database Operating Systems. Operating Systems: An Advance Cource, Springer-Verlag, 1978
[5] Gray J. N., Lorie R. A., Putzolu G. R., Traiger I. L.: Granularity of Locks and Degrees of Consistency in a Shared Data Base. RJ1654, IBM Res.Lab., San Jose, 1975
[6] Kim W., et al.: Composite Object Support in an Object-Oriented Database System. Proc. of Conf. on Object-Oriented Programming Systems, Languages and Applications, Orlando, Florida, 1987
[7] Kim W., Bertino E., Garza J. F.: Composite Objects Revisited. Proc. ACM SIGMOD Conf., 1989
[8] Korth H. F.: Locking Primitives in a Database System Journal of the ACM 30, 1, 1983.

## Ze świata

# Nowe mikroprocesory

W prasie pojawiają się doniesienia na temat uściślenia parametrów oraz terminu wprowadzenia do sprzedaży nowych mikroprocesorów. Firma IBM ogłosiła, że na swój procesor nowej generacji, nazywany dotychczas P5 lub 586, przyjmie nazwę Pentium. Przewiduje się, że będzie on 5–10 razy szybszy od mikroprocesora 486DX/33. Układ ten ma zawierać 3,2 mln tranzystorów, został zrealizowany w technologii 0,8 $\mu$m i jest zasilany napięciem 5 V. Szyna zewnętrzna układu jest 32-bitowa. Spodziewany termin wprowadzenia na rynek, to koniec I kwartału br. Bezwzględną szybkość działania procesora ocenia się na 100 MIPS przy początkowej częstotliwości zegara 66 MHz.

Natomiast firma Motorola ogłosiła wstępne dane dotyczące konkurenta Pentium – swego nowego mikroprocesora o symbolu 68060. Ma on zachować pełną zgodność oprogramowania z innymi procesorami serii 68000. Rozpoczęcie produkcji porzewidziano na pierwszą połowę 1994 r. Zakłada się ok. 3,5-krotne zwiększenie szybkości, w porównaniu do układu 68040 o częstotliwości zegara 25 MHz, oraz uzyskanie szybkości ok. 77 MIPS przy początkowej częstotliwości zegara 50 MHz, ale rozpatruje się możliwość dorównania pod tym względem parametrom Pentium. Kostka ma zawierać 2 mln tranzystorów, przy czym zakłada się zastosowanie nowszej niż w IBM technologii 0,5 $\mu$m. Układ ma działać zarówno przy zasilaniu 5, jak i 3,3 V.

Jeśli chodzi o architekturę, to zastosowano tu czterostopniowy system potokowy, przy czym pamięć notatnikowa (ang. *cache*) może przechowywać 256 rozkazów. Ocenia się, że 50–60% rozkazów będzie zrealizowane równolegle. Wskaźnik ten powinien zwiększyć się po zastosowaniu zoptymalizowanego kompilatora.

Mikroprocesor 68060 ma trzy podzespoły arytmetyczne: sumator oraz układy mnożący i dzielący. Dla liczb całkowitych wykorzystują one pierwszy układ potokowy. W układzie tym są realizowane również rozkazy zmiennego przecinka, ale wówczas obliczenia liczb całkowitych są przesyłane do drugiego układu potokowego. Cykle obu rodzajów rozkazów mogą zachodzić na siebie. Ładowanie oraz pamiętanie zmiennoprzecinkowe zajmuje jeden cykl zegarowy, dodawanie – trzy cykle, mnożenie – cztery, a dzielenie – 24 cykle. Ponadto procesor 68060 ma pamięć notatnikową o pojemności 32 kB dla rozkazów i danych oraz 32-bitową szynę dla danych i adresów, podobną do szyny w procesorze 68040.

W oczekiwaniu na procesor 68060, największy klient Motoroli – firma Apple będzie wykorzystywała do realizacji nowych generacji swych komputerów również mikroprocesory RISC. Procesory tego typu są opracowywane wspólnie przez firmy Intel i Motorola pod nazwą PowerPC.

JAN RYŻKO

MAREK MIŁOSZ
Katedra Informatyki
Politechnika Lubelska

# Mikrokomputerowa symulacja systemów – GPSS/PC

**Rozwój techniki mikrokomputerowej spowodował rozkwit oprogramowania przeznaczonego do bezpośredniego wykorzystania przez użytkowników. Wciąż rozszerzający się rynek zbytu na edytory tekstów, arkusze kalkulacyjne, bazy danych, programy graficzne lub wspomagania rysowania jest siłą napędzającą ich rozwój i stałe doskonalenie. Analogicznie ma się sprawa z popularnymi językami programowania. Praktycznie co drugi użytkownik mikrokomputerów nazywa siebie programistą i ma kompilator języka Pascal lub C. Niektóre obszary zastosowań mikrokomputerów pozostają w chwili obecnej jakby w cieniu tej popularnej informatyki. Należy do nich symulacja systemów. Czy słusznie?**

Historia symulacji cyfrowej systemów jest ściśle związana z histroią komputerów. Można z małym błędem stwierdzić, że pierwszy komputer powstał po to, by na nim symulować system rzeczywisty – reakcje jądrowe. Również późniejszy rozwój maszyn liczących wskazuje na ich silny związek z symulacją.

## Cyfrowa symulacja systemów

Lata sześćdziesiąte naszego wieku to nie tylko rozwój dużych, stacjonarnych systemów komputerowych, ale i rozwój języków symulacyjnych. Równolegle z komputerami IMB 360, CDC 6000 czy też PDP-10 powstają: NSS, OPS, GPSS, CSL, GASP, SEAL, Simpac, Simscript, Simula67, Siman, Slang, Sol, SPL, GSP, Midas, DES-1, DSL/90, Mimic, CSMP, ACSL, ANALOG, CSSL, GODYS, DYNAMO, SSPC, ...

Rozwija się także teorie symulacji cyfrowej systemów. Należy wspomnieć chociażby o takich przełomowych dziełach jak „System Simulation – The art and Science" Shannona, „Principles of Discrete Event Simulation" Fishmana czy też „System Simulation" Gordona. Po fali tłumaczeń [np. FIS81, GOR74] i polskich prac w latach siedemdziesiątych [7, 8, 9] zainteresowanie symulacją systemów w naszym kraju jakby osłabło. Chlubnym wyjątkiem jest tutaj praca [12].

Taki stan rzeczy powoduje dalszy spadek zainteresowania tą metodą badawczą oraz zmniejszenie się liczby ludzi umiejących się nią sprawnie posługiwać i stosować w praktyce. Dzieje się to w chwili, kiedy zmiany w gospodarce kraju stwarzają warunki i konieczność stosowania metod symulacyjnych do badań systemów, wspomagania procesu projektowania i podejmowania decyzji.

Symulacja komputerowa jest wygodnym narzędziem badawczym. Niezależnie od tego, czy obiektem zainteresowań jest praca reaktora jądrowego, lot rakiety kosmicznej do innej galaktyki, gospodarka państwa czy też funkcjonowanie sklepu, symulacja komputerowa jest bezpieczniejszym, tańszym, bardziej efektywnym i często jedynym możliwym do przeprowadzenia eksperymentem badawczym. Jest to eksperyment z dynamicznym (przeważnie) modelem systemu.

W zależności od celu badań i rodzaju badanego procesu określany jest typ modelu [6]. Podstawową klasyfikacją modeli jest z jednej strony ich podział na ciągłe, dyskretne oraz mieszane (hybrydowe), z drugiej strony – na deterministyczne i stochastyczne, a z trzeciej – na statyczne i dynamiczne. Dla tego samego systemu rzeczywistego można (w zależności od celu badań) budować modele różnego typu.

W modelach dyskretnych zmiana stanu odbywa się skokowo w ściśle określone momenty czasu. Pozwala to śledzić zmiany w modelu tylko w wybranych momentach czasu i całkowicie pomijać okresy pomiędzy nimi. Modele dyskretne doskonale nadają się do badań systemów organizacyjnych, produkcyjnych i niektórych technicznych (np. niezawodnościowych).

Symulacja komputerowa systemów dyskretnych jest wykorzystywana na etapie badania, projektowania i eksploatacji systemów. Dlatego też jej metody i narzędzia powinien poznać zarówno pracownik nauki, projektant, jak i zwykły inżynier.

Budowa modeli symulacyjnych i ich komputerowa realizacja jest procesem trudno poddającym się formalizacji. W zasadzie jest to sztuka. Sztukę tę pomagają opanować i efektywnie stosować w praktyce języki symulacyjne. Jednym z nich jest GPSS.

## Rys historyczny

GPSS (*General Purpose Simulation System*) jest językiem o strukturze blokowej, pozwalającym opisywać i symulować dynamiczne systemy dyskretne (stochastyczne lub deterministyczne) w postaci procesów quasi-równoległych. Język ten wykorzystuje metodę interakcji procesów [12]. Pod nawzą GPSS, oprócz języka do opisu modeli systemów, kryje się translator i program sterujący procesem symulacji.

GPSS powstał w 1961 r. w laboratoriach firmy IBM. Jego twórcą był Geoffery Gordon. Do najważniejszych implementacji GPSS na dużych systemach cyfrowych należy zaliczyć GPSS/360 i GPSS V (dialekty opracowane przez IBM) GPSS/ /Norden, GPSS/UCC, GPSSTS i GPSS-10 [10].

W świat mikrokomputerów GPSS wszedł dzięki firmom MINUTEMAN Software (wersja: GPSS/PC dla mikrokomputerów typu IBM/PC), Simulation Software (wersja: GPSSR/PC dla mikrokomputerów typu IBM/PC i różnych minikomputerów, a także stacji roboczych z systemem operacyjnym UNIX) oraz Wolverine Software Corporation (wersja: GPSS/H dla mikrokomputerów typu IBM PC, a także stacji roboczych

VAX/VMS i UNIX). Do interesujących implementacji języka należy zaliczyć mikro-GPSS, opracowany przez profesora I. Ståhla ze Sztokholmskiej Akademii Ekonomicznej. Mikro-GPSS jest dostępny w wersji dla IBM/PC, Macintosh i VAX/VMS.

Większość wersji GPSS ma interpretator języka. Wyjątkiem jest GPSS/H, który został zaopatrzony w kompilator zwiększający szybkości symulacji. GPSS/H nie ma jednak możliwości pracy z modelem w trybie interaktywnym, co znacznie utrudnia uruchamianie programów.

## Podstawy języka GPSS

Zdarzenia dyskretne, zachodzące w realnym systemie, w języku GPSS są odwzorowywane za pomocą ruchu obiektów-zadań (ang. *transaction*) między blokami w symulowanym czasie (czasie modelowym). Każdy z bloków wpływa na zmianę stanu określonego zadania lub grupy zadań. Kolejność przepływu zadań jest wyznaczana przez naturalne uporządkowanie bloków. Istnieją bloki kierujące zadania w inne miejsce modelu, w zależności od spełnienia się określonego warunku, lub też przypadkowo (z zadanym prawdopodobieństwem).

System GPSS ma wbudowany zegar, który odmierza upływ w czasu modelowego.

Model (a właściwie program) w języku GPSS składa się z segmentów, w których zachodzą równolegle (mierząc wg czasu modelowego) określone procesy. Oznacza to, że w każdej części programu jest ten sam czas modelowy. Każdy segment rozpoczyna się blokiem *GENERATE*, który tworzy zadania, a kończy się przeważnie blokiem *TERMINATE*, usuwającym je z modelu. Poszczególne segmenty modelu mogą imitować różne zjawiska, a zadania w nich – różne obiekty rzeczywiste.

Za pomocą parametrów bloku *GENERATE* można określić:
- rozkład statystyczny odstępów czasowych między tworzonymi zadaniami;
- opóźnienie generowania pierwszego zadania;
- maksymalną liczbę tworzonych zadań;
- priorytet zadania.

Oprócz poziomu priorytetu, z każdym zadaniem może być związany zbiór parametrów numerycznych.

Czas przejścia zadania przez blok (oprócz specjalnego bloku opóźniającego *ADVANCE*) jest równy zero.

Zadanie w swojej drodze może zostać zatrzymane przez:
- blok opóźniający *ADVANCE* (na ściśle określony czas);
- występienie braku możliwości wejścia zadania do następnego bloku (np. blokada lub zajęcie urządzenia, niespełnienie warunku itp.);
- czasowe usunięcie zadania z modelu.

Do najważniejszych bloków sterujących przepływem zadań w GPSS należą:

*TRANSFER* – blok skoku, który może pracować w trybie skoku bezwzględnego, satystycznego, warunkowego, przełącznikowego funkcjonalnego, parametrycznego, do podprogramu itd;
*LOOP* – blok organizacji pętli;
*TEST* – blok warunkowy, testujący warunek arytmetyczny;
*GATE* – blok przesunięcia zadania w zależności od wartości określonego wskaźnika logicznego, określającego stan poszczególnych elementów modelu.

Podstawowymi elementami języka GPSS, służącymi do odwzorowywania systemów masowej obsługi są: urządzenia(ang. *facility*) i magazyny (ang. *storage*).

Urządzenie jest elementem, w którym w danym momencie czasu może znajdować się tylko jedno zadanie. Wejście do urządzenia (zajęcie) odbywa się za pomocą bloku *SEIZE*. Wejście (za pomocą bloku *SEIZE*) innego zadania do zajętego już urządzenia jest niemożliwe. Próba dokonania tego spowoduje zatrzymanie (zawieszenie) zadania w ruchu na poprzednim bloku. Po zwolnieniu urządzenia (blok *RELEASE*), ruch zadania zawieszonego w ten sposób jest wznawiany automatycznie. Urządzenia mogą być także blokowane (blokowanie: *FUNAVAIL*, zwalnianie: *FAVAIL*)lub okupowane (okupowanie: *PREEMPT*, uwolnienie: *RETURN*).

Magazyn jest jakby grupą urządzeń o określonej liczności (pojemności). Miejsca w magazynie (urządzenia w grupie) są identyczne i nierozróżnialne. Za pomocą bloku *ENTER* zadanie może zająć określoną liczbę miejsc w magazynie (zwolnienie: blok *LEAVE*), pod warunkiem, że żądana liczba miejsc jest wolna. W przeciwnym przypadku zadanie oczekuje w bloku poprzednim na zmniejszenie zajętości magazynu. W dowolnym momencie czasu modelowego zajętość magazynu nie może bowiem przekroczyć jego pojemności. Dostępne jest blokowanie (*SUNAVAIL*) oraz zwalnianie blokady (*SAVAIL*) magazynu.

W miejscach warunkowego zatrzymania zadań tworzą się ich kolejki. Są one obsługiwane zgodnie ze standardową procedurą FIFO (ang. *First-In, First-Out*) w ramach każdego poziomu priorytetu zadań. Zmiana strategii obsługi kolejki jest możliwa przez przeniesienie zadań na listę użytkownika (ang. *user chain entity*) za pomocą bloku *LINK* i wprowadzenie do modelu przez blok *UNLINK*. Lista użytkownika może być uporządkowana wg strategii *FIFO, LIFO* (ang. *Last-In, First-Out*) lub wg wartości określonego parametru zadania. Umożliwia to realizację praktycznie dowolnych strategii obsługi kolejek.

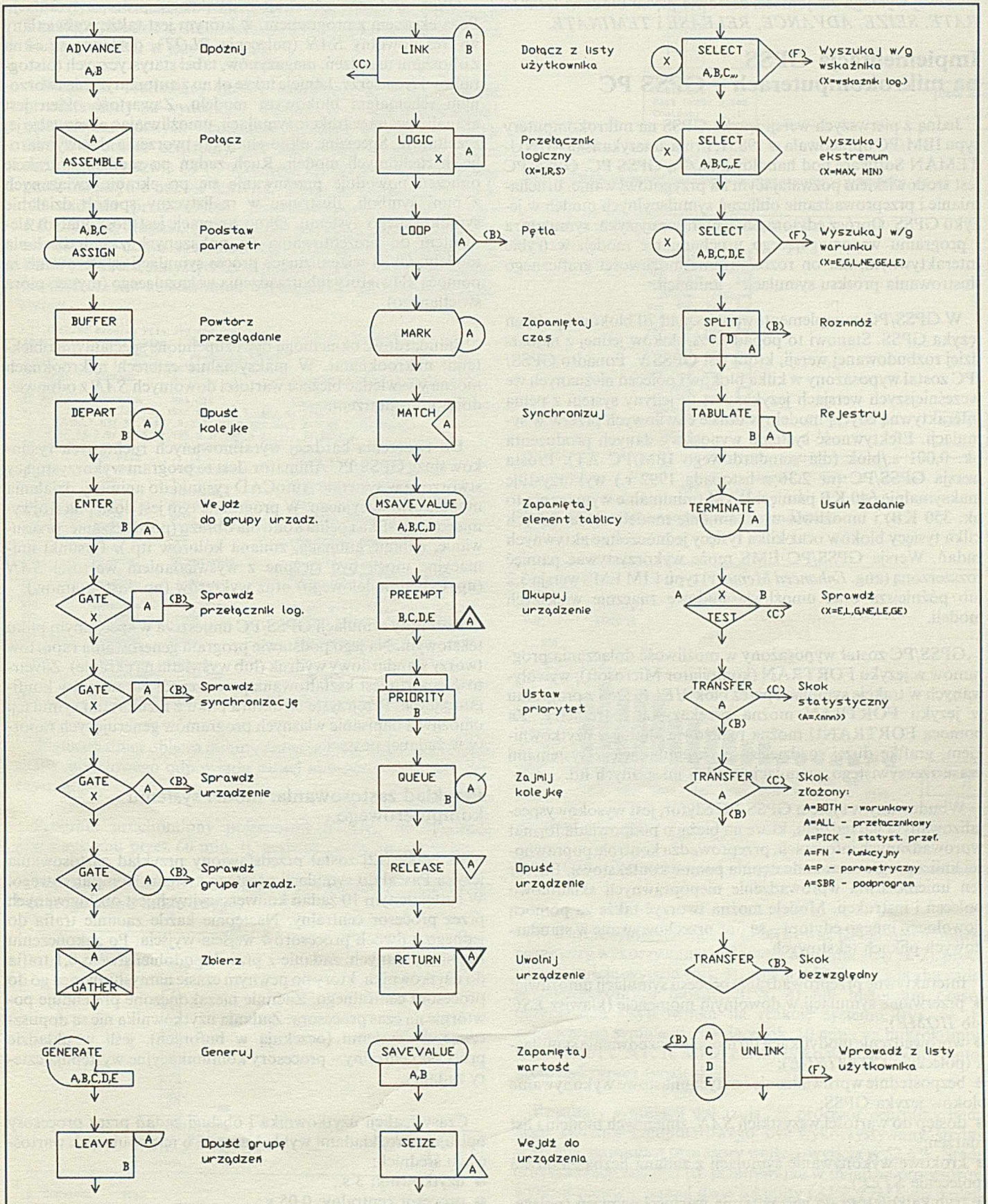
W GPSS istnieje pojęcie rodziny zadań (ang. *related transaction*). Rozmnożone w bloku *SPLIT* zadania mogą być łączone w jedno (blok *ASSEMBLE*), zbierane w jednym miejscu (blok *GATHER*) lub synchronizowane w różnych miejscach modelu (*MATCH*). Nieco bardziej skomplikowane operacje można dokonywać na zadaniach należących do tzw. grup (ang. *transaction group*).

GPSS jest wyposażony w cały arsenał środków służących do zbierania rezultatów symulacji. Niektóre dane (np. parametry pracy urządzeń, magazynów czy też poszczególnych bloków) są zbierane w sposób automatyczny. W celu rejestracji parametrów kolejek są tworzone specjalne obiekty-kolejki (ang. *queue*). Blok *QUEUE* określa miejsce wejścia zadania do kolejki (z zajęciem w niej określonej liczby miejsc), natomiast *DEPART* – punkt opuszczenia kolejki. Obiekty zwane kolejkami zbierają uśrednione dane (np. maksymalna, bieżąca i średnia długość kolejki, średnie czasy pobytu zadań w kolejce). Za pomocą deklaracji *QTABLE* można zdeklarować dla danej kolejki tabelę, w której będzie autoamtycznie tworzony histogram czasu pobytu zadań w kolejce.

Analogiczne tabele (deklaracja *TABLE* i blok rejestracji danych *TABULATE*) można tworzyć dla dowolnego systemowego artybutu numerycznego (ang. *System Numerical Attribute*). Atrybuty te są dostępne w każdym miejscu modelu i zawierają dane o praktycznie wszystkich jego elementach: od czasu modelowego, poprzez parametry zadań, kolejek, urządzeń magazynów, aż do funkcji, generatorów liczb losowych, zmien-

nych prostych (arytmetycznych i logicznych) i tablicowych, czy też przełączników logicznych i zmiennych pamięciowych. Systemowe atrybuty numeryczne (SAN) są wykorzystywane do budowy modeli, sterowania przebiegiem symulacji oraz zbierania jej rezultatów.

Blokowy charakter języka GPSS znalazł odbicie w bardzo przejrzystym sposobie odwzorowania modeli systemów za pomocą schematów blokowych. Dla każdego bloku języka został opracowany specjalny symbol graficzny. Większą część z nich przedstawia rys. 1.




**Rys. 1. Symbole stosowane w schematach blokowych języka GPSS**

Ten krótki przegląd elementów języka GPSS oczywiście nie wyczerpuje wszystkich jego możliwości w zakresie budowy modeli symulacyjnych. Łatwość interpretacji funkcji poszczególnych bloków oraz sposobu symulacji pozwala tworzyć dość skomplikowane modele już po poznaniu kilku – kilkunastu bloków. Najprostszy model jednokolejkowego systemu obsługi masowej wymaga znajomości zaledwie pięciu bloków: *GENERATE, SEIZE, ADVANCE, RELEASE* i *TEMINATE*.

## Implementacja GPSS na mikrokomputerach – GPSS PC

Jedną z pierwszych wersji języka GPSS na mikrokomputery typu IBM/PC opracowała w 1985 r. firma amerykańska MINUTEMAN Software pod handlową nazwą GPSS PC. GPSS/PC jest środowiskiem pozwalającym na przygotowywanie, uruchamianie i przeprowadzanie obliczeń symulacyjnych modeli w języku GPSS. Oprócz edytora, interpretatora języka, symulatora i programu wspomagającego uruchamianie modeli w trybie interaktywnym, ma on rozbudowane możliwości graficznego ilustrowania procesu symulacji – animacji.

W GPSS/PC zaimplementowano ponad 70 bloków i poleceń języka GPSS. Stanowi to ponad 95% bloków jednej z najbardziej rozbudowanej wersji, którą jest GPSS V. Ponadto GPSS/PC został wyposażony w kilka bloków i poleceń nieznanych we wcześniejszych wersjach języka. Jest to jedyny system z pełną interaktywną edycją modelu w czasie chwilowych przerw w symulacji. Efektywność systemu wynosi wg danych producenta ok. 0,001 s./blok (dla standardowego IBM/PC AT). Prosta wersja GPSS/PC (nr 2.36 z listopada 1992 r.) wykorzystuje maksymalnie 640 KB pamięci RAM (minimalne wymagania to ok. 350 KB) i umożliwia uruchamianie modeli zawierających kilka tysięcy bloków oraz kilka tysięcy jednocześnie aktywnych zadań. Wersja GPSS/PC EMS może wykorzystywać pamięć rozszerzoną (ang. *Enhanced Memory*) typu LIM EMS wersja 3.2 lub późniejsza oraz umożliwia budowę znacznie większych modeli.

GPSS/PC został wyposażony w możliwość dołączania programów w języku FORTRAN (kompilator Microsoft), wywoływanych w trakcie symulacji przez blok *HELP*. Do i z programu w języku FORTRAN można przekazywać parametry. Za pomocą FORTRANU można realizować dialog z użytkownikiem, grafikę dużej rozdzielczości, komunikację z systemami czasu rzeczywistego, symulację modeli mieszanych itd.

Wbudowany w system GPSS/PC edytor, jest wysokowyspecjalizowanym narzędziem, które na bieżąco podpowiada format wprowadzanych informacji, przeprowadza kontrolę poprawności linii z programem i udostępnia pomoc kontekstową. Edytor ten uniemożliwia wprowadzenie niepoprawnych składniowo poleceń i instrukcji. Modele można tworzyć także za pomocą dowolnego innego edytora – są one przechowywane w standardowych plikach tekstowych.

Interaktywne przeprowadzanie procesu symulacji umożliwia:
- przerwanie symulacji w dowolnym momencie (klawisz *ESC* lub *HOME*);
- wprowadzenie modyfikacji do modelu i wznowienie symulacji (polecenie *CONTINUE*);
- bezpośrednie wprowadzanie i natychmiastowe wykonywanie bloków języka GPSS;
- dostęp do wartości wszystkich *SAN*, zmiennych modelu i list zdarzeń;
- krokowe wykonywanie symulacji z zadaną liczbą zadarzeń (polecenie *STEP*);
- tryb kalkulatora do wyznaczania wartości wyrażeń (polecenie *SHOW*);
- wyświetlanie wykresu zmian wartości dowolnego *SAN* w trakcie symulacji (polecenie *PLOT*);
- śledzenie przebiegu symulacji (polecenia *TRACE* i *UNTRACE*);
- dynamiczne przełączanie się między oknami graficznymi.

Okna graficzne zawierają wizualizację procesu symulacji. Poza ekranem z programem, w którym jest także wyświetlany wykres dowolny *SAN* (polecenie *PLOT*), dostępne są okna z obrazami urządzeń, magazynów, tabel statystycznych (histogramów) i macierzy. Istnieje także okno z automatycznie tworzonym schematem blokowym modelu. Zawartość okien jest aktualizowana w trakcie symulacji, umożliwiając obserwację jej rezultatów. Specjalne okno służy do tworzenia dwuwymiarowych ruchomych modeli. Ruch zadań po modelu w trakcie obliczeń powoduje przesuwanie się po ekranie związanych z nimi symboli, ilustrując w realistyczny sposób działanie symulowanego systemu. Okno animacji jest doskonałym elementem do prezentowania na pokazanych zasad działania modelu. Okna wizualizujące proces symulacji są sterowane za pomocą klawiatury lub urządzenia wskazującego (myszy, pióra świetlanego).

Standardowe okna mogą być uzupełnione specjalnymi obiektami: mikrooknami. W maksymalnie czterech mikrooknach można wyświetlać bieżące wartości dowolnych *SAN* z odpowiednim komentarzem.

Do tworzenia bardziej wyrafinowanych ruchomych rysunków służy GPSS/PC Animator. Jest to program wykorzystujący stworzone w systemie AutoCAD rysunki do animacji działania modelu symulacyjnego. W programie tym jest dostępna trójwymiarowa grafika i pełna kontrola obrazu (powiększanie, przesuwanie, cofanie animacji, zmiana kolorów itp.). Rysunki animacyjne mogą być łączone z wyświetlaniem wartości *SAN* (np. czasu modelowego) oraz wykresów (np. histogramów).

Rezultaty symulacji GPSS/PC umieszcza w specjalnym pliku tekstowym. Na jego podstawie program generowania raportów tworzy standardowy wydruk (lub wyświetla na ekranie). Zawartość raportu jest kształtowana przez zespół parametrów konfigurujących. Przejrzysta struktura pliku z rezultatami symulacji umożliwia napisanie własnych programów generujących raporty w żądanej postaci.

### Przykład zastosowania: model systemu komputerowego

W pracy [12] został przedstawiony przykład zastosowania języka Pascal do symulacji prostego systemu komputerowego. W systemie tym 10 zadań konwersacyjnych jest obsługiwanych przez procesor centralny. Następnie każde zadanie trafia do jednego z dwóch procesorów wejścia-wyjścia. Po zakończeniu transmisji danych, zadanie z prawdopodobieństwem 0,1 trafia do użytkownika, który po pewnym czasie namysłu zwraca go do procesora centralnego. Zadanie niezakończone pretenduje powtórnie na czas procesora. Zadania użytkownika nie są dopuszczane do systemu (oczekują w buforach), jeśli w układzie procesor centralny – procesory komunikacyjne występują cztery zadania.

Czasy reakcji użytkownika i obsługi zadań przez procesory opisują się rozkładami wykładniczymi o następujących wartościach średnich:
- użytkownik: 3 s.;
- procesor centralny: 0,05 s.;
- procesor wejścia-wyjścia: 0,06 s.

Symulację systemu należy przeprowadzić dla 60 minut czasu pracy systemu. Należy wyznaczyć parametry wykorzystania mocy obliczeniowych (zajętości) poszczególnych elementów systemu, a także rozkład zmiennej losowej opisującej czas oczekiwania użytkownika na reakcję systemu.

Na tym, nieznacznie zmienionym w stosunku do opisanego w [12] przykładzie zostanie zilustrowana naturalność i prostota budowy modeli w języku GPSS.

Program, realizujący model symulacyjny systemu został przedstawiony na wydruku 1. Zawiera on, poza deklaracjami i definicjami różnych elementów, zaledwie 17 bloków języka GPSS. Procesor centralny jest imitowany w modelu przez urządzenie o nazwie *CPU*, natomiast procesory wejścia-wyjścia przez magazyn o nazwie *KANAL*. Zliczanie parametrów obróbki zadania przez system jest realizowane przez kolejkę *REALIZ*. Jej długość (*Q$REALIZ*) jest użyta także do testowania warunku dopuszczenia nowego zadania do procesora (blok *TEST*). Tabela statystyczna o nazwie OPOZ-HIST rejestruje rozkład czasów oczekiwania użytkownika na odpowiedź systemu.

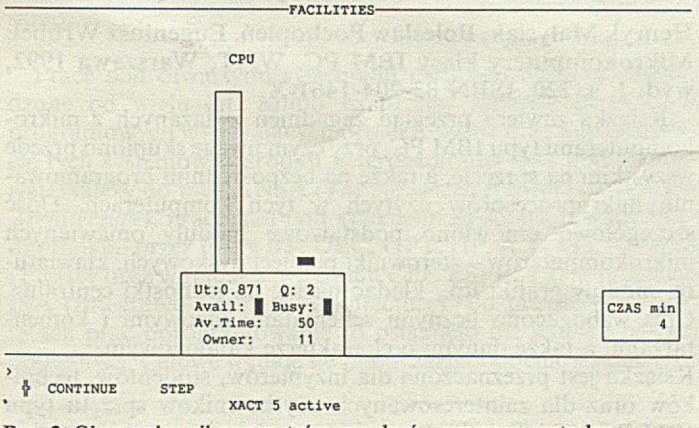
```
 ;  GPSS/PC Program File SYS_KOM.GPS.  (V 2, # 40079)  01-12-1993 12:11:50
10 *         Model systemu komputerowego:
20 *           procesor + 2 procesory we-wy
30 *           10 zadan uzytkownika
40 KANAL     STORAGE      2                  ;liczba procesorow we-wy
50 *
60 XPDIS     FUNCTION     RN3,C24            ;Exponential distribution function
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
70 OPOZ_HIST TABLE        MP1,0,500,19       ;histogram czasu reakcji
80           GENERATE     ,,,10              ;utworzenie 10 zadan
90 POCZ      ADVANCE      3000,FN$XPDIS      ;czas namyslu uzytkownika
100          MARK         1                  ;zapamietanie czasu
110          TEST L       Q$REALIZ,4         ;nie wiecej niz 4 zadania
120          QUEUE        REALIZ             ;zliczanie czasu realizacji zad.
130 POWROT   SEIZE        CPU                ;zajecie procesora
140          ADVANCE      50,FN$XPDIS        ;czas obrobki w procesorze
150          RELEASE      CPU                ;zwolnienie procesora
160          ENTER        KANAL              ;zajecie 1 procesora we-wy
170          ADVANCE      60,FN$XPDIS        ;czas komunikacji
180          LEAVE        KANAL              ;zwolnienie procesora we-wy
190          TRANSFER     .1,POWROT          ;srednio co 10 zadanie zakonczone
200          DEPART       REALIZ
210          TABULATE     OPOZ_HIST          ;rejestracja czasu reakcji
220          TRANSFER     ,POCZ              ;powrot zadania do uzytkownika
230 *         Segment - timer
240 LICZ     GENERATE     60000              ;zadanie-impuls co 1 min
250          TERMINATE    1
260 *         Polecenia sterujace
270          MICROWINDOW  1,N$LICZ           ;CZAS min
```

**Wydruk 1. Tekst programu – modelu działania systemu komputerowego**

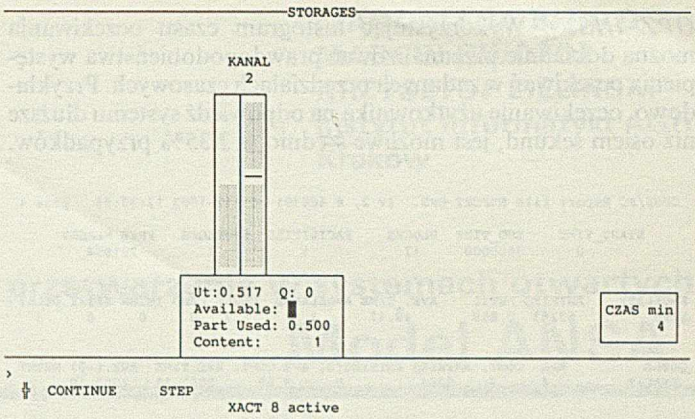
Segment-timer oblicza minuty czasu modelowego – jednostka czasu modelowego odpowiada jednej milisekundzie czasu rzeczywistego.

Program uruchomiony poleceniem START 60 symuluje pracę systemu przez 60 min. tj. godzinę czasu modelowego.

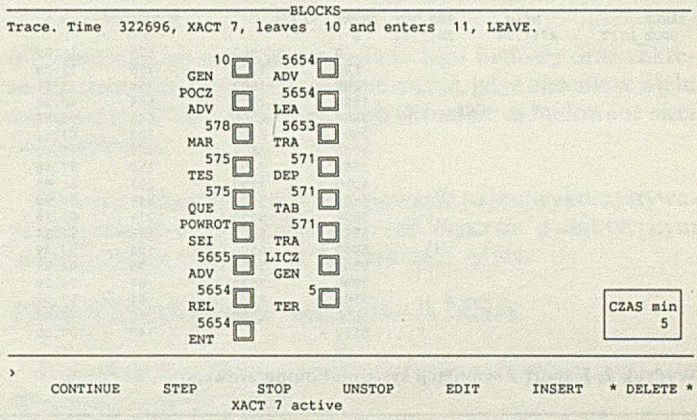
Polecenie w linii 270 powoduje wyświetlanie w mikrooknie bieżącego czasu modelowego.



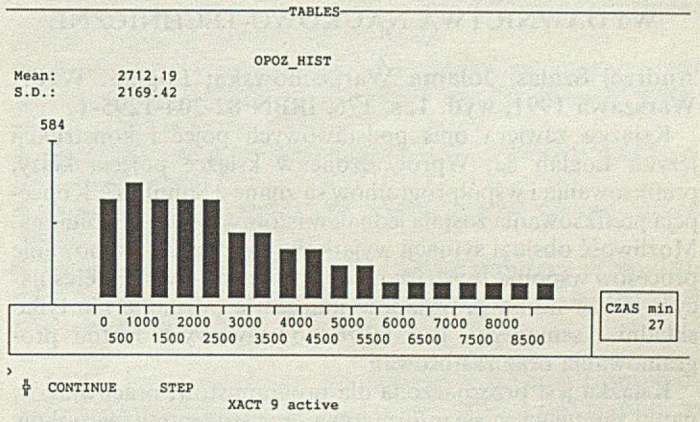

**Rys. 2. Okno animacji parametrów urządzeń – procesor centralny**




**Rys. 3. Okno animacji parametrów magazynów – procesory we-wy**




**Rys. 4. Okno ze schematem blokowym modelu**




**Rys. 5. Okno tabeli statystycznej – histogram czasu oczekiwania na reakcję systemu**

W trakcie symulacji można obserwować na ekranie monitora parametry wykorzystania procesora centralnego (rys. 2), procesorów wejścia-wyjścia (rys. 3), schemat blokowy z liczbą zadań w poszczególnych blokach (rys. 4) lub histogram czasów oczekiwania użytkownika na reakcję systemu (rys. 5). Po zakończeniu symulacji, trwającej ok. 10 min., na mikrokomputerze IBM/PC AT, 16 MHz, należy tylko wygenerować i zinterpretować raport (wydruk 2).

Rezultaty symulacji wskazują, że procesor centralny badanego systemu komputerowego był wykorzystywany średnio w 86,9%, natomiast procesory wejścia-wyjścia w 52,6%. Średni czas realizacji zadania (tj. przebywania w układzie procesorów) wynosił niewiele ponad dwie sekundy (*AVE.TIME* dla kolejki *REALIZ*). Natomiast średni czas oczekiwania przez użytkownika na reakcję systemu – 2,71 sek. (wartość średnia tabeli

*OPZ-HIST*). Wykorzystując histogram czasu oczekiwania można dokładnie przeanalizować prawdopodobieństwa występienia oczekiwań w zadanych przedziałach czasowych. Przykładowo, oczekiwanie użytkownika na odpowiedź systemu dłuższe niż osiem sekund, jest możliwe średnio w 2,35% przypadków.

```
GPSS/PC Report file REPORT.GPS.  (V 2, # 40079)  01-12-1993 12:32:55   page 1

    START_TIME    END_TIME  BLOCKS     FACILITIES  STORAGES   FREE_MEMORY
             0     3600000      17              1         1        291664


FACILITY     ENTRIES  UTIL.   AVE._TIME AVAILABLE   OWNER PEND INTER RETRY DELAY
 CPU           63263  0.869       49.47         1       9    0     0     0     3


QUEUE          MAX  CONT.  ENTRIES ENTRIES(0) AVE.CONT.  AVE.TIME  AVE.(-0) RETRY
 REALIZ          4     4     6296           0      3.50    2003.18   2003.18     3


STORAGE        CAP.  REMAIN.  MIN.   MAX.   ENTRIES AVL.   AVE.C.  UTIL.  RETRY DELAY
 KANAL            2       2     0      2     63262   1      1.05   0.526      0     0


TABLE           MEAN      STD.DEV.  RETRY  RANGE            FREQUENCY  CUM.%
 OPOZ_HIST    2711.36     2213.40       0
                                        0 -      500           698      11.09
                                      500 -     1000           798      23.78
                                     1000 -     1500           738      35.51
                                     1500 -     2000           684      46.38
                                     2000 -     2500           641      56.56
                                     2500 -     3000           507      64.62
                                     3000 -     3500           439      71.60
                                     3500 -     4000           350      77.16
                                     4000 -     4500           301      81.95
                                     4500 -     5000           249      85.90
                                     5000 -     5500           211      89.26
                                     5500 -     6000           156      91.74
                                     6000 -     6500           104      93.39
                                     6500 -     7000            96      94.91
                                     7000 -     7500            81      96.20
                                     7500 -     8000            51      97.01
                                     8000 -     8500            40      97.65
                                     8500 -                     148     100.00
```

**Wydruk 2. Raport z symulacji systemu komputerowego**

W ciągu 60 minut system zrealizował prawie 6300 zadań użytkowników. W tym czasie procesor centralny obrabiał zadania użytkowników ponad 63 200 razy.

Przedstawiony materiał nie wyczerpuje wszystkich problemów i możliwości zastosowań mikrokomputerów do symulacji procesów. Ta ciekawa i efektowna metoda badawcza powinna moim zdaniem znaleźć szersze niż dotychczas zastosowanie, szczególnie w procesach wspomagania podejmowania decyzji dotyczących systemów technicznych, ekonomicznych, zarządzania i planowania.

LITERATURA


[1] Chisman J. A.: Introduction to Simulation Modeling Using GPSS/PC. New Jersey, Prentice Hall, 1992

[2] Fishman G. S.: Symulacja komputerowa – pojęcia i metody. Warszawa, PWE, 1981

[3] Gordon G.: Symulacja systemów. Warszawa, WNT, 1974

[4] Gordon G.: The Aplication of GPSS V to Discrete System Simulation. NY, Prentice Hall, 1975

[5] Karian Z. A., Dudkiewicz E. K.: Modern Statistical, Systems, and GPSS Simulation. The First Course. New York, Computer Science Press, 1991

[6] Kącki E.: Symulacja komputerowa – metody, środki, zastosowania. Materiały V Krajowej Konferencji Informatyków, Poznań, 1985

[7] Kondratowicz L.: Modelowanie symulacyjne systemów. Warszawa, 1978

[8] Martin F. F.: Wstęp do modelowania cyfrowego. Warszawa PWN, 1976

[9] Perkowski P.: Technika symulacji cyfrowej. Warszawa, WNT, 1980

[10] Schriber T. J.: Simulation Using GPSS. New York, John Wiley and sons, 1990

[11] Thesen A., Travis L. E.: Simulation for Decision Making. St. Paul, West Publishing Company, 1992

[12] Tyszer J.: Symulacja cyfrowa. Warszawa, WNT, 1990.


## Nowe książki

### WYDAWNICTWA NAUKOWO-TECHNICZNE

**Andrzej Szałas, Jolanta Warpechowska; Loglan. WNT, Warszawa 1991, wyd. 1, s. 176. ISBN 82-204-1295-1**

Książka zawiera opis podstawowych pojęć i konstrukcji języka Loglan 82. Wprowadzone w książce pojęcia klasy, prefiksowania i współprogramów są znane z Simuli-67. Koncepcja prefiksowania została jednak w istotny sposób uogólniona. Możliwość obsługi sytuacji wyjątkowych oraz programowanie procesów współbieżnych czyni język nowoczesnym i interesującym. Duża liczba przykładów umożliwia poznanie nie tylko składni i semantyki, lecz także podstawowych metod programowania oraz zastosowań.

Książka jest przeznaczona dla programistów, pracowników nauki zajmujących się informatyką oraz studentów kierunków informatycznych.

**Jan Bielecki: Biblioteki ANSI C. WNT, Warszawa 1990, wyd. 1, s. 136. ISBN 83-204-1344-3**

Książka zawiera opisy funkcji bibliotecznych standardu ANSI C oraz opis składni języka standardowego. Jest przeznaczona dla użytkowników mikrokomputerów klasy IBM PC zainteresowanych programowaniem przenośnym w języku C zgodnym ze standardem ANSI.

**Janusz Szajna, Marian Adamski, Tomasz Kozłowski: Turbo Prolog. Programowanie w języku logiki. WNT, Warszawa 1991, wyd. 1, s. 168. ISBN 83-204-1395-8**

W książce w przystępnej formie przedstawiono zagadnienia związane z programowaniem w języku logiki. Jako narzędzie przedstawiono Turbo Prolog, jedną z najbardziej udanych implementacji Prologu. Czytelnik jest stopniowo wprowadzany w ideę programowania w tym języku, zapoznawany z dostępnymi mechanizmami i informowany o licznych różnicach w porównaniu z tradycyjnymi językami programowania, np. Pascalem. Część teoretyczna książki jest silnie związana z częścią praktyczną, przez pokazanie w licznych przykładach, jak teoretyczny opis problemu staje się programem w Turbo Prologu. W książce podano też wiele praktycznych informacji związanych z techniką pisania i uruchamiania programów oraz przykłady konkretnych zastosowań, m.in. przy projetowaniu układów cyfrowych.

Książka jest przeznaczona dla szerokiego kręgu Czytelników zainteresowanych sztuczną inteligencją, programowaniem w języku logiki i Turbo Pascalem.

**Ryszard Tadeusiewicz: ATARI Logo – Komputerowe przygody. WNT, Warszawa 1991, wyd. 2, s. 288. ISBN 83-204-1177-7**

Książka jest przeznaczona dla dzieci w wieku szkolnym. Bawiąc – uczy podstaw informatyki i oswaja z komputerem, jako przyszłym narzędziem pracy. Rodziców zachęca do wspólnej zabawy z dziećmi przy komputerze.

W książce zawarto pełny kurs języka Logo w wersji odpowiedniej do zastosowania w komputerze ATARI. Jest to seria zabaw z komputerem, które tym się różnią od gier, że czegoś uczą.

**Henryk Małysiak, Bolesław Pochopień, Eugeniusz Wróbel: Mikrokomputery klasy IBM PC. WNT, Warszawa 1992, wyd. 1, s. 220. ISBN 83-204-1461-X**

Książka zawiera przegląd zagadnień związanych z mikrokomputerami typu IBM PC, przy czym uwagę skupiono przede wszystkim na sprzęcie, a także na bezpośrednim programowaniu mikroprocesorów użytych w tych komputerach. Dość szczegółowo omówiono podstawowe moduły omawianych mikrokomputerów – sterowniki pamięci dyskowych, klawiaturę, moduły grafiki itp., kładąc nacisk na jednostki centralne. Opis wzbogacono licznymi schematami ideowymi i komentarzami, a także danymi o charakterze katalogowym.

Książka jest przeznaczona dla inżynierów, studentów, techników oraz dla zainteresowanych użytkowników sprzętu typu IBM PC/XT lub AT.

MAGDALENA BIJAŁD
MARCIN GWÓŹDŹ
KRZYSZTOF ZIELIŃSKI
Instytut Informatyki AGH
Kraków

## Rozproszone przetwarzanie w systemach otwartych

# Model ANSA*

Przetwarzanie rozproszone w systemach otwartych (*Open Distributed Processing*) stanowi jeden z najintensywniej rozwijanych obszarów zastosowań osiągnięć w zakresie sieci komputerowych, systemów operacyjnych i rozproszonych baz danych.

Otwarty system komputerowy stanowi autonomiczną całość zdolną do przetwarzania informacji, a składającą się z jednego lub kilku komputerów z odpowiednim oprogramowaniem, urządzeniami zewnętrznymi oraz środkami przekazywania informacji, którego funkcje można rozszerzać i modyfikować dzięki standaryzacji technik implementacji podstawowych funkcji systemu.

Pojęcie otwartości systemu komputerowego może dotyczyć różnych jego aspektów, takich jak np. języki programowania, system operacyjny, oprogramowanie sieciowe, model obliczeniowy, reprezentacja danych, interfejs użytkownik-program, czy też sposób udostępniania i rozszerzania funkcji użytkowych systemu. Zazwyczaj budowa systemów otwartych jest związana ze stosowaniem takich stnadardów, jak np. język C, system operacyjny UNIX, TCP/IP, X-11, Motif itp.

Zapewnienie otwartości rozproszonego heterogenicznego systemu komputerowego wnosi wiele nowych wymagań, jak np. protokoły komunikacji między procesami rozproszonymi, wspólna reprezentacja danych, nazewnictwo, lokalizacja i udostępnienie usług. Należy podkreślić, że przez procesy rozproszone rozumie się procesy realizowane w oddzielnych przestrzeniach adresowych, niezależnie czy wykonują się one w jednym fizycznym węźle sieci, czy też nie. Ważnym elementem charakteryzującym przetwarzanie rozproszone jest bowiem transparentność obliczeń. Oznacza to, że użytkownika systemu rozproszonego przetwarzania pewne fakty związane z fizycznym rozproszeniem są zakryte. Przykładowo, wywołanie procedur na lokalnej maszynie i maszynie zdalnej powinno odbywać się tak samo.

Prace nad otwartymi systemami rozproszonymi są prowadzone od wielu lat. Mimo to istnieje niewiele produktów programowych wspierających projektowanie oraz implementację tych systemów w środku heterogenicznej sieci komputerowej. Należy do nich ANSA (*The Advanced Networked Systems Architecture*). Prace nad tym systemem są sponsorowane między innymi przez firmy BT, DEC, GPT, HP, ICL, ITL, Olivetti i Plessey oraz w ramach programu CEC ESPRIT II w projekcie ISA (Integrated Systems Architecture). Po prawie dziesięciu latach prac oprogramowanie to osiągnęło już stopień pewnej dojrzałości. Celowe wydaje się zatem zaznajomienie środowiska informatycznego w kraju z zasadami jego budowy oraz zakresem zastosowań. Jest to szczególnie ważne, gdyż obecnie w wielu ośrodkach w Polsce już istnieją, lub aktualnie są budowane sieci komputerowe.

Należy podkreślić, że oprogramowanie to jest wykorzystywane od ponad roku w pracach oraz procesie dydaktycznym prowadzonym w Instytucie Informatyki AGH.

## Charakterystyka modelu ANSA

W punkcie tym zostaną omówione założenia projektowe oraz stosowane modele ANSA.

### Założenia projektowe ANSA

Zgodnie z przyjętymi założeniami prace związane z projektem ANSA obejmują cztery sfery działania:

- **architekturę** – rozwój architektury systemów rozproszonych jako zintegrowanego zbioru struktur, funkcji oraz zasad i wskazówek projektowych,
- **oprogramowanie** – opracowanie przykładowych praktycznych zastosowań,
- **standardy** – przekazywanie wyników prac do organizacji standaryzacyjnych,
- **transfer technologii** – wprowadzanie wyników prowadzonych badań do praktyki konstrukcji rozproszonych systemów komputerowych.

ANSA jest architekturą dla budowy systemów rozproszonych, które funkcjonują jako jednolita całość w taki sposób, że fakt rozproszenia jest niewidoczny dla programistów i użytkowników. Architektura ta pozwala na pełne wykorzystanie możliwości stwarzanych przez strukturę sprzętową systemu rozproszonego, takich jak np. współbieżność realizacji obliczeń, zwiększenie efektywności i niezawodności przez wykorzystanie heterogenicznych zasobów systemu i decentralizację oraz lepsze zamaskowanie wad, takich jak błędy komunikacji lub awarie części systemu.



*) Pracę wykonano w ramach grantu KBN nr: 8.0077.91.01

W systemie ANSA przyjęto, że dostosowanie programu do współpracy z systemem rozproszonym odbywa się na poziomie języka programowania. Możliwości rozproszonej realizacji obliczeń są reprezentowane przez dodatkowe konstrukcje syntaktyczne dodane do istniejących języków programowania, takich jak C czy Fortran. Zaletami tego rozwiązania są:
- prostota modelu programu,
- kontrola poprawności w czasie kompilacji, a nie wykonania programu,
- niezależność od systemu z punktu widzenia programu.

Celem systemu ANSA jest stworzenie takiej architektury systemów rozproszonych, która zapewniłaby:
- uniwersalność dla różnego rodzaju zasatosowań,
- przenośność na różne systemy operacyjne i języki programowania,
- możliwość pracy w heterogenicznym środowisku,
- strukturę modularną z maksymalnym wykorzystaniem istniejących funkcji,
- podstawę w dziedzinie rozproszonego nadawania nazw, współbieżności i kontroli błędów,
- rozszerzenie na różne komputery i topologie sieci bez ograniczeń rozmiaru,
- ukierunkowanie na żądania twórców oprogramowania,
- zapewnienie współpracy między autonomicznie zarządzanymi sieciami komputerowymi.

Przedsięwzięcie ANSA obejmuje zatem bardzo szerokie spektrum zagadnień dotyczących konstrukcji rozproszonych systemów komputerowych.

## Modele ANSA

Podczas tworzenia architektury systemu ANSA, zespół jego twórców zapoznawał się z aktualnymi badaniamia i osiągnięciami w dziedzinie przetwarzania rozproszonego oraz z technikami projektowania systemów. Badania te pozwoliły stwierdzić, że przy charakteryzowaniu pojęcia „rozproszone przetwarzanie" dla różnych ekspertów mają znaczenie różne sprawy kluczowe. W związku z tym opis systemu ANSA został zrealizowany w postaci zbioru pięciu modeli odpowiadających najczęściej spotykanym punktom widzenia.

Opis systemu rozproszonego w każdym z nich jest jednolity i zupełny. Określono następujące modele:
- **zadaniowy** – dostarcza ogólnego pojęcia o systemach przetwarzania informacji oraz opisuje wszystkie obiekty systemu pod kątem zadań i celów,
- **informacyjny** – dostarcza podstaw do opisu żądanych przez system informacji, tzn. struktur poszczególnych elementów informacji, zasad ustalania zależności, sposobów dzielenia informacji i atrybutów dostępu,
- **obliczeniowy** – opisuje szkielet struktury programu oraz narzędzia programistyczne, które powinny być dostępne dla twórcy programów rozproszonych,
- **inżynieryjny** – opisuje podstawowe części kompilatora i systemu operacyjnego do realizacji obliczeń w heterogenicznym otoczeniu,
- **technologiczny** – opisuje system pod kątem jego relacji z różnymi systemami operacyjnymi.

Mimo wielu punktów widzenia na projektowanie systemów rozproszonych, modele obliczeniowy i inżynieryjny są tymi, które mają największy wpływ na konstrukcję tych systemów; ich wzajemne zależności są bardzo ścisłe, a ich poznanie w zasadniczy sposób ułatwia zrozumienie całej filozofii systemu. Z tego powodu dalsze rozważania zostaną w tym artykule ograniczone tylko do tych dwu modeli.

### Model inżynieryjny ANSA

Z punktu widzenia praktycznego zastosowania środowiska programowego ANSA, dla informatyka najważniejszy jest model inżynieryjny. Model ten obejmuje:
- zarządzanie zadaniami,
- zarządzanie adresami,
- komunikację wewnątrz przestrzeni adresowej,
- rozproszone protokoły aplikacji,
- protokoły sieciowe,
- lokalizatora interfejsu,
- interfejs tradera,
- zarządcę konfiguracji systemu,
- zarządcę operacji atomowych,
- zarządcę replikacji interfejsu.

Wymienione składowe modelu inżynieryjnego ANSA zostaną poniżej szczegółowo omówione. Ich funkcje i budowa wynika z modelu przetwarzania przyjętego w rozważanej architekturze.

## Przetwarzanie informacji w modelu ANSA

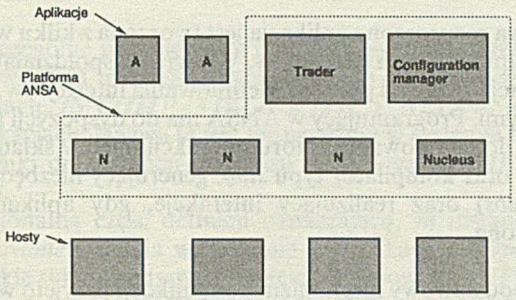
Przetwarzanie informacji w modelu ANSA jest ukierunkowane obiektowo i dotyczy trzech poziomów abstracji:
- kapsuły – odpowiadają procesom realizowanym pod nadzorem systemu operacyjnego UNIX; zapewniają wielowątkowe środowisko dla implementacji obiektów oraz typowe dla architektury ANSA funkcje systemowe;
- obiekty – podstawowe elementy, z których jest budowana aplikacja; obiekt jest określony przez definicję interfejsu, stanowiącą opis operacji – usług dostarczanych przez obiekt oraz struktury danych, na których one operują; obiekt może mieć wiele interfejsów, jak również kilka obiektów może być zaimplementowanych w ramach pojedynczej kapsuły;
- operacje – stanowią usługi dostarczane przez obiekt, czyli elementy akcji systemu.

Podstawowym elementem przetwarzania informacji w systemie ANSA jest usługa. Zgodnie z powszechnie przyjętą terminologią, elementy systemu, które korzystają z usług to klienci, a składowe dostarczające usług to serwery. Model obliczeniowy systemu ANSA zezwala obiektowi być jednocześnie klientem i serwerem. Dostarczane usługi dzielą się na aplikacyjne i systemowe. Celem usług systemowych jest zapewnienie wygodnych środków dla zarządzania obliczaniem rozproszonym przez zapewnienie różnego rodzaju transparentności. Usługi te są wbudowane w strukturę kapsuły i ukrywają szczególne aspekty złożoności programowania rozproszonego. System ANSA zapewnia kilka rodzajów transparencji, z których każdy może być stosowany lub nie stosowany w zależności od wymagań aplikacji. Obecnie dostępne wersje systemu dostarczają następujące rodzaje transparencji:
- dostępu – pozwala na ujednolicenie konstrukcji obiektów i ich wzajemną interakcję, niezależnie od tego, czy jest to obiekt lokalny, czy działający na zdalnej maszynie;
- lokalizacji – umożliwia interakcje między obiektami, niezależnie od ich fizycznej lokalizacji;
- współbieżności, sprawia, że niewidoczny jest fakt, iż z tej samej usługi korzysta współbieżnie wiele procesów klienta;
- błędów – zasłania efekt częściowego wykonania usługi; wymaga to implementacji mechanizmu atomowości operacji;
- replikacji – ukrywa fakt obecności w systemie serwerów dostarczających tej samej usługi.

Jak dotąd nie doczekały się jeszcze pełnej realizacji m.in. transparencje migracji i atomowości.

Model inżynieryjny ma na celu stworzenie płaszczyzny między modelem obliczeniowym, a licznymi modelami technologi-

cznymi. Jest on skonstruowany za pomocą odwzorowania obiektów modelu obliczeniowego w obiekty modelu inżynieryjnego. Elementy modelu obliczeniowego zostały przedstawione na rys. 1 i obemują usługę tradera dostarczającego mechanizmów lokalizacji funkcji w systemie, zarządcy konfiguracji (ang. *configuration manager*), pozwalającego na uruchomienie nowej aplikacji oraz modułów nucleusa. Moduły nucleusa muszą być obecne w każdym węźle sieci należącym do systemu. Nucleus spaja poszczególne węzły w jeden system dostarczając pewnych funkcji elementarnych, zasłaniających różnice między poszczególnymi węzłami heterogenicznej sieci komputerowej. Jednorodne środowisko dla realizacji usług systemowych ANSA oraz dostarczanych w ramach implementacji aplikacji w tym środowisku zapewniają tzw. kapsuły. Pozostałe elementy modelu inżynieryjnego, jak trader (ang. *trader*), zarządca węzła (ang. *node manager*) oraz factory są zrealizowane w postaci kapsuł.

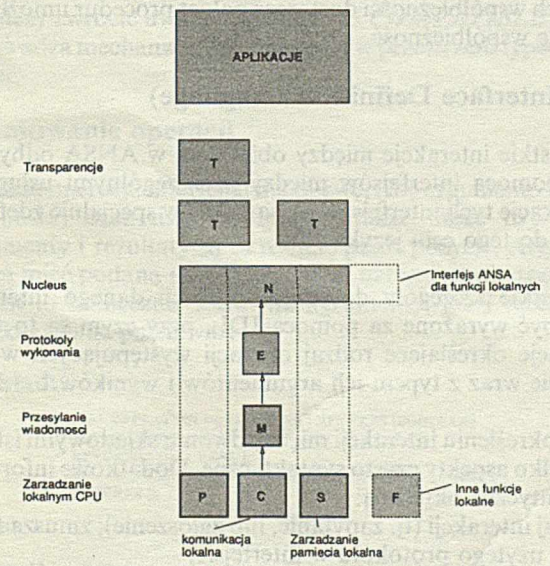



Rys. 1. Struktura systemu ANSA

## Węzły, kapsuły i nucleus

Zasoby węzła są zarządzane przez obiekt inżynieryjny zwany **nucleusem,** który udostępnia je obiektom inżynieryjnym zwanym **kapsułami.**

Struktura kapsuły oraz nucleusa została pokazana na rysunku 2. w odniesieniu do pojedynczego węzła systemu. Nucleus jest loklanym zarządcą zasobów niskiego poziomu. W praktyce jest on implementowany jako warstwa leżąca powyżej lokalnego systemu operacyjnego, zapewniająca dostęp do usług infrastruktury.




Rys. 2. Struktura kapsuły

Kapsuła jest odwzorowana w jeden proces pod nadzorem systemu operacyjnego i zapewnia wielowątkowe środowisko dla obiektów aplikacyjnych oraz usługi transparencji dostępu i lokalizacji. Funkcje kapsuły są wspierane przez zbiór operacji nucleusa.

Wątek jest niezależnie wykonywaną sekwencją operacji, natomiast zadanie jest procesorem wirtualnym, który wykonuje wątek przy użyciu wymaganych zasobów. Ogólnie ujmując, wątek może reprezentować zespół potencjalnie współbieżnych działań. Jednakże dla osiągania postępów wątek musi być dołączony do zadania (ang. task). Aby zapamiętać stan pośredni wątek w czasie wykonywania wymaga dodatkowych zasobów.

Wszystkie kapsuły są wielowątkowe i opcjonalnie mogą być wielozadaniowe. Wątki jedynie identyfikują ewentualną współbieżność, natomiast zadania dostarczają zasobów dla faktycznej współbieżności. Jeśli liczba wątków przewyższa liczbę zadań, to niektóre wątki muszą być szeregowane. Abstrakcja wątków jest dostarczana przez interpretera zadań – przez moduł nucleusa. Koncepcyjnie nucleus jest oddzielną kapsułą, wzywaną przez wywołane (ang. *call*). Ze względu na transparencję dostępu może on być jednak zaimplementowany jako kombinacja wywołań funkcji biblitecznych w kapsule oraz operacji zdalnych.

Specyfikacja nucleusa zawiera definicje protokołów wymaganych dla komunikacji między nucleusami oraz sposobu implementacji usług w postaci trzech warstw. Warstwa górna (ang. *session service*), dostarcza struktur dialogu i synchronizacji, które odpowiadają funkcji nucluesa dla wewnątrzkapsułowej komunikacji. Warstwa środkowa dostarcza protokołu (ang. *execution protocol*), który odpowiada za określoną dla modelu obliczeniowego semantykę interakcji. Warstwa dolna (ang. *message passing service*) zapewnia usługi transportowe między nucleusami. Do implementacji tej usługi mogą być użyte protokoły połączeniowe (np. TCP) oraz bezpołączeniowe (np. UDP).

ANSA definiuje usługi, które zapewniają dostępną w całej sieci infrastrukturę dla obiektów rozproszonych. W najnowszej wersji implementacji systemu (ANSAware 4.0) są to usługi:
- Trading service,
- Factory service,
- Node Manager service,

które stanowią oddzielne kapsuły.

## Trading

Trading jest usługą polegającą na kojarzeniu ofert poszczególnych interfejsów z żądaniami wykonania dostarczonych przez nie usług. Po pomyślnym skojarzeniu dwa obiekty mogą ze sobą współpracować. Trader jest połączeniem dwóch różnych obiektów:
- zarządcy nazw kontekstów – obiekt ten zarządza przestrzenią nazw kontekstów i zapewnia rejestrację, przeglądanie i usuwanie ofert usług w tej przestrzeni,
- zarządcy typów – obiekt ten zarządza przestrzenią nazw typów interfejsów i utrzymuje między typami relację *IsASubtypeOf*. Silna zależność między parą oferta-żądanie będącą w relacji *IsASubtypeOf*, narzuca umieszczenie w jednej kapsule dwóch obiektów.

Serwer usługi zgłasza swoją gotowość działania przez zarejestrowanie interfejsu wraz z dodatkowymi informacjami dotyczącymi kontekstu i typu w traderze; klient znajduje potrzebny serwer przez wykonaie na traderze operacji *Lookup*. Rezultatem pomyślnego wykonania tej usługi jest wskaźnik służący do instancji interfejsu usługi. Jedną z dodatkowych informacji dostarczanych do tradera jest kontekst. Baza danych tradera ma wewnętrzną strukturę drzewiastą, przypominającą drzewo katalogów. Poszczególne pola nazwy kontekstu są oddzielone

znakiem "/" i określają ścieżkę prowadzącą do danego kontekstu. Gdy nazwa kontekstu rozpoczyna się od znaku "/", oznacza to że poszukiwanie kontekstu powinno rozpocząć się od wierzchołka drzewa. Każdy węzeł w drzewie zawiera swoją nazwę, wskaźniki do następnego elementu w drzewie na tym samym poziomie, listy dołączonych poniżej kontekstów oraz listy interfejsów oferowanych w danym kontekście. Podanie kontekstu przy rejestracji specyfikuje umiejscowienie usługi. Rozwiązanie takie pozwala na uporządkowanie usług ze względu np. na ich przeznaczenie.

Posiadanie wskaźnika do interfejsu nie gwarantuje reakcji usługi na późniejsze wywołania. Może się to zdarzyć w przypadku, gdy usługa jest zarejestrowana, ale jej serwer faktycznie nie jest aktywny. System zapewnia specjalne funkcje dla usunięcia takiej oferty.

### Factory

Większość systemów obiektowo zorientowanych dostarcza mechanizmów dynamicznego tworzenia obiektów. W systemie ANSAware dynamiczne tworzenie usług jest związane z tworzeniem kapsuł zawierających obiekty z interfejsami. Tak więc usługa *Factory* dostarcza mechanizmów do tworzenia i niszczenia kapsuł. Odbywa się to w następujący sposób:
- Klient żąda od *Factory* utworzenia kapsuły.
- Factory tworzy nową kapsułę.
- Nowa kapsuła jest tworzona z pojedynyczym obiektem, który ma interfejs typu *Capsule*. Ten obiekt można traktować jako kreatora innych obiektów. Interface *Capsule* ma operację *Capsule*$ – Instantiate, która jest używana do tworzenia obiektów w kapsule.
- Referencja do utworzonego w ramach nowej kapsuły interfejsu *Capsule* jest zwracana co *Factory*.
- *Factory* zwraca tę referencję do klienta, który wywołał operację utworzenia kapsuły.
- Klient może użyć tej referencji do stworzenia interesujących go obiektów.

### Node Manager

*Node Manager* (NM) jest usługą pozwalającą na tworzenie, proste monitorowanie i usuwanie usług w danym węźle. Moduł ten korzysta przy tym z mechanizmów dostarczanych przez usługę *Factory*. Zasadniczą częścią usługi *Node Manager* jest baza danych do zapisu i zarządzania usługami w danym węźle. Każdy opis usługi jest identyfikowany przez alias. Każdy alias może mieć różną wartość atrybutów specyfikujących sposób jego zarządzania. Alias może być zdefiniowany jako stały, co oznacza, że NM automatycznie uruchamia określoną usługę po jej zakończeniu. Baza danych NM zawiera następujące informacje:
- alias jednoznacznie identyfikujący opis usługi,
- nazwę interfejsu dla usługi (używaną przez Trader),
- kontekst,
- atrybuty własne,
- ścieżkę do programu wykonywalnego, zawierającego serwer danej usługi,
- nazwę kapsuły,
- nazwny obiektów,
- argumenty wywołania,
- parametry środowiska,
- maksymalną liczbę wywołań usługi.

Dynamiczne tworzenie usług za pomocą NM odbywa się w następujący sposób:
- NM rejestruje ofertę na *Traderze,*
- klient importuje usługę,
- *Trader* stwierdza istnienie oferty i importuje ją z *Node Managera,*
- NM tworzy kapsułę serwera za pomocą *Factory,*
- NM wywołuje operację *Instantiate* nowo utworzonej kapsuły interfejsu *Capsule,*
- NM zwraca do *Tradera* referencję do interfejsu,
- *Trader* zwraca tę referencję do klienta,
- Klient może wywołać usługi dostarczone w ramach interfejsu, którego referencję otrzymał.

## Model obliczeniowy ANSA

Zgodnie z wcześniejszymi rozważaniami, model ten dotyczy technik tworzenia oprogramowania w środowisku ANSA. Jednym z głównych celów ANSA jest uproszczenie projektowania, konstruowania, rozbudowy i konserwacji rozproszonych systemów informacyjnych. Narzędzia programistyczne ułatwiają to zadanie na poszczególnych etapach tworzenia systemu.

Typowa rozproszona aplikacja jest tworzona z kilku wzajemnie współpracujących elementów. Właściwe współdziałanie części aplikacji zależy od precyzji zdefiniowania interfejsów między składowymi. Programujący w ANSA ma do dyspozycji specjalny język do definiowania zbioru interakcji między składowymi aplikacji oraz kompilator typu *stub,* generujący niezbędny kod (ang. *stubs*) oraz realizujący interakcje, gdy aplikacja jest rozproszona.

Jako podstawowy mechanizm komunikacji przyjęto w modelu ANSA RPC (*Remote Procedure Call*), czyli zdalne wywołanie procedury; klient wywołuje operację zdefiniowaną w specyfikacji interfejsu łączącego go z serwerem. Istnieją dwie formy wywołań:
- zapytanie – klient zawiesza swoją działalność i czeka, aż serwer wykona operacje i zwróci rezultat,
- zawiadomienie – klient kontynuuje działanie i nie czeka na zakończenie pracy serwera.

ANSA dostarcza też programiście preprocesor, który przekształca źródłowy program w języku C, zawierający komendy języka PREPC (język, w którym są napisane wywołania operacji), na zwykłe pliki źródłowe w C. Zapewnia także synchronizację operacji elementarnych, z których mogą być tworzone obszary krytyczne, oraz dla systemów operacyjnych nie zapewniających współbieżności dostarcza pakiet procedur umożliwiających tę współbieżność.

### IDL (Interface Definition Language)

Wszystkie interakcje między obiektami w ANSA odbywają się za pomocą interfejsów między poszczególnymi usługami. Specyfikacje tych interfejsów są napisane w specjalnie zdefiniowanym do tego celu języka IDL.

Wszystkie szczegóły dotyczące operacji danego interfejsu muszą być wyrażone za pomocą IDL, przy czym są to tylko informacje określające rodzaj operacji występujących w tym interfejsie wraz z typem ich argumentów i wyników.

Przy określeniu interakcji między dwoma składowymi istotne są nie tylko aspekty czysto syntaktyczne. Dodatkowe informacje semantyczne określają:
- rodzaj interakcji (tj. zapytanie, lub zgłoszenie); zmusza to do wyboru użytego protokołu w interfejsie;
- transparencje (przezroczystość) na poziomie interakcji.

IDL w ANSA jest zmodyfikowaną wersją języka zdefiniowanego przez Xerox Network Systems.

Specyfikacje akcji realizuje się w języku C rozszerzonym

o specjalne instrukcje metajęzyka DPL (*Distributed Processing Lanquage*).

## Ogólna struktura specyfikacji interfejsu

Specyfikacja napisana w języku IDL ma następującą strukturę ogólną:

```
1=>  IFtypeName1 : INTERFACE =
2=>  IS COMPATIBLE WITH IFTypeName2;
3=>  NEEDS IFTypeName3;
     BEGIN
4=>  - definicje typu danych
5=>  - definicje operacji
     END.
```

**Wydruk 1**

Poszczególne pozycje (1–5) definicji interfejsu określają:
1. Nazwę typu interfejsu. Odwołania do tej nazwy występują w sekcjach *'IS COMPATIBLE WITH'* i *'NEEDS'* w innych specyfikacjach. Nazwa ta jest też używana w systemie tradingu. Wskaźniki do instancji interfejsu *Name*1 są typu *IFTypeName*1*Ref*.
2. Opcjonalną część definicji oznaczającą, że instancja typu *IFTypeName*1 spełnia wymagania specyfikacji *IFTypeName*2. Wszystkie składowe typy i operacje zdefiniowane w *IFTypeName*2 są dodane do *IFTypeName*1. Tylko dodatkowe typy i operacje są zdefiniowane w tej specyfikacji. Jeżeli *IFTypeName*2 jest kompatybilny z *IFTypeNameX*, to *IFTypeName*1 jest też kompatybilny z *INFypeNameX*. W tej samej specyfikacji może wystąpić więcej niż jedna sekcja 'IS COMPATIBLE WITH'.
3. Opcjonalną sekcję określającą, że serwer instancji typu *IFTypeName*1 odwołuje się do instancji typu *INFTypeName*3. Wszystkie typy definiowane w *IFTypeName*3 są dodane do *IFTypeName*1. W specyfikacji może występować kilka sekcji *'NEEDS'*. Jest to przydatne, gdy wskaźniki do instancji typu *IFTypeName*3 są przesyłane do instancji typu *IFTypeName*1 jako argumenty lub rezultaty wywołań operacji.
4. Opcjonalne definicje typów danych.
5. Opcjonalne definicje operacji.

Należy zwrócić uwagę, że sekcja *'IS COMPATIBLE WITH'* wprowadza mechanizm dziedziczenia w przestrzeni typów interfejsów.

## Definiowanie operacji

Każda operacja interfejcu ma nazwę, listę argumentów (może być pusta) oraz listę rezultatów (także może być pusta). Argumenty i rezultaty są określane przez pozycję. Argumenty muszą mieć podaną nazwę formalną, natomiast dla rezultatów nie jest to konieczne. Zarówno agrumenty, jak i rezultaty są przezkazywane przez wartość.

**Przykład:**

```
Foo : OPERATION [ a : INTEGER ] RETURNS [BOOLEAN];
Bar : INTERROGATION OPERATION [ x, y : REAL]
RETURNS [ sum : REAL ];
Baz : ANNOUNCEMENT OPERTION [ i : INTEGER ; z : REAL ]
RETURNS [];
```

**Wydruk 2**

## Kompilator stubc

Kompilator ten przetwarza specyfikacje interfejsów napisane w języku IDL na pliki nagłówkowe (ang. *include*) oraz szkielety plików źródłowych klienta–serwera reprezentowane w języku C.

Dla specyfikacji w IDL zaczynającej się od:

*Foo: INTERFACE =*

w aktualnym katalogu są tworzone następujące pliki:
1. *Foo.sif* – zawiera informacje o specyfikacji przeznaczone dla preprocesora C,
2. *tFoo.h.* – definicje typów *typedef*, odpowiadające konstrukcjom typów w specyfikacji IDL,
3. *eFoo.h* – deklaracja *extern* dla funkcji zawartych w pliku szkieletu klienta,
4. *cFoo.c* – źródłowy plik szkieletu klienta,
5. *sFoo.c* – źródłowy plik szkieletu serwera.

Nazwy plików są generowane na podstawie nazwy specyfikacji, a nie na podstawie nazwy pliku zawierającego specyfikację. Stosuje się jednak konwencję, wg której specyfikacja interfejsu o nazwie *Foo* jest umieszczana w pliku *Foo.idl*.

## Język DPL

Kod programu używającego i dostarczającego usług w ANSA-ware jest napisany przy użyciu komend dla preprocesora PREPC, wstawionych w tekst źródłowy zapisany w języku C. Komendy te umożliwiają:

- deklarowanie typów interfejsów i leksykalnego łączenia zmiennych typu *ansa-InterfaceRef*,
- dynamiczne tworzenie – niszczenie instancji interfejsów,
- wywoływanie operacji instalacji interfejsu.

Wszystkie komendy PREPC zaczynają się od wykrzyknika (!) w pierwszej kolumnie programu źródłowego. Jeśli komenda jest dłuższa niż jedna linia, to każda jej linia, z wyjątkiem ostatniej, musi być zakończona backslashem (\); kontynuacja linii nie zaczyna się wykrzyknikiem. Komendy *PREPC* nie są zakończone średnikiem (;). Podstawowe komendy języka DPL wraz ze sposobem ich użycia podano w poniższym przykładzie.

## Przykład programu – przesyłanie wiadomości

Poniższy przykład zastosowania ANSAware jest implementacją usługi przesłania wiadomości wraz z pomiarem czasu tej operacji. Interface *MsgPas* zawiera jedną operację.

```
MsgPas : INTERFACE =

BEGIN
--      definicja typow danych
        BuffType : TYPE = SEQUENCE OF CHAR ;

--      specyfikacje operacji
Myecho: OPERATION [ Src : BuffType]
       RETURNS [ BuffType ];

END.
```

**Wydruk 3**

Serwer dla tej usługi zawiera definicję *Myecho* interfejsu oraz funkcję *body* () odpowiadającą funkcji *main* () w języku C. W funkcji tej następuje utworzenie i zarejestrowanie instancji interfejsu na traderze.

```
#include "AWstdio.h"
#include "AWunistd.h"

#define PROPSIZE 1024
char propbuf[PROPSIZE];

! MANAGED MsgPas          /*  Umozliwienie tworzenia kapsuly za pomoca  */
                          /*  Factory                                   */

! USE MsgPas              /*  Deklaracja interfejsow, ktore beda        */
! USE Trader               /*  uzyte w kapsule                           */

! DECLARE { ir, p->ref } : MsgPas SERVER     /* Deklaracja referencji   */
                                             /* do interfejsu           */
```

```
void body( argc, argv, envp )
int argc;
char *argv[];
char *envp[];
{
    ansa_InterfaceRef ir;

!   {ir} :: MsgPas$Create( 16 )     /* Utworzenie instancji interfejsu */
    (void) system_init_properties( propbuf, PROPSIZE, argc, argv );
!   {} <- traderRef$Export( "MsgPas", "/ansa/testservices", propbuf, ir )
                            /*       Export uslugi do tradera          */
}


int MsgPas_Myecho( _attr, src, res)  /* Kod operacji Myecho interfejsu MsgPas */
ansa_InterfaceAttr *_attr;
BuffType src;
        BuffType *res;
{
        res->length=src.length;
        res->data=&(src.data[0]);

return 1;
}
```

**Wydruk 4**

Proces klienta korzystającego z usługi przesłania wiadomości zawiera odpowienie deklaracje oraz wywołanie operacji interfejsu.

```
#include "AWtime.h"
#include "AWstdio.h"
#include "kbinput.h"
#define S sizeof(char)
#define BLOCK 1024*S
#define END 2500

! USE MsgPas                 /*  Deklaracja interfejsow, ktore beda      */
                            /*  uzyte w kapsule                         */
! DECLARE {ir} : MsgPas CLIENT                /* Deklaracja referencji   */
                                              /* do interfejsu           */
ansa_InterfaceRef ir;
void body( argc, argv, envp )
int argc;
char *argv[];
char *envp[];
{
 int i;
 struct timeval start;
 struct timeval stop;
 struct timezone tz;
 char *tab;
 BuffType buf,buf1;
 long rtime=0;
 int j;

!   {ir} <- traderRef$Import("MsgPas", "/", "")
                                    /*  Import uslugi z tradera  */
        tab=calloc(1,1024);
        memset(tab,'k',1024);
        buf.data=tab;
        buf.length=1024;
        for (j=0;j<100;j++)
          {
            gettimeofday(&start, &tz);
!           {buf1} <- ir$Myecho(buf)          /*  Wykonanie operacji */
            gettimeofday(&stop, &tz);
            rtime =rtime + ((stop.tv_sec-start.tv_sec)*1000000
                          - start.tv_usec+stop.tv_usec);
          }
        printf(" Sredni czas przeslania danych %ld \n", rtime/(2*100));
        free(tab);
!   ir$Discard                      /* Zakonczenie uzywania interfejsu */
}
```

**Wydruk 5**

## Kierunki rozwoju środowiska ANSA

Projekt ANSA jest rozwijany przez firmę Architecture Projects Mangement Ltd. utworzoną w 1989 r. w jej centralnym laboratorium w Cambridge z inicjatywy sponsorujących ten projekt firm. Prace są prowadzone w kilku kierunkach. Pierwszym z nich jest poszerzenie liczby dostępnych dla systemu platform sprzętowych i w konsekwencji pełniejsza realizacja założenia o heterogeniczności środowiska tworzonego oprogramowania. Kolejnym kierunkiem prac nad projektem ANSA jest implementacja wszystkich poziomów transparentności, które zostały określone w założeniach projektu. Poziomy, które zaimplementowano dotychczas zostały opisane w rozdziale „Przekazanie informacji w modelu ANSA". Zgodnie z założeniami projektu do zrealizowania pozostały jeszcze dwa poziomy transparentności:

- migracja procesów – jest formą dynamiczną transparencji lokalizacji; zmiana lokalizacji usługi nie ma być widoczna dla procesu korzystającego z tej usługi;
- pełna komunikacja grupowa,
- dalsze prace nad zapewnieniem odporności systemu ANSA na uszkodzenia. Prace te dotyczą odporności na uszkodzenia poszczególnych węzłów oraz przywrócenia stanu systemu przed awarią. Rozwojowi podlegają także aplikacje demonstracyjne dostarczone z ANSAware.

* * *

Architektura ANSA jest obecnie dostępna jako system o nazwie ANSAware w wersji 4.0. System taki jest zainstalowany w Instytucie Informatyki AGH w Krakowie. W obecnej konfiguracji pracuje on na komputerach firmy SUN o architekturze SPARC. Zrealizowano również przeniesienie ANSA na komputery IBM RS6000 oraz CONVEX 3200. Prowadzone są testy systemu, a także badania nad implementacją mechanizmu *„load balancig"* oraz bardziej zaawansowanych mechanizmów transparentności. ANSAware jest również wykorzystywany w procesie dydaktycznym jako przykład nowoczesnego narzędzia do konstrukcji i implementacji systemów rozproszonych.

LITERATURA


[1] ANSA: An Engineer's Introduction to the Architecture. 1989 Architecture Projects Mangement Limited

[2] ANSAware 4.0 Application Programer's Manaual. 1989 Architecture Projects Management Limited

[3] Marques J. A. and Guedes P.: Extending the Operating System to Support an Object-Oriented Enviroment. OOPSLA'89, 1989

[4] Olsen Michael H.: A Persistent Object Infrastructure for Heterogenous Distributed Systems. Hewlett-Packard Laboratories, Bristol, England

[5] Warne John: ANSA-Assumptions, Principles, and Structure. Conference proceedings of Software Engineering Enviroments'91, University College of Wales, Aberystwyth, 1991

ZYGMUNT DRĄŻEK
PETER J. A. REUSCH

# System FCS jako środowisko programowe do budowy systemów wspomagania decyzji

Ze względu na dużą liczbę istniejących narzędzi programowych do tworzenia systemów wspomagania decyzji (SWD) i różnego obszaru ich zastosowań, w literaturze przedmiotu nie występuje jednolita definicja dla tych narzędzi. Od programowych narzędzi do wspomagania decyzji oczekuje się wspomagania użytkowników przy tworzeniu modeli, zarządzaniu danymi, budowie prognoz dla przedsiębiorstwa oraz pomocy podczas analizy i oceny otrzymywanych wyników. W związku z tym do istotnych części składowych tych narzędzi zalicza się:

- język modelowania, który umożliwia formułowanie równań modelu w języku akceptującym notację wykorzystywaną przez użytkownika. Tego rodzaju forma tworzenia modelu odróżnia go od arkuszy kalkulacyjnych, w których model jest tworzony przez połączenie komórek tabeli arkusza kalkulacyjnego;
- ukierunkowany na różnych użytkowników zestaw funkcji programowych. Funkcje te umożliwiają wykorzystanie języka do rozwiązywania problemów planowania w następujących dziedzinach:
  - kosztów i budżetu,
  - inwestycji i finansów,
  - zbytu i marketingu,
  - przedsięwzięć;
- wbudowane funkcje analizy w postaci:
  - *What-if-Analyse* – do badania zachowania się modelu po zmianie jednej lub kilku wartości w modelu. Po dokonaniu zmiany następuje przeliczenie modelu z podanymi wartościami i zapamiętanie wyników. Celem zmniejszenia pracochłonności operowania modelem, polegającej na ustaleniu nowych wartości i przeliczeniu modelu, powinna istnieć możliwość zdefiniowania wartości granicznych i przyrostu ich wartości w jednym przebiegu. Po zapoczątkowaniu modelu następuje jego krokowe przeliczanie oraz udokumentowanie wyników;
  - *How-to-achieve-Analyse* – wartości zmiennej wejściowej są ustalane w ten sposób, że wielkość wyjściowa modelu osiągnie pożądaną wartość.

Narzędzia programowe do wspomagania decyzji oprócz elastycznych form wyprowadzania obliczonych wyników powinny umożliwiać zapamiętanie otrzymanych danych w postaci sformatowanej lub binarnej, w celu ich późniejszego wykonania lub przekazania do innych systemów. Ponadto powinno być możliwe dokonywanie porównań wyników z różnych przebiegów oraz ich ocena.

Narzędzia te powinny ponadto być wyposażone w:

- rozkazy i funkcje do rozwiązywania typowych problemów z zakresu planowania,
- szczegółową kontrolę składni i efektywne narzędzia do poprawiania błędów,
- łatwe w użytkowaniu standardowe funkcje analizy i prognozowania,
- rozkazy i procedury do elastycznego formułowania modeli oraz ich zmiany i zarządzania,
- możliwości generowania zestawień w różnych układach na ekranie, drukarce oraz ploterze.

Do najbardziej rozpowszechnionych narzędzi do tworzenia SWD zalicza się system FCS, którego polską wersję przygotowano w Instytucie Cybernetyki Ekonomicznej i Informatyki Uniwersytetu Szczecińskiego.

System Micro-FCS (*Financial Control System*) został opracowany przez firmę EPS-Consultants (*Evaluation and Planning Systems*) i obecnie jest rozpowszechniany przez firmę PILOT. FCS został przystosowany do tworzenia i obsługi modeli przez różnych użytkowników na ich stanowiskach pracy. Pracuje na tzw. płaszczyźnie systemu, z której użytkownik przechodzi do wybranego poziomu. FCS jako system konwersacyjny pracuje każdorazowo z aktualnym modelem (nazwanym logiką), tabelą danych, specyfikacją zestawień oraz zbiorem instrukcji służących do wykreślenia wyników modelu. Dysponuje także generatorem zestawień oraz interaktywnymi narzędziami do tworzenia wykresów.

Budowa modeli jest wykonywana w następujących fazach:

- specyfikacja równań modelu,
- wprowadzanie danych,
- specyfikacja wyjść i grafiki,
- eksperymenty na modelu z wykorzystaniem funkcji do analizy.

Tworzenie równań modelu jest wspomagane pełnoekranowym edytorem tekstu. Każda linia modelu zawiera jedno równanie, które może być identyfikowane przez nazwę lub numer linii. W modelu mogą być wykorzystane rozkazy do formułowania warunków operatorów logicznych, zdanie *GO TO* oraz funkcje dialogu z modelującym. Dane wejściowe są wprowadzane jako pojedyncze wartości do tabeli danych utworzonej na podstawie równania modelu. Przyporządkowanie wartości tabeli dla zmiennej modelu następuje przez podanie numeru linii lub nazwy zmiennej. Uzupełnienie tabeli wynikami następuje podczas wykonywania obliczeń. Za pomocą standardowych rozkazów wynikowa tabela może być poddana wymaganym przekształceniom. Mogą być też wykonane standardowe wydruki. Dostosowane do potrzeb użytkowników zestawienia oraz grafika mogą być zrealizowane po napisaniu sekwencji poleceń, zapisaniu ich w zbiorze tekstowym i wywołaniu specjalnej funkcji języka. Oddzielna realizacja czynności tworzenia modelu, zarządzania danymi i specyfikacji wydruków przyczyniły się do podniesienia efektywności tworzenia modeli planowania, gdyż mogą one współpracować z różnymi zestawami danych, a wyniki obliczeń mogą być na żądanie użytkownika przedstawiane w dowolnej formie.

Istotną zaletą FCS jest jego ukierunkowanie na użytkownika i duża elastyczność w tworzeniu oprogramowania. Te właściwości zrealizowano na podstawie rozbudowanych funkcji zew-

nętrznych użytkownika, oddzieleniu danych wejściowych i wyjściowych od struktury modelu oraz niezależną od tej struktury formę prezentacji i analizę wyników. W związku z powyższym model zostaje każdorazowo podzielony i jest realizowany w następujących fazach:

- opracowania wprowadzania logiki modelu,
- wprowadzania zmiany danych wejściowych,
- wykonania modelu,
- opracowania wykonania zestawień w formie tabulogramów lub grafiki,
- analizy.

Pracę z systemem FCS rozpoczyna się od zdefiniowania liczby kolumn dla arkusza roboczego, można ją podzielić na następujące pięć etapów:

I – opracowanie logiki modelu, która jest formułowana w pełnoekranowym edytorze i zawiera – przedstawione w postaci równań oraz ujmowane z różnych punktów widzenia – związki zakładowe i gospodarcze;

II – wprowadzenie danych i wartości początkowych dla obliczenia równań modelu. Dane są wprowadzane do wygenerowanej na podstawie równania modelu tabeli danych, w której kolumny przedstawiają okresy czasu, natomiast w wierszach są zawarte dane z konkretnymi wartościami dla tych okresów;

III – przeliczanie modelu na podstawie sformułowanej logiki i wprowadzanych danych. W trakcie wykonywania obliczeń następuje połączenie wierszy logiki z wprowadzonymi danymi i w wyniku daje wypełnioną tabelę. W celu uelastycznienia pracy, opracowano rozkazy umożliwiające selektywne wykonywanie obliczeń oraz testowanie modelu po zmianie wybranych warunków;

IV – wykonanie zestawień. Wyniki modelu, które zostały zapamiętane w postaci w tabeli danych, mogą być przedstawione, zgodnie z wymaganiami użytkownika, w formie zestawień roboczych, zestawień indywidualnych oraz grafiki. Tworzenie zestawień indywidualnych i grafiki odbywa się w pełnoekranowym edytorze przy użyciu standardowych poleceń i klawiszy funkcyjnych lub jest wykonywane za pomocą generatorów zestawień i grafiki. Wygenerowanie zestawień jest dokonywane na podstawie wprowadzonych przez użytkownika informacji w Popup menu. W dalszej kolejności wyniki modelu mogą być wykreślone na wysokorozdzielczych urządzeniach w wybranych kolorach i w wielu formach graficznych;

V – analiza modelu. W etapie tym jest możliwe przeprowadzenie analizy wrażliwości i poszukiwania celu. Analiza wrażliwości ma zadanie udzielenia odpowiedzi, bez konieczności modyfikacji modelu, na pytania typu: *„Co będzie gdy...?"*. Po podaniu nowych wartości wejściowych, następuje przeliczenie modelu i przedstawienie wyników. Uzyskane wyniki są przechowywane w obszarze roboczym i nie zmieniają w nim danych. Na podstawie uzyskanych danych można przeprowadzić dalsze eksperymenty i, zmieniając wybrane elementy modelu, poszukiwać najlepszego wariantu dla sytuacji decyzyjnej. Poszukiwanie celu: w tym przypadku poszukuje się warunków wejściowych, które przy zachowaniu aktualnej logiki, prowadzą do uzyskania pożądanej wartości. Po wydaniu odpowiedniego rozkazu, po którym występuje podanie pozycji wielkości docelowej i wyspecyfikowanie jej wartości oraz zmiennej do estymacji, następuje iteracyjne przeliczenie modelu i przedstawienie wartości dla poszczególnych iteracji.

Język FCS zalicza się do wysokozorganizowanych języków programowania. Jest wyposażony w standardowe narzędzia do:

- tworzenia złożonych warunków,
- rozwiązywania równań jednoczesnych,
- definiowania przez użytkownika własnych funkcji i rozkazów,
- korzystania z funkcji standardowych dla przetwarzania tekstów, obsługi dat, obsługi ekranu i zbiorów oraz funkcji matematycznych.

Ponadto język FCS oferuje całą gamę możliwości do tworzenia profesjonalnego oprogramowania, np. przez:

- wykorzystywanie podprogramów,
- stosowanie technik programowania strukturalnego,
- sterowanie zastosowaniami przez zbiory JOB,
- sterowanie pracą ekranu,
- obsługę wielu typów drukarek,
- obsługę wielu typów ploterów,
- wykorzystywanie narzędzi wspomagających tworzenie złożonych modeli.

Otwarta struktura zbioru HELP pozwala na każdorazowe wprowadzanie opisów dla zdefiniowanych przez użytkownika funkcji.

Przy opracowywaniu modeli wykorzystywanych w planowaniu, istotne znaczenie ma możliwość odwzorowywania wielowymiarowych struktur. FCS dysponuje w tym celu dodatkowymi komponentami, umożliwiającymi:

- odwzorowywanie struktur składających się z maksymalnie 12 wymiarów,
- hierarchiczną konsolidację wyników,
- komunikację z użytkownikiem za pomocą okienek oraz elastyczne techniki zapytań.

Każdy wymiar zawiera wybrane elementy struktury modelowanego obiektu (np. filii, produktów). Po wykonaniu obliczeń wyniki z tych podstruktur są dostępne na wyższych poziomach modelu.

Zasygnalizowane powyżej możliwości języka FCS wykazują, że może być on wykorzystywany jako narzędzie programowe do tworzenia modeli i komputerowego wspomagania podejmowania decyzji, ponieważ zawarte w nim konstrukcje językowe są stosowane do języka, jakim posługują się specjaliści zajmujący się tymi problemami. Stosowanie tego języka w znacznym stopniu zmniejsza nakład pracy przy tworzeniu i oprogramowywaniu modeli, a także redukuje możliwości wystąpienia błędów.

---

**W związku z wprowadzeniem powszechnego podatku dochodowego i koniecznością informowania właściwych Urzędów Skarbowych o wszystkich wypłatach zwracamy się do Autorów wysyłających teksty do opublikowania o podawanie następujących danych:**

- **nazwisko, imiona (pierwsze i drugie),**
- **imiona: ojca i matki,**
- **miejsce i data urodzenia,**
- **numer identyfikacyjny PESEL (wpisany przez Biuro Meldunkowe do dowodu osobistego – nie mylić z numerem dowodu osobistego!),**
- **dokładny adres (miejsce zameldowania),**
- **adres Urzędu Skarbowego właściwego dla miejsca zamieszkania Autora (bardzo ważne – bez tej informacji nie możemy przekazać honorarium do wypłaty!).**

---

ZYGMUNT TOPOLEWSKI
Wrocław

# Klucze szyfrujące w metodzie RSA

Od 1978 r. mówi się dość głośno o nowej amerykańskiej metodzie szyfrowania informacji. Natomiast już od 1983 r. mówi się oficjalnie o metodzie szyfrowania za pomocą dwóch kluczy szyfrujących, zwaną metodą RSA *Public-Key-Cryptosystem*. Nazwa metody pochodzi od skrótu nazwisk jej autorów (Riverst, Shamir, Adleman). Dość obszernie na ten temat napisano w ostatnio wydanej książce „Kryptografia i ochrona danych" D. E. Robling Denninga, WNT, Warszawa 1992 r.

Zainteresowało mnie zdanie opublikowane w czasopiśmie *Scientific American* przez Martina Gardena, który stwierdził: *...jest to nowy rodzaj szyfru, którego przełamanie zajmie miliony lat*. Przyjrzyjmy się bliżej tej ciekawej metodzie w świetle powyższego stwierdzenia.

Po pierwsze, metoda ta opiera się na elementarnej zasadzie szyfru CEZARA, która obowiązuje od ponad 2000 lat i nosi nazwę „tabeli przyporządkowania". Według tej zasady wszystkie litery alfabetu i nie tylko, otrzymują z góry określone wartości liczbowe, np.: $A = 01$, $B = 02$, .... , $Z = 24$.

Po drugie, metoda została nazwana „metodą jawnego klucza", co jest nazwą co najmniej mylącą. Zastanówmy się więc nad cechą jawności kluczy.

Oficjalnie podano, że w metodzie występują dwa klucze – jeden jawny, a drugi – tajny. Już to stwieedzenie stawia pod znakiem zapytania samą nazwę. Model matematyczny klucza jawnego, czyli szyfrującego, jest następujący:

$$y = z^u (mod\, n)$$

Natomiast model matematyczny klucza tajnego, czyli deszyfrującego wyrażony został wzorem:

$$y^* = y^b (mod\, n)$$

Podane powyżej wzory matematyczne, chociaż autorzy nazwali je kluczami, faktycznie roli kluczy nie spełniają. Jednak, jak sam autor wspomnianej wyżej książki, na 2. 21, stwierdza: *przełamanie szyfru przy znajomości wyłącznie metody szyfrowania powinno być niemożliwe*, tak i jest w omawianej metodzie RSA. Znając więc metodę i modele matematyczne, postarajmy się znaleźć klucz.

Na początku analizy wyjaśnijmy znaczenie użytych we wzorach symboli:
$z$ – grupa dwóch znaków meldunku, pobrana z tabeli przyporządkowania – jest to wartość tajna,
$u$ – określona liczba pierwsza – wartość jawna,
$n$ – iloczyn dwóch liczb pierwszych – wartość jawna,
$b$ – obliczona liczba – wartość ściśle tajna.

Dokonajmy teraz analizy pozostałych wielkości liczbowych, które w widoczny sposób nie występują w zaprezentowanych modelach matematycznych:

- wielkość $n$ obliczamy z dwóch dowolnie przyjętych liczb pierwszych $p$ i $q$, za pomocą wzoru: $n = p * q$. Przyjęte wielkości liczb pierwszych $p$ i $q$ są wielkościami tajnymi;
- wartość $b$ obliczamy ze wzoru $b * u = 1 \,(\text{mod}\, t)$. Widzimy, że w tym przypadku występuje nowa wielkość $t$, którą obliczamy ze wzoru $t = (p - 1)(q - 1)$.

W ostateczności wzór możemy zapisać następująco:

$$\frac{b * u}{t} = 1 \qquad \text{czyli} \qquad \frac{b * u}{(p-1)(q-1)} = 1$$

Znając wielkości $u$, $p$ oraz $q$ dobieramy taką wartość dla $b$, ażeby po prawej stronie równania otrzymać jedność.

Z tej krótkiej analizy wynika jasno, że na podstawie jawnej wielkości $n$ obliczymy i to w bardzo prosty sposób, dwie podstawowe wielkości $p$ i $q$. Znając te dwie wielkości praktycznie dokonaliśmy złamania szyfru, gdyż możemy obliczyć $t$, a następnie szukaną wielkość $b$, która służy do deszyfracji. Stąd rzeczywistym kluczem są dwie wielkości, dwie liczby pierwsze, $p$ oraz $q$, które są trzymane w ścisłej tajemnicy. Nie ma więc tu mowy o rzekomym „jawnym" kluczu.

Co jednak jest istotą o dużym znaczeniu w amerykańskiej metodzie jawnego klucza RSA? Do sukcesu autorów należy zaliczyć wspaniałe ukrycie rzeczywistego klucza w dwóch różnych modelach matematycznych. W tej sytuacji posłuchajmy, co na temat metody mówią zachodnie czasopisma i pracownicy naukowi.

F. Weber w artykule *Der Traum von ser perfekten Verschlüsselungsmethode, Tchnische Rundschau* 32/85, pisze. ... *zaszyfrowana może być tylko właściwa informacja użytkowa. Nagłówek meldunku wraz z potrzebnymi do sterowania informacjami, musi być przekazywany otwartym tekstem ... Ponieważ we wszystkich węzłach sieci, informacje sterujące są podawane tekstem otwartym, zachodzi konieczność dodatkowego zabezpieczenia węzłów.* Podobne wypowiedzi możemy znaleźć także w innych publikacjach zachodnich.

Ta troska, wyrażona przez E. Webera wynika stąd, że metoda ta ma możliwość generowania tylko jednego rodzaju szyfru. Tymczasem w sieciach komputerowych zachodzi konieczność dodatkowego szyfrowania nagłówka oraz informacji sterujących, odrębnych rodzajem szyfru. Stąd praktyczne zastosowanie metody staje pod znakiem zapytania.

Kolejne zagadnienie, to stosowane w metodzie liczby pierwsze. Prof. Wolfgang Biegert, z Fachhochschule für Technik-Stuttgart, analizując liczby pierwsze stosowane w metodzie stwierdza, że każda z liczb powinna zawierać więcej niż 45 znaków. Wskazana jest liczba 150 a nawet 200 znaków. Dla potwierdzenia konieczności stosowania tak długich liczb pierwszych podaje, że po zastosowaniu liczb pierwszych o długości 40 znaków nastąpiło złamanie szyfru w przeciągu 90 minut a więc

jednak nie miliony lat! Złamania szyfru dokonano za pomocą komputera CRAY-2.

Zastanówmy się czy tak długie liczby pierwsze mogą być praktycznie wykorzystane w metodzie. Podam przykład szyfrowania opracowany przez prof. W. Biegerta, który najlepiej ilustruje zasady szyfrowania stosowane w metodzie jawnego klucza RSA.

**Dane:**
$n = p * q = 52961$
$u = 131$
Tabela przyporządkowania: $A = 01$, $B = 03$, $C = 07$, $D = 04$, $E = 0{,}5$, $F = 10$, $G = 09$, $H = 08$, itd.

**Zadanie:**
Zaszyfrować i przesłać grupę liter *EG*. W nagłówku, tekstem otwartym przesyłamy wartości $n$ oraz $u$. Jednocześnie w sposób tajny adresatowi zostały przekazane wielkości liczbowe $p$ oraz $q$.

**Rozwiązanie:**
Z tabeli przyporządkowania odczytujemy wartości liczbowe: $E = 05$ oraz $G = 09$, co zapisujemy jako: $z = 509$. Wielkości te podstawiamy do wzoru:
$y = 509^{131} \pmod{52961}$
Kolejne potęgi $y$ otrzymujemy według następującej zasady:
$509^{1} \pmod{52961} = 509$
$509^{2} \pmod{52961} = 47237$
$509^{4} \pmod{52961} = 34278$
$509^{8} \pmod{52961} = 41499$
$509^{16} \pmod{52961} = 34164$
$509^{32} \pmod{52961} = 24378$
$509^{64} \pmod{52961} = 11503$
$509^{128} \pmod{52961} = 22431$

Potęgę liczby 131 otrzymujemy z sumy potęg $128 + 2 + 1$:
$(128 + 2)$ $22431 * 47237 \pmod{52961} = 35381$,
$(\ldots + 1)$ $35381 * 509 \pmod{52961} = 2189$.

Otrzymany wynik 2189 przesyłamy do adresata w postaci szyfrogramu.

Adresat po otrzymaniu szyfrogramu najpierw oblicza na podstawie przedstawionego powyżej wzoru ściśle tajną wielkość $b$. W wyniku obliczeń otrzymuje się wynik $b = 37271$. Wartości te podstawia do wzoru deszyfracyjnego $y^* = 2189^{37271} \pmod{52961}$.

Nie analizując poszczególnych faz obliczeń, które przebiegają zgodnie z wyżej podanym przykładem obliczania kolejnych potęg, adresat otrzymuje wynik: $y^* = 509$. Z tabeli przyporządkowania odczytuje litery $05 = E$ oraz $09 = G$.

W ten sam sposób postępujemy z każdą grupą kolejnych dwóch znaków szyfrogramu i meldunku, nawet w przypadku, gdy meldunek zawiera kilka tysięcy znaków. Przykład dotyczył bardzo małych liczb pierwszych, a przecież metoda wymaga stosowania liczb minimum 45 znaków każda. Wnioski pozostawiam Czytelnikowi.

Tyle uwag ogólnych na temat metody jawnego klucza RSA. Zachodzi w tym miejscu pytanie, czy istnieje możliwość spełnienia wszystkich wymagań stawianym szyfrogramom przesyłanym w sieciach komputerowych. Odpowiedź w tym przypadku jest twierdząca. Metoda taka została opracowana, oprogramowana i wdrożona przez zespół pracowników naukowych Politechniki Wrocławskiej. Wspominiana metoda otrzymała nazwę ZT-UNIKAKOD. Ogólny jej opis, wraz z przykładowymi szyfrogramami, wymaga odrębnego opracowania.

# PROGRESS-em przez ocean danych

**dokończenie ze s. 4**

różnymi językami. Wersja systemu przeznaczona dla projektanta jest obecnie dostępna w językach: angielskim, duńskim, holenderskim, niemieckim, norweskim, polskim, portugalskim, szwedzkim i włoskim. Wersja polska umożliwia wprowadzanie polskich liter wg wybranych standardu kodowania oraz sortowanie pól tekstowych zawierających polskie znaki; po polsku są też wyświetlane wszelkie komunikaty.

Jedną ze składowych systemu PROGRESS w wersji siódmej będzie *Translation Manager*, czyli moduł zarządzający przechowywaniem i udostępnianiem kilku wersji językowych łańcuchów znakowych będących integralną częścią aplikacji. Umożliwi to pisanie aplikacji wielojęzycznych (z możliwością dynamicznej zmiany języka komunikacji z użytkownikiem), co może znaleźć zastosowanie np. w obsłudze ruchu turystycznego.

## PROGRESS w Polsce

Po złagodzeniu ograniczeń COCOM w 1991 r. PROGRESS zaczął być sprzedawany również w Polsce. Początkowo mało widoczny, zdobywa sobie coraz większą popularność, jest prezentowany na różnych konferencjch czy Szkołach PTI, pojawiają się publikacje na jego temat. W maju 1992 r. odbyła się konferencja założycielska Polskiego Klubu Użytkowników PROGRESS-a. Wydawany jest dość regularnie biuletyn klubowy, powstał też katalog polskich aplikacji w języku PROGRESS (na razie niewielki objętościowo). Polski dystrybutor PROGRESS-a (CSBI) zapewnia użytkownikom serwis techniczny i konsultacje, prowadzi też kursy programowania w języku PROGRESS oraz prezentacje systemu. Kilkadziesiąt polskich firm software'owych pisze programy w PROGRESS-ie, przy czym niektóre z nich mają już status *Value Added Resselera* (VAR).

LITERATURA

[1] Campbell John: Programmer's Progress. White Star Software, 1991
[2] Codd E. F.: A Relational Model of Data for Large Shared Data Banks. CACM, June 1970, No. 6.
[3] Dokumentacja i materiały firmy Progress Software Corporation
[4] FROM THE LAB: Work In Progress. HP Professional, September 1992
[5] Gillin Paul: Fast action, not luck, puts casino on a roll. COMPUTERWORLD (USA), June, 8, 1992
[6] Ostaszewska Anna: Optymalizacja podziału ról klient – serwer. COMPUTERWORLD (PL), 7.09.1992
[7] Perschke Susan: The Secret Life of 4GLs. DBMS, November 1992
[8] Saad David: Robust 4GL system aids client-server development. DIGITAL NEWS, June, 8, 1992
[9] Simovici Dan: Progress. COMPUTER, June 1992
[10] TP1 Benchmark Report. Progress Software Corporation 1991.

# Rynek mikroprocesorów X86

Szeroka oferta procesorów typu 386 ze strony takich firm, jak Advanced Micro Devices Inc. (ADM) zmusiła Intel Corp. do zwiększenia produkcji układów 486 oraz obniżenia ich ceny. W rezultacie ceny systemów komputerowych działających na procesorach 486 spadły poniżej trzech tysięcy dolarów, a niektórych z nich – nawet poniżej dwóch tysięcy.

Przewidywane wprowadzenie do sprzedaży mikroprocesora typu 586 i wytwarzanie mikroprocesora typu 486 przez wiele firm pozwala przewidywać, że ceny te nadal będą spadać. Według danych, badającej ten rynek firmy International Data Corpotation, w 1991 r. sprzedano 705 tys. systemów komputerowych typu 486, na 1992 r. przewidywano 170% wzrostu sprzedaży (do 1,9 mln), natomiast na 1993 r. – 137% wzrost (do 4,5 mln sztuk). Tanie procesory i współzawodnictwo wśród sprzedawców systemów komputerowych powodują znaczne obniżenie stopy zysku. Systemy typu 486 spełniają wiele wymagań, którym systemy 386 nie mogły sprostać, co poszerza krąg kilentów o tych, którzy wykorzystują nowe 32-bitowe systemy operacyjne i wieloprzetwarzanie oraz potrzebują większych możliwości zapamiętywania i przetwarzania graficznego.

Główną zaletą systemów 486 jest ich szybkość obliczeniowa. Przechodząc od systemów 386 do 486 uzyskuje się od 100 do 300% wzrostu szybkości działania. Wydajność systemu jest większa nawet przy mniejszych częstotliowściach zegara. Np. system 486SX 20 MHz jest ok. 33% szybszy od systemu 386DX 33 MHz, a najszybszy na rynku (sierpień 1992 r.) system 486DX2/50 (o częstotliwości zegara 50 MHz) jest trzykrotnie szybszy od 386DX/33 oraz o 33% od 486DX/33.

Inną atrakcyjną systemów 486 jest możliwość łatwego poprawienia parametrów komputera za cenę zaledwie około 100 dolarów przez przejście na procesory o wyższej częstotliwości zegara lub wykorzystanie techniki podwajania częstotliwości, stosowanej ostatnio przez firmę Intel. Przykładem jest tu system 486DX2 66 MHz, który może pracować w systemach o częstotliwości 33 MHz.

Faktyczne usprawnienie szybkości zależy od kodu aplikacji oraz od konfiguracji sprzętu, załwaszcza stacji dyskowych. Np. firma Tolerate Inc. oceniła, że system 486/25 firmy Wyse Technology jest 75% szybszy od systemów 386/25 tej samej firmy, a system 486/33 – ponad dwukrotnie szybszy. Wzrost szybkości osiąga się głównie dzięki zastosowaniu koprocesora matematycznego. Wzrost ten po wprowadzeniu systemów 486 zależy też od zastosowanego systemu operacyjnego. Np. przy pracy pod 32-bitowym UNIXEM firma Select Sales Inc. (SSI) osiągnęła dwukrotny wzrost szybkości przy przejściu z systemu 386/33 do 486/33, ale przy pracy z niektórymi 16-bitowymi systemami operacyjnymi, systemy 386/40 są w niektórych przypadkach szybsze od systemów 486/25. Wspomina firma SSI nabyła systemy 486 od firm AST Research, Compaq Computer Corp i Wyse Technology. Do podobnych wniosków doszła też firma Star Technologies, która sama wytwarza takie systemy, wykorzystując płyty główne firm American Micronics Inc. oraz Gemini Computers. Według oceny tej firmy, używanie układów 486 jest szczególnie korzystne w zastosowaniach sieciowych i pracujących pod systemem Windows.

Mimo szybkiego rozwoju układów 486, projektanci kompletnych systemów (ang. *integrators*) nie osiągają zbyt wielkich korzyści, związanych zwykle z wprowadzeniem nowej jednostki centralnej. Systemy 486 stają się coraz bardziej powszechne, tak jak jeszcze niedawno zjawisko to obserwowano na rynku systemów 386. Powszechność ta wynika głównie z ich użycia w złożonych zastosowanich (systemy wieloprocesorowe, projektowanie wspomagane komputerem, systemy wydawnicze, graficzne itp.).

Jednocześnie Intel zapowiada na lato 1993 r. pojawienia się mikroprocesora typu 586 zwanego też P5, którego dostawy rozpoczną się pod koniec tego lub na początku przyszłego roku. Ma on osiągnąć szybkość przetwarzania 100 mln operacji na sekundę, a więc dwukrotnie więcej niż najlepsze wersje procesora 486. Powinno to zmniejszyć różnicę między komputerami o skróconej (RISC) i złożonej liście rozkazów (CISC), aczkolwiek w dającej się przewidzieć przyszłości kostki RISC będą przy prostych operacjach nadal szybsze od układów X86. Mimo tej przewagi na rynku dominują mikroprocesory Intela. Wg. firmy InfoCorp w 1991 r. sprzedano tylko 308 tys. mikroprocesorów RISC, w porównaniu do 20,4 mln procesorów Intela. Ponadto na komputerach osobistych opartych na układach X86 działa około 20 tys. programów użytkowych, a więc pięciokrotnie więcej od zastosowań na mikropocesorach SPARC firmy Sun.

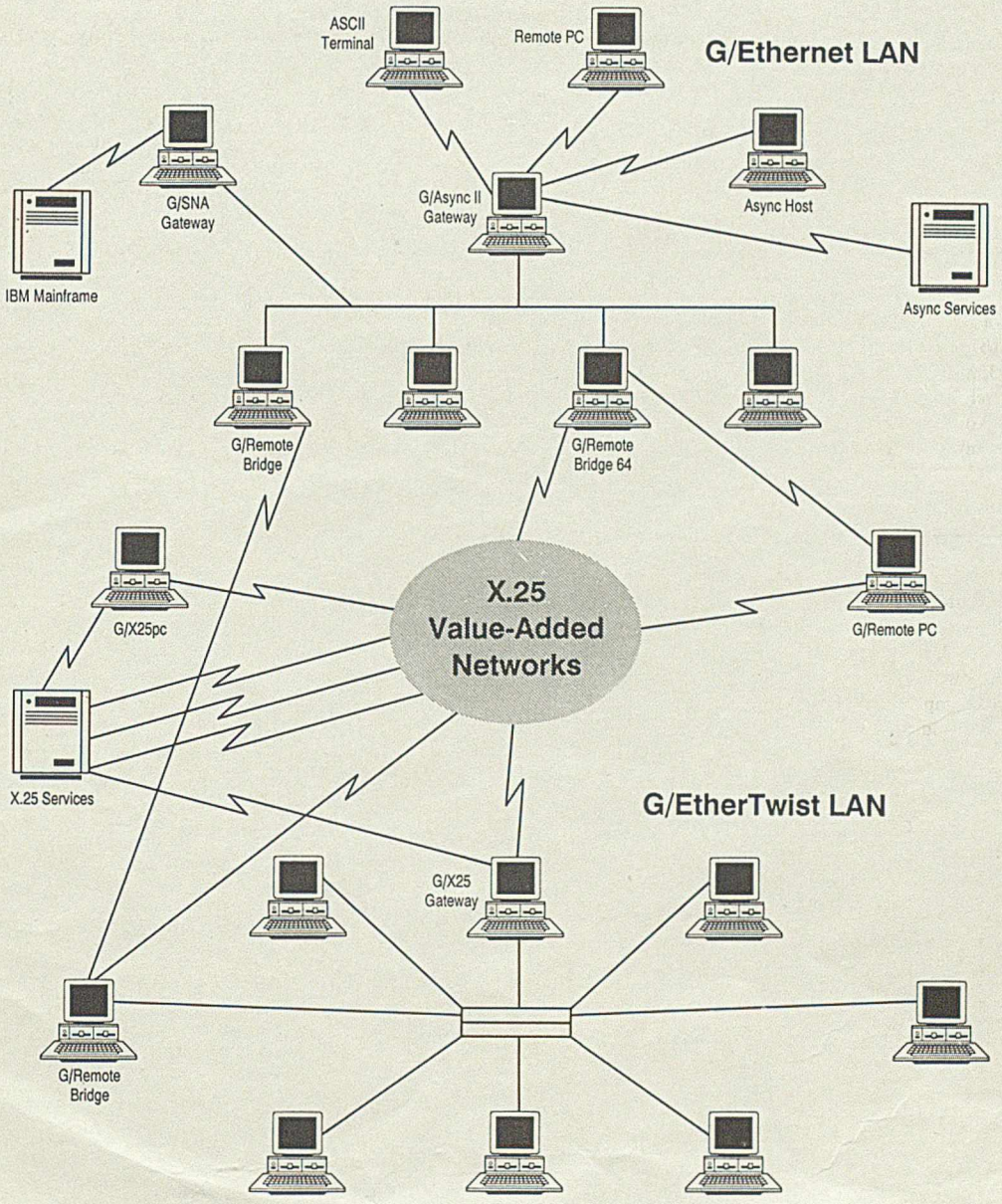
Tak więc układy oparte na mikroprocesorach 486 i 586 będą dominować w kategorii systemów złożonych, natomiast nawet w połowie lat dziewięćdziesiątych ilościowo dominować będą systemy oparte na komputerach osobistych klasy 386SX, głównie ze względu na znaczny rozwój komputerów przenośnych oraz laptopów. **(JR)**

**Informujemy, że począwszy od marca br. wpłaty na prenumeratę INFORMATYKI są przyjmowane nie tylko przez Zakład Kolportażu Wydawnictwa SIGMA-NOT, lecz także przez wszystkie urzędzy pocztowe nadawczo-odbiorcze oraz doręczycieli na terenie całego kraju.**

| | | |
|---|---|---|
| **Ostaszewska A., Fuglewicz P.: PROGRESS-em przez ocean danych**<br>INFORMATYKA 1993, nr 3, s. 1<br>Zwięzła charakterystyka cech i zalet systemu zarządzania relakcyjną bazą danych PROGRESS oraz zasady rozpowszechniania tego systemu w Polsce. | **Ostaszewska A., Fuglewicz P.: With PROGRESS across data ocean**<br>INFORMATYKA 1993, No. 3, p. 1<br>Concise characteristics of features and advantages of the PROGRESS relational database management system and the distribution of the system in Poland. | **Ostaszewska A., Fuglewicz P.: Mit PROGRESS über den Datenozean**<br>INFORMATYKA 1993, Nr. 3, S. 1<br>Eine kurzgefasste Charakteristik von Eigenschaften und Vorteilen des PROGRESS-Relativdatenbankverwaltungssystems und das Verbreiten dieses Systems in Polen. |
| **Wieczerzycki W.: Zarządzanie współbieżnością transakcji w obiektowych bazach danych**<br>INFORMATYKA 1993, nr 3, s. 5<br>Omówienie podstawowych algorytmów zarządzania współbieżnym wykonywaniem transakcji w obiektowych bazach danych. | **Wieczerzycki W.: Management of transaction concurency in object-oriented databases**<br>INFORMATYKA 1993, No. 3, p. 5<br>Discussion of basic algorithms for management of concurrent transaction execution in object-oriented databases. | **Wieczerzycki W.: Verwaltung von Transaktionsparallelverarbeitung in objektorientierten Datenbanken**<br>INFORMATYKA 1993, Nr. 3, S. 5<br>Eine Besprechung von Grundalgorithmen für die Verwaltung der Transaktionsparallelverarbeitung in objektorientierten Datenbanken. |
| **Miłosz M.: Mikrokomputerowa symulacja systemów – GPSS/PC**<br>INFORMATYKA 1993, nr 3, s. 11<br>Charakterystyka problemów symulacji systemów i języka GPSS oraz implementacji tego języka na mikrokomputery typu IBM PC. | **Miłosz M.: Microcomputer simulation of systems – the GPSS/PC**<br>INFORMATYKA 1993, No. 3, p. 11<br>Characteristics of system simulation problems and of the GPSS language, as well as of the language implementation on IBM/PC microcomputers. | **Miłosz M.: Mikrocomputersimulation von Systemen – GPSS/PC**<br>INFORMATYKA 1993, Nr. 3, S. 11<br>Eine Charakteristik von Systemsimulationsproblemen und der GPSS-Sprache, sowie von Implementierung dieser Sprache auf Mikrocomputern der IBM/PC–Klasse. |
| **Bijałd M., Gwóźdź M., Zieliński K.: Rozproszone przetwarzanie w systemach otwartych. Model ANSA**<br>INFORMATYKA 1993, nr 3, s. 17<br>Charakterystyka struktury oraz funkcji poszczególnych modułów modelu oprogramowania ANSA, a także budowy oprogramowania użytkowego i perspektyw rozwoju tego modelu. | **Bijałd M., Gwóźdź M., Zieliński K.: Distributed processing in open systems. The ANSA model**<br>INFORMATYKA 1993, No. 3, p. 17<br>Charakteristics of the structure and individual functions of the ANSA software model, as well as of application software building and the model development perspectives. | **Bijałd M., Gwóźdź M., Zieliński K.: Verteilte Verarbeitung in offenen Systemen. Das ANSA-Modell**<br>INFORMATYKA 1993, Nr. 3, S. 17<br>Eine Charakteristik von Struktur und Funktionen einzelner Modulen des ANSA-Softwaremodells, sowie von Bau der Anwendungssoftware und Entwicklungsaussichten dieses Modells. |
| **Drążek Z., Reusch P.J.A.: System FCS jako środowisko programowe do budowy systemów wspomagania decyzji**<br>INFORMATYKA 1993, nr 3, s. 23<br>Charakterystyka konstrukcji i możliwości funkcjonalnych systemu FCS jako efektywnego narzędzia do tworzenia systemów wspomagania decyzji. | **Drążek Z., Reusch P.J.A.: The FCS system as software environment for building of decision supporting system**<br>INFORMATYKA 1993, No. 3, p. 23<br>Characteristic of the construction and functional possibilities of the FCS system as effective tool for building of decision supporting systems. | **Drążek Z., Reusch P.J.A.: Das FCS-System als die Softwareumwelt für den Bau von Systemen zur Entschlussunterstützung**<br>INFORMATYKA 1993, Nr. 3, S. 23<br>Eine Charakteristik von Konstruktion und Funktionsmöglichkeiten des FCS-Systems als leistungsfähiges Werkzeug für Herstellung von Entschlussunterstützungssystemen. |
| **Topolewski Z.: Klucze szyfrujące w metodzie RSA**<br>INFORMATYKA 1993, nr 3, s. 25<br>Krytyczna ocena skuteczności amerykańskiej metody szyfrowania informacji RSA. | **Topolewski Z.: Secret code keys in the RSA method**<br>INFORMATYKA 1993, No. 3, p. 25<br>Critical evaluation of efficiency of the american RSA method for secret coding of information. | **Topolewski Z.: Kodierungsschlüssel in der RSA-Methode**<br>INFORMATYKA 1993, Nr. 3, S. 25<br>Eine kritische Beurteilung von Wirksamkeit der amerikanischen RSA-Geheimschriftmethode. |

# Produkty sieciowe LAN i WAN

Gateway communications, inc.




BEZKONKURENCYJNE PRODUKTY
SIECI KOMPUTEROWYCH NAGRADZANE PRZEZ:

**PC MAGAZINE, LAN MAGAZINE, INFO WORLD**

OFERUJE AUTORYZOWANY DYSTRYBUTOR:

MIKROB

**P.W.P.T. MIKROB SP. Z O.O.**

**20-346 Lublin, ul. Długa 5**
**Tel. (0-81) 420-61, faks 415-43, teleks 643776 MIKRO PL**

PC MAGAZINE EDITORS' CHOICE January 31, 1989 G/Ethernet AT

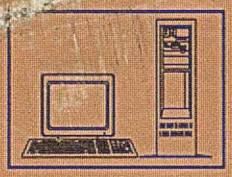
PC MAGAZINE EDITORS' CHOICE October 16, 1990 G/EtherTwist

*0/15/91*

Na życzenie wysyłamy katalog produktów. Atrakcyjny program współpracy dla dealer'ów.

# JUNISOFTEX®

*Przedsiębiorstwo Produkcyjno-Usługowe Innowacji Technicznych Biuro Techniczno-Handlowe 44-100 Gliwice ul. Konstytucji 11.*
***tel. 31-57-00, 31-51-31, 31-75-10, fax 31-75-10, centr. 31-90-81 do 88, telex 036233***
*Zespół wdrożeń i analiz systemów komputerowych wewn. 15 ,25, 282, Zespół inżynierii oprogramowania wewn. 14, 20, 245, Zespół d/s serwisu systemów Novell i sprzętu komputerowego wewn. 16, 245, Dział Handlowy wewn. 18, 28, 201, Szkoła Komputerowa wewn. 282, Sekretariat wewn. 250, Prezes wewn. 13, 272, Dyrektor wewn. 12, 272, Główny Księgowy wewn. 256,*
** Konto: PKO I/O Gliwice 27661-225198-136 **



## OFERUJEMY

***Kompleksową komputeryzację przedsiębiorstw obejmująca dostawę i wdrożenie zintegrowanych systemów komputerowych takich jak system:***

*finansowo-kosztowy (automatyczne rozliczanie kosztów)*
*+*
*gospodarka materiałowa*
*+*
*środki trwałe*
*+*
*wyposażenie*
*+*
*obrót towarowy fakturowanie sprzedaży*
*+*
*kadry-płace*
*+*
*techniczne przygotowanie produkcji*

***Sprzęt komputerowy w tym:***

*Komputery*
*Monitory*
*Dyski twarde*
*Drukarki (mozaikowe i laserowe)*
*osprzęt sieciowy (karty sieciowe, kable zwykłe i światłowodowe)*

***Systemy i oprogramowanie licencjonowane***

*DOS*
*Novell*
*Btrieve*
*SQL Base*
*Windows*

***Szkolenia komputerowe***

*podstawy obsługi komputerów*
*arkusze kalkulacyjne (Qpro, Lotus, EXCEL)*
*edytory tekstów (WordPerfect, TAG, ChiWriter)*
*bazy danych (dbase IV, Clipper, Paradox)*
*systemy operacyjne (DOS, Novell)*

# SOFTWARE SUPPORT FOR DOS, NOVELL 2.2, 3.11, 4.0, SQL BASE firmy GUPTA

*

## OPROGRAMOWANIE DLA PRZEDSIĘBIORSTW

*oferujemy własnej produkcji zakładowy system informatyczny PHU-Perfect(N) obejmujący swoim zakresem:*
*system finansowo-kosztowy umożliwiający automatyczne rozliczanie kosztów, atomatyczną dekretację, analizę rozrachunków*
*system obrotu materiałami, wyrobami, towarami (kody paskowe) w tym: stany magazynowe, ewidencje zamówień, fakturowanie sprzedaży (VAT) z automatyczną dekretacją rozdzielników kosztów i sprzedaży do systemu finansowo-kosztowego,*
*system ewidencji majątku trwałego i wyposażenia*
*system kadrowo-płacowy (SQL) z automatyczną dekretacją rozdzielników płacowych do systemu finansowo-kosztowego*
*system technicznego przygotowania produkcji (uk. 1993 r)*
*(oprogramowanie w systemie SQL BASE w przygotowaniu)*

*

## OPROGRAMOWANIE DLA URZĘDÓW MIAST I GMIN

*oferujemy własnej produkcji system finansowo-podatkowy UMG-Perfect(N) obejmujący swoim zakresem:*
*system finansowo-księgowy,*
*systemy podatkowe obejmujące: ustalanie wymiarów, naliczanie i kontrolę płatności należności podatkowych dla dowolnych podatników i/lub płatników podatków na ich kontach i kontach dochodów ze swobodną korektą dowolnych informacji w dowolnym okresie a w tym:*
*ewidencja zapłat z wykorzystaniem kodów paskowych, emisja wezwań do zapłaty, naliczanie odsetek.*
*system ewidencji majątku*

*

**Zapraszamy do współpracy w zakresie wdrożeń naszych systemów partnerów, - firmy informatyczne z całego kraju !!!**

0/7/92