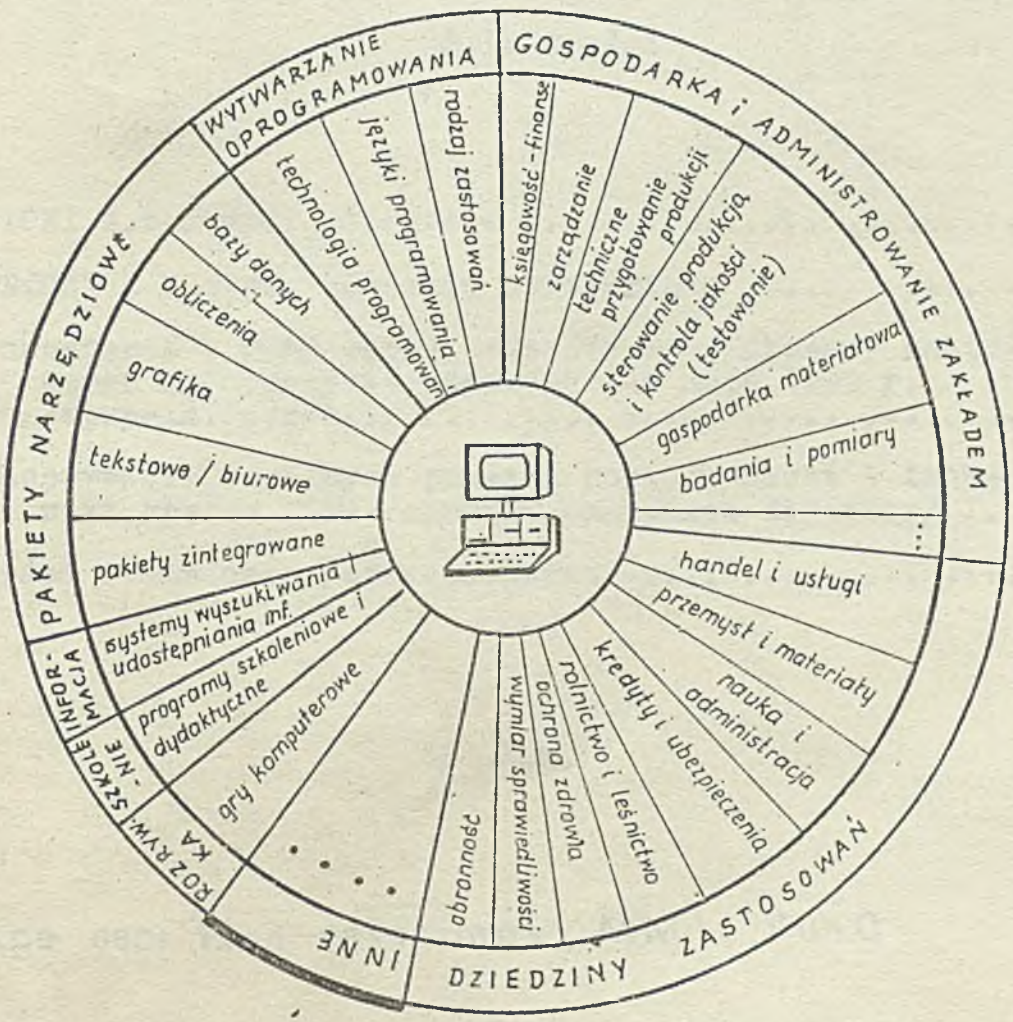
techniki komputerowe

5 '89

IMM

BIULETYN INFORMACYJNY



INSTYTUT MASZYN MATEMATYCZNYCH
BRANŻOWY OŚRODEK INTE

Rysunek na okładce: Ważniejsze obszary zastosowań mikrokomputerów

Druk IMM zam. 28/89 nakł. 1060 egz.

TECHNIKI KOMPUTEROWE

Rok XXVII Nr 5 1989

Spis treści

DWUMIESIĘCZNIK

Wydaje:

INSTYTUT MASZYN MATEMATYCZNYCH
ul. Krzywickiego 34 02-078 WARSZAWA tel. 28-37-29

BRANŻOWY OŚRODEK INFORMACJI NAUKOWEJ, TECHNICZNEJ I EKONOMICZNEJ

dr Andrzej ROWICKI

Instytut Maszyn Matematycznych

JĘZYKI OBIEKTOWE

WSTĘP

Na początku lat osiemdziesiątych coraz częściej używano w literaturze określeń takich, jak "języki obiektowe" i "języki obiektowo-zorientowane". Określenia te były używane nie zawsze jednoznacznie. Języki tak różniące się między sobą jak Smalltalk i Ada były czasami określane jako języki obiektowe. Język Smalltalk jest powszechnie uznawany za język obiektowy, natomiast język Ada trudno uważać za język obiektowy. Ada jest językiem pascalopodobnym, bardzo rozbudowanym, pozwalającym uzyskać dużą modularność programowania. Zawieranie (encapsulation) oraz konstrukcje takie, jak pakiet (package) i ogólny moduł programowy (generic program unit) można traktować jako konstrukcje obiektowe.

Historycznie rzecz biorąc pierwszym językiem obiektowym była Simula 67 opracowana na uniwersytecie w Oslo pod kierunkiem prof. O. Dahla. Przez długi czas nie zdawano sobie sprawy z obiektowości konstrukcji języka Simuli. Inspiracji do języków obiektowych można doszukiwać się w programowaniu strukturalnym.

Pierwsze elementy podejścia obiektowego pojawiły się w języku Algol. W Algolu pojawiły się elementy programowania strukturalnego takie, jak blok i procedura, które można traktować jako konstrukcje obiektowe. Z pojęciem bloku w Algolu łączy się pojęcie bloku dynamicznego, związane z realizacją bloku a więc z rezerwacją pewnego obszaru pamięci komputera na zmienne, wyniki, no i oczywiście na sam blok. Na ogół po wykonaniu bloku czy też procedury nie ma do nich użytkownik bezpośredniego dostępu, dostęp realizuje się przez ponowne wykonanie bloku czy też procedury. W Simuli (i w innych językach obiektowych) występuje twór programowy zwany obiektem, do którego można się odwoływać wielokrotnie bez konieczności wykonania programu zawartego w obiekcie. Wynika to z faktu, że obiekt może być wartością zmiennej. Obiekt można traktować jako egzemplarz utworzony według pewnego wzorca zwanego klasą. Klasa jest konstrukcją syntaktyczną, a więc jest również tworem statycznym. Odwołanie się do klasy powoduje utworzenie obiektu dynamicznego określonego strukturą klasy nazywanego obiektem klasy (reprezentantem klasy) w skrócie nazywanego powszechnie obiektem. Dokładniej mówiąc, obiekt zostaje utworzony w wyniku wykonania określonego programu w definicji klasy. Teoretycznie z klasy można utworzyć (wygenerować) nieskończenie wiele obiektów mających wspólne cechy (atrybuty) określone przez klasę. Utworzenie pojęcia obiektu i klasy stanowi istotny krok w rozwoju języków programowania. Powyższe pojęcia ułatwiają modularyzację programowania i znaczną dekompozycję rozwiązywanych problemów. Z języków obiektowych znaczne rozpowszechnienie uzyskała Simula, a ostatnio Smalltalk. Opracowany na UW język obiektowy Loglan mimo znacznych zalet nie uzyskał tak znacznej popularności. Z wielu istniejących języków obiektowych bardziej znane są języki: LOOPS, CLU, FLAVORS, oraz obiektowe wersje języka C takie, jak OBJECTIVE-C i C++.

W związku z postępem technologii układów scalonych stały się możliwe efektywne implementacje Smalltalku. Język Smalltalk uzyskuje ostatnio coraz większą popularność ze względu na bogate otoczenie programowe i rozbudowaną grafikę. Rozbudowane implementacje Smalltalku i innych języków obiektowych zawierające znaczną liczbę zdefiniowanych klas można traktować jako języki deklaratywne, gdyż komunikat określa "czego oczekujemy odwołując się do klasy" nie precyzując

sposobu uzyskania pożądanej informacji. Natomiast w zaimplementowanych klasach jest podany sposób (algorytm) uzyskania pożądanej informacji.

W pierwszej kolejności omówimy język Simula 67, na przykładzie którego omówimy podstawowe pojęcia języków obiektowych takie, jak klasa, obiekt, dziedziczenie. Następnie omówimy język Loglan kładąc nacisk na konstrukcje obiektowe. Najobszerniej omówimy język Smalltalk ze względu na składnię odbiegającą od powszechnie przyjętych konwencji w językach programowania oraz bardzo rozbudowaną hierarchię klas. Składnia języka Smalltalk jest zbliżona do języka naturalnego, co w intencji autorów miało ułatwić korzystanie z języka użytkownikom nie znającym języków programowania. Wydaje się, że takie podejście może prowadzić do wieloznaczności przez skojarzenia z językiem naturalnym i utrudniać właściwe zrozumienie konstrukcji języka Smalltalk.

Ostatnio Smalltalk jest reklamowany jako język sztucznej inteligencji na równi z językami LISP i PROLOG. Pojawiły się systemy doradcze (expert systems) implementowane w Smalltalku. System doradczy THE ANALIST przeznaczony do automatyzacji prac biurowych i sprawnej informacji, zawiera edytory tekstowe i graficzne, tabele i wykresy, systemy banków danych i grafikę handlową. Natomiast system HUMBLE jest jądrem systemu doradczego, umożliwiającym jednoczesne istnienie wielu baz wiedzy, wnioskowanie do "tyłu i przodu", stosującym model "pewności" zaczerpnięty z systemu MYCIN.

Simula 67

Na przykładzie Simuli omówimy podstawowe pojęcia języków obiektowych takie jak: klasa, obiekt, dziedziczenie.

Podstawowym nowym pojęciem w Simuli w stosunku do innych języków programowania (nie obiektowych) jest pojęcie obiektu. Obiekt jest "reprezentowany" przez samodzielny fragment programu (blok w terminologii algolowskiej) zdefiniowany w deklaracji klasy.

Formalna deklaracja klasy zawiera

```
class <nazwa klasy (lista parametrów formalnych)>;
<zbiór wartości>;<specyfikacja parametrów>;
begin <zbiór deklaracji lokalnych klasy>;
<lista instrukcji>;
end
```

Pozycja <zbiór wartości> w deklaracji klasy jest związana ze sposobem przekazywania parametrów formalnych. Gdy oczywisty jest sposób przekazywania parametrów (wynikający z typów parametrów) to ta pozyja w deklaracji klasy nie występuje.

Jeżeli w deklaracji klasy nie występuje lista instrukcji to klasa definiuje pewne struktury danych.

Utworzenie obiektu klasy następuje w wyniku wykonania wyrażenia

```
new <nazwa klasy (lista parametrów aktualnych)>;
```

Każdy obiekt klasy ma własną kopię zmiennych, a więc ma zdolność reprezentowania stanu dynamicznego. Obiekt klasy tak długo "żyje" jak długo są odwołania do niego.

Tytułem przykładu zdefiniujemy teraz klasę "liczba zespolona".Deklaracja klasy będzie miała następującą postać:

```
class liczba zespolona (x,y); real x,y;
begin
real r, alfa;
r:= rgst (x..2 + y..2);
alfa: = arctan (y,x);
end
```

Wykonanie wyrażenia

```
new liczba zespolona (3,4);
```

powoduje utworzenie obiektu, którym jest liczba zespolona

3 + i 4

W językach obiektowyczh można się odwoływać do obiektów. Wymaga to wprowadzenia nowego typu zmiennych. W Simuli wprowadzono nowy typ zmiennych, których wartościami mogą być obiekty danej klasy.

Typ ten nazwano typem referencyjnym (reference type).

Deklaracją zmiennej tego typu jest napis o postaci

```
ref (<nazwa klasy>) <nazwa zmiennej>;
```

Np. jeżeli A jest nazwą klasy a X nazwą zmiennej, to mówimy, że zmienna X jest zmienną referencyjną typu A.

Zmienną referencyjną, której wartością jest obiekt danej klasy uważa się za nazwę tego obiektu. Umożliwia to rozróżnianie obiektów tej samej klasy.

Nadawanie wartości zmiennym referencyjnym realizuje się za pomocą instrukcji przypisania, nazywanej instrukcją przypisania referencyjnego. Instrukcja ta ma następującą postać:

```
<nazwa zmiennej>: - <wyrażenie referencyjne>;
```

Wartością wyrażenia referencyjnego jest obiekt danej klasy. A więc jeżeli chcemy się odwołać do obiektu to musimy zadeklarować zmienną referencyjną i nadać wartość tej zmiennej za pomocą instrukcji przypisania referencyjnego. Gdy odwołujemy się do obiektu klasy liczba zespolona, należy napisać następujący ciąg deklaracji:

```
ref (liczba zespolona) Z;
Z: - new liczba zespolona (3,4);
```

Teraz możemy odwołać się do obiektu klasy liczba zespolona przez zmienną Z. Dostęp do zmiennych x,y,r, alfa uzyskujemy przez następujące wyrażenia Z.x, Z.y, Z.r i Z.alfa. Wykonanie ciągu instrukcji

```
print (Z.x);
```

```
print (Z.y);
print (Z.r);
print (Z.alfa);
```

dla obiektu new liczba zespolona (3.4) da nam w wyniku następujący ciąg liczb:

3.4, 5, 0.927

Pokażemy teraz jak korzystając z obiektów mniej złożonych można tworzyć obiekty bardziej złożone. Rozważymy to na przykładzie definiowania klasy punkt, a następnie korzystając z tej definicji utworzymy klasę wektor.

Klasę punkt definiujemy w sposób następujący:

```
class P(x,y); real x,y;;
```

Klasa o nazwie P jest wzorcem, według którego są budowane obiekty przedstawiające punkty. W treści klasy nie ma deklaracji lokalnych oraz listy instrukcji. Treścią klasy jest instrukcja pusta ograniczona dwoma średnikami.Przykładami obiektów klasy P mogą być np. punkty o współrzędnych (0,0) i (3,2)

Zdefiniujemy teraz klasę wektor odwołując się do definicji klasy P (punkt) przez zmienne referencyjne

```
class W (A,B); ref (P) A,B;;
```

W celu utworzenia obiektu klasy W, aby móc się odwołać do tej klasy należy jeszcze wprowadzić zmienną referencyjna

```
ref(W) wektor;
```

Utworzymy teraz obiekty klasy P będące punktami o współrzędnych (0,0) i (3,2) za pomocą instrukcji przypisania referencyjnego

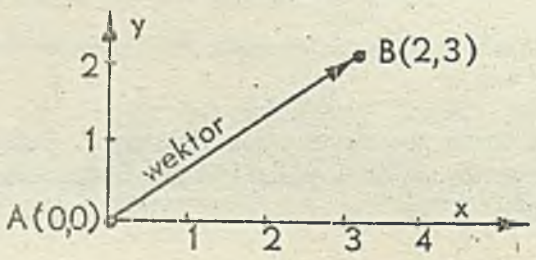
```
A: - new P (0,0);
B: - new P (3,2);
```

Wykonanie instrukcji przypisania referencyjnego powoduje utworzenie obiektów A i B na podstawie definicji klasy P będących punktami o współrzędnych (0,0) i (3,2).

Utworzymy teraz na podstawie definicji klasy W nowy obiekt będący wektorem

```
wektor: - new (A,B);
```

co możemy zinterpretować graficznie w sposób następujący:



Wracając do definicji klasy należy podkreślić, że w deklaracji klasy jest konieczna specyfikacja parametrów formalnych (określenie typu parametrów). W deklaracji klasy dopuszcza się następujące typy parametrów formalnych:

- proste (real, integer, Boolean, character)
- tekstowe (text)
- referencyjne
- złożone, tzn. tablice typów prostych, tekstowych i referencyjnych.

Należy zwrócić uwagę, że parametrem formalnym klasy nie może być procedura, etykieta ani też klasa.

Możliwe są tylko dwa sposoby przekazywania prarametrów a mianowicie przez wartość i przez referencję. Parametry typów prostych są przekazywane tylko przez wartość,

Natomiast typów referencyjnych oraz typów złożonych takich, jak tablice typów referencyjnych i tablice typów tekstowych tylko przez referencję. Wyjątek stanowią parametry typu tekstowego oraz tablice typów prostych, które mogą być przekazywane obydwoma sposobami.

Jeżeli chcemy parametr typu text czy też typu tablica typów prostych przekazać przez wartość, to należy nazwę tych parametrów poprzedzić słowem value i wstawić w deklaracji klasy między listę parametrów formalnych i zbiór specyfikacji. Jeśli nie podamy nazw tych parametrów po słowie value to przekazanie tych parametrów nastąpi przez referencję. W języku Simula 67 istnieje możliwość zdefiniowania nowej klasy, która będzie rozszerzeniem klasy A o nowe pożądane własności. Tak określoną klasę B nazywamy podklasą klasy A lub klasę prefiksowaną przez klasę A, co zapisujemy w sposób następujący.

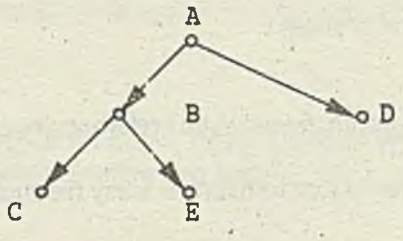
```
A class B; .......
```

Nazwanie klasy B podklasą klasy A wynika z tego, że obiekty tej klasy tworzą podzbiór w zbiorze obiektów klasy A. Obiekty tego podzbioru mają cechy zdefiniowane zarówno w klasie A, jak i w klasie B. Nazwa klasa prefiksowana bierze się bezpośrednio z postaci deklaracji podklasy.

Prefiksowanie umożliwia definiowanie dowolnie rozbudowanych klas. Wynika to stąd, że klasa, która jest prefiksowana sama może być z kolei prefiksem innej klasy. W ten sposób można utworzyć całą hierarchię klas. Mechanizm prefiksowania daje możliwość definiowania pojęć złożonych na bazie pojęć prostszych uprzednio zdefiniowanych. Klasa prefiksująca wprowadza podstawowe pojęcia, za pomocą których klasa prefiksowana definiuje nowe pojęcia. Hierarchia klas odpowiada więc hierarchii pojęć, w której pojęcia bardziej złożone są podrzędne względem prostszych. Rozpatrzmy następujący ciąg deklaracji klas

```
  class A
A class B
B class C
A class D
B class E
```

Klasy te tworzą pewną hierarchię, co graficznie można przedstawić w sposób następujący:



Strzałki na tym grafie określają kierunek dziedziczenia własności.

Na prefiksowanie klas w Simuli 67 są nałożone pewne ograniczenia mianowicie klasa prefiksująca musi być zadeklarowana na tym samym poziomie, na którym jest użyta jako prefiks.

Prefiksowanie może być również stosowane do bloków. Przyjęto, że klasa może być prefiksem nie tylko innej klasy, ale także bloku.Prefiksowanie bloków ma jednak trochę inne znaczenie. Istotą prefiksowania klas jest głównie rozszerzenie problemu zdefiniowanego w klasie prefiksującej. Natomiast blok prefiksowany stanowi program, który korzysta z narzędzi dostarczonych przez klasę prefiksującą w celu rozwiązania konkretnego zadania. Istotna różnica między blokiem prefiksowym, a klasą prefiksowaną polega na tym, że klasa jest deklaracją a blok jest instrukcją. Klasa prefiksowana stanowi wzorzec, według którego będą dopiero tworzone dymaniczne jednostki programu zwane obiektami. Natomiast instrukcja bloku prefiksowanego powoduje natychmiastowe utworzenie jednostki dynamicznej i wykonanie zawartych w niej instrukcji.

Język Simula 67 jest uniwersalnym językiem programowania i nie należy go traktować jako języka stworzonego do programowania tylko określonych zadań. Ponieważ jest uniwersalny może służyć jako baza do definiowania różnorakich języków ukierunkowanych problemowo. W tym celu wystarczy zdefiniować klasę opisującą wszystkie właściwości takiego języka. Użycie tej klasy jako prefiksu do bloku umożliwia posługiwanie się pojęciami zdefiniowanymi w klasie prefiksującej, a więc pozwala na programowanie w zdefiniowanym języku.

W Simuli 67 zdefiniowano różne klasy systemowe SIMSET i SIMULATION.Ponieważ definicje tych klas są zawarte w systemie umożliwiającym programowanie w Simuli 67, to można nimi prefiksować dowolny program (blok) i własne klasy.

Klasa SIMSET opisuje strukturę danych złożoną z list symetrycznych, co pozwala na posługiwanie się w programach listami symetrycznymi. Klasa SIMULATION definiuje środki pozwalające na pisanie programów symulujących systemy rzeczywiste. Każdy program symulacyjny oparty na klasie SIMULATION można traktować jako model badanego systemu. Podstawowym pojęciem używanym przy konstruowaniu modelu jest pojęcie procesu. W klasie SIMULATION jest zadeklarowana klasa o nazwie proces. Są również zadeklarowane inne klasy ułatwiające tworzenie modelu symulacyjnego badanego systemu rzeczywistego.

Loglan

Język programowania Loglan został opracowany w Instytucie Informatyki UW. Prace nad Loglanem rozpoczęły się w drugiej połowie lat siedemdziesiątych. W 1977 r. był gotowy pierwszy raport Loglanu. W 1982 r. został uruchomiony kompilator na minikomputer Mera 400 napisany w Fortranie. W 1983 r. powstała oszczędniejsza wersja (dwukrotny zysk pamięciowy) tego kompilatora napisana w języku asemblerowym Mera. Do końca 1986 r. powstały interpretery na maszynę Siemens (zgodną z IBM 370), VAX 780/VMS, IBM PC. Od 1985 r. trwają prace nad nową udoskonaloną wersją języka i projektem środowiska programistycznego, którego język byłby składową, zawierającą między innymi edytor sterowany składnią i system monitorowania wykonania programów. W roku 1987 rozszerzono środowisko Loglanu o dwie interesujące klasy: TextWindow i ANSI.

. Klasa TextWindow umożliwia działanie na obiektach będących okienkami zawierającymi teksty. Każdy obiekt klasy TextWindow może być wyświetlony na ekranie, wymazany z ekranu (schowany), przesuwany, rozciągany i ściskany, może być wyposażony w ramkę, można mu "zlecić" wydrukowanie tekstu w nim zapisanego lub wczytanie tekstu, można też dokonać inwersji lub zmiany kolorów jasnego i ciemnego.

Klasa ANSI umożliwia wyświetlanie na ekranie różnych rodzajów pisma: wytłuszczone, odwrócone, migające i podkreślone, a także kombinacje tych pism. Ponadto klasa ta zawiera procedury sterujące położeniem kursora (znacznika) i procedurę odczytywania znaków z klawiatury. Klasa ANSI może służyć jako otoczenie (prefiks) programu, w którym chcemy wyświetlać na ekranie komunikaty o różnych krojach pisma.

Loglan jest językiem pascalopodobnym. Podstawowe konstrukcje "obiektowe" zostały zaczerpnięte z Simuli 67. W Loglanie usunięto pewne istotne ograniczenia występujące w Simuli 67, unowocześniono syntaktykę, wprowadzono nowe mechanizmy (np. obsługę sytuacji wyjątkowych).

Podstawową nowością w stosunku do takich języków programowania, jak Pascal czy też Ada jest możliwość wprowadzania w Loglanie nowego typu modułów. Moduły te są oznaczone jako CLASS, COROUTINE, PROCESS, HANDLER i umożliwiają obliczenia równoległe i prawie równoległe, obsługę sygnałów i sytuacji wyjątkowych w modułach (Handler) obsługę sytuacji wyjątkowych. Loglan dopuszcza całkowicie dynamiczne tablice i ma niewiele ograniczeń na przekazywanie parametrów (np. można przekazać nazwy modułu jako parametr. Język Loglan można traktować jako język obiektowy, gdyż moduły CLASS, COROUTINE, PROCESS i HANDLER służą jako wzorce do konstrukcji obiektów zgodnie z opisami zawartymi w deklaracjach tych modułów. Z drugiej strony język zachowuje wszystkie pożyteczne i sprawdzone mechanizmy tradycyjnego programowania imperatywnego.

Podobnie jak w Simuli 67, najbardziej interesującą cechą Loglanu jest operacja składania modułów nazywana operacją prefiksowania. Prefiksowanie jest dwuargumentową operacją na modułach programu. W Loglanie prefiks powinien być klasą (CLASS), współprogramem (COROUTINE) lub procesem (PROCESS), natomiast w Simuli 67 może być tylko klasą. Moduł prefiksowany może być dowolnego rodzaju: procedurą, klasą, współprogramem, funkcją, procesem lub blokiem (w Simuli 67 tylko klasą lub blokiem). W Simuli występują pewne ograniczenia na prefiksowanie, których uniknięto w Loglanie, a mianowicie moduł prefiksowany i prefiksujący muszą być zadeklarowane na tym samym poziomie. Ogranicza to znacznie możliwości dziedziczenia własności i ogranicza elastyczność pisania programów. Wyraźnie to widać na przykładzie klas systemowych SIMULATION i SIMSET w Simuli 67, które można traktować jako języki problemowo ukierunkowane przeznaczone do symulacji i wykonywania operacji na listach symetrycznych. Klasa SIMULATION jest prefiksowana klasą SIMSET. W Simuli 67 nie ma mechanizmów umożliwiających rozszerzenie tych klas, ograniczenie do deklaracji klas na tym samym poziomie uniemożliwia rozszerzenie biblioteki klas zgodnie z potrzebami użytkowników. Powoduje to również trudności w niezależnej kompilacji modułów, co ogranicza znacznie zakres zastosowań Simuli 67.

Dla porządku omówimy teraz najważniejsze własności języka Loglan. Podstawowe instrukcje występujące w Loglanie można podzielić na następujące rodzaje:

- przypisanie
- wywołanie procedury
- instrukcje strukturalne warunkowe i iteracyjne
- instrukcje przepływu sterowania

Po lewej stronie instrukcji przypisania może wystąpić ciąg zmiennych oddzielonych przecinkami, wtedy nastąpi równoczesne przypisanie wartości wyrażenia po tej stronie wszystkim zmiennym występującym po lewej stronie. Wyrażenie

```
A(i), i := 1;
```

oznacza, że wartości zmiennej wyniku wykonania instrukcji przypisania będzie równa 1.(A(1),1).

Procedurę wywołuje się za pomocą instrukcji call. Wywołanie procedury następuje w wyniku wykonania wyrażenia

```
call < identyfikator procedury (<lista parametrów aktualnych >)>;
```

Składowymi instrukcji strukturalnych mogą być ciągi instrukcji, a nie tylko pojedyncze instrukcje. Dla uniknięcia niejednoznaczności instrukcje strukturalne kończą się słowem kluczowym będącym odróżnieniem słowa kluczowego zaczynającego instrukcję

```
if...fi
do...od
case...esac
```

Instrukcjami warunkowymi w Loglanie są instrukcje alternatywy (instrukcje if) oraz instrukcja wyboru warunkowego (instrukcja case).

Najbardziej ogólną postacią instrukcji iteracji jest nieskończona pętla - ciąg instrukcji ujęty w nawiasy syntaktyczne do...od. Wyjście z takiej pętli umożliwia instrukcja sterująca exit. W Loglanie występują jeszcze instrukcje while i for. Nie występuje natomiast instrukcja repeat, jak w Pascalu, co jednak nie stanowi istotnego ograniczenia.

Podprogramem może być procedura lub funkcja. Ogólna struktura programu jest następująca:

```
unit <identyfikator podprogramu>;
<typ podprogramu> (tryb przekazywania parametrów
  <lista parametrów formalnych>;<typ parametrów>);
<zbiór deklaracji lokalnych podprogramu>;
begin <lista instrukcji>
end <identyfikator podprogramu>;
```

Ponieważ podprogram może być procedurą lub funkcją to w deklaracji podprogramu w zależności od sytuacji należy umieścić słowo kluczowe procedure lub function. Jeżeli podprogramem jest funkcja to po typie parametrów w deklaracji procedury należy jeszcze podać typ wyniku. Mogą występować następujące tryby przekazywania parametrów określone słowami kluczowymi: inout, input, output.

Tryb inout oznacza, że w chwili wywołania procedury wartości jej parametrów aktualnych będą przypisane parametrom formalnym, a po wykonaniu procedury nastąpi ich przypisanie w odwrotnym kierunku - wartości parametrów formalnych zostaną przypisane parametrom aktualnym. Jeżeli parametr jest przekazywany w trybie input (taki tryb jest przyjmowany domyślnie, jeżeli nie ma żadnej specyfikacji), to w momencie wywołania podprogramu następuje przypisanie parametrowi formalnemu parametru aktualnego. Jeżeli parametr jest przekazywany w trybie output to w momencie powrotu z podprogramu wartośc parametru formalnego jest przypisywana parametrowi aktualnemu. Ze sposobu przekazywania parametrów widać, że w trybach inout i output odpowiadający im parametr aktualny musi być zmienną, a nie ogólnym wyrażeniem (ze względu na końcowe przypisanie wartości).

Jako przykład podprogramu podamy rekurencyjną wersję definicji funkcji silnia.

```
unit silnia: function (n:integer):integer;
begin
if n<0 then return fi;
if n=0 OR n=1 then result:=1
   else
   result:= silnia (n-1)*n
   fi
end silnia;
```

Blok jest strukturą zaczerpniętą z Algolu. Może służyć do specyfikacji podobliczenia. Jest jedynym modułem, którego deklaracja pokrywa się z wykonaniem. Z tego powodu występuje w innych modułach w ciągu instrukcji. Blok składa się z lokalnych deklaracji i listy instrukcji do wykonania. Wielkości zadeklarowane w bloku nie są widoczne na zewnątrz. Program główny jest jedynym blokiem, który ma nazwę. Program główny może mieć postać zwykłego bloku. Wówczas jest on dostępny pod nazwą main.

Ogólna struktura bloku jest następująca:

```
block
<zbiór deklaracji lokalnych>;
begin
<lista instrukcji>
end;
```

Deklaracja klasy ma postać następującą:

```
unit  <nazwa klasy>: class (<lista parametrów formalnych> :
      <specyfiacja parametrów>);
<zbiór deklaracji lokalnych klasy >;
<lista instrukcji>
end <nazwa klasy>;
```

Jeżeli w definicji klasy lista instrukcji jest pusta, to klasa definiuje strukturę danych. W Loglanie jest możliwe zadeklarowanie procedury lub funkcji wewnątrz deklaracji klasy. Zgodnie z zasadami widoczności przyjętymi w językach o strukturze blokowej, z wnętrza takiej procedury czy też funkcji jest możliwy dostęp do wszystkich danych lokalnych klasy. Procedury i funkcje zadeklarowane w

klasie są tak samo atrybutami obiektu klasy jak zmienne. Tak samo można odwoływać się do nich z zewnątrz obiektu podając zmienną wskazującą obiekt oraz identyfikator procedury lub funkcji.

Do tworzenia obiektów klas służy operator new. Utworzenie obiektu klasy następuje w wyniku wykonania wyrażenia

```
<zmienna klasowa>:= new <nazwa klasy (lista parametrów aktualnych)>;
```

Natomiast zmienna klasowa wskazuje utworzony obiekt o atrybutach określonych przez listę parametrów aktualnych.

Zmienne lokalne zadeklarowane w klasie oraz jej parametry są atrybutami obiektu klasy. Do wartości tych atrybutów można się dostać z zewnątrz tego obiektu, podobnie jak do pól rekordu w Pascalu. Należy tylko podać zmienną wskazującą obiekt oraz identyfikator atrybutu. Wykonanie wyrażenia

```
<zmienna klasowa>. <identyfikator artrybutu> : = <podstawianie wartości>
```

spowoduje nadanie nowych wartości atrybutom obiektu (zmienne lokalne lub parametry). W Loglanie istnieje możliwość zabezpieczenia atrybutu przed niepożądaną ingerencją z zewnątrz obiektu klasy. W tym celu należy na początku deklaracji klasy (przed zbiorem deklaracji lokalnych klasy) umieścić wyrażenie

```
close <lista atrybutów chronionych>;
```

Lista atrybutów umieszczona po close jest dostępna tylko dla instrukcji tego obiektu. Daje to możliwość ukrycia szczegółów implementacyjnych typu danych.

W Loglanie jedynymi typami złożonymi poza typami tablicowymi są typy klasowe. Typy klasowe można składać za pomocą operacji prefiksowania. Prefiksowanie umożliwia definiowanie dowolnie rozbudowanych klas. Klasa, która jest prefiksowana sama może być również prefiksem innej klasy. W ten sposób można utworzyć hierarchię klas. Mechanizm prefiksowania daje możliwość definiowania pojęć złożonych na bazie pojęć prostszych uprzednio zdefiniowanych. Jeżeli klasa A prefiksuje klasę B to syntaktycznie ten fakt zapisujemy w sposób następujący:

```
unit B: A class (.....);
```

Gdy blok jest prefiksowany klasą, to istnieje możliwość ograniczenia dostępu do atrybutów klasy prefiksującej z bloku prefiksowanego tą klasą. Ograniczenie dostępu może wystąpić zarówno w klasie prefiksującej, jak też w bloku prefiksowanym. W klasie przyjmuje formę specyfikacji hidden dla atrybutów, a w bloku specyfikacji taken. W bloku dostępne są tylko te atrybuty klasy, które zostały wymienione na liście specyfikacji taken w bloku oraz nie umieszczone na liście specyfikacji hidden w klasie.

Ogólnie rzecz biorąc prefiksowanie jest dwuargumentową operacją na modułach programu. Loglan w stosunku do innych języków obiektowych na ogół mało narzuca ograniczeń na prefiksowanie. W Loglanie prefiks może być klasą, procesem lub współprogramem. Moduł prefiksowany może być dowolnego rodzaju: procedurą, klasą, współprogramem, funkcją, procesem lub blokiem.

Smalltalk

Pierwsze prace nad językiem programowania Smalltalk rozpoczął Alan Kay w firmie Xerox w 1971 r. W następnych latach zostały opracowane kolejne wersje języka, mianowicie Smalltalk -72, -74, -76, -78 i Smalltalk -80. Zostały również opracowane wersje języka na komputery personalne. Dużą popularność zdobył język Smalltalk V opracowany na komputery personalne. Język Smalltalk jest reklamowany jako język sztucznej inteligencji na równi z takimi językami, jak Lisp i Prolog. Ostatnio pojawiły się systemy doradcze (expert systems) implementowane w Smalltalku.

System doradczy THE ANALIST przeznaczony jest do automatyzacji prac biurowych, a system HUMBLE jest jądrem systemu doradczego umożliwiającym jednoczesne istnienie wielu baz wiedzy.

Ogólna charakterystyka Smalltalku

Język Smalltalk jest językiem obiektowym, którego główne koncepcje, jak obiekt i klasa zostały zaczerpnięte z Simuli. W Smalltalku silnie zaakcentowana jest hierarchia klas. Ogólną charakterystykę Smalltalku podamy opierając się na implementacji języka Smalltalk 80.

Język Smalltalk 80 jest wyposażony w bogaty zestaw środków programowych jak:

- edytory tekstowe i graficzne
- narzędzia symulacyjne
- narzędzia ułatwiające i przyspieszające działania systemu.

W języku można wyróżnić dwie warstwy, tj. język programowania i język komunikowania.Jednakże nie można przedstawić ostrego podziału: warstwy te są wzajemnie powiązane. Obiekty są podstawowymi składnikami systemu Smalltalk 80. Obiekty mogą reprezentować np:

- liczby
- ciągi znaków
- kolejki
- słowniki
- figury geometryczne
- katalogi zbiorów
- edytory tekstowe
- programy
- kompilatory
- procesy obliczeniowe
- operacje finansowe
- zobrazowanie graficzne informacji

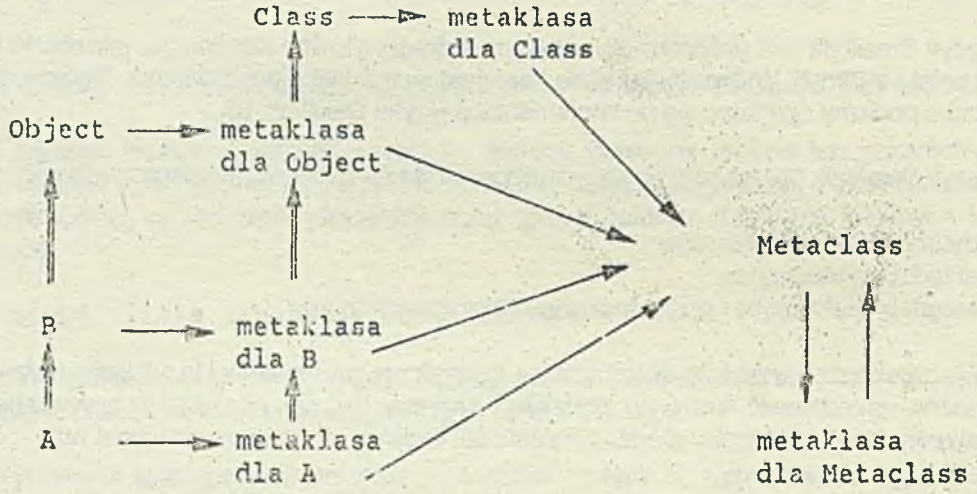
Działanie systemu Smalltalk 80 polega na komunikowaniu się zbioru obiektów między sobą.

Obiekt składa się z prywatnej pamięci i zbioru operacji. Każdy obiekt jest obiektem klasy (instance of a class). W przeciwieństwie do Simuli 67 w Smalltalku 80 obiektem klasy może być klasa. Klasą, której reprezentantem (obiektem) klasy jest klasa nazywamy metaklasą. Jeżeli tworzymy nową klasę, to jest automatycznie tworzona (generowana) w systemie dla niej metaklasa. Klasa opisuje implementację zbioru obiektów mających wspólne cechy. Indywidualny obiekt wygenerowany z klasy nazywa się obiektem (reprezentantem) tej klasy. Klasa opisuje pamięci prywatne biektów oraz operacje wykonywane przez obiekty. Metaklasy są podobne do klas, gdyż zawierają opis algorytmów używanych przez obiekty do wykonywania określonych operacji. Jednakże metaklasy różnią się tym od klasy, że nie mogą być obiektami metaklasy (wygenerowane z metaklasy). Są one obiektami pewnej klasy zwanej Metaklas. Metaklasy nie mają odrębnej nazwy. Dostęp do metaklasy realizuje się przez przesłanie komunikatu do obiektu tej metaklasy o postaci "class". Natomiast klasa Class jest abstrakcyjną superklasą (abstract superclass) dla wszystkich metaklas. Class opisuje ogólną strukturę klas. Abstrakcyjną nadklasę tworzymy ze względów formalnych, gdy kilka klas ma wspólne cechy, ale żadna z nich nie jest podklasą względem siebie. Abstrakcyjna nadklasa nie generuje obiektów.

W Smalltalku jest silnie zaakcentowana hierarchia klas w sensie dziedziczenia własności. Każda klasa ma na ogół więcej niż jedną nadklasę. Każdy obiekt jest obiektem klasy. Hierarchia klas spełnia następujące warunki:

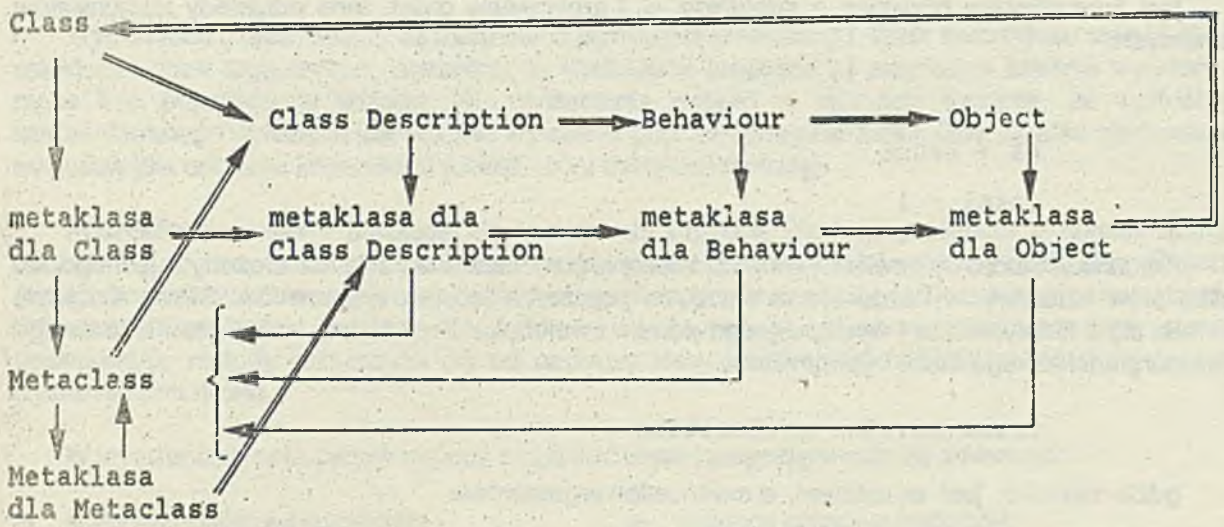
- każda klasa jest podklasą klasy Object (z wyjątkiem klasy Object, która nie ma nadklasy)
- każdy obiekt jest obiektem klasy
- każda klasa jest obiektem metaklasy
- każda metaklasa jest podklasą klasy Class
- każda metaklasa jest obiektem klasy Metaclass.

Niech A będzie podklasą klasy B. Hierarchię klas i metaklas w systemie Smalltalk 80 można przedstawić w sposób, jak na rysunku 1.



Rys 1. Hierarchia klas i metaklas w systemie Smalltalk 80, gdzie
"⇾" oznacza relację bycia obiektem ("A⇾metaklasa dla A" oznacza, że A jest obiektem metaklasy dla A)
"⇨"oznacza relację dziedziczenia własności ("A⇨B" oznacza, że klasa A jest podklasą klasy B).

Na powyższym rysunku podano tylko fragment hierarchii klas. Jeżeli nowa klasa jest wprowadzana do systemu, to jest ona definiowana jako podklasa klasy w istniejącej w systemie Smalltalk 80 hierarchii klas, na szczycie której jest klasa Object. Dla wprowadzonej klasy jest automatycznie generowana przez system odpowiadająca jej metaklasa. Do definiowania nowych klas są wymagane w systemie dwie klasy Behavior i Class Description. Klasa Behavior definiuje pewne stany wymagane w procesie interpretacji i kompilacji oraz do generowania obiektów. Klasa Class Description jest podklasą klasy Behavior i zawiera dodatkowe informacje potrzebne do pełnej reprezentacji klasy takie jak, reprezentacja zmiennych obiektowych, nazwa klasy i komentarze. Klasa Class Description zawiera dwie podklasy: Class i Metaclass. Klasa Class opisuje reprezentacje zmiennych klasowych oraz operacje na tych zmiennych. Klasa Metaclass steruje procesem generowania metaklasy dla nowo utworzonej klasy. Wyżej opisane klasy "współdziałając między sobą" ułatwiają opis i generację nowo definiowanych klas. Relacje między tymi klasami można przedstawić w sposób jak na rys.2.



Rys.2 Relacje między klasami w Smalltalku.
Relacje "→" i "⟹" mają znaczenie jak poprzednio.

Podstawowe elementy

Podstawowym elementem języka Smalltalk jest klasa, której reprezentantami (obiektami klasy) są obiekty. Obiekt można traktować jako zbiór danych i operacji (programów). Dane obiektu są dostępne poza nimi jedynie pośrednio jako jego operacje. Obiekt można traktować więc jako strukturę danych. Operacje (programy) w obiekcie są oznaczane selektorami i są nazywane metodami (methods).Komunikacja między obiektami odbywa się w ten sposób, że do wskazanego obiektu wysyła się polecenie wykonania operacji (programów) zawartych w obiekcie, określonych komunikatem. Wykonanie komunikatu polega na odnalezieniu operacji (oznaczonej takim samym selektorem jak w komunikacie) i jej wykonaniu. Wyliczona wartość operacji jest przesyłana do nadawcy (obiektu, który wygenerował komunikat).

Komunikaty (messages)

Komunikaty są wyrażeniami języka Smalltalk 80.Komunikat określa odbiorcę, podaje selektor i może zawierać argumenty. Wysłanie komunikatu (watrościowanie wyrażenia) jest interpretowane jako polecenie wykonania metody określonej przez selektor w obiekcie będącym odbiorcą komunikatu.

W opisie komunikatu na pierwszym miejscu występuje wyrażenie oznaczające odbiorcę komunikatu (nazwa obiektu) a następnie selektor i ewentualnie argumenty.

Komunikat nie zawierający argumentów jest nazywany komunikatem unarnym a selektor, selektorem unarnym. Wyrażenie theta sin jest komunikatem unarnym. Odbiorcą jest liczba określona przez zmienną theta i selektor sin.

Komunikat binarny zawiera jeden argument i selektor binarny, który jest jednoznakowym lub dwuznakowym symbolem jak np. `+, -, <=, ==.`

Wyrażenie `origin + offset`

jest komunikatem binarnym o selektorze + i argumencie offset. Inne przykłady komunikatów binarnych:

```
3 + 4
45 + count
total - 1
```

Komunikat kluczowy zawiera jeden lub więcej argumentów oraz selektor złożony z jednego lub kilku słów kluczowych. Każde słowo kluczowe poprzedza jeden z argumentów. Słowo kluczowe składa się z identyfikatora i występującego po nim dwukropka. Przykładem komunikatu kluczowego jednoargumentowego może być wyrażenie

```
frame moveTo: newlocation
```

gdzie moveTo: jest selektorem, a newlocation argumentem.

Przykładem komunikatu dwuargumentowego może być wyrażenie:

```
list at: index put: element
```

w którym selektor jest utworzony ze słów kluczowych at: i put: a argumentami są index i element.

Po wysłaniu komunikatu do obiektu zawsze uzyskujemy odpowiedź, która jest wyniklem wartościowania metody. Odpowiedź jest również obiektem. Komunikat może być odbiorcą odpowiedzi na komunikat. Odpowiedź na komunikat może być również argumentem dla innego komunikatu. Przykładem komunikatu unarnego opisującego odbiorcę innego komunikatu unarnego może być wyrażenie

```
window frame center
```

Zgodnie z zasadą analizowania komunikatów z lewa na prawo odbiorcą komunikatu o selektorze unarnym frame będzie obiekt window, a odbiorcą komunikatu o selektorze unarnym center będzie obiekt uzyskany w wyniku wartościowania wyrażenia window frame.

Jeżeli w wyrażeniu występują różnego typu selektory, to najpierw są analizowane komunikaty zawierające selektory unarne, następnie binarne, a na końcu zawierające selektory kluczowe. Nawiasy również zmieniają kolejność analizowania (wartościowania) wyrażeń.

A więc na przykład wyrażenie 2 + 3 x 4, zgodnie z przyjętą zasadą wartościowania ma wartość 20 a nie 14. Natomiast wyrażenie 2 + (3 x 4) ma wartość 14 zgodnie z zasadą wartościowania w pierwszej kolejności wyrażeń w nawiasach.

Metody (methods)

Klasa opisuje zbiór obiektów zwanych jej reprezentantami (obiektami klasy - instances). W ramach opisu klasa zawiera zbiór metod określających reakcje reprezentantów na komunikaty. Metoda opisuje ciąg akcji podejmowanych w przypadku uaktywnienia jej przez komunikat. Akcje te polegają na wywołaniu komunikatów, przypisywania wartości zmiennym, wyliczaniu wartości dla początkowego komunikatu. W opisie metody można wyróżnić następujące części:

- wzór komunikatu (message pattern)
- nazwy zmiennych czasowych - roboczych (temporary)
- wyrażenia

Wymienione części metody są rozdzielane pionowymi kreskami (|). Wzór komunikatu składa się z selektora i nazw argumentów. Wyrażenia są rozdzielane kropkami (.) przy czym ostatnie wyrażenie może być poprzedzone strzałką (↑). Wystąpienie strzałki w metodzie oznacza, że wartością wartościowanego wyrażenia jest wartość wyrażenia poprzedzonego strzałką. Gdy strzałka nie wystąpi wartością jest odbiorca komunikatu (obiekt, który uaktywnił metodę).

Poszukiwanie metody pasującej do komunikatu (mającej identyczny selektor z komunikatem) zaczyna się od klasy odbiorcy komunikatu i przebiega przez wszystkie jej nadklasy aż do klasy Object (będącej na szczycie hierarchii klas). Jeżeli metoda nie zostanie znaleziona to jest sygnalizowany błąd. Gdy komunikat będzie wysłany do odbiorcy wyznaczonego przez nazwę zastrzeżoną super poszukiwanie metody rozpoczyna się od nadklasy klasy zawierającej metodę, w której występuje wysłanie komunikatu.

W wyrażeniach opisujących metody mogą być użyte następujące rodzaje zmiennych:

- zmienne obiektowe odbiornika
- pseudo-zmienna self
- argumenty komunikatu
- zmienne czasowe (robocze)
- zmienne kluczowe
- zmienne globalne

Zmienne obiektowe występują w klasie opisującej odbiorcę komunikatu. Pseudozmienna self odnosi się do odbiorcy komunikatu. Dostęp do tej zmiennej jest realizowany podobnie jak do innych zmiennych. Zmienna ta tym się różni od innych, że nie można jej nadawać nowej wartości za pomocą instrukcji przypisania.

Zmienne robocze i argumenty mają wspólne cechy. Są one deklarowane w metodzie i istnieją tylko podczas wykonywania metody. Argumenty są automatycznie inicjowane, nie dotyczy to zmiennych roboczych. Można zmieniać wartość zmiennych roboczych za pomocą instrukcji przypisania.

Zmienne klasowe są dostępne dla wszystkich reprezentantów klasy i dla samej klasy. Mimo że wartości tych zmiennych mogą być zmieniane, to w typowych zastosowaniach są traktowane jak stałe. Są inicjowane w momencie tworzenia klas.

Zmienne globalne są dostępne dla wszystkich obiektów systemu Smalltalk 80. Słownik globalny o nazwie Smalltalk zawiera wszystkie nazwy zmiennych globalnych i ich wartości.

Część metod z różnych klas jest zapisana w języku maszyny docelowej. Są one dostępne bezpośrednio procesorami języka Smalltalk. Metody te nazywane są metodami pierwotnymi. Metody pierwotne zawierają najczęściej używane metody systemowe (np. operacje arytmetyczne, elementarne operacje na obiektach i klasach) od których może w sposób istotny zależeć efektywność całego systemu. Metody pierwotne są wywoływane przez numery oznaczające je, dokładniej mówiąc są oznaczane przez

```
wyrażenie <primitive: numer metody pierwotnej>
```

Klasy

Podstawową konstrukcją w języku Smalltalk jest klasa. Cechą charakterystyczną Smalltalku jest występująca w formie jawnej hierarchia klas, na szczycie której jest klasa Object. Można powiedzieć, że programowanie w języku Smalltalk sprowadza się do tworzenia lub modyfikowania istniejących klas opisujących obiekty interesujące programistę. Opis klasy tworzymy według wzorca podanego na rys. 3.

nazwa klasy	identyfikator
nazwy zmiennych obiektowych	identyfikatory
metody	metoda : metoda

Rys.3 Wzorzec opisu klasy.
Nazwa klasy rozpoczyna się od dużej litery; nazwy zmiennych rozpoczynają się od małych liter.

Nazwa klasy rozpoczyna się od dużej litery: nazwy zmiennych rozpoczynają się od małych liter.Jako przykład podamy klasę liczba zespolona uprzednio zdefiniowaną w Simuli 67. Opis klasy podamy korzystając z podanego wzorca:

```
nazwa klasy                          liczba zespolona
nazwy zmiennych obiektowych           r  alfa
metody
rdla: x i : y |x y| ↑ r←sqrt  (x..2 + y..2)
alfa dla: x i : y | x y| ↑ alfa←arctan (y,x).
```

gdzie "←" oznacza operację przypisania wartości zmiennej.

Wysłanie komunikatów

Liczba zespolona rdla: 3 i :4

Liczba zespolona alfa dla: 3 i :4

spowoduje nadanie wartości 5 zmiennej r i wartości 0.927 zmiennej alfa.

Należy podkreślić, że opis klas, metody i komunikaty mogą być opisywane w języku zbliżonym do naturalnego, co było między innymi intencją twórców języka Smalltalk.

Pełny opis klasy wymaga określenia położenia klasy w hierarchii klas, dokładniej mówiąc określenia nadklasy dla klasy definiowanej. Pełny opis klasy należy utworzyć według wzorca podanego na rys.4.

nazwa klasy	identyfikator
nadklasa	identyfikator
nazwy zmiennych obiektowych	identyfikatory
nazwy zmiennych klasowych	identyfikatory
metody klasowe	metody . . . metody
metody obiektowe	metody . . . metody

Rys.4 Pełny wzorzec opisu klasy.

Struktury sterowania i bloki

Struktury sterowania zapewniają odpowiednią kolejność podejmowania akcji. Podstawowe struktury sterowania w Smalltalku 80 zapewniają sekwencyjne wykonywanie (wartościowanie) wyrażeń (np. wartościowanie wyrażeń w metodach). Pozostałe struktury sterowania opierają się na obiektach zwanych blokami. Wyrażenie blokowe składa się z ciągu wyrażeń oddzielonych kropkami lub z jednego wyrażenia. Wyrażenia blokowe są oznaczane nawiasami kwadratowymi np.

```
[index←index + 1]
[index←index + 1.array at: index put: 0]
```

Wyrażenie blokowe nie jest wartościowane w miejscu jego tekstowego wystąpienia, a dopiero wtedy, gdy do bloku zostanie wysłany specjalny komunikat. Komunikat uaktywniający blok jest unarnym selektorem value (gdy blok nie ma parametrów) albo selektorem kluczowym będącym tylokrotnym złożeniem klucza value: ile argumentów zawiera blok. Wartością bloku jest wartość ostatniego wyrażenia w bloku, o ile żadne wyrażenie w bloku nie jest poprzedzone strzałką. Jeżeli występuje wyrażenie poprzedzone strzałką to wartościowanie tego wyrażenia powoduje zakończenie wykonywania bloku i metody, a jego wartość staje się wartością bloku i metody.

Bloki wykorzystuje się głównie do zapisu struktur sterowania. Najprostszymi strukturami sterowania są struktury pierwotne takie, jak wybór warunkowy (conditional selection) i iteracja warunkowa (conditional repetition).

Wybór warunkowy i iteracja warunkowa są realizowane za pomocą metod w klasie Boolean. Aktywizacja wyboru warunkowego następuje po wysłaniu komunikatów zawierających selektory: ifTrue: ifFalse: których argumentami są bloki. Wybór warunkowy jest strukturą podobną do struktury if...tnen...else występującej w językach algolopodobnych. A więc np. wyrażeniom zapisanym w języku algolopodobnym

```
if a<b then
a: = a + 1 else
a: = a - 1;
```

odpowiada w Smalltalku 80 następujące wyrażenie

```
a<b
ifTrue:[a←a + 1].
ifFalse:[a←a - 1].
```

Aktywizacja iteracji warunkowej następuje po wysłaniu komunikatów zawierających selektory: whileTrue:, whileFalse:, których argumentami są bloki. Wyrażenia bloku wskazane przez selektor są tak długo wartościowane, aż zostanie spełniony odpowiedni warunek. Iteracja warunkowa ma podobną strukturę do struktur while i until występujących w językach algolopodobnych. A więc np. wyrażeniu zapisanemu w języku algolopodobnym

```
while i<10 do begin
x : = x + a [i]
i : = i + 1
end;
```

odpowiada w Smalltalku 80 następujące wyrażenie

```
[i<10]
whileTrue: [x←x + [a at:i].
           i←i + 1].
```

Podstawowe klasy Smalltalku 80

Język Smalltalk 80 zapewnia jednolitą składnię ułatwiającą operacje na obiektach, wysyłanie komunikatów i definiowanie klas.

Celem ułatwienia zachowania jednolitej składni system Smalltalk 80 zawiera opisy klas takich, jak Object, Class, Message, CompiledMethods, Context oraz podklasy BlockContex i MethodContext. Do koordynacji niezależnych procesów są przewidziane klasy ProcessorScheduler, Process i Semaphore. Wyróżniony obiekt nil jest jedynym reprezentantem klasy UndefinedObject. Powyższe klasy stanowią jądro systemu Smalltalk 80.

Klasy tworzą hierarchię, na szczycie której jest klasa nazwana Object. Każda klasa w systemie Smalltalk 80 dziedziczy własności swoich nadklas w hierarchii. Klasa Object opisuje zachowanie wszystkich obiektów systemu. Zawiera ona metody umożliwiające testowanie klasy obiektów, tworzenie kopii obiektów, drukowanie symbolicznych reprezentacji obiektów.

System Smalltalk 80 zawiera również klasy służące do opisu podstawowych struktur danych. Klasy te opierają się na liczbach i ciągach (zbiorach). Klasa Number służy do opisu wszystkich numerycznych obiektów. Podklasy Float, Fraction Integer klasy Number opisują specyficzne reprezentacje liczb. Klasa Integer zawiera trzy podklasy: SmallInteger, LargePositiveInteger, LargeNegativeInteger. Liczby więc są obiektami (reprezentantami) klas Float, Fraction, SmallInteger, LargePositiveInteger, LargeNegativeInteger.

Kasa Collection służy do opisu struktur danych zawierających ciągi i zbiory obiektów. Podklasami tej klasy między innymi są następujące klasy: Bag, Set, OrderedCollection, LinkedList, MappedCollection, SortedCollection i IndexedCollection, której podklasami są klasy String i Array. Klasy Bag i Set opisują ciągi elementów nieuporządkowanych. W klasie Bag dopuszcza się wielokrotne wystąpienie identycznych elementów, a w klasie Set jest niedopuszczalne wystąpienie identycznych elementów. Nazwy pozostałych klas pozwalają z grubsza wyrobić sobie opinię, do jakich obiektów one służą. Pełny wykaz klas systemu Smalltalk 80 jest podany na rys. 5.

```
Object

Magnitude
    Character
    Date
    Time

    Number
        Float
        Fraction
        Integer
            LargeNegativeInteger
            LargePositiveInteger
            SmallInteger

    LookupKey
        Association

Link

    Process

Collection

    SequenceableCollection
        LinkedList

            Semaphore

        ArrayedCollection
            Array

            Bitmap
                DisplayBitmap

            RunArray
            String
                Symbol
            Text
            ByteArray

        Interval
        OrderedCollection
            SortedCollection
    Bag
    MappedCollection
    Set
        Dictionary
            IdentityDictionary

Stream
    PositionableStream
        ReadStream
        WriteStream
            ReadWriteStream
                ExternalStream
                    FileStream

    Random

File
FileDirectory
FilePage
UndefinedObject
Boolean
    False
    True

ProcessorScheduler
Delay
SharedQueue

Behaviour
    ClassDescription
        Class
        Metaclass

Point
Rectangle
BitBit
    CharacterScanner

    Pen

DisplayObject
    DisplayMedium
        Form
            Cursor
            DisplayScreen
    InfiniteForm
    OpaqueForm
    Path
        Arc
            Circle
        Curve
        Line
        LinearFit
        Spline
```

Rys.5 Pełny wykaz klas w Smalltalku 80.
Wcięcia tekstu w przedstawionej hierarchii oznaczają relację bycia podklasą.

LITERATURA

[1] BYTE 1981 t.6 nr 8 (numer specjalny o systemie Smalltalk 80)

[2] Goldberg A.,Robson D.,: Smalltalk-80:the Language and its Implementation.Addison - Wesley, Reading, Mass. 1983

[3] Goldberg A. Smalltalk-80: The Interactive Programming Environment. Addison-Wesley, Reading, Mass. 1983

[4] Krępski A.: Język Smalltalk-80. Warszawa 1987 IPI PAN

[5] Oktaba H., Ratajczak W.: Simula 67. Warszawa 1980 WNT

[6] Salwicki A. Tworzenie modułów programistycznych w oparciu o abstrakcyjne typy danych. Jesienna Szkoła PTI, Mrągowo 1987

[7] Warpechowska J.:Podręcznik Loglanu. Skrypt dla studentów UW.Warszawa.

mgr Andrzej PAPROCKI
Instytut Maszyn Matematycznych

Oprogramowanie narzędziowe

Potrzeba masowej produkcji oprogramowania i stale rosnący jego koszt wymuszają szybki rozwój narzędzi wspomagających tworzenie i pielęgnację produktów programowych. Obecnie liczba produktów zaliczanych do klasy oprogramowania narzędziowego jest tak wielka, że nie sposób nie tylko ich wymienić, ale nawet trudno jest je poklasyfikować. Różnorodność ta często utrudnia programiście wybór właściwych narzędzi. Poniżej przedstawiony jest przegląd ogólnych kierunków rozwoju oprogramowania narzędziowego pod kątem możliwości zwiększenia efektywności wytwarzania produktów programowych.

Efektywność narzędzi programowych

Szacuje się, że światowe wydatki na opracowanie oprogramowania rosną w tempie 12% rocznie. W roku 1985 na wytworzenie oprogramowania wydano w USA 70 mld $, a na świecie 140 mld $. Przewiduje się, że w 1995 roku liczby te będą wynosiły odpowiednio: 225 mld $ i 450 mld $ [1].

Podstawową przyczyną wzrostu kosztów oprogramowania jest wzrost wielkości i stopnia skomplikowania produktów programowych. W większości firm wykorzystujących oprogramowanie krzywa wzrostu rozmiaru pojedynczych produktów (w instrukcjach maszynowych) w zależności od czasu, przypomina w kształcie krzywą wykładniczą. Dobrym przykładem tej tendencji może być rozwój programu kosmicznego realizowanego przez NASA. Pierwszy projekt MERCURY w latach 62-63 obejmował około 1.5 mln instrukcji

maszynowych, podczas gdy przewiduje się, że projekt SpaceStation, o planowanym terminie zakończenia w 91 roku, będzie liczył około 80 mln instrukcji.

Powyższe dane wskazują, że nawet niewielki procentowo wzrost wydajności w tworzeniu oprogramowania pozwoliłby na znaczne oszczędności. Z analizy kosztów opracowania produktu programowego wynika, że podstawowy wpływ na ten koszt ma jego rozmiar i jakość zespołu realizującego oraz sposób zarządzania. Bezpośrednio za tą grupą cech plasują się cechy związane z narzędziami programowymi wykorzystywanymi do tworzenia i zarządzania produktem. Należy zdawać sobie jednak sprawę, że wprowadzenie nowych narzędzi jest stosunkowo kosztowne. Przykładem może być wprowadzenie nowego języka programowania. Jeżeli dany projekt programowy będzie realizowany w nowym, nieznanym zespołowi języku programowania, to przy niezmienionych innych warunkach, czas opracowania produktu będzie o 20% dłuższy niż wówczas, gdy zespół ma doświadczenie w używaniu tego języka. Wynika z tego, że nie opłaca się wprowadzać nowego języka programowania, jeżeli jego poziom jest zbliżony do poziomu starego języka.

Oprogramowanie narzędziowe używane przez zespoły specjalizujące się w wytwarzaniu oprogramowania powinno spełniać następujące warunki:

- Narzędzia programowe powinny wspomagać wszystkie fazy cyklu życia projektu

Obecne produkty programowe realizowane są przez wielkie zespoły ludzkie. Zatrudnianie przy opracowywaniu pojedynczego produktu kilkuset osób jest bardzo częste. Zespół taki musi współpracować ze sobą, musi więc używać takiej samej technologii wytwarzania oprogramowania. Koszty zmiany narzędzi używanych przez zespół są duże i dlatego firmy produkujące oprogramowanie starają się o prowadzenie bardzo konsekwentnej polityki w zakresie doboru narzędzi i technologii tworzenia oprogramowania.

- Oprogramowanie narzędziowe powinno stanowić zintegrowane środowisko

Oprogramowanie narzędziowe jest dużo efektywniejsze, jeżeli stanowi jednolite środowisko, a nie zbiór pojedynczych narzędzi. Środowisko to musi być przystosowane do typu modelu cyklu życia produktu i metodologii projektowania. Najefektywniejsze narzędzia pracują na podstawie informacji gromadzonych w bazie danych projektu. Ostatnio używa się nawet baz wiedzy. Istnieje wiele takich środowisk. Są one najczęściej związane z językami programowania i wtedy mają zwykle tą wadę, że nie obejmują początkowych faz cyklu życia. Najbardziej znane i najpełniejsze jest środowisko języka Ada.

- Oprogramowanie narzędziowe powinno być na tyle maszynowoniezależne, aby przejście na nowe konfiguracje sprzętowe nie wymagało zmiany technologii wytwarzania

Przykładem takiego środowiska może być system UNIX z językiem C, oraz środowisko języka Ada. Program napisany w Adzie, przy przenoszeniu z jednej maszyny na inną, zazwyczaj nie wymaga żadnych zmian.

- Narzędzia programowe powinny być "silne"

Jedną z miar efektywności tworzenia produktów jest liczba linii tekstu źródłowego produkowanego miesięcznie przez członka zespołu realizującego produkt. Okazuje się, że wydajność mierzona w ten sposób jest zbliżona dla różnych klas języków programowania. Dużo efektywniejsze jest więc używanie, przy dużych projektach, języków wysokiego poziomu. Tłumaczy to popularność języków klasy Ady i języków czwartej generacji.

- Używana metodologia musi zapewniać wielokrotne użycie poszczególnych elementów

Oprogramowaniu narzędziowe powinno zapewniać możliwości włączania do nowo opracowywanego produktu elementów z innych produktów programowych. Przykładem może być stosowanie wyspecjalizowanych bibliotek, generatory środków komunikacji z użytkownikiem itp.

Oprogramowania narzędziowe dla poszczególnych faz cyklu życia produktu

W latach sześćdziesiątych i pierwszej połowie lat siedemdziesiątych oprogramowanie narzędziowe koncentrowało się przede wszystkim na etapie kodowania produktu. Był to okres gwałtownego rozwoju języków programowania. Zaniedbywany był rozwój narzędzi wspomagających inne fazy cyklu życia produktu programowego. Obecnie, przede wszystkim w wyniku dużego rozwoju inżynierii programowania opracowanych zostało wiele różnorodnych narzędzi wspomagających praktycznie wszystkie fazy cyklu życia produktu programowego. Niestety nie istnieje jeden, ogólnie przyjęty model cyklu życia produktu. W celu zilustrowania narzędzi programistycznych przyjęto w tej pracy model kaskadowy. Cykl życia projektu programowego [2] składa się w tym modelu z następujących faz:

- specyfikacja założeń
- projekt strukturalny
- projekt szczegółowy
- kodowanie
- testowanie i weryfikacja
- integracja
- pielęgnowanie

Narzędzia wspomagające specyfikację produktu

Obecnie specyfikacja projektu jest na ogół najmniej sformalizowana i narzędzi do specyfikowania jest stosunkowo niewiele. Mówi się nawet o kryzysie specyfikacji, ukazując ją jako najsłabsze ogniwo w cyklu życia projektu.

Istnieje wiele sposobów opisu specyfikacji założeń. Często stosuje się opis słowny (w języku naturalnym), jednak tak zbudowana specyfikacja jest trudna do sprawdzenia poprawności i kompletności. Jedną z dość często spotykanych metod jest przedstawienie produktu w postaci skończonego automatu deterministycznego lub opisu za pomocą sieci Petri. Zaletą tych metod specyfikacji jest to, że nadają się one również do specyfikacji systemów rozproszonych [3]. Dla metod tych powstały specjalne języki np. SXL pozwalające nie tylko na opis specyfikacji, ale również na jej wykonanie. W tradycyjnym podejściu do tworzenia produktów programowych pierwszą wykonywalną postacią projektu jest dopiero kod źródłowy. Język SXL pozwala na uzyskanie wykonywalnej postaci projektu już w fazie specyfikacji, co niezwykle ułatwia wczesne wykrywanie błędów polegających na niekompletności lub sprzeczności założeń.

Inne podejście do specyfikacji proponują technologie oparte na szybkim tworzeniu prototypów. U podstaw tych technologii leży założenie, że użytkownik zamawiający produkt programowy najczęściej nie może wyspecyfikować założeń. Jedyną skuteczną metodą uzyskania specyfikaji produktu od takiego użytkownika jest zrobienie prototypu i analiza jego zachowań. Co gorzej, zazwyczaj nie wystarczy zrobienie jednego prototypu. Dla metody szybkiego tworzenia prototypów powstały liczne narzędzia. Są to narzędzia o charakterze symulacyjnym oraz generatory środków komunikacji z użytkownikiem.

Narzędzia wspomagające projektowanie

Oprogramowanie wspomagające projektowanie produktów programowych stanowi bardzo dużą część oprogramowania narzędziowego. Narzędzia wspomagające projektowanie są bardzo różnorodne zarówno pod względem skali rozwiązywanego problemu, jak i obsługiwanych metod projektowania. W chwili obecnej nie ma jednej, uznanej powszechnie metody projektowania systemów informatycznych. Do najbardziej znanych należą:

- projektowanie obiektowe,
- projekowanie strukturalne,
- projektowanie ukierunkowane na Adę

Bradzo szybko rozwija się metoda projektowania obiektowego. Powstały dla niej specjalne narzedzia w postaci systemów wspomagających projektowanie np ODESSY [4], jak również pełne środowiska np. rozwijane orzez firmę DEC środowisko języka Trellis [5].

Dla projektu strukturalnego używa się różnych narzędzi wspomagających graficzne zobrazowanie takiego projektu. Są to wszelkiego rodzaju redaktory graficzne tworzone dla poszczególnych metod projektowania i dla przyjętego, graficznego schematu przedstawiania powiązań.

Dla opisu projektu szczegółowego popularne staje się używanie języków programowania wysokiego poziomu jako języków projektowania. Próbuje się zastosować język programowania do danej metodologii np. Adę lub Modulę-2 do projektowania obiektowego, lub stworzyć metodologię opartą na języku np. metodologia projektowania w Adzie [6]. Oparcie projektu na sformalizowanym języku programowania ma tę zaletę, że pozwala na skompilowanie takiego projektu i sprawdzenie zarówno kompletności, jak i poprawności używania poszczególnych fragmentów. Jako podstawowy język dla programowania obiektowego używany jest obecnie Smalltalk-80.

Przejście od projektu strukturalnego do szczegółowego ułatwiają programy półautometycznego generowania kodu projektu szczegółowego.

Narzędzia wspomagające kodowanie projektu

Narzędzia wspomagające kodowania projektu stanowią najliczniejszą grupę w oprogramowaniu narzędziowym, co wiąże się z tym, że były one najwcześniej rozwijane. Do podstawowych narzędzi wspomagających kodowanie należą kompilatory i interpretery języków programowania. Liczba różnych języków i ich dialektów wynosi kilka tysięcy, ale

tylko nieliczne zdobyły popularność. Przy wyborze języka implementacji projektu należy unikać języków mało znanych przede wszystkim z tego powodu, że zazwyczaj mają one mało programów wspomagających i mała jest możliwość powtórnego korzystania z istniejącego już kodu (np. istniejące biblioteki wyspecjalizowane). Z drugiej strony popularność niektórych języków tradycyjnych tj. COBOLu czy FORTRANu nie odpowiada ich zaletom z punktu widzenia współczesnej inżynierii oprogramowania.

Wybór języka programowania rzadko uwarunkowany jest jedynie przydatnością tego języka dla projektu. Najczęściej dochodzą inne względy, takie jak dostępność danego języka na posiadanym sprzęcie, doświadczenia zespołu realizującego projekt itp. Warto pamiętać, że jeżeli nie ma innych istotnych powodów, to nie należy używać języków niskiego poziomu. Wydajność mierzona w liniach uruchomionego kodu na jednostkę czasu jest zbliżona dla języków wysokiego i niskiego poziomu. Z punktu widzenia efektywności celowe jest więc używanie języków "silnych".

Przy kodowaniu należy pamiętać, że przy dużych projektach, tekst programu jest nie tylko środkiem komunikacji człowiek-maszyna lecz również (a nawet przede wszystkim) środkiem komunikacji pomiędzy poszczególnymi członkami zespołu. Czytelny sposób kodowania jest z tego względu bardzo ważny. Należy unikać sztuczek dostępnych w niektórych językach programowania, jeżeli zmniejszają one przejrzystość kodu. Dla celów czytelnego redagowania programów powstało wiele formaterów i programów redagujących, które transformują kod na określony standard zapisu (pretty-writer).

Sam proces kodowania może być przyspieszony i ułatwiony przez zastosowanie specjalnych, przystosowanych do danego języka programowania programów redagujących. Mogą być to redaktory sterowane składnią lub redaktory ukierunkowane językowo. Redaktory takie pozwalają na pisanie kodu poprawnego pod względem składniowym, a niektóre z nich pozwalają także na sprawdzenie semantyki statycznej. Używanie takich redaktorów wydatnie skraca czas kompilacji

programu. Ze względu na dużą liczbę używanych języków programowania, dość duże koszty wytworzenia takich redaktorów oraz możliwość formalnego sprecyzowania dla nich założeń dla szerokich klas języków powstały generatory redaktorów [7]. Jako przykład zastosowania tej metody może służyć redaktor VAX/LSE (Language Sensitive Editor) [8] firmy DEC, obsługujący wszystkie podstawowe języki programowania, których kompilatory są dostarczane dla maszyn typu VAX.

Podstawową cechą współczesnych produktów programowych jest ich duży rozmiar. Zazwyczaj są one kodowane częściami, konieczne są więc środki zapewniające rozłączną kompilację poszczególnych części produktu i możliwość sprawdzenia poprawności użycia modułów już skompilowanych. Trzeba też przechowywć historię kompilacji poszczególnych modułów. W większości środowisk programistycznych istnieją narzędzia programowe do tworzenia i obsługi bazy danych projektu. Przykładem może być tworzenie bazy danych dla produktów pisanych w Adzie, gdzie konieczność jej istnienia wynika już ze standardu języka. Narzędzia wspomagające utrzymanie porządku kompilacji poszczególnych modułów są obecnie popularne i dostępne nawet na małych maszynach np. TURBO PASCAL 4.0 jest wyposażony w takie możliwości.

Narzędzia wspomagające uruchamianie i testowanie

Narzędzia wspomagające uruchamianie produktów programowych, to przede wszystkim różnego rodzaju programy uruchomieniowe. Programy te początkowo były dość proste i ich koncepcja polegała na określeniu miejsc przerwania wykonywania programu. W miejscu przerwania można było oglądać i zmieniać wartości poszczególnych zmiennych. Bardzo proste programy uruchomieniowe operują adresami maszynowymi lub symbolicznymi na poziomie asemblara. Rozwój języków wysokiego poziomu doprowadził do rozwoju symbolicznych programów uruchomieniowych, których działanie jest związane z danym językiem programowania. W programach tych istnieje możliwość odwoływania się do linii tekstu programu źródłowego i odwołania do zmiennych tego programu przez

podawanie ich nazw. Takimi programami uruchomieniowymi dysponuje większość poważnych firm dostarczających oprogramowanie. Często, podobnie jak w wypadku ukierunkowanych językowo programów redagujących, firmy te dostarczają jeden program uruchomieniowy, który może współpracować z kompilatorami wielu języków wysokiego poziomu np. VAX/VMS Symbolic Debugger. Podejście do uruchamiania programów na zasadzie przerywania programu i stwierdzania, że napotkany został błąd, nie zawsze daje dobre wyniki, gdyż często błąd powstał w innym miejscu niż się on objawia. W ostatnim czasie zaczęły pojawiać się programy uruchomieniowe wykorzystujące wiedzę o programie. Opierają się one na ostatnich osiągnięciach z dziedziny sztucznej inteligencji i systemów rzeczoznawczych i pozwalają na precyzyjną lokalizację błędów, wycofując się od miejsca, w którym błąd został ujawniony, aż do instrukcji, która ten błąd spowodowała. Przykładem takiego systemu uruchomieniowego jest system PALAS [9].

Następną grupą narzędzi wspomagających uruchamianie i testowanie programów są programy zarządzające testami. Powstało wiele metod testowania i sprawdzania kompletnosci testów tzn. sprawdzania, czy testy pokrywają wszystkie ścieżki w programie. W dużych systemach istnieje potrzeba sprawdzania, czy nowe poprawki nie zmieniły sposobu zachowań przetestowanych już modułów. Jedna z metod sprawdzania polega na automatycznym wykonywaniu testów dla modułów już sprawdzonych i porównywaniu ich wyników z zapamiętanymi wynikami wzorcowymi. Przykładem takiego sposobu zarządzania testami jest VAX/VMS Test Manager firmy DEC.

Osobną grupę stanowią programy wspomagające uruchamianie systemów wbudowanych. Często oprogramowanie przygotowywane jest na maszynie nazywanej technologiczną i uruchamiane na innej maszynie nazywanej docelową. Powstały liczne narzędzia ułatwiające uruchamianie takich programów na maszynie technologicznej (symulatory). Są też narzędzia wspomagające sam proces przeniesienia uruchomionego, lub częściowo przygotowanego produktu na maszynę docelową.

Narzędzia wspomagające integrację

Narzędzia wspomagające integrację programu to przede wszystkim programy zarządzające wersjami, programy pozwalające na tworzenie produktu w zależności od wymaganej konfiguracji. Programy te umożliwiają zarządzanie różnymi wersjami produktu już na poziomie kodu źródłowego np. DEC Code Management System. Do grupy narzędzi wspomagających integrację należą także programy obsługi bibliotek i rozbudowane programy łączące, pracujące nie tylko na podstawie analizy odwołań, lecz także bazy danych produktu programowego. Przykładem takich narzędzi są programy łączące dla projektów pisanych w Adzie.

Narzędzia wspomagające pielęgnację oprogramowania

Faza następująca po przygotowaniu produktu jest najczęściej najdłuższa i najtrudniejsza do zarządzania. Najczęściej produkt jest pielęgnowany przez inny zespół niż ten, który przygotowywał dany produkt. Poprawki w istniejącym projekcie zazwyczaj wymagają modyfikacji większości poprzednich faz cyklu życia projektu i dlatego oprócz narzędzi specyficznych dla fazy pielęgnacji używa się w niej praktycznie wszystkich poprzednio wymienionych narzędzi. Faza pielęgnacji produktu jest ułatwiona, jeżeli otoczenie programowe wymaga tworzenia bazy danych projektu i wyposażone jest w narzędzia zarządzające taką bazą. Możliwe jest wtedy automatyczne wykonywanie wielu czynności takich jak: powtórna kompilacja jednostek zależnych, wskazywanie części projektu powiązanych z danym modułem itp. W oparciu o metody analizy programu powstały nawet narzędzia odtwarzające projekt z istniejącego kodu, ale na razie sprawdzają się one tylko dla małych projektów. W fazie pielęgnacji szczególnie użyteczne są systemy testowania przedstawione powyżej. Do narzędzi wspomagających tę fazę należą także wszelkiego rodzaju analizatory odwołań.

Środowisko narzędziwe na przykładzie otoczenia Ady

Jak już było wspomniane, w celu zwiększenie efektywności opracowywania produktu programowego należy raczej dążyć do używania całych środowisk narzędziowych, niż do posługiwania się pojedynczymi programami narzędziowymi. W chwili obecnej istnieje wiele takich środowisk. Są one najczęściej związane albo z konkretnymi firmami produkującymi oprogramowanie, albo z konkretnymi metodologiami wytwarzania oprogramowania. Przy wyborze środowiska ważne jest zapewnienie sobie możliwości dalszego rozwoju projektu. Ważne jest też, aby środowisko takie pozwoliło na obsługę szerokiej gamy zastosowań i sprzętu. Obecnie najpopularniejszymi, pełnymi środowiskami narzędziowymi są:

- środowisko VAX/VMS,
- system UNIX z językiem C,
- środowisko programowe Ady.

Bardzo znane, choć niepełne, jest zintegrowane środowisko dla TURBO PASCALA firmy Borland w wersji 4.0 i wyższych.

Przykładem pełnego środowiska programowego jest środowisko języka Ada. Założenia dla niego sformułowane zostały w dokumencie STONEMAN [11]. Istnieje wiele implementacji środowisk programowych Ady i w przeciwieństwie do samego języka nie są one niestety w pełni zestandaryzowane. Minimalne środowisko Ady (MAPSE) powinno zawierać co najmniej:

- język poleceń, najlepiej podobny do Ady,
- językowo ukierunkowany program redagujący,
- językowo ukierunkowany program formatujący,
- kompilator (może ich być kilka),
- konsolidator - może ich być kilka, szczególnie gdy maszyna technologiczna nie jest maszyną docelową,
- analizator statyczny,
- analizator przepływu sterowania,
- program uruchomieniowy,
- administrator bazy projektu,

- program konfigurujący umożliwiający generowanie różnych wersji projektu.

Środowisko takie może być albo zintegrowane albo w postaci osobnych narzędzi sterowanych językiem `poleceń. Obecnie wiele firm posiada gotowe środowiska Ady. Do najbardziej znanych należy ALS (Ada Language System) firmy SofTech. Istnieją także otoczenia dla komputerów klasy IBM PC np.IntegrAda firmy AETECH [12],

Tendencje do automatyzacji - eliminowanie faz cyklu

Rozwój środków narzędziowych, a przede wszystkim rozwój inżynierii oprogramowania, umożliwił próbę automatyzacji niektórych faz cyklu życia produktu programowego. Systemy takie określa się często wspólną nazwą CASE (Computer Aided Software Engineering) [13]. Narzędzia te ułatwiają projektowanie, zawierają środki graficznego zobrazowania projektu, możliwość generowania projektu szczegółowego i półautomatycznego generowania kodu wynikowego. Dodatkowo wyposażone są one w bogate możliwości dokumentowania produktu. Stosunkowo najlepiej rozwinięte są narzędzia półautomatycznego tworzenia kodu. Przykładem takiego narzędzia może być system AnimAID firmy Generic Software Limited pozwalający na półautomatyczne tworzenie programów napisanych w Adzie na podstawie projektu opartego na technice projektowania obiektowego.

Bardzo spektakularnym skróceniem fazy kodowania projektu jest zastosowanie generatorów aplikacji do tworzenia typowych produktów obsługi baz danych i generacji raportów z tych baz. Wzrost efektywności wytwarzania takich produktów w porównaniu z tradycyjnym kodowaniem w językach klasy COBOL-u wynosi często 3000%.

Wielokrotne wbudowywanie fragmentów oprogramowania

Przy tworzeniu nowego produktu programowego nie ma konieczności tworzenia wszystkich jego składowych od początku. Istnieje możliwość zastosowania fragmentów już istniejących produktów. Wielokrotne używanie tych samych modułów jest znane praktycznie od początku istnienia oprogramowania np. biblioteki funkcji matematycznych. Ostatnio technologia ta bardzo się rozpowszechniła przede wszystkim w związku z ogólnym wzrostem rozmiaru projektów. Przy typowych projektach dla danej klasy zastosowań fragmenty wbudowane z istniejących bibliotek stanowią często ponad 70% całego projektu.

Warunkiem koniecznym korzystania z istniejącego już oprogramowania przy tworzeniu nowego projektu, jest standaryzacja zasad wymiany danych. Standaryzacja taka jest w tej chwili typowa dla produktów jednej firmy wytwarzającej oprogramowanie. Przykładem może być oprogramowanie rodziny VAX firmy DEC, w którym wszystkie kompilatory mają tę samą konwencję przekazywania parametrów pomiędzy procedurami i w ten sam sposó generują odwołania do systemu operacyjnego i otoczenia. Dużą rolę w standaryzacji oprogramowania odegrał system UNIX.

Oprócz standaryzacji zasad komunikacji pomiędzy składowymi produktów programowych poszczególnych firm podejmowane są próby stworzenia standardów ułatwiających łączenie produktów różnych firm. Przykładem może być porozumienie X/OPEN [10] obejmujące następujące firmy: BULL, DEC, ERICSSON, HEWLETT-PACKARD, ICL, NIXDORF, OLIVETTI, PHILIPS, SIMENS, UNISYS. Porozumienie to pracuje nad wypracowaniem standardów umożliwiających przenoszenie programów pomiędzy sprzętem przez nie produkowanym.

Podsumowanie

Obecnie oprogramowanie narzędziowe jest bardzo różnorodne i rozbudowane. Należy się spodziewać, że czynniki ekonomiczne będą działały w kierunku zmniejszenia tej różnorodności i

zaniku poszczególnych narzędzi na rzecz rozbudowanych środowisk tworzenia oprogramowania. Już w tej chwili rozbudowane narzędzia korzystają z bazy danych projektu. Należy spodziewać się wzrostu roli tej bazy i przekształcenia jej w bazę wiedzy oraz rozwój środków tworzenia oprogramowania opartych na systemach rzeczoznawczych.

Liczni eksperci twierdzią, że w przyszłości podstawowym narzędziem programistycznym będzie Ada w systemie UNIX. Twierdzą tak zarówno zwolennicy, jak i przeciwnicy obu tych narzędzi.

Literatura:

[1] Boehm B: Improving Software Produktivity. Computer 1987 nr 8

[2] Rook P. Controlling software projects. Software Engineering Journal 1986 nr 1

[3] Wang Yu: A Distributed Specification Model and Its Prototyping. IEEE Transactions on Software Engineering 1988nr 8.

[4] Diederich J., Milton J: An Object-Oriented Design System Shell. OOPSLA'87 Object-Oriented Programming Systems, Languages and Aplications Conference Proceedings. SigPlan Notices grudzień 1987.

[5] O'Brien P., Halbert D., Kilian M: The Trellis Programming Environment. OOPSLA'87 Object-Oriented Programming Systems, Languages and Aplications Conference Proceedings. SigPlan Notices grudzień 1987.

[6] Buhr R: System Design with Ada. Prentice-Hall 1984

[7] Terma T i inni: A System for Generating Language-Oriented Editors. IEEE Transactions on Software Engineering 1988 nr 6.

[8] VAX/VMS Software Language & Tools Handbook. Digital Equipment Corporation 1985

[9] Korel B: PALAS - Program Error-Locating Assistant System. IEEE Transactions on Software Engineering 1988 nr 9.

[10] X/OPEN Portability Guide 1987

[11] Requierements for the Programming Environment for the Common High Order Languages, STONEMAN. Departament of Defense 1980

[12] Nyberg K i Udell J: IntegrAda. Byte 1989 nr 1

[13] Byte 1989 nr 4.

Sprawozdanie z konferencji "Współczesne Obszary Zastosowań Informatyki" Szczyrk 3-5 maja 1989.

W okresie od 3 do 5 maja 89 Górnośląski Oddział Polskiego Towarzystwa Informatycznego zorganizował w Ośrodku Konferencyjno-Wypoczynkowym "Klimczok" w Szczyrku konferencję "Współczesne Obszary Zastosowań Informatyki". przeznaczoną przede wszyskim dla członków PTI z branżowych ośrodków informatycznych. Podstawowym celem konferencji było uzupełnienie wiedzy uczestników o aktualnych kierunkach rozwoju informatyki.

Pierwszy dzień konferencji poświęcony był problematyce systemów operacyjnych. Wygłoszone zostały następujące referaty:

dr K. Nałęcki "Sprzętowe uwarunkowania architektury systemów operacyjnych"

Referat stanowił wprowadzenie do problematyki systemów operacyjnych. Podano metody zarządzania zasobami: czasem procesora, pamięcią wirtualną, pamięciami zewnętrznymi. Przedstawione zostały też problemy związane z systemem plików i sposobami ochrony plików.

dr A. Bukowy "Wykorzystanie mechanizmów sprzętowych w systemie operacyjnym XENIX"

Przedstawiona została podstawowa koncepcja systemu XENIX i sposób, w jaki jest ona realizowana przez procesory INTEL 80286 i 80386. Referent omówił metody buforowania danych i wpływ tych metod na efektywność systemu dla różnych konfiguracji sprzętowych.

mgr P. Krukowski "Architektura systemu QNX"

Referent przedstawił budowę i podstawowe zasady działania systemu QNX. System QNX jest UNIX-podobnym systemem czasu rzeczywistego, przeznaczonym przede wszystkim do sterowania.

Po referatach nastąpiła dyskusja. Dyskutanci skoncentrowali się przede wszystkim na porównaniach zalet i niedomagań systemów XENIX i QNX.

Drugi dzień konferencji rozpoczął referat dr P. Fiszera "System transmisji danych i funkcji aplikacyjnych SITA w obszarze działań linii lotniczych". Przedstawione były funkcje sieci SITA wykorzystywanej przez Polskie Linie Lotnicze LOT. Omówiona została struktura i możliwości sieci SITA.

Następnie przedstwiony został blok referatów związanych z inżynierią programowania.

dr J. Zalewski "Okres istnienia oprogramowania"

W referacie podane zostały normy ANSI, którym powinny odpowiadać wyniki poszczególnych faz cyklu życia produktu programowego.

mgr A. Paprocki "Oprogramowanie narzędziowe"

Referent dokonał przeglądu narzędzi programistycznych, wspomagających masowe wytwarzanie oprogramowania.

dr Z. Szyjewski "Standaryzacja procesu projektowania systemów informatycznych dla celów komputerowego wspomagania"

Referent zaproponował metodologię wytwarzania oprogramowania do celów zarządzania opartą na

punktach węzłowych, oraz przedstawił komputerowe narzędzia wspomagające używanie tej metody,

doc W. Cholewa "Systemy doradcze - struktura, zasady, konstruowanie"

Referat stanowił wprowadzenie do problematyki systemów doradczych (systemów ekspertowych).

Na zokończenie dnia odbyła się dyskusja, która koncentrowała się przede wszystkim na zagadnieniach jakości wytwarzanego oprogramania.

Trzeci dzień konferencji przeznaczono na omówienie problematyki stacji roboczych. Mgr J. Rybnik i mgr J. Solak przedstawili najpopularniejsze na rynku stacje robocze i szczegółowo omówili rodzinę SUN-3.

Przedstawione referaty charakteryzowały obecny stan rozwoju wybranych dziedzin zastosowań informatyki. Wszystkie referaty przyjęte zostały z dużym zainteresowaniem. Ważną częścią konferencji były także rozmowy kuluarowe pozwalające na nawiązanie kontaktów i integrujące środowisko.

Andrzej Paprocki

OPRACOWYWANIE WSPÓLNYCH PROGNOZ ROZWOJU NAUKI I TECHNIKI KRAJÓW RWPG

Współpraca krajów RWPG nabiera nowych znaczeń i przypuszczalnie będzie przybierała nowe formy. Nie będzie to zapewne współpraca przypadkowa. Planowanie tej współpracy, jej odpowiednie stymulowanie, stosowanie właściwych mechanizmów rynkowych i ekonomicznych, przy rezygnacji ze starych niesprawdzonych mechanizmów administracyjnych, formalnych i zarazem nieefektywnych, spowoduje, że powinny nabrać istotnego znaczenia dobre rozpoznania prognostyczne. Ich opracowywanie może wymagać zupełnie innego podejścia, zarówno do meritum zagadnienia, jak do metod ich wykonywania, natomiast tryb postępowania, zakres i sposób formułowania zagadnień nie musi ulec większym zmianom. Dlatego wydaje się, iż zaprezentowanie niniejszego materiału właśnie teraz jest celowe i może okazać się pożyteczne.

WPROWADZENIE

Zasady opracowywania wspólnych prognoz rozwoju nauki i techniki krajów RWPG zostały przygotowane w roku 1987 w ramach specjalnej roboczej grupy prognozowania naukowo-technicznego. Wyspecyfikowano w nich najważniejsze problemy organizacyjne i metodyczne, umożliwiające sterowanie współpracą naukowo-techniczną i ekonomiczną krajów RWPG.

Przy opracowywaniu omawianych zasad wykorzystano poprzednie prace o podobnym charakterze, uwagi krajów i ich propozycje oraz aktualne zalecenia sekretariatu RWPG. Uwzględniono też ogólne zasady współpracy w RWPG.

Dokument stanowi formalne wytyczne, zarówno dla krajów RWPG, jak i dla poszczególnych organów RWPG i innych międzynarodowych organizacji naukowo-technicznych.

PODSTAWOWE ZAŁOŻENIA

Wspólne prognozowanie rozwoju nauki i techniki - formą współpracy naukowo-technicznej krajów RWPG

Prowadzenie wspólnych prac w zakresie prognozowania rozwoju nauki i techniki ma na celu określenie zasadniczych kierunków działań w długoterminowych planach współpracy naukowo-technicznej krajów RWPG.

Współpracę przy prognozowaniu realizuje się w ramach organów RWPG i innych organizacji wielostronnej współpracy.

Wspólne prognozy rozwoju nauki i techniki wypracowywane przez specjalistów z poszczególnych krajów RWPG, zawierają koncepcję rozwoju w wybranych dziedzinach oraz informacje o warunkach, jakie muszą być spełnione dla realizacji takiego rozwoju, jak też propozycje niezbędnych rozwiązań organizacyjnych. Prognozy takie mają wprawdzie jedynie charakter zaleceń, ale stanowią istotny element całego zbioru informacji, niezbędnej dla skutecznego sterowania współpracą naukowo-techniczną w ramach RWPG.

Omawiane wspólne prognozy rozwoju nauki i techniki powinny zawierać następujące elementy:

a/ identyfikację przedmiotu /obiektu/ prognozy, charakterystykę jego stanu w krajach RWPG i na świecie, a w tym:
   - podstawowe charakterystyki techniczno-technologiczne, ekonomiczne i socjalne,
   - takie czynniki /ekonomiczne, socjalne, międzynarodowe, ekologiczne, techniczne i inne/, które mają wpływ na rozwój przedmiotu prognozy, a które pozostają niezmienne w całym objętym prognozą okresie czasu,
   - stan współpracy w przedmiocie prognozy,

b/ tendencje rozwoju przedmiotu prognozy w krajach RWPG, w głównych krajach kapitalistycznych i w krajach rozwijających się,

c/ prawdopodobne wyraźne przełomy w światowym rozwoju naukowo-technicznym i ich wpływ na rozwój nauki i techniki w kra-

jach RWPG, w zakresie przedmiotu prognozy,

d/ przewidywany poziom potrzeb krajów RWPG, osiągnięcie którego jest związane z rozwojem przedmiotu prognozy,

e/ rola współpracy naukowo-technicznej w zwiększeniu efektywności przemysłu krajów RWPG, w odniesieniu do przedmiotu prognozy,

f/ podstawowe problemy naukowo-techniczne, rozwiązanie których zapewniałoby osiągnięcie zamierzonych celów w objętym prognozą okresie czasu,

g/ sposoby i uwarunkowania najszybszego osiągnięcia zamierzonych celów, wraz z oceną podstawowych rozwiązań alternatywnych.

Opracowywane prognozy powinny być wykorzystywane do:

a/ przygotowywania przyszłego KPPNT, stanowiącego bazę dla wypracowywania jednolitej polityki naukowo-technicznej krajów RWPG,

b/ formułowania podstawowych kierunków współpracy naukowo-technicznej interesujących wszystkie kraje RWPG,

c/ zestawianie planów współpracy naukowo-technicznej w formie wielostronnych przedsięwzięć krajów RWPG,

d/ opracowywanie długoterminowych programów współpracy naukowo-technicznej,

e/ opracowywanie szczegółowych programów współpracy naukowo-technicznej w zakresie konkretnych problemów - na najbliższy okres pięcioletni.

Przedmiot i zadania wspólnego prognozowania

Wspólne prognozy powinny obejmować wszystkie podstawowe kierunki i problemy nauki i techniki, decydujące o postępie krajów RWPG.

Zasadniczym zadaniem opracowywania wspólnych prognoz rozwoju nauki i techniki jest identyfikacja aktualnych, w objętym prognozą okresie, kierunków i problemów rozwoju nauki i techniki, z uwzględnieniem konsekwencji socjalno-ekonomicznych. Zadaniem prac prognostycznych jest też określenie dróg

rozwiązania wyspecyfikowanych problemów w ramach współpracy naukowo-technicznej krajów RWPG.

Tak więc, podstawowymi zadaniami współpracy przy prognozowaniu rozwoju nauki i techniki są:

- identyfikacja podstawowych kierunków rozwoju przedmiotu prognozy, identyfikacja parametrów określających ten rozwój, jak też i ocena możliwego wpływu postępu w zakresie przedmiotu prognozy na ogólny rozwój krajów RWPG,
- identyfikacja nierozwiązanych, węzłowych problemów naukowo-technicznych odnoszących się do przedmiotu prognozy,
- określenie najbardziej realnych i najbardziej efektywnych dróg rozwiązywania poszczególnych wymienionych wyżej problemów naukowo-technicznych i ich wdrażanie do przemysłu,
- przygotowywanie propozycji w zakresie organizacji współpracy naukowo-technicznej krajów RWPG, ocena możliwych terminów rozwiązania problemów proponowanych do współpracy oraz ocena niezbędnych środków /nakładów/ na rozwiązanie tych problemów,
- sformułowanie propozycji, wdrożenie i efektywne stosowanie w praktyce szczegółowych wyników współpracy naukowo-technicznej, z uwzględnieniem specjalizacji i kooperacji przyjętej w ramach RWPG.

Prace prognostyczne w zakresie rozwoju nauki i techniki charakteryzują się strukturą hierarchiczną. Najwyższy poziom to określenie zasadniczych, długofalowych kierunków rozwoju, mających istotne znaczenie dla wszystkich krajów socjalistycznych. Kierunki te z zasady mają charakter międzyresortowy /międzygałęziowy/.

Drugi poziom obejmuje istotne problemy naukowo-techniczne, rozwiązanie których wymaga wprawdzie znaczących środków, ale które umożliwią dokonanie istotnego wzrostu potencjału naukowo-technicznego, a zwłaszcza przemysłowego RWPG. Każdy taki problem powinien charakteryzować się kompleksowym ujęciem rozwiązywanych zagadnień.

Na trzecim poziomie znajdują się poszczególne zadania naukowo-techniczne, wymagające przeprowadzenia konkretnych badań

i opracowań, np. skonstruowanie nowych lub istotne zmodernizowanie istniejących maszyn, czy oprzyrządowania dla technologii, opracowania nowych metod, nowych materiałów, lub nowych gatunków materiałów znanych i stosowanych, a nawet nowych form organizacji produkcji, organizacji pracy, czy też nowych sposobów zarządzania itp. Wprowadzenie każdego z tych opracowań ma zapewnić przyspieszenie postępu naukowo-technicznego oraz zwiększenie efektywności współpracy krajów RWPG.

Formalną podstawą do opracowywania prognozy rozwoju nauki i techniki jest postanowienie właściwego organu RWPG, określające zadanie prognostyczne, okres, na jaki należy opracowywać prognozę i kraj koordynujący prace nad prognozą.

Omawiane zasady mają zapewnić jednolitość prac prognostycznych, ułatwić organizację ich przygotowania i uzgadniania. Powinny też umożliwić wykorzystanie opracowywanych prognoz w zarządzaniu współpracą naukowo-techniczną oraz rozszerzyć możliwość zastosowania techniki komputerowej przy opracowywaniu prognoz.

ORGANIZACJA WSPÓŁPRACY PRZY OPRACOWYWANIU PROGNOZY

Organizacyjne formy prac prognostycznych

Prognozy są sporządzane przez organy RWPG i organizacje międzyrządowej współpracy naukowo-technicznej na podstawie zaleceń i postanowień stałych komitetów i komisji RWPG oraz innych organów współpracy międzynarodowej analogicznej rangi.

W kraju koordynującym opracowywanie danej prognozy, określa się główną organizację, która z kolei powołuje krajową grupę roboczą do opracowania danej prognozy. W skład takiej krajowej grupy roboczej powinni wchodzić czołowi specjaliści z właściwych instytucji naukowo-badawczych i doświadczalno-konstrukcyjnych, a ponadto specjaliści metodolodzy prognozowania naukowo-technicznego. Natomiast w krajach uczestniczących jedynie w opracowywaniu prognoz mogą, ale nie muszą, powstawać krajowe robocze grupy do opracowania tejże prognozy.

Przygotowywanie materiałów roboczych do prognozy oraz projekty wspólnych dokumentów wykonuje główna organizacja przy współpracy organizacji współpracujących lub krajowych grup roboczych.

W badaniach prognostycznych mogą być rozróżnione dwa rodzaje prognoz: prognozy jednokrotne i nieprzerwane prace prognostyczne /prognozowanie towarzyszące/.

Prognozy jednokrotne są opracowywane dla wybranych problemów naukowo-technicznych, których rozwiązywanie wydaje się celowe w ramach współpracy wielostronnej. W tych prognozach określa się drogi i środki wspólnego rozwiązywania najpilniejszych problemów rozwoju nauki i techniki.

Prognozowanie towarzyszące należy prowadzić w tych kierunkach postępu naukowo-technicznego, którymi są zainteresowane wszystkie lub większość krajów RWPG, zwłaszcza gdy charakter analizowanych problemów wymaga stałego okresowego przeglądu i uszczegółowiania podstawowych rozwiązań.

Kolejność prac przy opracowywaniu wspólnych prognoz

Opracowanie prognozy obejmuje cztery etapy: analityczny, badawczy, programowy i organizacyjny.

Na etapie analitycznym powinny być rozpatrywane następujące elementy:

- aktualny stan przedmiotu /obiektu/ prognozy w poszczególnych krajach RWPG, w całym RWPG, w innych krajach oraz najwyższy poziom światowy,
- identyfikacja aktualnych potrzeb społecznych, związanych z prognozowanym obszarem,
- stan nauki i techniki w prognozowanym okresie, niezbędny do osiągnięcia założonego poziomu zaspokojenia potrzeb społecznych,
- formy współpracy naukowo-technicznej, które należy przyjąć dla uzyskania wymaganego poziomu rozwoju nauki i techniki /w przedmiocie prognozy/,

Zadaniem etapu badawczego jest dostarczenie odpowiedzi na następujące grupy pytań:

- jaki będzie możliwy do osiągnięcia rozwój przedmiotu prognozy w objętym prognozowaniem okresie, przy zachowaniu obecnych tendencji rozwojowych,
- jakie problemy naukowo-techniczne muszą być rozwiązane, aby osiągnąć założony poziom rozwoju obiektu prognozy,
- dla których z tych problemów musi się zorganizować współpracę naukowo-techniczną krajów RWPG i jakich rezultatów tej współpracy należy oczekiwać.

Etap programowy prognozy ma dać odpowiedzi na następujące pytania:

- jakie są możliwe drogi wspólnego rozwiązywania problemów naukowo-technicznych w ramach prac wielostronnych i jakie są możliwości wdrożenia wyników tych prac do praktyki, oraz jakie będą skutki dodatnie, a jakie negatywne rozwiązania tych problemów,
- ile czasu zajmie realizacja każdej z możliwych dróg,
- jaki jest stopień pewności w realizacji każdej z możliwych dróg.

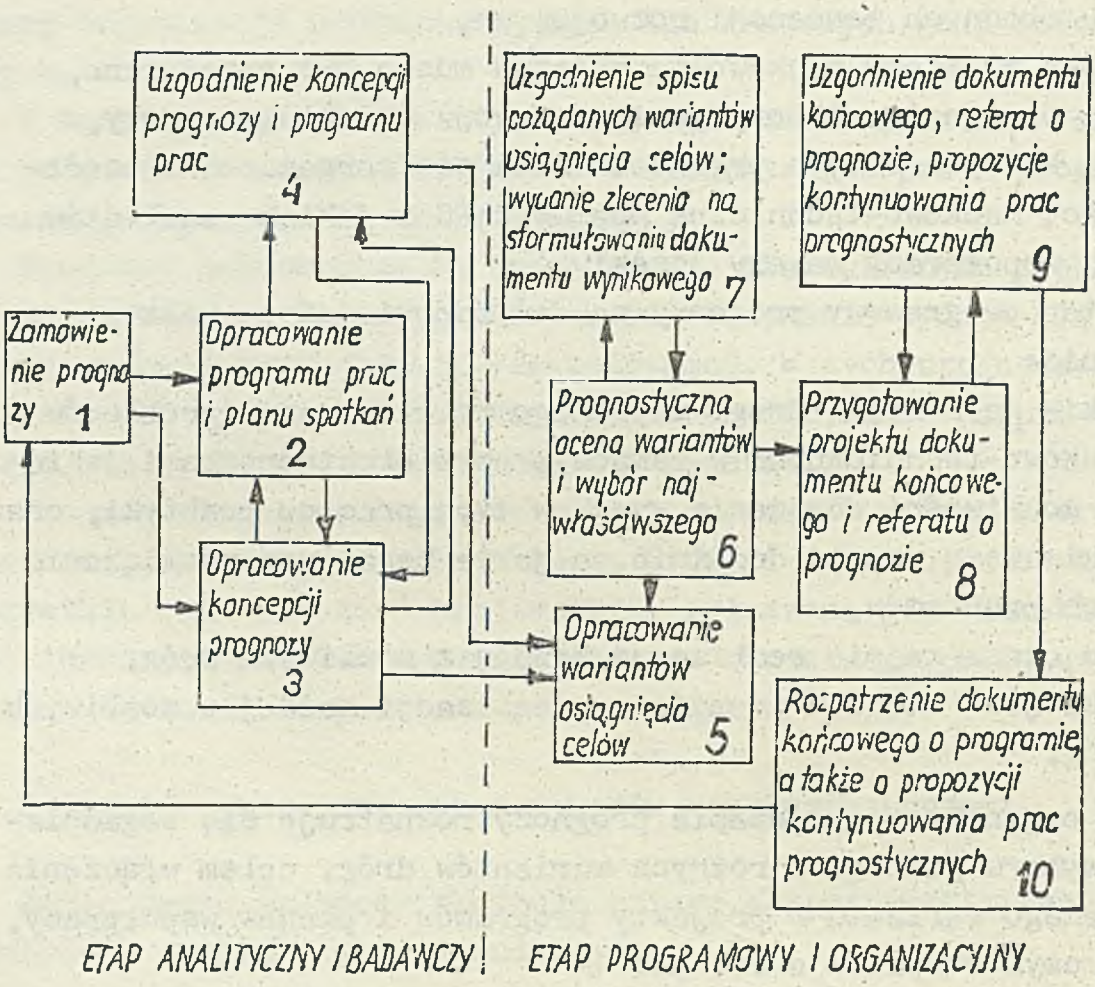
W organizacyjnym etapie prognozy rozpatruje się zagadnienie wyboru jednego z różnych wariantów dróg, celem włączenia wybranego wariantu w projekty programów i planów współpracy, przy czym rozpatruje się tu:

- kadrowe, materialne, techniczne i finansowe zasoby niezbędne do realizacji każdego z możliwych wariantów,
- drogi współpracy w prognozowanym okresie możliwe i najbardziej wskazane dla najskuteczniejszego rozwiązania poszczególnych problemów naukowo-technicznych,
- niezbędne przesłanki dla wdrożenia rezultatów przewidywanego rozwoju, np. specjalizacja, kooperacja, itp. przemysłów współpracujących krajów,
- warianty rozwiązania całego kompleksu problemów organizacyjnych, które wydają się najbardziej racjonalne.

Tryb opracowania prognozy

Opracowanie wspólnej prognozy naukowo-technicznej przez

zainteresowane kraje RWPG prowadzi się według schematu przedstawionego na rys. 1.



Rys. 1. Schemat organizacji prac prognostycznych

Pokazuje on następujące etapy i bloki prac:

Etapy: analityczny i badawczy

Blok 1. Zamówienie na opracowanie prognozy

W zamówieniu określa się cel i zadanie prognozy oraz kolejność jej opracowywania. Zamówienia formułują organy RWPG lub inne organizacje współpracy wielostronnej.

Zamówienie powinno zawierać:

- nazwę zamawiającego,
- określenie obiektu prognozowania

- okres, na jaki ma być opracowana prognoza,
- sformułowanie ogólnych /socjalno-ekonomicznych i politycznych/ celów opracowania prognozy,
- określenie informacyjnych podstaw dla opracowania prognozy /dane o poprzednich prognozach/,
- przewidywane formy organizacji prac prognostycznych,
- określenie krajów RWPG, które będą współuczestniczyć w opracowywaniu prognozy i kraju koordynującego prace, oraz określenie organizacji głównej i współpracujących,
- terminy opracowania całej prognozy i poszczególnych jej etapów,
- czas i miejsce przeprowadzenia pierwszego spotkania specjalistów,
- formy wykorzystania rezultatów prognozy.

Blok 2. Opracowanie programu prac

Główna organizacja opracowuje program pracy, zawierający identyfikację poszczególnych działań i przedsięwzięć, terminy ich wykonania, wykonawców formy przedstawienia wyników. Równocześnie określa się terminy przeprowadzenia i kraje organizujące poszczególne spotkania.

Czas wykonania poszczególnych prac programu powinien być skorelowany z terminami posiedzeń ekspertów. Przygotowanie materiałów na posiedzenie i ich rozesłanie przeprowadza główna organizacja nie później niż na miesiąc przed spotkaniem. Przygotowany program prac nad prognozą kieruje się, w celu uzgodnienia, do krajów uczestniczących w opracowywaniu prognozy.

Blok 3. Opracowanie koncepcji prognozy

Główna organizacja formułuje projekt koncepcji i program prac nad prognozą, oraz rozsyła go, celem uzgodnienia, do krajów uczestniczących.

Koncepcję opracowuje się w ramach etapów analitycznego i badawczego, wykorzystując jakościową i ilościową ocenę stanu

obecnego oraz kierunków rozwoju obserwowanych na świecie.

Koncepcja, oprócz wymienionych elementów, powinna zawierać:

- opis zjawisk i warunków, mogących wpłynąć ograniczająco lub przyśpieszająco na przebieg rozwoju,
- zbiór różnych celów rozwojowych, wraz z zadanym stopniem prawdopodobieństwa osiągnięcia każdego z tych celów,
- wielkość parametrów techniczno-ekonomicznych i socjalnych, jakie powinny być osiągnięte na koniec prognozowanego okresu.

W skład zbioru celów wchodzą przeznaczone do wspólnego opracowywania podstawowe kierunki rozwoju nauki i techniki krajów RWPG. Kierunki te powinny zapewnić osiągnięcie zamierzonego poziomu rozwoju obiektu prognozy i spełnić sformułowany w zadaniu, ostateczny cel prognozy. Są to więc węzłowe problemy naukowo-techniczne, proponowane do wspólnego opracowania w ramach wskazanego podstawowego kierunku oraz wybrane konkretne zadania naukowo-techniczne niezbędne dla rozwiązania problemów węzłowych.

Przy opracowywaniu koncepcji wskazane jest zbadanie możliwości posługiwania się takimi metodami prognozowania, jak statystyczna ekstrapolacja tendencji, opracowanie scenariuszy rozwoju, analizy i studia literaturowe, analiza morfologiczna, oceny ekspertów lub inne możliwe kompleksowe metody prognozowania.

Blok 4. Uzgadnianie koncepcji prognozy i programu prac

Uzgodnienia te przeprowadza się na spotkaniu ekspertów krajów uczestniczących w opracowaniu prognozy. W czasie posiedzenia ocenia się i uzgadnia:

- koncepcję prognozy wraz z układem celów,
- program prac,
- kolejność prac,
- jeżeli potrzeba, zaleca się uszczegółowienie lub uzupełnienie dokumentów prognozy.

Etapy: programowy i organizacyjny

Blok 5. Różne sposoby osiągania celów

Sposoby te, to zbiory problemów i zadań z zakresu poszczególnych podstawowych kierunków rozwoju nauki i techniki. Rozwiązanie tych problemów pozwoli osiągnąć założono wskaźniki charakteryzujące rozwój przedmiotu prognozy. W zależności od złożoności przedmiotu prognozy, możliwa jest dekompozycja problemów na podproblemy, przy czym owa dekompozycja może być wielopoziomowa.

Przy tworzeniu owych różnych sposobów wiodących do osiągnięcia celów wskazane jest rozważenie zarówno metody grafów, czy metody analizy morfologicznej, jak też i innych metod prognozowania.

Blok 6. Ocena prognostyczna

Ocenie prognostycznej podlegają węzłowe problemy naukowo-techniczne, czyli wymienione sposoby osiągania celów prognozy. Ocenę taką należy opracowywać na podstawie analizy:

- światowych tendencji rozwiązywania tego typu problemów, uwzględniając niezbędną w tym zakresie współpracę międzynarodową,
- celów i form współpracy przy rozwiązywaniu danego problemu,
- interesów, jakie poszczególne kraje mogą upatrywać w rozwiązaniu danego problemu,
- warunków koniecznych do rozwiązania problemu, w tym: warunków materialnych, kadrowych, organizacyjno-technicznych, czasowych.

Przy omawianej ocenie, celowe jest stosowanie metod prognozowania, opartych zarówno na informacji statystycznej, jak i na informacji uzyskiwanej od ekspertów.

Wynik oceny każdego z tak przeanalizowanych problemów, powinien być odnotowany w karcie informacyjnej danego problemu. Karta problemu prowadzona jest przez krajowe grupy

robocze, następnie jest scalana przez kraj koordynujący /można tu zastosować komputer/.

Blok 7. Uzgodnienie wykazu sposobów osiągnięcia celów, wydanie zlecenia na sformułowanie dokumentu wynikowego

W toku prac główna organizacja, wraz z organizacjami współpracującymi, ostatecznie uzgadniają propozycję współpracy naukowo-technicznej. Obejmuje ona wykaz podstawowych kierunków i konkretne warianty badań. Ponadto organizacje te opracowują założenia tzw. dokumentu końcowego całego procesu prognozowania.

To działanie kończy właściwe prace etapów programowego i organizacyjnego. Pozostają jeszcze trzy końcowe bloki, mające charakter bardziej formalny.

Blok 8. Przygotowanie projektu dokumentu końcowego i referatu o prognozie

Poszczególne organizacje współpracujące /krajowe grupy robocze/ opracowują swoje materiały do projektu dokumentu końcowego i swój projekt referatu o prognozie.

Równocześnie rozpatruje się celowość kontynuacji prac prognostycznych. Przygotowane materiały rozsyła się do wszystkich krajów uczestniczących w pracach prognostycznych w celu uzgodnienia tychże dokumentów.

Blok 9. Uzgodnienie dokumentu końcowego, referatu o prognozie i propozycji kontynuowania prac prognostycznych

Posiedzenie ekspertów uzgadnia dokument końcowy i referat o prognozie. W tych materiałach mogą być też rozpatrzone propozycje kontynuowania prac prognostycznych. Następnie oba opracowania przedkłada się odpowiedniemu organowi RWPG lub właściwej organizacji współpracy międzynarodowej.

Blok 10. Rozpatrzenie dokumentu końcowego i referatu o prognozie oraz propozycji kontynuowania prac prognostycznych

Właściwy organ RWPG lub właściwa organizacja międzynarodowa rozpatruje dokument końcowy i referat o prognozie, a następnie podejmuje decyzje o ich wykorzystaniu. Ponadto, jeśli uzna celowość kontynuacji prac prognostycznych, kieruje te materiały do Komitetu Współpracy Naukowo-Technicznej RWPG.

Struktura wskaźników prognostycznych

Strukturę wskaźników prognostycznych określa się w zależności od natury konkretnej prognozy. Podstawowym wymaganiem dla wyboru wskaźników prognostycznych jest możliwość jednoznacznego przedstawienia danych wyjściowych prognozy w następujących typach dokumentów:

- dokument prognozy,
- zamówienie na prognozę,
- tablica opisu zbioru celów,
- karta informacyjna kierunku,
- karta informacyjna problemu,
- uogólniona forma przedstawienia materiałów prognostycznych.

Zalecane formy prac przy opracowaniu prognozy

Opracowywanie prognoz jednorazowych

Przeprowadza się dwa lub trzy spotkania ekspertów. Na pierwszym uzgadnia się wspólną koncepcję prognozy, układ celów i dokumenty organizacyjne, takie jak np. program opracowania prognozy. Na drugim posiedzeniu uzgadnia się spis pożądanych sposobów osiągnięcia celów i przydziela się zadania do opracowania i ujęcia w dokumencie końcowym. Jeżeli to konieczne, to te materiały mogą być uzgadniane pomiędzy krajami w trybie roboczym. Na trzecim posiedzeniu uzgadnia się końcowy

dokument prognostyczny i jeżeli to konieczne, przyjmuje się wniosek o kontynuacji prac prognostycznych.

Przy tego typu prognozach operuje się następującymi rodzajami dokumentów:

1/ zamówienie na prognozę,
2/ program prac, zawierający kolejność i terminy opracowywania prognozy,
3/ koncepcja prognozy, zawierająca opis stanu obiektu /obszaru/ prognozy, porównanie z poziomem światowym, opis pożądanego poziomu zaspokojenia potrzeb społecznych, ocenę pożądanego poziomu rozwoju prognozowanego obiektu /obszaru/,
4/ spis podstawowych kierunków rozwoju nauki i techniki, proponowanych do wspólnego opracowania, czyli zbiór celów,
5/ wykaz sposobów osiągnięcia celów,
6/ końcowy dokument prognostyczny,
7/ referat o prognozie.

Ciągłe prace prognostyczne /prognozowanie towarzyszące/

Prace prognostyczne prowadzi się równolegle z innymi pracami, realizowanymi wspólnie przez różne kraje RWPG w danym zakresie tematycznym, przy czym realizuje się /w pełnym zakresie/ wszystkie czynności pokazane na schemacie /rys. 1/. Jednakże wobec ciągłego charakteru prac prognostycznych, nie wszystkie dokumenty prognostyczne /np. koncepcja, zbiór celów, itp./ opracowuje się ciągle całkowicie na nowo. Zwykle są one okresowo jedynie odnawiane. Pozwala to, z jednej strony, bardziej szczegółowo przeanalizować najnowsze tendencje rozwojowe, a z drugiej strony - skrócić terminy wykonania poszczególnych etapów.

PREZENTACJA REZULTATÓW PRAC NAD PROGNOZĄ

Rezultaty prognozy /dokument końcowy i referat o prognozie/ przekazuje się do organów RWPG lub właściwej organizacji międzynarodowej, która zamówiła wykonanie prognozy. Dokumenty te kieruje się również do Komitetu Współpracy Naukowo-Technicznej RWPG.

Zawartość dokumentu końcowego

Końcowy dokument prognostyczny składa się z tekstu podstawowego i załączników i ma następującą strukturę i formę:

1/ karta tytułowa,
2/ typ prognozy,
3/ charakterystyka obiektu /przedmiotu, obszaru/ prognozy i jego roli w zaspokojeniu społecznych celów i potrzeb,
4/ podstawa przeprowadzania prac prognostycznych,
5/ charakterystyka oddzielnych elementów obiektu prognozy,
6/ rezultaty badań analitycznych:
   - porównywanie stanu i tendencji rozwoju obiektu prognozy i jego składowych /elementów/ w krajach uczestniczących w opracowywaniu prognozy z poziomem światowym,
   - ocena tendencji rozwoju obiektu prognozy oraz znaczenie tych tendencji dla nauki i gospodarki narodowej krajów uczestniczących w pracach,
7/ zbiór celów rozwoju obiektu /obszaru/ prognozowanego i jego elementy, czyli:
   - podstawowe kierunki rozwoju nauki i techniki,
   - węzłowe problemy naukowo-techniczne,
   - konkretne zadania naukowo-techniczne,
8/ uwarunkowanie rozwoju obiektu prognozy i jego elementów w prognozowanym okresie:
   - podstawowe zmiany, które mogą zajść w czasie objętego prognozą okresu, a mające istotny wpływ na rozwiązanie prognozowanych problemów,
   - zalecenie najbardziej efektywnych sposobów dróg osiągnięcia zamierzonych celów,

- dane o efektywności gospodarki narodowej, możliwej do osiągnięcia w wyniku rozwiązania objętych prognozą problemów,

9/ ocena podstawowych zasobów: materialnych, energetycznych, kadrowych, finansowych i innych, niezbędnych dla realizacji poszczególnych, przewidywanych sposobów osiągnięcia założonych celów,

10/ zalecenia co do skali i zakresu wykorzystania prognozowanych obiektów, oraz ocena skutków socjalno-ekonomicznych tych zastosowań. Propozycje, co do realizacji prognozowanych prac z uwzględnieniem podziału zadań pomiędzy kraje RWPG, jak też podział pomiędzy kraje RWPG zadań w zakresie badań naukowych, opracowywania i wdrażania do produkcji i zastosowania obiektów będących przedmiotem prognozy,

11/ propozycje, co do dalszych wspólnych prac prognostycznych,

12/ dopuszcza się następujące formy prezentacji podstawowych rezultatów badań prognostycznych:

- zamówienie na prognozę,
- tablice opisu zbioru celów prognozy,
- karta informacyjna kierunku,
- karta informacyjna problemu,
- uogólniona forma przedstawienia materiałów prognostycznych,

13/ informacja o zespołach badawczych,

14/ wykaz przebadanych materiałów,

15/ załączniki do dokumentu końcowego /zestaw i objętość załączników wynika z konkretnej sytuacji/. W załącznikach przedstawia się materiały uzupełniające, wykorzystywane przy opracowaniu prognozy.

Zawartość referatu o prognozie

Referat, czyli wykład o prognozie powinien zawierać podstawowe rezultaty prac prognostycznych. Struktura wykładu powinna być zgodna z końcowym dokumentem prognostycznym i odzwierciedlać następujące zagadnienia:

1/ podstawę przeprowadzenia prac prognostycznych,
2/ charakterystykę obiektu prognozy i jego rolę w zaspokojeniu konkretnych celów i potrzeb socjalno-ekonomicznych,
3/ porównanie stanu i tendencji rozwoju obiektu prognozowanego w krajach opracowujących prognozę i w krajach przemysłowo rozwiniętych,
4/ ocena tendencji rozwojowych w obszarze prognozy i ich znaczenie dla nauki i gospodarki narodowej krajów prognozujących,
5/ propozycje najbardziej efektywnych sposobów osiągnięcia zamierzonych celów, z uwzględnieniem różnych możliwych sposobów,
6/ ocena zasobów niezbędnych dla realizacji poszczególnych sposobów osiągnięcia celów,
7/ propozycje odnoszące się do skali i zakresu wykorzystania prognozowanych obiektów w dłuższym okresie, jak też i oceny socjalno-ekonomicznych konsekwencji zaistnienia prognozowanych zdarzeń,
8/ wnioski i propozycje, co do organizacji współpracy naukowo-technicznej krajów RWPG, dla każdego z rozpatrywanych sposobów osiągnięcia celów.

ZAKOŃCZENIE

"Organizacyjno-metodyczne podstawy opracowywania wspólnych prognoz rozwoju nauki i techniki krajów RWPG" są podstawowym dokumentem systemu dotyczącego metodyki wspólnego opracowywania prognoz rozwoju naukowo-technicznego. Uzupełniają go, opracowane przez Roboczą Grupę Współpracy Prognozowania Naukowo-Technicznego, zalecenia dotyczące wyboru metod wspólnego prognozowania, metodyki problemowo ukierunkowanych ocen potencjału naukowo-technicznego, metodyki formułowania kompleksowych programów naukowych, międzyresortowe i resortowe metodyki wspólnego prognozowania. Opracowywane są też w poszczególnych organach RWPG inne, niezbędne materiały metodyczne, tworzące cały system metodyk wspólnego prognozowania.

Na podstawie materiałów RWPG opracowała:
dr inż.St.Bonkowicz-Sittauer

NOWOŚCI TECHNICZNE

64-bitowy mikroprocesor firmy Intel - i860

Jest to pierwsza w świecie realizacja układu, który zawiera ponad milion tranzystorów w jednej kostce i oferuje moc obliczeniową superkomputera w jednym elemencie bardzo dużej skali integracji /VLSI/. Handlowe oznaczenia układu to odpowiednio A80860-33 i A80860-40 w zależności czy wykorzystywany jest generator zegarowy o częstotliwości 33 czy 40 MHz. Zapewnia to operacje na liczbach całkowitych, działania o zmiennym przecinku i zastosowania graficzne. Można z powodzeniem stosować go do skomputeryzowanych stanowisk inżynierskich, przy obliczeniach naukowych, graficznych stacjach roboczych i systemach dla wielu użytkowników. Równoległa architektura pozwala na uzyskiwanie wysokiej wydajności wykorzystując potokowe /pipelined/ układy przetwarzania i duże pamięci pomocnicze /caches/. Zastosowano tu krzemową technologię CHMOS IV o wymiarze podstawowym 1/um.

Mikroprocesor i860 składa się z 9 podzespołów; są to:

1/ główna jednostka przetwarzania,
2/ zmiennoprzecinkowy układ sterujący,
3/ sumator zmiennoprzecinkowy,
4/ zmiennoprzecinkowy układ mnożący,
5/ układ graficzny,
6/ układ podziału na strony /paging/,
7/ pamięć pomocnicza rozkazów,
8/ pamięć pomocnicza danych,
9/ układ sterowania magistralami i pamięciami.

Główna jednostka przetwarzania steruje wszystkimi operacjami mikroprocesora i860. Wykonuje ona rozkazy ładowania, pamiętania, działań na liczbach całkowitych i bitach, a także steruje przesyłaniem i ładowaniem do pamięci rozkazów dla części zmiennoprzecinkowej. Do działania na liczbach całkowitych wykorzystuje się zespół 32 rejestrów 32-bitowych. Rozkazy ładowania i pamiętania wprowadzają lub wyprowadzają 8, 16 i 32-bitowe dane do lub z tych rejestrów. Pełna lista rozkazów logicznych, sterujących przesłaniami i działających na liczbach całkowitych pozwala jednostce głównej realizować pełne oprogramowanie systemowe i aplikacyjne. Mechanizm pułapek pozwala na szybkie reagowanie na wyjątki i przerwania zewnętrzne. Usuwanie błędów ułatwione jest możliwością śledzenia danych i rozkazów.

Sprzęt układów zmiennoprzecinkowych dołączony jest do oddzielnego zespołu rejestrów zmiennoprzecinkowych, które mogą tworzyć strukturę 16 rejestrów 64-bitowych lub 32 rejestrów 32-bitowych. Specjalne rozkazy ładowania i pamiętania mają również dostęp do tych samych rejestrów jako struktura 8x128 bitów. Wszystkie rozkazy zmiennoprzecinkowe wykorzystują te rejestry jako źródło i przeznaczenie operandów.

Zmiennoprzecinkowy układ sterujący kontroluje zarówno sumator, jak i zmiennoprzecinkowy układ mnożący wydając rozkazy zajmujące się wszystkimi źródłowymi i wynikowymi wyjątkami i uaktualniając bity stanu w zmiennoprzecinkowym rejestrze stanu. Sumator i układ mnożący mogą działać równolegle dając do dwóch wyników na cykl zegarowy. Zmiennoprzecinkowe typy danych, rozkazy oraz postępowanie w sytuacjach wyjątkowych zgodne są ze standardem dla binarnej arytmetyki zmiennoprzecinkowej /ANSI-IEEE Std 754-1985/.

Sumator zmiennoprzecinkowy wykonuje dodawanie, odejmowanie, porównanie i konwersje 64 i 32-bitowych wartości zmiennoprzecinkowych. Normalnie rozkaz dodawania realizowany jest w trzech cyklach zegarowych, natomiast w systemie potokowym wynik uzyskuje się w jednym cyklu.

Zmiennoprzecinkowy układ mnożący dokonuje mnożenia liczb zmiennoprzecinkowych i całkowitych oraz określania odwrotności liczb zmiennoprzecinkowych na 64- i 32-bitowych wartościach zmiennoprzecinkowych. Rozkazy te wykonywane są normalnie w trzech do czterech cykli zegarowych, lecz w systemie potokowym można uzyskać wynik zwykłej dokładności w jednym cyklu, a podwójnej dokładności w dwóch cyklach.

Układ graficzny obejmuje specjalny podzespół logiczny, pozwalający na uzyskiwanie obrazów trójwymiarowych o zmiennej intensywności barw i eliminację powierzchni zasłaniających za pomocą specjalnego algorytmu. Graficzne własności mikroprocesora i860 pozwalają na uzyskiwanie obrazów bardzo zbliżonych do oryginalnych przedmiotów.

Układ podziału na strony zapewnia zabezpieczoną, podzieloną na strony, wirtualną pamięć dołączoną przez 64 wejścia, oraz 4-torową, ustawianą asocjacyjnie pamięć zwaną TLB /Translation Lookaside Buffer - bufor indeksów transakcji/. Układ wykorzystuje tę pamięć do translacji adresu logicznego na fizyczny i sprawdzania nieprawidłowości dostępu. Układ zabezpieczenia dostępu wykorzystuje dwa poziomy uprzywilejowania użytkownika i administratora.

Pamięć pomocnicza rozkazów jest dwutorową, ustawianą asocjacyjnie pamięcią o pojemności 4 kB w blokach 32-bajtowych. Przesyła ona 64 bity na cykl zegarowy /320 Mbajt/s przy 40 MHz/.

Pamięć pomocnicza danych jest dwutorową, asocjacyjnie ustawianą pamięcią o pojemności 8 kB, która przesyła dane z prędkością 128 bitów na cykl zegarowy /640 MB/s przy 40 MHz/. Mikroprocesor i860 normalnie uaktualnia stan pamięci pomocniczej przy zapisie do pamięci bez potrzeby natychmiastowego uaktualniania pamięci głównej. Gdy zachodzi potrzeba, użyciu pamięci pomocniczej zapobiega oprogramowanie.

Układ sterowania magistralami i pamięciami pomocniczymi zapewnia jednostce głównej dostęp do danych i rozkazów. Odbiera on żądania i specyfikacje cykli z jednostki głównej, powoduje opuszczenie przetwarzania w pamięciach pomocniczych, steruje translacją TLB i zapewnia sprzężenie z magistralą zewnętrzną. Jego struktura potokowa pozwala jednocześnie realizować trzy istotne cykle magistrali.

Na podstawie Advance Information firmy Intel

Pierwsze systemy RISC firmy Data General

Data General Corp. oferuje serwer i cztery stanowiska robocze oparte na zredukowanej liście rozkazów, a zbudowane na mikroprocesorach 88000 firmy Motorola realizowanych w technologii CMOS. Pracują one w systemie operacyjnym Unix w wersji firmowej VDG/VX. Pierwsza jednostka to system bezdyskowy z monochromatycznym monitorem 20-calowym i pamięcią operacyjną o pojemności 4 Mbajty. Cena jej ma być znacznie poniżej 10 tys. dolarów.

Pierwszy komputer na mikroprocesorze 88000

Przed ofertą General Corp. pojawił się komputer zbudowany na procesorze 88000. Jest to Model 8000 firmy Sanyo/Icon International z Orem. Jest to trzyprocesorowy system osiągający 15 milionów rozkazów na sekundę w cenie 155 tys. dolarów. Stanowi on konkurencję dla średnich komputerów VAX firmy DEC a także Pyramid i Sequent.

Pamięć EPROM firmy Texas Instruments

Nie trzeba stosować pamięci pomocniczych do przechowywania kodu procesora, gdy średni czas dostępu wynosi tylko 20 ns jak w pamięci EPROM o nazwie Burst Mode TI o pojemności 1Mbitów. Zawiera ona matrycę 1Mbitową o rozstępie 1,4 µm w technologii MOS i 256-bitowy bufor, działający jak pamięć pomocnicza, do którego wprowadza się sekwencyjnie informację z matrycy zanim zapotrzebuje ją mikroprocesor. Pamięć ma być sprzedawana pod koniec 1989 roku.

Obniżenie kosztów przetwarzania w maszynach Encore Computer Corp.

Oparte na mikroprocesorze 32532 firmy National Semiconductor superminikomputery Multimax 500 firmy Encore Computer z Malborough zawierają dwuprocesorową kartę jednostki centralnej, każdy z nich wykonuje 8,5 milionów operacji na sekundę. Można połączyć 10 takich kart i wówczas koszt przetwarzania wyniesie poniżej 3500 dolarów za 1 Mrozkaz/s.Cały system, który miał być sprzedawany od lipca 1989 r. miał kosztować 155 tysięcy dolarów. Bądący w podobnej cenie VAX 6300 firmy DEC daje tylko około 5 Mrozkazów/s. Równie korzystnie wypada konkurencja z systemem 6550 firmy Prime Computer i MV/40000 Data General.

Sprzętowa kompresja danych

Procesor 9703 firmy STAC z Pasadeny realizuje algorytm dwukrotnej kompresji danych, co pozwala co najmniej 30-krotnie zwiększyć szybkość przesyłania w porównaniu z implementacjami programowymi. Przyjmuje on dane wejściowe z prędkością 500 Kbajtów/s i wysyła ścieśnione dane na taśmę lub dysk z prędkością 250 Kbajtów/s. Układy te dostępne są od drugiego kwartału 1989 roku z przeznaczeniem dla stacji taśmowych typu QIC w cenie około 50 dolarów.

Wysoka wydajność mikroprocesora firmy NEC

Mikroprocesor V80 NEC Corp. jest 32-bitowym układem o rozszerzonej liście rozkazów /CISC/ pracującym z częstotliwością zegara 33 MHz. Wykonuje on 16,5 miliona rozkazów na sekundę, a więc 2,5 raza więcej niż poprzedni układ V70 i 5 razy więcej niż 16-bitowy V16. Układ zawdzięcza swe dobre wyniki 7-stopniowej architekturze potokowej /pipeline/ i 1 Kbajtowym pamięciom pomocniczym dla danych i rozkazów. Ponadto układ zawiera przewidywanie rozgałęzień i zarządzanie pamięcią. Zapewnione jest też ciągłe wykrywanie błędów na magistralach danych i adresowych. Kostka o wymiarach 14,5x15,5 mm zawiera 930 tysięcy tranzystorów i realizowana jest w technice dwuwarstwowej /aluminium/ z rozstępem 0,8 um. Cena wersji 25 MHz wynosi 960 dolarów, a wersji 33 MHz 1200 dolarów. W roku 1990 przewiduje się wersję 45 MHz dającą 22,5 miliona rozkazów na sekundę.

Electronics 3/89

Oprogramowanie sieciowe firmy Novell

Najbardziej popularnym systemem operacyjnym w dziedzinie sieci lokalnych jest NetWare firmy Novell z Provo w stanie Utah. Użytkownicy tego systemu mogą współdzielić pliki i oprogramowanie aplikacyjne zapamiętane na dyskach twardych serwera plików, a także drukarki i inne drogie urządzenia peryferyjne. Struktura wieloprocesorowa i wielozadaniowa umożliwia wykonywanie wielu operacji jednocześnie, zwiększając wydajność sieci. Systemy baz danych, oprogramowanie najczęściej spotykane w zarządzaniu ma lepszą wydajność w sieci sterowanej NetWare niż wykonywane na samodzielnych mikrokomputerach. System wprowadza ograniczenia dostępu do zasobów sieci zgodnie z profilem użytkownika określonym przez zarządzającego siecią. Umożliwia on wykonanie praktycznie każdego programu aplikacyjnego pracującego pod DOS-em, a także stosowanie IBM PS/2 jako stacji roboczych i serwerów plików.

Najprostszą wersją tego systemu jest ELS NetWare Poziom I v.2.0a, która zaprojektowana jest dla nie więcej jak 4 aktywnych użytkowników, choć liczba zarejestrowanych użytkowników może być większa. Jest ona zgodna z DOS-em 3.0÷3.3 i Windows/386. Jest to wstępne rozwiązanie dla małych

przedsiębiorstw i biur, i ma charakter niededykowany, to znaczy, że serwer plików może być używany jako stacja robocza. Wersję tę łatwo wymienić na jeden z systemów wyższego poziomu. Maksymalna konfiguracja serwera obejmuje 1000 jednocześnie otwartych plików, 5 drukarek sieciowych, 2 dyski twarde.

Nieco bardziej rozbudowanym systemem jest ELS NetWare Poziom II, który umożliwia jednoczesną pracę do 8 użytkowników. Jego podstawowe cechy i mechanizmy zarządzania są identyczne jak w Advanced NetWare v.2.1x. W systemie tym tabele alokacji Plików z dostępem indeksowym TURBO-FAT umożliwiają szybkie przeszukiwanie, co jest ważne dla dużych plików (powyżej 2 MB). Ponadto data wygaśnięcia ważności i ograniczenia uprawnień, a także ograniczenia czasu sesji, liczba nieudanych prób nawiązania sesji i stacje robocze, z których można sesję nawiązać mogą tu być połączone z każdym kontem użytkownika, przy czym system sprawdza co pół godziny ograniczenia związane z kontem użytkownika oraz poziom jego funduszy i automatycznie odłącza użytkowników nieuprawnionych do pracy. System umożliwia też obciążenie użytkowników za używanie zasobów sieci i ograniczenie rozmiarów pamięci dyskowej dostępnej użytkownikowi. Mechanizm Hot Fix na bieżąco wykrywa defekty nośnika dysków twardych i usuwa błędy podczas pracy systemu. Mogą tu być wykonywane (w trybie 286 pracy systemu operacyjnego) procesy dowartościowujące, z którymi komunikacja umożliwiona jest przez nowe łącze z programem użytkowym, które pozwalają też na rozliczenie użycia zasobów, zarządzanie kolejkami, diagnostykę, tworzenie wirtualnych kontroli i niestandardowe zabezpieczenie zasobów. System upraszcza pracę zarządzającego siecią przez wprowadzenie sterowanych przez menu programów narzędziowych, zawiera oprogramowanie zdalnego mostu łącz asynchronicznych oraz emulator NetBIOSa i umożliwia stosowanie standardowego sprzętu sieciowego. Działa on we wszystkich powszechnie używanych topologiach sieci lokalnych włączając Ethernet, ARCNET i Token-Ring. Jest kompatybilny z DOS-em 2.x, 3.x i 4.x oraz Windows/386 i OS/2 Standard i Exended Edition 1.x i umożliwia wykonanie wszystkich popularnych programów edukacyjnych. Może pracować zarówno jako system dedykowany, jak i niededykowany. Maksymalna konfiguracja serwera to tak jak przy poziomie I 1000 otwartych plików, a ponadto 32 tysiące obiektów katalogowych w wolumenie, 32 napędy dysków twardych, 32 wolumeny, każdy po maksymalnie 255 MB, 1500 buforów pamięci kieszeniowej, 2 GB pamięci masowej i 12 MB pamięci operacyjnej.

Znacznie szerszą wersją jest Advanced NetWare v.2.1x przystosowana do pracy 100 użytkowników jednocześnie. Oprócz cech poprzednich systemów ma i tę, że obejmuje łączenie sieci lokalnych, wielokrotne połączenia zdalne oraz współpracę z dużymi komputerami i minikomputerami. Zapewnia też ochronę danych, rozliczanie użytych zasobów, ulepszony poziom bezpieczeństwa i mechanizmy programowania zwiększające moc sieci lokalnych, oraz umożliwia jednoczesną pracę do 4 różnych adapterów sieciowych. Jest to pełny system NetWare zaprojektowany dla dużych przedsiębiorstw i biur. Maksymalna konfiguracja serwera jest taka sama jak w ELS NetWare poziom II.

Najnowszą i najbardziej rozbudowaną konfiguracją jest SFT NetWare v.2.1x, która również pozwala na jednoczesną pracę 100 użytkowników. Mamy tu jednak do czynienia ze zwiększoną niezawodnością serwera plików przez umożliwienie automatycznego powielenia tych samych danych na dwóch identycznych dyskach dołączonych do tego samego stanowiska (dyski lustrzane) lub do różnych sterowników (dyski zdublowane). System śledzenia transakcji chroni informacje w bazie danych przed uszkodzeniem lub utratą spójności w momencie awarii.

Na podstawie materiałów reklamowo-dokumentacyjnych firmy Novell

Cena zł. 380.-

ISSN 0239-8044