Autor: *Michał Kapiczyński*

1. Cechy charakterystyczne i zastosowania programowania obiektowego

Programowanie obiektowe - jest jednym z paradygmatów programowania, czyli sposobu patrzenia programisty na przepływ sterowania i wykonywania programu komputerowego. Zgodnie z nim programy definiuje się za pomocą obiektów, komunikujących się między sobą w celu wykonania określonego zadania.Obecnie jest to najpopularniejsza technika programistyczna.

Programowanie obiektowe jest potoczną, ogólnie przyjętą nazwą programowania zorientowanego obiektów.

Istnieje jeszcze:

Object based design - programowanie oparte na obiektach - paradygmat, który wykorzystuje jedynie pojęcie obiektu i związaną z tym hermetyzację kodu i ochronę pól,

Object orientated programming - programowanie zorientowane obiektowo - tutaj mamy do czynienia z czymś więcej - tworzenie hierarchii klas i obiektów, definicja ich wzajemnych zależności, zmiana zachowania klasy w zależności od typów itd.

Obiekt - materialny lub abstrakcyjny byt, który można wyizolować ze środowiska,

Obiekt (bardziej programistycznie - struktura danych, która występuje łącznie z operacjami dozwolonymi do wykonania na niej, Obiekt może być złożony tzn. składać się z innych obiektów.

Obiekt jest charakteryzowany poprzez:

- tożsamość, która odróżnia go od innych obiektów (miejsce zajmowane przez ten obiekt w pamięci komputera),
- stan, który może zmieniać się w czasie (bez zmiany tożsamości obiektu). Stan obiektu w danym momencie jest określany przez aktualne wartości atrybutów i powiązań z innymi obiektami.
- zachowania do niego przypisane, tj. zestaw operacji, które wolno stosować na danym obiekcie,

Atrybuty obiektów mogą być:

- proste,
- złożone,
- wskaźnikowe,
- powtarzalne,
- opcjonalne,
- domyślne,
- pochodne (wyliczane),
- wspólne dla zestawu obiektów,
- inne obiekty.

```
class A {
public:
  ...
  void rob_cos(B* dawca);
  ...
private:
  ...
};

class B {
public:
  ...
  void rob_cos(A* dawca);
  ...
private:
  ...
};


int main() {
  A obA;
  B obB;
  ...
  obA.rob_cos(&obB);
  ...
}
```

Relacje między obiektami:

```
class A {
public:
  ...
  void setB(B *_s);
  void rob_cos();
  ...
private:
  B *czesc
};

class B {
public:
  ...
private:
  ...
};


int main() {
  A obA;
  B obB;
  A.setB(&B);
  ...
  obA.rob_cos();
  ...
}
```

- **Skojarzenie** - obiekt typu A wykorzystuje obiekt typu B i/lub B wykorzystuje A w celu wykonania swoich zadań. A i B są tworzone i pamiętane zupełnie niezależnie.

- **Agregacja** - obiekt typu A zawiera w sobie obiekt typu B, ale oba są tworzone niezależnie od siebie. Obiekt typu A może składać się z wielu obiektów innych typów.

```
class A {
public:
  ...
  void rob_cos();
  ...
private:
  B czesc
};

class B {
public:
  ...
private:
  ...
};


int main() {
  A obA;
  ...
  obA.rob_cos();
  ...
}
```

- **Kompozycja** - obiekt typu A zawiera w sobie obiekt typu B i jest jego panem i władcą tzn. A zarządza czasem życia obiektu typu B - A tworzy i niszczy B. Obiekt B nie może istnieć (nie ma racjonalnego wyjaśnienia takiej możliwości) bez obiektu A.

- **Uogólnienie** - obiekt typu B dziedziczy po obiekcie typu A. A jest klasą nadrzędną, B - podrzędną.

Przykład obiektu: Samochód:

Jego atrybutami mogą być:

- pozycja,
- prędkość,
- stan paliwa,
- części składowe np. silnik, koła,

Czynności, które może wykonywać:

- start silnika,
- przemieszczanie się,
- przyspieszanie,
- hamowanie,
- ...

Klasa to szablon obiektu:

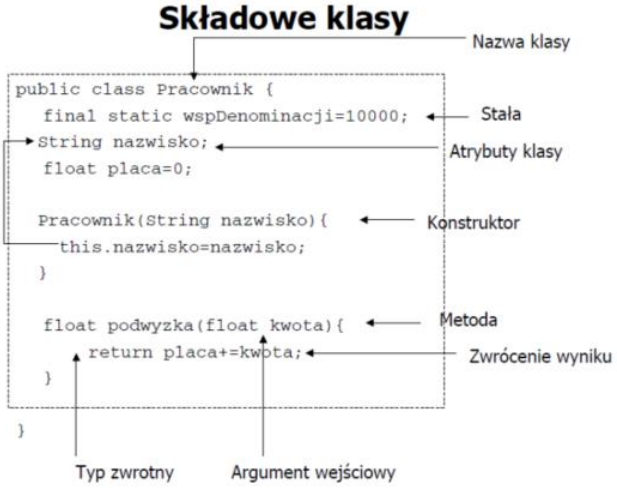
- Jest miejscem przechowywania (definiowania) tych własności grupy obiektów, które są niezmienne (co do definicji, a nie wartości) dla wszystkich członków grupy,
- Ogranicza kontekst, w którym odwołania do obiektu może być użyte w programie,
- Dobrze zbudowana klasa jest starannie wydzieloną abstrakcją pochodzącą ze słownictwa dziedziny danego problemu.
- Obejmuje pewien mały, dobrze określony zbiór zobowiązań, z których jest w stanie się w pełni wywiązać. Zapewnia oddzielenie specyfikacji abstrakcji od jej implementacji. Jest zrozumiała i prosta, a przy tym rozszerzalna i dająca się łatwo dostosować do potrzeb.

Struktura klasy:

- pola - atrybuty (różne wartości dla różnych instancji),
- metody - (takie same dla wszystkich).

Ochrona danych - definiowanie obszaru widoczności zmiennej lub funkcji:

- publiczne (public) - dostępne z dowolnego miejsca w programie,
- prywatne (private) - dostępne jedynie z wnętrza danej klasy,
- chronione (protected) - dostępne z wnętrza danej klasy i klas pochodnych.



Podstawowe zasady programowania obiektowego:

- Zasada dekompozycji (metoda walki ze złożonością) - rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i od całości,
- Zasada abstrakcji (metoda walki ze złożonością) - budowa abstrakcyjnych struktur i operowanie na nich bez wnikania w ich wewnętrzną strukturę. Eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji. Wyodrębnienie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzenie pojęć lub symboli oznaczających takie cechy,
- Zasada ponownego użycia - wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, bibliotek, komponentów projektu itd.
- Zasada sprzyjania naturalnym ludzkim własnościom (cel nadrzędny obiektowej analizy) - dopasowanie modeli pojęciowych i modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

Cechy paradygmatu obiektowego:

- Hermetyzacja

Ukrywanie implementacji, **enkapsulacja**. Zamknięcie pewnego zestawu bytów programistycznych w "kapsułę" o dobrze określonych granicach. Oddzielenie abstrakcyjnej specyfikacji tej kapsuły (obiektu, klasy, moduły etc.) od jej implementacji, ukrycie części informacji zawartej w tej kapsule dla operacji z zewnątrz obiektu. Hermetyzacja jest podstawową techniką abstrakcji tj. ukrycia wszelkich szczegółów danego przedmiotu lub bytu programistycznego, które na danym etapie rozpatrywania (analizy, projektowania, programowania) nie są istotne.

Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Każdy typ obiektu prezentuje innym obiektom swój interfejs, które określa dopuszczalne metody współpracy.

Pewne języki osłabiają to założenie, dopuszczając pewien poziom bezpośredniego (kontrolowanego) dostępu do wnetrza obiektu, ograniczając w ten sposób poziom abstrakcji. Przykładowo w niektórych kompilatorach C++ istnieje możliwość tymczasowego wyłączenia mechanizmu enkapsulacji i otwarcia w ten sposób dostępu do wszystkich pól i metody prywatnych.

- Dziedziczenie

To związek między klasami obiektów określający przekazywanie cech (definicji atrybutów, metod) z nadklasy do jej podklasy. Porządkuje i wspomaga polimorfizm i enkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności tylko tę, której nie ma obiekt ogólny.

W typowym przypadku powstają grupy obiektów zwane klasami oraz grupy klas zwane drzewami. Odzwierciedlają one wspolne cechy obiektów.

Istnieje wiele rodzajów dziedziczenia np.: dziedziczenie oparte na klasach, na prototypach, wielokrotne dziedziczenie.

- Abstrakcja

Każdy obiekt w systemie służy jako model abstrakcyjnego "wykonawcy", które może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez ujawniania, w jaki sposób zaimplementowano dane cechy. Ograniczenie zakresu cech manipulowanych obiektów wyłącznie do cech kluczowych dla algorytmu, a jednocześnie niezależnych od implementacji.

- Polimorfizm

Zdolność obiektów do różnych zachować w zależności od bieżącego kontekstu wykonania programu. Wywołane mogą być różne wersje tej samej funkcji. Pozwala to na rozszerzalność i łatwą modyfikację programu.

Zastosowania programowania obiektowego:

- Inżyniera oprogramowania - w języku modelowania UML do analizy oraz modelowania rzeczywistości. Podejście obiektowe ułatwia zrozumienie przez człowieka złożonych bytów informatycznych.
- Języki programowania - na programowaniu obiektowym bazują całe języki programowania takie jak np. język JAVA
- Komunikacja z relacyjnymi bazami danych - ORM (Object Rational Mapping) - sposób odwzorowania obiektów bazodanowych o relacyjnym charakterze poprzez obiektową architekturę systemu informatycznego,
- Biblioteki - programowanie obiektowe ułatwia pisanie reużywalnych bibliotek programistycznych, z których programista ma możliwość korzystania poprzez wystawiony interfejs bez wnikania w szczegóły implementacyjne,
- Komunikacja sieciowa - podejście obiektowe ułatwia komunikację sieciową między programami napisanymi w różnych językach programowania. Przykładem jest ujednolicony format przesyłania danych JSON oparty na podejściu obiektowym.
- Graficzne środowiska tworzenia oprogramowania - RAD (Rapid Application Development) - szybkie tworzenie aplikacji. Jest to ideologia i technologia polegająca na udostępnianiu programiście zestawu gotowych komponentów (obiektów), z których ten ma możliwość tworzenia złożonych programów.
- Współdziałanie systemów heterogenicznych - Przykładem zastosowania jest technologia **CORBA** (Common Object Request Broker Architecture) zapewniająca komunikację między obiektami pracującymi w heterogenicznych (różnorodnych) systemach komputerowych. Obiekty pełniące dowolne funkcje mogą być zaimplementowane w róznych językach programowania, na dowolnej platformie sprzętowej pod kontrolą różnych systemów operacyjnych.

  Opis obiektów, a właściwie ich interfejsów znajduję się w specjalnym pliku IDL (Interface Definition Language), które jest kompilowany na kod zajmujący się kontrolą komunikacji w systemie.

  Obiekty mają swoje adresy IOR (Interpretable Object Reference) będące kilkusetznakowymi adresami kodującymi wiele informacji m.in. o adresie komputera, programu, nr obiektu, informacje o kolejności zapisu bajów itd.

Podział programowania obiektowego:

- oparte na klasach - definiowane są klasy, czyli typy zmiennych, a następnie tworzone są obiekty, czyli instancje tych typów
- oparte na prototypach - W tym podejściu nie istnieje pojęcie klasy. Nowe obiekty tworzy się w oparciu o istniejący już obiekt - prototyp, po którym dziedziczone są pola i metody. Dodatkowo można go rozszerzać o nowe. Typ ten spotykany w językach interpretowanych np. JavaScript.

2. Mechanizm dziedziczenia, ochrona pól - przykład implementacji

Dziedziczenie - to w programowaniu obiektowym mechanizm polegający na współdzieleniu funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje ("dziedziczy") także te pochodzące z klasy, po której dziedziczy.

Klasa dziedzicząca nazywana jest **klasą pochodną / potomną.**

Klasa, z której następuje dziedziczenie **klasą bazową.**

Klasy pochodne obok swoich własnych pól i metod posiadają również kompletny interfejs klasy bazowej.

W językach programowania z prototypowaniem (np. JavaScript) nie występuje pojęcie klasy, dlatego dziedziczenie zachodzi tam pomiędzy poszczególnymi obiektami.

Każdy obiekt o typie klasy pochodnej jest jednocześnie obiektem o typie klasy bazowej. Innymi słowy w C++ jeżeli mamy dwie klasy Figura i Elipsa, przy czym Elipsa dziedziczy po klasie Figura to wszędzie tam, gdzie spodziewamy się Figury geometrycznej lub wskaźnika do Figury możemy wstawić Elipsę.

W C++ dziedziczenie jest realizowane jako relacja zawierania. Innymi słowy jeśli zadeklarujemy, że klasa B dziedziczy po klasie A, to wówczas wewnętrznie będzie to zaimplementowane w ten sposób, że obiekt klasy B będzie w sobie zawierał obiekt klasy A. Przy czym jest to wewnętrzny mechanizm języka C++.

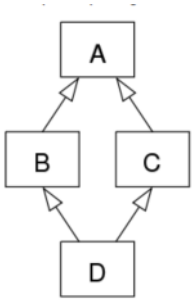
Kolejność tworzenia obiektów klas bazowych i pochodnych:

Generalnie podczas alokacji pamięci na obiekt klasy pochodnej najpierw alokowana jest pamięć na obiekt klasy bazowej, a dopiero później na obiekt klasy pochodnej. Dealokacja przebiega odwrotnie.

Wielodziedziczenie - nie wspominać, ale dobrze wiedzieć

Mechanizm umożliwiający klasie pochodnej dziedziczenie po więcej niż jednej klasie bazowej. Jest to cecha specyficzna danego języka np. w C++ wielodziedziczenie jest możliwe, a w Javie nie, aczkolwiek z Javą 8 wchodzi możliwość implementowania metod w interfejsach.

Z wielodziedziczeniem związany jest problem diamentu:



Klasa B i C dziedziczą po klasie A, czyli każda z nich ma swoją własną kopię klasy B. Jeśli klasa D dziedziczy zarówno po klasie B jak i C i na obiekcie klasy D spróbujemy wywołać metodę klasy A kompilator nie będzie wiedział, którą kopię metody wywołać (z tę z klasy B, czy z klasy C?).

Rodzaje dziedziczenia - też nie wspominać, ale dobrze wiedzieć

W języku C++ wyróżnia się trzy rodzaje dziedziczenie: prywatne, chronione i publiczne. Jednak dziedziczenie prywatne i chronione są stosowane bardzo rzadko i jest to cecha specyficzna języka C++ (np. w Javie dziedziczenie prywatne i chronione nie istnieje).

Ochrona pól:

Jest to mechanizm umożliwiający programiście określanie, do których pól i metod będzie można odwołać sie spoza danej klasy. Zazwyczaj jest to realizowane za pomocą tzw. modyfikatorów dostępu:

- private - do pól i metod objętych tym modyfikatorem można odwołać się tylko i wyłącznie z metod tej klasy oraz z funkcji i klas zaprzyjaźnionych z tą klasą,
- protected - podobnie jak private, ale do pól i metod oznaczonych tym modyfikatorem mogą odwoływać się również klasy pochodne,
- public - pola i metody oznaczone tym modyfikatorem są dostępne w całym programie.

W teorii w programowaniu obiektowym w ogóle nie ma mowy o czymś takim jak pola publiczne (publiczne mogą być tylko metody), jednak większość języków (C++, Java) umożliwia programiście deklarację pól publicznych.

Przykład ochrony pól:

```
struct B { // default access modifier inside struct is public
    void set_n(int v) { n = v; }
    void f()          { cout << "B::f" << endl; }
  protected:
    int m, n; // B::m, B::n are protected
  private:
    int x;
};

struct D : B {
    using B::m;               // D::m is public
    int get_n() { return n; } // B::n is accessible here, but not outside
//  int get_x() { return x; } // ERROR, B::x is inaccessible here
 private:
```

```
    using B::f;              // D::f is private
};

int main() {
    D d;

//  d.x = 2; // ERROR, private
//  d.n = 2; // ERROR, protected
    d.m = 2; // protected B::m is accessible as D::m

    d.set_n(2); // calls B::set_n(int)
    cout << d.get_n() << endl; // output: 2

//  d.f();   // ERROR, B::f is inaccessible as D::f


    B& b = d; // b references d and "views" it as being type B

//  b.x = 3; // ERROR, private
//  b.n = 3; // ERROR, protected
//  b.m = 3; // ERROR, B::m is protected

    b.set_n(3); // calls B::set_n(int)
//  cout << b.get_n(); // ERROR, 'struct B' has no member named 'get_n'

    b.f();   // calls B::f()
    return 0;
}
```

Dziedziczenie, a zmiany modyfikatorów dostępu.

Klasa pochodna dziedziczy wszystkie pola i modyfikatory klasy bazowej niezależnie od ich modyfikatorów dostępu, jednak:

- zmienne i pola zadeklarowane w klasie bazowej jako **publiczne** będą publiczne w klasie pochodnej,
- zmienne i pola zadeklarowane w klasie bazowej jako **chronione** będą chronione w klasie pochodnej i klasa pochodna będzie miała do nich dostęp,
- zmienne i pola zadeklarowane w klasie bazowej jako **prywatne** zostaną odziedziczone (tak samo jak wszystkie inne), jednak nie będzie do nich bezpośredniego dostępu. Można się do nich dostać tylko przez metody dostępowe odziedziczone z klasy bazowej.

Uwaga Wnuka:

Przyjmijmy, że mamy następujące dwie klasy:

```
class A {
private:
  int a,b;
}
```

```
class B : public A {
 protected:
   int a;
}
```

W klasie bazowej i klasie pochodnej mamy zadeklarowaną zmienną **a** o tej samej nazwie, ale o innych modyfikatorach dostępu. Haczyk polega na tym, że ktoś może widząc takie coś pomyśleć, że umieszczona w klasie B deklaracja zmiennej chronionej **a** zmienia modyfikator dostępu zmiennej a odziedziczonej po klasie A. Nic takiego nie ma miejsca.

Dziedziczenie jest realizowane jako zawieranie, w związku z tym w klasie B znajdą się wszystkie zmienne klasy bazowej A wraz z modyfikatorami dostępu zgodnymi z zzasadami dziedziczenia modyfikatorów oraz dodatkowo w klasie B znajdzie się drugie pole o nazwie **a**, ale o modyfikatorze protected. Będziemy więc mieli jedno pole **a** typu int odziedziczone po klasie bazowej, do którego nie będziemy mieli bezpośredniego dostępu oraz drugie pole **a** o modyfikatorze dostępu protected.

Klasy zaprzyjaźnione (dodatek dobrze wiedzieć, że coś takiego istnieje)

W tym wypadku obiekt klasy B może wywołac prywatną metodę klasy A.

```
class B
{
    // B declares A as a friend...
    friend class A;

private:
    void privatePrint()
    {
        std::cout << "hello, world" << std::endl;
    }
};

class A
{
public:
    A()
    {
        B b;
        // ... and A now has access to B's private members
        b.privatePrint();
    }
};

int main()
{
    A a;
    return 0;
```

3. Polimorfizm - sposób działania, opisać na przykładzie

Polimorfizm jest pojęciem ściśle związanym z programowaniem obiektowym oraz dziedziczeniem. Jest to mechanizm pozwalający na definiowanie jednej metody w wielu postaciach oraz uzależnianie jej działania od typu obiektu, dla którego jest wywoływana.

Polimorfizm możemy określić jako wirtualizację operacji. Jest to możliwość dynamicznego (późnego, realizowanego w fazie wykonania) wiązania nazwy operacji do wielu implementacji (metod) tej operacji w różnych klasach pozostających w relacji dziedziczenia. Wiązaniu towarzyszy mechanizm wyboru konkretnej implementacji. Wybór implementacji zależy od nazwy metody oraz od typu dynamicznego tego obiektu, dla którego została wywołana operacja, a nie od typu zmiennej, wskazującej ten obiekt.

Co nam daje?
Dzięki niemu mamy pełną kontrolę nad wykonywanym programem, nie tylko w momencie kompilacji (**wiązanie statyczne**) ale także podczas działania programu (**wiązanie dynamiczne**) – niezależnie od różnych wyborów użytkownika.

Obiekt klasy pochodnej może być wskazywany przez wskaźnik typu klasy bazowej.
Typem statycznym obiektu wskazywanego przez wskaźnik jest typ tego wskaźnik, a **typem dynamicznym** obiektu wskazywanego przez wskaźnik jest typ na jaki dany wskaźnik wskazuje.

Mechanizm polimorfizmu jest możliwy dzięki metodom wirtualnym.

Metoda wirtualna jest to funkcja składowa klasy poprzedzona słowem kluczowym **virtual**, której sposób wywołania zależy od typu dynamicznego wskaźnika, a nie od typu statycznego.

Metoda wirtualna to taka metoda, której ciało może zostać przykryte w klasach dziedziczących po klasie bazowej. W ten sposób możemy zdefiniować zupełnie inne zachowania klas dziedziczących po klasie bazowej.

Specyficznym przykładem metody wirtualnej jest metoda czysto wirtualna, która w ogóle nie posiada ciała. Klasa posiadająca taką metodę staje się klasą abstrakcyjną. Oznacza to tyle, iż nie jest możliwe stworzenie obiektu tej klasy. Klasa taka służy jedynie temu, aby zdefiniować pewnego rodzaju interfejs i jest przeznaczona jedynie po to, by po niej dziedziczyć.

W przykładzie poniżej o ile mogą istnieć konkretne figury będące kwadratami, kołami itp. to nie powinien istnieć żaden obiekt klasy Figura ponieważ jest to jedynie abstrakcyjny byt. Natomiast dziedziczenie po tej klasie i jej rozszerzanie powoduje, że stworzymy już konkretną figurę geometryczną. Metodę czysto wirtualną w języku C++ deklaruje się tak:

```
class Figura
{
  public:
    virtual float pole() = 0;
};
```

Taka deklaracja metody wirtualnej uniemożliwia stworzenie jakiegokolwiek obiektu klasy *Figura* oraz zmusza do określenia metody *float pole()* na jednym z poziomów z dziedziczenia. Nie jest możliwe pominięcie takiej implementacji.

Metody wirtualne:

- Definiowane przy wykorzystaniu słowa kluczowego **virtual** (przy implementacji słowa **virtual** nie używamy),
- Przy wywołaniu metody wirtualnej zostanie przeszukana tablica funkcji wirtualny przynależąca do danego obiektu i wyszukana wersja najbliższa w hierarchii dziedziczenia,
- Metody wirtualne w danej hierarchii powinny mieć pełną zgodność. Nie może się różnić liczba parametrów metody. Wartość zwracana może się różnić, ale tylko jeśli w klasie pochodnej metoda zwraca klasę dziedziczącą po typie wartości zwracanej przez metodę klasy bazowej.
- Funkcja zaczyna zaczynać się jak wirtualna w momencie pierwszego pojawienia się słowa **virtual**,
- Zachowanie wirtualne może (ale nie musi) skończyć się po pierwszym wsytąpieniu funkcji bez **virtual** w hierarchii dziedziczenia,
- Rodzaj dziedziczenia nie wpływa na zachowanie sie funkcji wirtualnych (zmienia się jedynie ich widoczność),
- Metody statycznie nie mogą być wirtualne,
- W przypadku argumentów domyślnych wykorzystana będzie wartość odpowiadająca wersji funkcji według typu wskaźnika, a nie dynamicznego typu (związane z określaniem wartości domyślnych na etapie kompilacji),

- Konstruktor nie może być wirtualny,
- Destruktor powinien być wirtualny ze względu na niebezpieczeństwo wycieków pamięci jeśli w klasie została zadeklarowana jakakolwiek metoda wirtualna. Deklaracja destruktora powinna znajdować się w klasie bazowej.

Polimorfizm uzyskujemy dzięki:

- Niejawnemu przekształceniu typów wskaźników do klas pochodnych na wskaźniki do klas typu podstawowego,
- Mechanizmowi funkcji wirtualnych,
- Operacji rzutowania *dynamic_cast* oraz operatorom *typeid*

Polimorfizm kosztuje - Klasy **polimorficzne** zajmują więcej miejsca w pamięci, ponieważ kompilator automatycznie dodaje do nich wskaźnik *vptr* wskazujący na tablicę *vtab*. Dla każdej klasy musi istnieć osobny wskaźnik i osobna tablica. Tablica jest generowana automatycznie i zawiera wskaźniki do funkcji, wygenerowane przez kompilator.

W języku JAVA wszystkie metody są wirtualne.

Przykład polimorfizmu C++

Kod

```
#include <QCoreApplication>
#include <iostream>
using namespace std;


class A {
public:
 virtual void f1() { cout << "A f1\n"; }
 void f2() { cout << "A f2\n"; }
};
class B : public A {
public:
 void f1() { cout << "B f1\n"; }
 virtual void f2() { cout << "B f2\n"; }
};
class C : public B {
public:
 void f1() { cout << "C f1\n"; }
 virtual void f2() { cout << "C f2\n"; }
};
```

```
int main(int argc, char *argv[])
{
    QCoreApplication g(argc, argv);
    A a; B b; C c;
     A *pAb, *pAc;
     B *pBc;
     pAb = &b;
     pAc = &c;
     pBc = &c;
     cout << "Klasa A:\n";
     a.f1();
     a.f2();
     cout << "Klasa B:\n";
     b.f1();
     b.f2();
     cout << "Klasa B jako A:\n";
     pAb->f1();
     pAb->f2();
     cout << "Klasa C:\n";
     c.f1();
     c.f2();
     cout << "Klasa C jako A:\n";
     pAc->f1();
     pAc->f2();
     cout << "Klasa C jako B:\n";
     pBc->f1();
     pBc->f2();
    return g.exec();
}
```

Rezultat

```
Klasa A:
A f1
A f2
Klasa B:
B f1
B f2
Klasa B jako A:
B f1
A f2
Klasa C:
C f1
C f2
Klasa C jako A:
C f1
A f2
Klasa C jako B:
C f1
C f2
```

Klasa B jako A:

- metoda *f1()* zostanie związana dynamicznie z typem na, który wskaźnik pAb będzie wskazywał (typ klasy B).
- metoda *f2()* nie jest zadeklarowana jako wirtualna zatem zostanie związana statycznie z typem wskaźnika w fazie kompilacji - stąd wywołanie metody *f2()* na wskaźniku pAb wywołuje metodę z klasy A

Klasa C jako A taka sama sytuacja.

Klasa C jako B:

- metoda *f1()* była zadeklarowana jako wirtualna w klasie A, zatem zostanie związana dynamiczna i wywołana implementacja z klasy C,
- metoda *f2()* została zadeklarowana jako wirtualna w klasie B, więc również zostanie związana dynamicznie i wywołana implementacja z klasy C

Jawne wywołanie funkcji bazowej jest możliwe poprzez poprzedzenie metody dwukropkiem i nazwą klasy.

4. Pojęcie zbioru rozmytego, definicja i interpretacja funkcji przynależności.

Zbiorem rozmytym A w pewnej niepustej przestrzeni **X**, co zapisujemy $A \subseteq X$, nazywamy zbiór par:

$$A = \{(x, \mu_A(x)); x \in X\}$$

gdzie:

$$\mu_A: X \rightarrow [0,1]$$

$\mu_A(x)$ jest **funkcją przynależności** zbioru rozmytego A. Funkcja ta każdemu elementowi $x \in X$ przyporządkowuje jego stopień przynależności do zbioru rozmytego A, przy czym można wyróżnić 3 przypadki:

- $\mu_A(x) = 1$ - oznacza to pełną przynależność do zbioru rozmytego A, tzn. $x \in A$,
- $\mu_A(x) = 0$ - oznacza to brak przynależności elementu x do zbioru rozmytego A, tzn. $x \notin A$
- $0 < \mu_A(x) < 1$ - oznacza to częściową przynależnośc elementu x do zbioru rozmytego A.

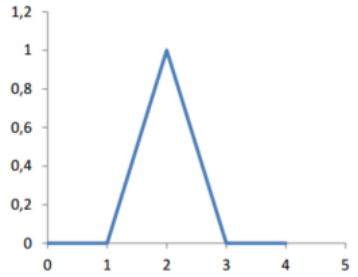
Funkcja przynależności jest uogólnieniem funkcji charakterystycznej określonym na zbiorach rozmytych.

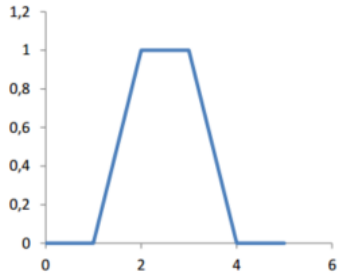
Funkcja charakterystyczna zbioru Z to funkcja φ(x), która elementom przestrzeni X przyporządkowuje wartości 0 i 1:

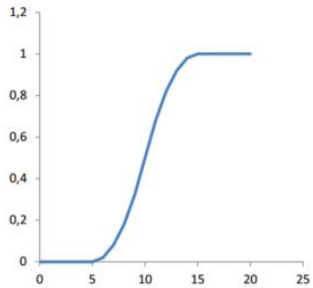
$$\varphi(x) = \begin{cases} 0 \; dla \; x \notin Z \\ 1 \; dla \; x \in Z \end{cases}$$

Funkcja charakterystyczna określa, czy dany element x przestrzeni X należy, czy nie należy do zbioru Z, a funkcja przynależności określa dodatkowo stopień przynależności do tego zbioru.

W teorii zbiorów rozmytych istnieją różne funkcje przynależności. Najczęściej stosowane są funkcje trójkątne, trapezowe, singleton, s-funkcje oraz funkcje Gaussa (dwie ostatnie ze względu na to, że są różniczkowalne).



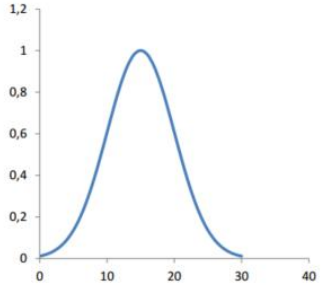




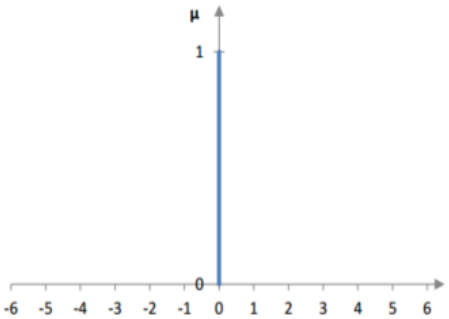
$$f(x) = \begin{cases} 0 \; dla \; x < A \\ 2\left(\frac{x-A}{C-A}\right)^2 \; dla \; A < x \leq B \\ 1 - 2\left(\frac{C-x}{C-A}\right)^2 \; dla \; B < x \leq C \\ 1 \; dla \; x > C \end{cases}$$

gdzie:

- A – początek s-funkcji
- B – wartość argumentu dla którego wartość funkcji wynosi 0,5
- C – koniec s-funkcji



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$



5. Elementy składowe wnioskowania rozmytego.

Wnioskowanie rozmyte, podobnie jak wnioskowanie w logice klasycznej polega na ocenie prawdziwości zdania logicznego (**wniosku** lub inaczej **konkluzji)** na podstawie prawdziwości innych zdań logicznych (**przesłanek)**. Różnica polega na tym, że w logice klasycznej zdanie może być prawdziwe lub fałszywe, a w logice rozmytej mówimy o stopniu prawdziwości zdania. Zdanie dotyczy zazwyczaj przynależności danej liczbowej do określonego zbioru.

Wnioskowanie rozmyte odbywa się według poniższego schematu:

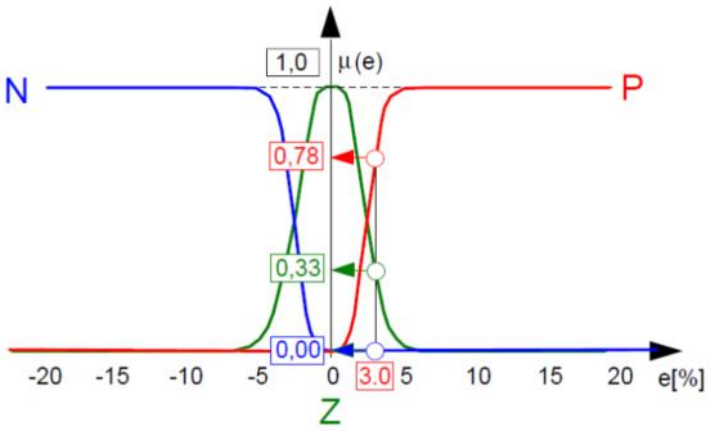
Rys. 1 Ogólny schemat procesu wnioskowania rozmytego

Blok rozmywania

Rozmywaniem nazywamy przekształcenie wartości ostrych wielkości wejściowej (np. modelu) do wartości rozmytych. Każdej wartości ostrej w określonej przestrzeni rozważań (czyli określonym zbiorze) zostają przyporządkowane zbiory wartości funkcji przynależności do określonych zbiorów rozmytych. **Zbiorem rozmytym** nazywamy zbiór uporządkowanych par - wartość ostra x, wartość funkcji przynależności μ(x)**.** Funkcja przynależności określa stopień przynależności elementu do danego zbioru.

W ramach rozmywania w pierwszej kolejności należy nazwać wejścia. Następnie należy zaprojektować zbiory rozmyte. Odbywa się to poprzez przyporządkowanie wartości określonym w poprzednim kroku zmiennym słownym (lingwistycznym). Końcową fazą rozmywania wejścia jest określenie zbioru wartości funkcji przynależności dla każdej wartości nazwy lingwistycznej na podstawie wartości ostrej wejścia oraz stworzonych zbiorów rozmytych.

Przykład - rozmywanie odchyłki regulacji



Rys. 8 Przykład operacji rozmywania odchyłki regulacji *e*.
- μ*N(e)* zbioru rozmytego *N ("ujemna odchyłka regulacji"),*
- μ*Z(e)* zbioru rozmytego *Z ("zerowa odchyłka regulacji"),*
- μ*P(e)* zbioru rozmytego *N ("dodatnia odchyłka regulacji").*

Blok wnioskowania

Wnioskowaniem nazywamy wyznaczanie wyjścia o charakterze rozmytym na podstawie nieostrego wejścia oraz bazy reguł między wejściem, a wyjściem.

Baza reguł jest zbiorem implikacji, mówiących o przynależności zmiennej wyjściowej do danego zbioru rozmytego w zależności od przynależności zmiennych wejściowych do odpowiednich zbiorów rozmytych. Reguły mają postać IF ... THEN ...

Mówimy, że baza reguł jest zupełna jeśli definiuje wszystkie relacje wejście-wyjście. Liczba reguł zależy od liczby wejść, wartości lingwistycznych im przypisanych i liczby wyjść. Przesłanki (reguły) mogą być złożone. Wówczas poszczególne elementy przesłanki są łączone spójnikami AND lub OR, ponadto używa się w zapisie reguł operatora zaprzeczenia NOT. W anaalogiczny sposób następnik może być również złożony.

Oprócz bazy reguł do przeprowadzenia procesu wnioskowania niezbędne są **funkcje przynależności zmiennych wyjściowych** do ich zbiorów rozmytych.

Wnioskowanie może być oparte na pojedynczej regule bądź ich złożeniu.

Wnioskowanie na pojedynczej regule odbywa się zgodnie z implikacją **Mamdaniego**. Stopień spełnienia następnika reguły nie może przekroczyć stopnia spełnienia przesłanki. Stopień spełnienia przesłanki złożonej jest wyznaczany zgodnie z zasadą minimum (dla AND) lub maksimum (dla OR). Zasada minimum/maksimum mówi o tym, że stopień spełnienia w tym wypadku przesłanki jest równy minimalnej/maksymalnej wartości funkcji przynależności jednej ze składowych przesłanki. Dla operatora NOT stopień spełnienia przesłanki w postaci NOT(x = A) jest równy $(1-\mu_A)$.

Stopień w jakim spełniony jest wniosek możemy nazwać **poziomem zapłonu reguły**.

W trakcie wnioskowania na podstawie jednej reguły ważne jest odnalezienie reguły, która została spełniona w największym stopniu. Wnioskowanie odbywające się w ten sposób jest proste i nie wymaga dużych nakładów obliczeniowych.

Możliwe jest także wnioskowanie skalowane, polegające na wyznaczeniu stopnia spełnienia przesłanki zgodnie z wyżej podanymi zasadami i przeskalowaniu zbioru rozmytego następnika o wyznaczony poziom (mnożenie każdej wartości funkcji przynależności wniosku przez stopień spełnienia przesłanki).

Z kolei wnioskowanie na zbiorze reguł odbywa się na zasadzie utworzenia sumy mnogościowej wniosków cząstkowych, które są wyznaczane na podstawie pojedynczych reguł.

Reguła w postaci zdania warunkowego

$$If(e = Ne)then(dCV = NdCV)$$

Wnioskowanie Mamdani[1]

$$If(\mu(e) = 0,7)then(\mu(dCV) = 0,7)$$

- Zasada minimum

$$If[(\mu(e) = 0,7)AND(\mu(de) = 0,1)]then(\mu(dCV) = 0,1)$$

- Zasada maksimum

$$If[(\mu(e) = 0,7)OR(\mu(de) = 0,1)]then(\mu(dCV) = 0,7)$$

- Dopełnienie

$$If[NOT(\mu(e) = 0,7)]then(\mu(dCV) = 0,3)$$

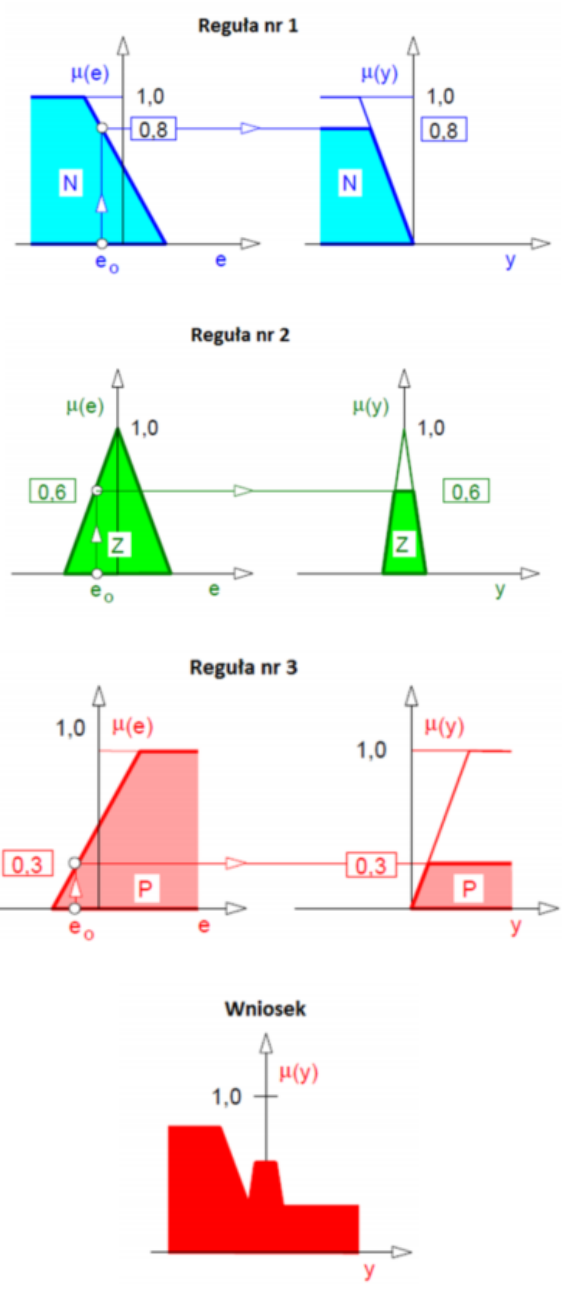
Gdzie:

- e - odchyłka regulacji,
- Ne - zbiór rozmyty "Ujemna odchyłka regulacji",
- de - zmiana odchyłki regulacji,
- dNe - zbiór rozmyty "Ujemna zmiana odchyłki regulacji",
- dCV - zmiana sterowania,
- $NdCV$ - zbiór rozmyty "Ujemna zmiana sterowania".

Blok wyostrzania

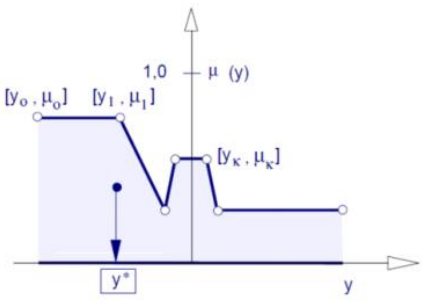
Wyostrzaniem nazywamy przekształcenie zbioru rozmytego do wartości ostrej wielkości wyjściowej.

Przykład dla jednego wejścia rozmytego i trzech funkcji przynależności:



Odwzorowanie wzoru rozmytego w wartość ostrą nie jest jednoznaczne, tzn. istnieje wiele metod wyostrzania:

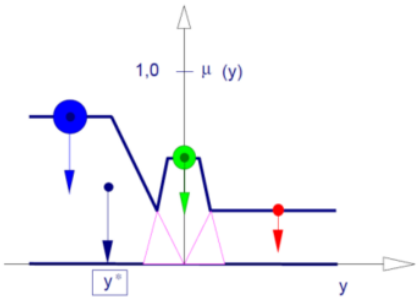
- **metody obszarowe**
  - metoda środka ciężkości (COG)



Polega na znalezieniu pierwszej współrzędnej środka ciężkości obszaru pod wykresem wniosku rozmytego.

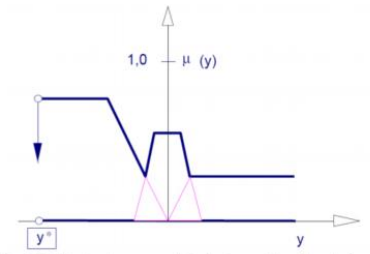
- metoda środka sum (COS)
- metoda środka największego obszaru (COLA)

- metody wysokościowe
  - metoda wysokości (HM)



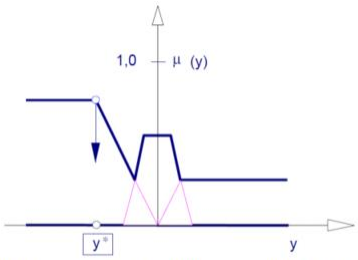
Wyjście jest średnią ważoną wysokości zbiorów rozmytych. Metoda prostrza od metod obszarowych.

  - metoda wysokości "pierwszy z największych" (FOM)



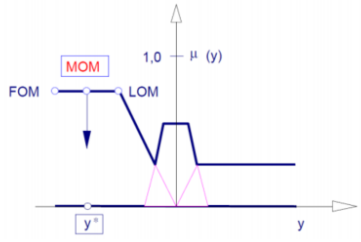
Rys. 15 Metoda polega na znalezieniu pierwszej współrzędnej punktu leżącego na lewym skraju najwyższego obszaru. Metoda jest bardzo szybka.

  - metoda wysokości "ostatni z największych" (LOM)



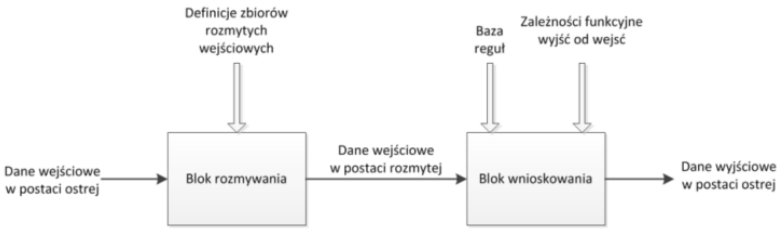
Rys. 16 Metoda polega na znalezieniu pierwszej współrzędnej punktu leżącego na prawym skraju najwyższego obszaru. Metoda jest bardzo szybka.

  - metoda wysokości "środkowy z największych" (MOM)



Rys. 17 Metoda polega na znalezieniu pierwszej współrzędnej punktu leżącego na środku najwyższego obszaru (średniej arytmetycznej współrzędnych skrajnych punktów obszaru). Metoda jest bardzo szybka

Model TSK (Takagi-Sugeno-Kang'a)



Rys. 18 Schemat wnioskowania rozmytego w modelu TSK.

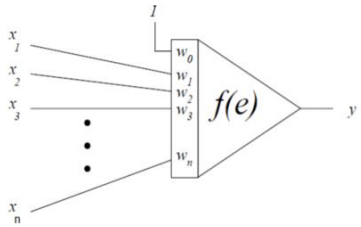
W tym modelu wniosek (reguła) jest w postaci ostrej. Nie ma zatem bloku wyostrzania. Reguły jednak też są w postaci IF...THEN...

Wnioskiem z pojedynczej reguły jest zależność funkcyjna zmiennej wyjściowej (z konkluzji) od zmiennych wejściowych (z przesłanek). Do przeprowadzenia procesu wnioskowania jest zatem niezbędna znajomość zależności funkcyjnych zmiennej wyjściowej od wejściowej. Wnioskiem ogólnym z kilku reguł jest średnia ważona konkluzji z pojedynczych reguł, gdziaktye wagami są poziomy aktywacji pojedynczych reguł.

6. Typy sieci neuronowych i ich zastosowanie.

Sztuczne sieci neuronowe to struktury, składające się z prostych jednostek obliczeniowych (sztucznych neuronów), przetwarzających dane, komunikujących się między sobą i pracujących równolegle. Powstały na gruncie wiedzy o działaniu systemu nerwowego istot żywych i stanowią próbę wykorzystania zjawisk zachodzących w systemach nerwowych przy poszukiwaniu nowych rozwiązań technologicznych.

Sztuczny neuron można rozpatrywać jako specyficzny przetwornik sygnałów.



Podstawowe elementy składowe neuronu:

- n wejść neuronu wraz z wagami w_i (wektor wag w oraz wektor sygnałów wejściowych x),
- pobudzenie **e** neuronu:

$$e = \sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w}^T \mathbf{x}$$

- funkcja aktywacji (przejścia),

$$y = f(e)$$

- jeden sygnał wyjściowy **y**,

Zatem formuła opisująca działanie neuronu:

$$y = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

Jak wynika z powyższego wzoru działanie neuronu jest bardzo proste. Sygnały wejściowe $x_1, x_2 \ldots x_n$ zostają ponożone przez odpowiednie wagi $w_1, w_2 \ldots w_n$, a następnie otrzymane wartości są sumowane. W wyniku tych operacji powstaje sygnał poddawany działaniu funkcji aktywacji **f**. O właściwościach neuronu decydują wagi oraz rodzaj i parametry funkcji aktywacji. Wagi są dobierane w procesie uczenia sieci.

Podstawowe typy funkcji aktywacji:

- liniowa $y = k \cdot e$
- nieliniowe - ciągłe, nieciągłe, unipolarne i bipolarne,
- funkcja skoku jednostkowego, progowa (McCulloch i Pitts), $f(e) = \begin{cases} 1 \text{ dla } e \geq \Theta \\ 0 \text{ dla } e < \Theta \end{cases}$
- funkcja sigmoidalna: $f(e) = \frac{1}{1+\exp(-\beta e)}$
- funkcja tangens hiperboliczny

Uczenie sieci

- Z nauczycielem

Uczenie z nauczycielem polaga na tym, że sieci podaje się przykłady poprawnego działania, które powinna ona porem naśladować w swoim bieżącym działaniu. Przykład należy rozumieć w ten sposób, że nauczyciel podaje konkretne sygnały wejściowe i wyjściowe, pokazując jaka jest wymagana odpowiedź sieci dla pewnej konfiguracji danych wejściowych. Mamy do czynienia z parą wartości - przykładowym sygnałem wejściowym i pożądanym (oczekiwanym) wyjściem, czyli wymaganą odpowiedzią sieci na ten sygnał wejściowy. Zbiór przykładów zgromadzonych w celu ich wykorzystaniu w procesie uczenia sieci nazywa się zwykle ciągiem uczącym. Zatem w typowym procesie uczenia sieć otrzymuje od nauczyciela ciąg uczący i na jego podstawie uczy się prawidłowego działania, stosując jedną z wielu znanych dziś strategii uczenia.

- Bez nauczyciela

Obok opisanego wyżej schematu uczenia z nauczycielem występuje też szereg metod tak zwanego uczenia bez nauczyciela (albo samouczenia sieci). Metody te polegają na podawaniu na wejście sieci wyłącznie szeregu przykładowych danych wejściowych, bez podawania jakiejkolwiek informacji dotyczącej pożądanych czy chociażby tylko oczekiwanych sygnałów wyjściowych. Odpowiednio zaprojektowana sieć neuronowa potrafi wykorzystać same tylko obserwacje wejściowych sygnałów i zbudować na ich podstawie sensowny algorytm swojego działania - najczęściej polegający na tym, że automatycznie wykrywane są klasy powtarzających się sygnałów wejściowych i sieć uczy się (spontanicznie, bez jawnego nauczania) rozpoznawać te typowe wzorce sygnałów.

Można wyróżnić kilka rodzajów sieci neuronowych:

- sieci jednokierunkowe,
- sieci rekurencyjne (Hopfielda),
- sieci samoorganizujące się (Kohena),
- sieci radialne (RBF).

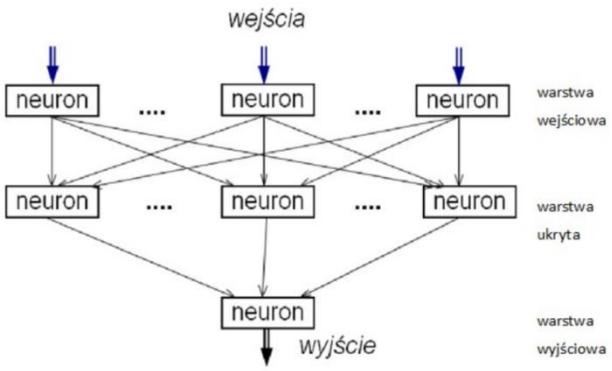
1. Sieci jednokierunkowe

Przepływ sygnałów odbywa się w jednym kierunku - od wejścia do wyjścia. Zwykle neurony ułożone są w warstwach, a powiązania dotyczą tylko neuronów w sąsiednich warstwach.

Wyróżnia się w niej następujące warstwy:

- warstwę wejściową,
- warstwy ukryte,
- warstwę wyjściową.

Perceptron wielowarstwowy



W warstwie wejściowej następuje normalizacja sygnałów, a w warstwie wyjściowej wyliczana jest wartość sygnału wyjściowego z sieci (neuron sigmoidalny lub o liniowej funkcji aktywacji). Warstwy ukryte tworzą najczęściej neurony o sigmoidalnej funkcji aktywacji. Każdy neuron kolejnej warstwy otrzymuje sygnały wyjściowe wszystkich neuronów warstwy wcześniejszej i przekazuje swój sygnał do wszystkich neuronów kolejnej warstwy. W obrębie tej samej warstwy neurony nie mogą się łączyć ze sobą. Ilość warstw ukrytych i ilość neuronów w warstwie ukrytej zależy od charakteru zadania. Do modelowania procesów przemysłowych zazwyczaj stosuje się perceptrony z jedna warstwą ukrytą.

Największym zainteresowaniem cieszy się sieć jednokierunkowa, wielowarstwowa o neuronach typu sigmoidalnego, zwana perceptronem wielowarstwowym.

Uczenie sieci polega na modyfikacji wag. Podczas uczenia dąży się do minimalizacji odpowiednio zdefiniowanej funkcji celu, której argumentem jest wektor szukanych wag wejść neuronów.

Do uczenia sieci najczęściej stosuje się metodę propagacji wstecznej. Algorytm propagacji wstecznej (nazwa tego algorytmu wynika z kolejności sygnałów obliczania sygnałów błędu, która przebiega w kierunku odwrotnym niż przechodzenie sygnałów przez sieć, to znaczy od warstwy wyjściowej poprzez warstwy ukryte w kierunku warstwy wejściowej) określa strategię doboru wag przy wykorzystaniu gradientowych metod optymalizacji. Podstawę algorytmu stanowi funkcja celu, definiowana jako suma kwadratów różnić między aktualnymi wartościami sygnałów wyjściowych, a wartościami zadanymi.

Najskuteczniejsze są gradientowe metody uczenia sieci, których podstawą działania jest znajomość gradientu funkcji celu. Opierają się na rozwinięciu w szereg Taylora funkcji celu w najbliższym sąsiedztwie znanego aktualnie rozwiązania.

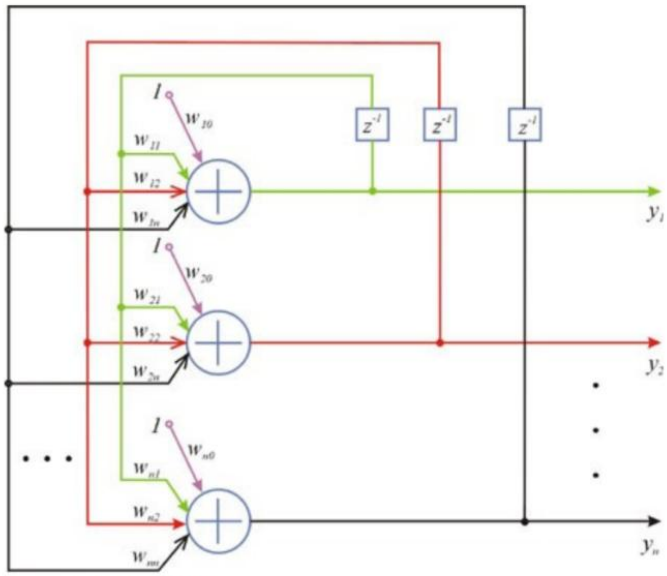
Zastosowania sieci jednokierunkowych:

- budowanie prostych układów logicznych (np. XOR),
- aproksymacja dowolnej nieliniowej, ale statycznej funkcji,
- aproksymacja zależności wielkości wyjściowych od wejściowych, gdy zależność ta jest trudna do opisu matematycznego na podstawie zależności fizycznych, ale mamy doświadczalne wartości wejść i wyjść do nauki sieci (diagnostyka procesów, walidacja czujników)

2. Sieci rekurencyjne (Hopfielda)

Różnią się od sieci jednokierunkowych występowaniem sprzężenia zwrotnego między warstwami wyjściowymi lub ukrytymi i wejściowymi. Podstawową cechą wyróżniającą te sieci są zależności dynamiczne na każdym etapie działania. Zmiana stanu jednego neuronu przenosi się w skutek masowego sprzężenia zwrotnego na całą sieć, wywołując stan przejściowy, kończący się określonym stanem ustalonym, ogólnie innym niż poprzedni.

Jest to sieć jednowarstwowa o regularnej budowie, składająca się z wielu neuronów połączonych każdy z każdym. Wagi sieci są symetryczne tzn. $w_{ij} = w_{ji}$. Neurony mają funkcję aktywacji typu signum, przyjmującą wartości ze zbioru {-1,1). Podczas uczenia sieć Hopfielda modyfikuje swoje wagi w zależności od wektora uczącego. W trybie odtworzeniowym wagi nie ulegają modyfikacjom, natomiast sygnał wejściowy pobudza sieć, która poprzez sprzężenie zwrotne wielokrotnie przyjmuje na swoje wejście sygnał wyjściowy, aż do ustabilizowania się odpowiedzi.



Rys. 7. Sieć Hopfielda

Uczenie sieci rekurencyjne odbywa się według **reguły Hebba**, zgodnie z którą wagi są modyfikowane według zależności:

$$w_{kj}=\frac{1}{N}\sum_{i=1}^{M} x_k^i x_j^i$$

gdzie: **N** - ilość wyjść, **M** - ilość wektorów uczących, **i** - nr wektora uczącego $x^i = [x_1^i, \dots, x_n^i]$

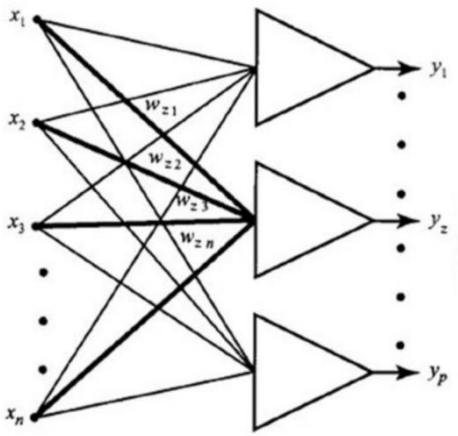
Zastosowanie:

- jako pamięć asocjacyjna, tzn. ma za zadanie odtworzenie pewnego wzorca na podstawie zaszumionego sygnału wejściowego. Stąd jest stosowana np. do klasyfikacji i rozpoznawania obrazów.
- optymalizacja przy użyciu specjalizowanej struktury obwodu (sieć realizująca programowanie liniowe i kwadratowe, sieć rozwiązująca problem komiwojażera, podział grafu na 2 części - zapewnia minimalną liczbę łączeń między częściami),
- przetwarzanie sygnałów - przetworniki A/C. transformacja Fouriera, przetwarzani i dekompozycja sygnałów

3. Sieci samoorganizujące się (Kohena)

Sieć, której uczenie odbywa się "bez nauczyciela". Ciąg uczący składa się jedynie z wartości wejściowych.

Sygnał wejściowy trafia na wejście wszystkich neuronów. Tutaj następuje wyznaczenia miary podobieństwa (odległości w sensie dobranej metryki) sygnału wejściowego **x** od wszystkich wektorów wag. Następnie wyłania się zwycięzcę, czyli neuron o wagach o najmniejszej odległości od wektora wejściowego i modyfikuje się jego wagi oraz wagi jego sąsiadów. Wielkość korekty zależy od odległości wektora wejściowego od wektora wag zwycięskiego neuronu. Ten algorytm uczenia nazywa się *Winner Takes All*. Jest to przykład uczenia bez nauczyciela.

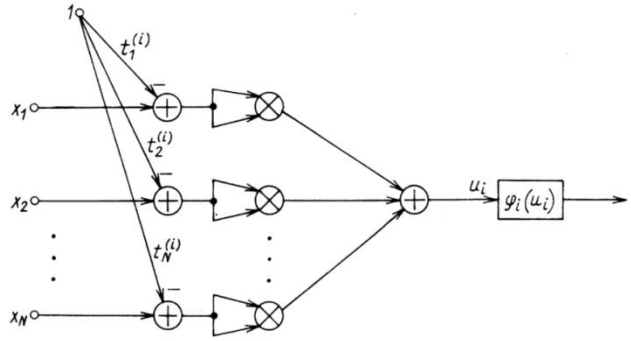


Zastosowanie:

- grupowanie danych,
- kompresja danych - odwzorowanie topologii danych wejściowych poprzez dużo mniejszą liczbę wag neuronów tworzących sieć,
- przetwarzanie mowy
- prognozowanie (np. obciążeń systemu elektroenergetycznego),

4. Sieci radialne

Zawierają neurony o radialnych funkcjach aktywacji.



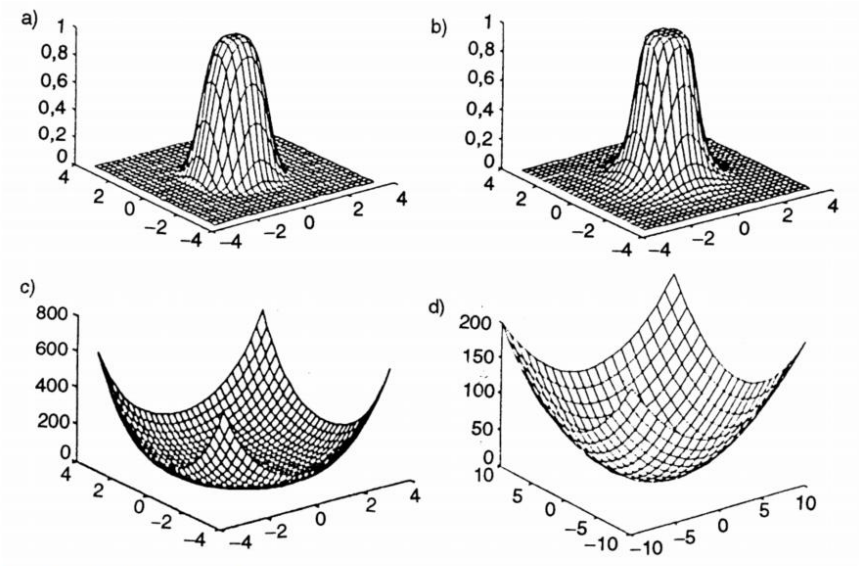
Rys. 4. Neuron radialny

Funkcje aktywacji realizują odwzorowanie:

$$x \to \varphi(\|x - t\|)$$

gdzie: $\|x - t\|$ - to norma euklidesowa.

Funkcja φ() jest nazywana radialną funkcją bazową. Jej wartość zmienia się radialnie wokół środka t. Oznacza to, że argument funkcji aktywacji jest różnica między wektorem wyznaczonym przez wartości sygnałów wejściowych i wektorem współczynników t.



Rys. 5. Radialne funkcje aktywacji

W porównaniu do sieci perceptronowych, sieci radialne zawierają znacznie więcej neuronow. W realizacji programowej działają wolniej niż sieci perceptronowe. Stosują inny sposób przetwarzania danych, przez co następuje skrócenie procesu uczenia się. Mają z góry ustalona architekturę składającą się z trzech warstw: wejściowej, ukrytej i wyjściowej. Nauron warstwy wyjściowej realizuje operację ważone sumy sygnałów. W warstwie ukrytej można stosować różne funkcje bazowe.

Uczenie składa się z dwóch kroków:

- Dobór położenia oraz kształtu funkcji bazowych wybranymi metodami (dobór losowy, samoorganizacja, metoda wstecznej propagacji błędu).
- Dobór wag neuronu wyjściowego.

Zastosowanie:

Uczenie się tych sieci bywa bardzo wolne, ale działanie po procesie uczenia bardzo szybkie. Wykorzystywane są przykładowo do:

- aproksymacja funkcji nieliniowych (jak w jednokierunkowych),
- przewidywanie np. przewidywanie trendów ekonomicznych na podstawie danych z długiego okresu czasu,
- monitoring np. monitoring dźwięków wydawanych przez silniki samolotów i pociągów,
- kontrola procesów chemicznych,
- kompresja danych,.

7. Cechy charakterystyczne i zastosowania systemów czasu rzeczywistego

System czasu rzeczywistego (ang. real-time system) to urządzenie techniczne, którego wynik i efekt działania jest zależny od chwili wypracowania tego wyniku. Istnieje wiele różnych definicji naukowych takiego systemu. Ich wspólną cechą jest zwrócenie uwagi na równoległość w czasie zmian w środowisku oraz obliczeń realizowanych na podstawie stanu środowiska. Z tego wyściugu dwóch stanów: zewnętrznego i wewnętrznego wynikają kryteria ograniczające czas wypracowania wyniku.

Systemy czasu rzeczywistego najczęściej są tworzone w oparciu o komputery, ale nie jest to konieczne. Można tym pojęciem określić np. pneumatyczny regulator.

System operacyjny czasu rzeczywistego (ang. real-time operating system - RTOS) to komputerowy system operacyjny, który został opracowany tak, aby spełnić wymagania narzucone na czas wykonywania zadanych operacji. Systemy takie stosuje się jako elementy komputerowych systemów sterowania pracujących w reżimie czasu rzeczywistego.

Ogólnie można przyjąć założenie, że zadaniem systemu operacyjnego czasu rzeczywistego oraz oprogramowania pracującego pod jego kontrolą i całego sterownika komputerowego jest wypracowanie odpowiedzi (np. sygnałów sterujących kontrolowanym obiektem) na skutek wystąpienia pewnych zdarzeń (zmiana sygnałów z czuników sterownika). Biorąc to pod uwagę podstawowym wymogiem dla systemów operacyjnych czasu rzeczywistego jest określenie najgorszego najdłuższego czasu, po jakim urządzenie komputerowe wypracuje odpowiedź po wystepianiu zdarzenia. Ze względu na to kryterium systemy operacyjne czasu rzeczywistego dzielą się na:

- **Twarde** - takie, dla których znany jest najdłuższy czas odpowiedzi oraz wiadomo, ze nie zostanie on przekroczony,
- **Miękkie** - takie, które starają się odpowiedzieć najszybciej jak to możliwe, ale nie wiadomo jest jaki może być najgorszy czas odpowiedzi,

Najczęstsze schematy działania RTOS'ów:

- Wywołane zdarzeniami (Event-driven) - przełączają wykonywanie procesu, gdy nadejdzie proces o wyższym priorytecie - szeregowanie priorytetami. Głównie dla systemów miekkich.
- Dzielące czas (Time-sharing) - przełączające zadania w równych odstępach czasowych kontrolowanych przez zegar procesora i algorytm szeregowania,

W systemach dzielących czas kluczową kwestią jest opracowanie (wybranie) odpowiedniego algorytmu szeregowania (scheduling) oraz sposobu podziału czasu procesora. Należy określić, któremu z procesów należy przydzielić procesor oraz na jak długi czas, aby wszystkie wykonywane procesy spełniały zdefiniowane dla nich ograniczenia czasowe.

Algorytm szeregowania (scheduler - planista) - to algorytm rozwiązujący jedno z najważniejszych zagadnień informatyki - jak rozdzielić czas procesora i dostęp do innych zasobów między zadania o te zasoby konkurujące.

Najczęściej algorytm szeregowania jest implementowany jako część wielozadaniowego systemu operacyjnego. Oprócz systemów operacyjnych dotyczy w szczególności także serwerów bazodanowych.

Używane najczęściej algorytmy szregowania:

- **FIFO** - często stosowany algorytm, jeden z prostszych w realizacji. Daje dobre efekty w systemach ogólnego przeznaczenia. Zadanie wykonuje się aż nie zostanie wywłaszczone przez siebie lub inne zadanie o wyższym priorytecie.
- **Planowanie rotacyjne** (round-robin) - Znane również jako algorytm karuzelowy. Każde z zadań otrzymuje kwant czasu, po spożytkowaniu którego zostaje wywłaszczone i ustawione na końcu kolejki.
- **Planowanie sporadyczne** - zadania otrzymują tak zwany "budżet czasu". Ten algorytm pomaga pogodzić wykluczające się regułu dotyczące szregowania zadań okresowych i nieokresowych. Wciąż nie jest implementowany przez wiele systemów, jednak znalazł się w standardzie POSIX.

Najbardziej znane systemy operacyjne czasu rzeczywistego:

- LynxOS,
- OSE,
- QNX twardy,
- RTLinux twardy,
- VxWordks twardy,
- Windows CE - system miękki,
- MacOS - system miękki

Systemy czasu rzeczywistego znajdują zastosowane:

- W przemyśle do nadzorowania procesów technologicznych,
- Do nadzorowania eksperymentów naukowych,
- W urządzeniach powszechnego użytku jak sterowniki układów ABS i ESP czy wtrysku paliwa do silników samochodowych, bądź też w urządzeniach gospodarstwa domowego,
- W medycynie, w lotnictwie, zastosowaniach wojskowych i komicznych

8. Podstawowe właściwości sieci przemysłowych i ich znaczenie praktyczne.

Sieć przemysłowa to sieć teleinformatyczna umożliwiająca komunikację pomiędzy różnymi urządzeniami cyfrowymi w ustandaryzowany sposób i w warunkach przemysłowych.

Wymagania:

- niezawodność,
- przewidywalność procesu komunikacji,
- odporność na kolizje komend,
- możliwość pracy w trudnych warunkach.

Działanie większości sieci przemysłowych opiera się na modelu ISO/OSI, będącego specyfikacją ustanowioną w celu stworzenia wspólnego modelu sieciowego, która jest traktowana jako model odniesienia dla większości rodzin protokołów komunikacyjnych.

Zgodnie z tym modelem proces komunikacji został podzielony na 7 etapów, nazywanych warstwami:

- Warstwy niższe:

    1. Fizyczna - transmisja sygnałów w sieci, przetwarzanie sygnałów na bity, media transmisyjne,
    2. Łącza danych - nadzór nad prawidłową transmisją, pakowanie danych w ramki, kontrola błędów,
    3. Sieciowa - dysponuje wiedzą na temat topologii sieci, odpowiada za znajdowanie dróg między poszczególnymi urządzeniami,
    4. Transportowa - segmentuje dane i sklada je w strumień. Zapewnia całościowe połączenie między stacją źródłową, a docelową.

- Warstwy wyższe:

    5. Sesji - synchronizacja danych pomiędzy sesjami systemu nadawcy i odbiorcy. Nadzór nad połączeniem i wznawianie połączenia.

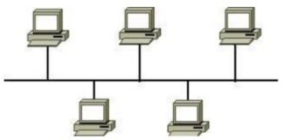
6. Prezentacji - Przetwarzanie danych do postaci zrozumiałej dla danego systemu. Kompresja, dekompresja, szyfrowanie itd.
7. Aplikacji - interfejs wykorzystywany przez aplikację do przesyłania danych.

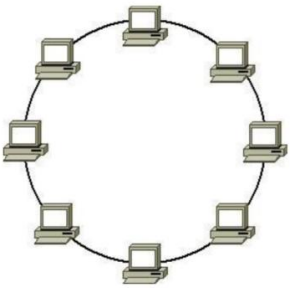
Sieci przemysłowe mogą mieć różne **topologie** tzn. model układu połączeń między elementami sieci. Rodzaj topologii określa fizyczną realizację sieci, jej układu przewodów i mediów transmisyjnych.

Podstawowe rodzaje topologii:

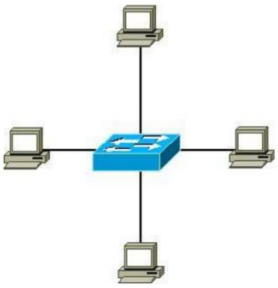
- Topologia liniowa - urządzenia sieciowe połączone są z dwoma sąsiednimi. Dane przesyłane są przez kolejne połączenia aż do dotarcia do celu. Cechuje ją małe zużycie przewodów, niska skalowalność oraz fakt, iż awaria pojedynczego przewodu lub urządzenia sieciowego powoduje przerwanie sieci.

- Topologia magistrali - cechuje się tym, że wszystkie elementy sieci są podłączone do jednej magistrali. Charakteryzuje ją małe zużycie kabla, niska cena, łatwość instalacji i małe koszty produkcji, jednak również trudność lokalizacji usterek, potencjalnie duża liczba kolizji, awaria głównego kabla unieruchamia całą sieć, słaba skalowalność i niskie bezpieczeństwo.

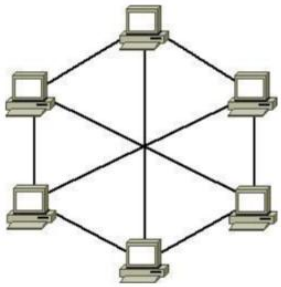




- Topologia pierścienia - Metoda transmisji oparta o token-ring. Małe zużycie przewodów, mała skalowalność, awaryjność i trudna diagnostyka. Trudne dołączanie nowych urządzeń.



- Topologia gwiazdy - Kable połączone są w wspólnym punkcie dostępu, w którym znajduje się koncentrator lub przełącznik. Topologię tą cechuje większa przepustowość, łatwa lokalizacja uszkodzeń, wydajność, łatwa rozbudowa, przejrzystość, awaria jednego urządzenia nie blokuje sieci, duże zużycie kabli, awaria centralnego punktu zatrzymuje sieć.



- Topologia siatki - wszystkie (lub jakaś część) urządzenia są ze sobą wzajemnie połączone. Zapewnia wysoką niezawodność, brak kolizji, wysoką przepustowość, łatwą diagnostykę, ale wysoki koszt i skomplikowaną budowę.

Sieci dzielą się również pod względem kierunku przesyłania danych:

- Dupleks - przesył możliwy w obu kierunkach jednocześnie,
- Half dupleks - przesył możliwy w obu kierunkach, ale nie jednocześnie,
- Simpleks - przesył możliwy tylko w jednym kierunku

Ważna cechą sieci przemysłowych jest ich **determinizm** tzn. zdolność do określenia czasu, w którym zostanie zrealizowane żądanie.

Sieci charakteryzuje również system **rozwiązywania kolizji.** Kolizja zachodzi, gdy dwa urządzenia chcą jednocześnie nadawać po jednym medium komunikacyjnym. Do mechanizmów wykrywania i zapobiegania kolizji należą tokeny, natychmiastowe przerwanie nadawania i ponowną próbę po określonym czasie, odpowiednie topologie (np. topologia siatki).

Sygnał w sieciach może być **kodowany** na wiele sposobów, a konkretna metoda kodowania wpływa na bezpieczeństwo i prędkość transmisji.

Przykładowe metody kodowania:

- Kodownaie Manchester,
- Modulacja amplitudowa (ASK) typ modulacji cyfrowej reprezentującej sygnał w postaci zmieniającej się amplitudy fali nośnej,
- Modulacja częstotliwościowa (FSK),
- Modulacja fazowa (PSK).

Popularne sieci:

- ASI
    - sieć polowa (do łączenia czujników, enkoderów, zaworów itp.)
    - Mono-master,
    - Max. 62 slav'y.
    - Dwuprzewodowa,
    - Niepotrzebne jest zewnętrzne zasilanie urządzeń,
    - Przystosowana do pracy w trudnych warunkach,
    - Odporna na zakłócenia,
- LonWorks
    - Przystosowana do obsługi inteligentnych budynków,
    - Dwuprzewodowa,
    - Kodowanie Manchester,
- Modbus
    - Wykorzystywane warstwy fizyczne: łącze szeregowe, Ethernet.
    - Rodzaje sieci: RTU (szeregowy), ASCII (szeregowy), TCP/IP (Ethernet), UDP (Ethernet),
    - Max 246 urządzeń,
    - Mono master
- Profibus DP
    - Multi-master (przekazywanie tokenów),
    - Warstwa fizyczna oparta o RS-485,
    - Max 127 węzłów,
    - Transmisja do 12Mbit/s
- Profibus PA
    - Zasilanie z przewodu sieciowego,
    - przystosowana do pracy w trudnych warunkach,
    - Prędkość 31.25 kb/s,
    - Pół-dupleks
    - Do 32 stacji,
    - Dwuprzewodowa,
- Canopen
    - Czteroprzewodowa,
    - Mono-master
    - Do 1Mbit/s
- Foundation FieldBus H1
    - 31.25 kbit/s
    - Dwuprzewodowa
    - Half-duplex,
    - Używa RS485
    - Do zastosowań w trudnych warunkach,
    - Odporna na zaklócenia
- HAART
    - prędkość 1200 b/s,
    - dwuprzewodowa,
    - Master/Slave,

9. Struktury funkcjonalne i sprzętowe systemów automatyki

Systemy automatyki - układy urządzeń zajmujące się automatyzacją produkcji i procesów przemysłowych

Proces przemysłowy - ciąg celowych działań realizowanych w ustalonym czasie przez określony zbiór maszyn i urządzeń przy określonych dostępnych zasobach.

Podział na struktury:

Systemy automatyki często są przeznaczone do realizacji złożonych zadań, które wymagają nadania systemowi odpowiedniej struktury. Można rozróżnić:

- Strukturę funkcjonalną,
- Strukturę sprzętową.

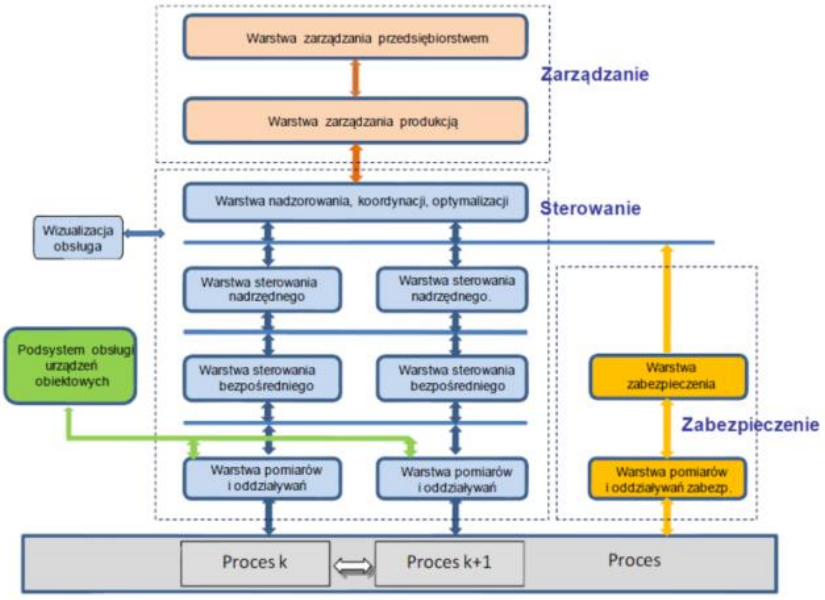
Struktura **funkcjonalna** definiuje zbiór realizowanych zadań oraz powiązania między nimi. Współczesne systemy sterowania i zarządzania produkcją mają zawsze hierarchiczną strukturę funkcjonalną.

Struktura **sprzętowo-programowa** określa sposób technicznej realizacji zadań z zastosowaniem sprzętu i oprogramowania. Strukturze funkcjonalnej odpowiadać może wiele różnych rozwiązań sprzętowo-programowych.

Struktura funkcjonalna systemów automatyki

W strukturze funkcjonalnej systemu automatyki i zarządzania procesu można wyróżnić:

- System automatyki zawierający:
  - podsystem sterowania,
  - podsystem wizualizacji i obsługi,
  - podsystem bezpieczeństwa,
  - podsystem obsługi urządzeń obiektowych.
- System zarządzania w skład, którego wchodzi:
  - podsystem zarządzania produkcją,
  - podsystem zarządzania przedsiębiorstwem



Rys.2.1. Struktura funkcjonalna systemu sterowania i zarządzania produkcją

W skład **podsystemu sterowania** wchodzą następujące warstwy:

Warstwa **realizacji pomiarów i oddziaływań sterujących**. W strukturze sprzętowej jest to zbiór urządzeń pomiarowych i wykonawczych. Coraz częściej są to tzw. inteligentne urządzenia pomiarowe i wykonawcze, wyposażone w jednostki

mikroprocesorowe, realizujące wiele funkcji wstępnego przetwarzania sygnałów, regulacji położenia elementów nastawczych oraz komunikacji z jednostkami sterującymi przez sieci polowe.

Warstwa **sterowania bezpośredniego,** której głównymi zadaniami jest regulacja i sterowanie binarne. Fizycznie w warstwie tej występują różnego rodzaju sterowniki i regulatory.

Warstwa **sterowania nadrzędnego**, w której realizowane są algorytmy regulacji, kompensacji zakłóceń, adaptacji oraz sterowania optymalnego poszczególnych węzłów technologicznych. Sygnały sterujące wypracowane w tej warstwie nie oddziałują bezpośrednio na urządzenia wykonawcze, lecz stanowią wejścia zadane dla algorytmów w warstwie sterowania bezpośredniego. Algorytmy tej warstwy mogą być realizowane zarówno przez sterowniki dużej mocy jak i komputery.

Warstwa **nadzorowania, koordynacji i optymalizacji procesu.** Nadzorowanie obejmuje zadania wykrywania, rejestrowania i sygnalizacji alarmów. Zadaniami sterowania, realizowanymi dla procesów ciągłych w tej warstwie jest koordynacja strumieni materiałów i energii przepływających między różnymi częściami procesu (węzłami technologicznymi) oraz optymalizacja punktów pracy procesu. Dla procesów dyskretnych realizowane są algorytmy koordynacji pracy grupy maszyn i urządzeń. Funkcje te realizują komputery.

Zadaniem **podsystemu wizualizacji i obsługi** jest współpraca z operatorami, automatykami i innymi użytkownikami systemu. Służą do tego panele i stacje operatorskie, stacje inżynierskie i nadzorcze. Stacje operatorskie umożliwia oddziaływanie na proces, stacje inżynierskie pozwalają na konfigurację sprzętową i programową (algorytmiczną) systemu oraz na wprowadzanie modyfikacji, natomiast stacje nadzorcze umożliwiają jedynie wizualizację przebiegu procesu i danych.

Każdy proces jest wyposażony w **podsystem zabezpieczeń**, który w warstwie pomiarów i oddziaływań sterujących oraz w warstwie sterowania jest niezależny od systemu sterowania. Oznacza to, że funkcje zabezpieczenia realizowane są z wykorzystaniem innych urządzeń (pracujących zwykle w strukturze redundancyjnej) niż zadania sterowania. W warstwie wizualizacji i obsługi systemy sterowania i zarządzania mogą być niezależne lub zintegrowane.

Podsystem obsługi urządzeń obiektowych to osobny podsystem zawierający bazę danych, w której gromadzone są wszelkie parametry konfiguracyjne urządzeń pomiarowych i wykonawczych, informacje o dokonywanych modyfikacjach tych parametrów, naprawach urządzeń itp. Podsystem ten umożliwia zdalną konfigurację i kalibrację urządzeń obiektowych, a także diagnostykę w trybie off-line (gdy urządzenie nie funkcjonuje w procesie) oraz coraz częściej diagnostykę bieżącą (online) pracujących urządzeń pomiarowych i wykonawczych.

Współczesne systemy automatyki są integrowane z **systemami zarządzania.** Taki zintegrowany system sterowania i zarządzania ma jeszcze dwie dodatkowe warstwy: **warstwę zarządzania produkcją** i **warstwę zarządzania przedsiębiorstwem**, realizowane wyłącznie w technice komputerowej. Zadania zarządzania produkcją realizują systemy MES (Manufacturing Execution Systems), a zarządzania przedsiębiorstwem systemy ERP (Enterprise Resource Planning).

W strukturach sterowania algorytmy warstw wyższych wyznaczają parametry dla algorytmów warstw niższych, natomiast algorytmy warstw niższych realizowane są zwykle z większymi częstotliwościami niż algorytmy warstw wyższych. Przykładowo algorytmy sterowania nadrzędnego wyznaczają wartości zdane algorytmów regulacji, a algorytmy regulacji przekazują sygnały sterujące do warstwy oddziaływana na proces, gdzie urządzenia wykonawcze (często serwomechanizmy) nadążają za zmianami sygnałów sterujących. Przy tym algorytmy sterowania nadrzędnego realizowane są zwykle ze znacznie mniejszymi częstotliwościami (np. kilka razy na godzinę) niż algorytmy regulacji (kilku lub kilkadziesiąt razy na sekundę).

Struktura sprzętowa systemów automatyki

W strukturze sprzętowej systemów automatyki wyróżnić można następujące rodzaje uniwersalnych jednostek funkcjonalnych:

- urządzenia pomiarowe i wykonawcze,
- urządzenia sterujące,
- urządzenia obserwacji i obsługi,
- sieci,
- serwery

Urządzenia pomiarowe i wykonawcze

Urządzenia pomiarowe realizują pomiar różnych wielkości fizycznych analogowych i cyfrowych.

Typowymi urządzeniami wykonawczymi są: zawory regulacyjne, klapy, zasuwy, podajniki materiałów sypkich, wentylatory, dmuchawy. Są one napędzane przez siłowniki pneumatyczne lub silniki elektryczne.

Urządzenia sterujące

Spośród urządzeń sterujących można wyróżnić:

- Regulatory aparatowe (PID, uniwersalne z ustalonym podziałem pamięci, wielofunkcyjne).

Regulator PID stanowi najlepsze rozwiązanie w przypadku braku wiedzy na temat obiektu regulacji. Poprzez odpowiedni dobór nastaw regulatora uzyskuje sie regulacje dostosowana dla danego obiektu.

Regulatory uniwersalne z ustalonym podziałem pamięci są bardziej elastyczne i mają większe możliwości funkcjonalne. Charakteryzuje je ustalony podział pamięci oznacza to, że w pamięci regulatora wydzielone są podzbiory bloków o określonej liczności i określonych rozmiarach uporządkowane zwykle w strukturze warstwowej. Bloki w danej warstwie maja taki sam rozmiar, ale rozmiar bloków w różnych warstwach może się różnić. Do każdego bloku w danej warstwie przyporządkowany może zostać jeden z algorytmów przeznaczony do realizacji w danej warstwie. Liczba sygnałów wejściowych i wyjściowych w tych algorytmach jest ustalona. W obszarach bloków przechowywane są wartości sygnałów i ich parametrów odpowiadające algorytmowi przyporządkowanemu do bloku. Muszą one zajmować zatem obszar nie większy niż rozmiar bloku.

Regulatory wielofunkcyjne z dynamiczną rezerwacją pamięci są najbardziej zaawansowaną funkcjonalnie i technicznie grupą przyrządów aparatowych. Realizują funkcje przetwarzania sygnałów, regulacji i sterowania binarnego. Elastycznośc funkcjonalną przyrządu zapewnia dynamiczny przydział pamięci. Każdemu algorytmowi dostępnemu w bibliotece przyrządu odpowiada określony rozmiar bloku na wartości sygnałów wejściowych i wyjściowych oraz wartości parametrów algorytmu. Rezerwacja pamięci dla danego bloku następuje w sposób dynamiczny w trakcie konfiguracji przyrządu po wybraniu bloku do realizacji. Liczba zastosowań poszczególnych algorytmów nie jest ograniczona tak jak w przypadku ustalonego podziału pamięci (określona licbza bloków w warstwie), jedynym ograniczeniem jest rozmiar pamięci przyrządu.

- Sterowniki programowalne PLC

W stosunku do urządzeń aparatowych modułowe sterowniki PLC mogą zwykle obsługiwać znacznie więcej sygnałów, a tym samym sterować zarówno procesami małymi, jak też średnimi i dużymi. Są także urządzeniami zapewniającymi krótsze czasy realizacji algorytmów sterowania w stosunku do urządzeń aparatowych. Wynika to z stosowania odpowiednio mocnych procesorów. Sterowniki PLC nie mają wbudowanych elementów (wyświetlaczy, linijek diodowych, klawiszy itp.) umożliwiających sterowanie ręczne, prostą wizualizację i obsługę operatorską sterownika. Funkcje wizualizacji i obsługi procesu zapewniają panele operatorskie lub systemy SCADA. Do programowania oraz konfiguracji struktury sprzętowej wykorzystywane są komputery PC.

- Sterowniki programowalne PAC

Programowalne sterowniki PAC (Programmable Automation Controller) to nowa generacja urządzeń sterujących, które łączą zalety sterowników PLC oraz komputerów PC. Zamiast prostej pętli programowej w sterownikach PAC stosowane są systemy operacyjne czasu rzeczywistego. Kluczowa różnica między PAC i PLC polega na znacznie bardziej zaawansowanym oprogramowaniu oraz dużej jego elastyczności, polegającej m.in. na łatwym przenoszeniu na różne platformy sprzętowe.

- Stacje procesowe w systemach DCS

Sterowniki występujące w systemach klasy DCS określane są powszechnie jako stacje procesowe. Stacja procesowa zawiera: zasilacze, jednostkę centralną, moduły wejść – wyjść procesowych, moduły sieci do sprzężenia z magistralą systemu, zdalnymi modułami wejść wyjść oraz inteligentnymi urządzeniami. Typowa budowa odbiega zatem od struktury sprzętowej sterownika programowalnego. Istotne różnice polegają na wykorzystaniu systemów operacyjnych czasu rzeczywistego w stacjach procesowych, możliwości dołączania nowych urządzeń pod napięciem bez przerywania pracy systemu oraz oznaczania pomiarów aktualnym czasem określanym z dużą dokładnością (tzw. stempel czasowy), zwykle 1 ms. Jest to realizowane przez moduły wejściowe. Charakterystyczne jest także stosowanie redundancji w klasycznych systemach DCS. W systemach hybrydowych redundancja jest opcjonalna.

Urządzenia obserwacji i obsługi

Urządzenia te zapewniają komunikację i współpracę systemu sterowania z obsługą systemu, tj. operatorami i inżynierami. Wyróżnić można:

- Panel operatorski - urządzenie umożliwiające obserwację i kontrolę maszyn i innych urządzeń oraz procesów przemysłowych,
- Stacje operatorskie - umożliwiają operatorom śledzenie przebiegu procesu, obserwację alarmów i zdarzeń oraz sterowanie ręczne,
- Stacje inżynierskie - są elementami dużych zintegrowanych systemów sterowania (klasy DCS). Stacje inżynierskie służą do konfiguracji i modyfikacji systemu sterowania. Z ich wykorzystaniem inżynierowie projektują system, określając jego

strukturę sprzętową, struktury algorytmiczne układów przetwarzania sygnałów regulacji, sterowania binarnego, kontroli procesu (wykrywania i sygnalizacji alarmów).
- Stacje informacyjne - (nadzorcze). Przeznaczone dla innych użytkowników systemu (technologów, kierownictwa produkcyjnego, służb utrzymania ruchu itp.). Nie pozwalają na ingerencję w pracę systemu. Ułatwiają jedynie pozyskiwanie informacji z systemu, które przeznaczone są dla danego użytkownika.

Sieci

Sieci łączą wszystkie pozostałe urządzenia w jeden system sterowania i zarządzania. Wyróżniamy sieci szeregowe i równoległe. W sieciach równoległych równocześnie przesyłana jest informacja wielobitowa, do czego wykorzystywana jest wiązka przewodów. Określone podzbiory przewodów wykorzystywane są do przekazywania danych, adresów przerwań, informacji sterujących itp. Sieci równoległe mogą zapewniać większą szybkość transmisji, lecz ich wadą jest mały zasięg (do kilku metrów) i duża liczba linii przesyłowych. Rozwiązania takie stosowane są w systemach wieloprocesorowych. Były one także wykorzystywane w rozwiązaniach stacji procesowych systemów DCS. W sieciach szeregowych informacja przesyłana jest jako ciąg kolejnych bitów. W przypadku połączeń elektrycznych do transmisji stosowana jest najczęściej para skręconych przewodów. Ten typ transmisji jest dominujący. W systemach sterowania wyróżniamy następujące rodzaje sieci szeregowych:

- sieci polowe (Fieldbus),
- lokalne sieci komputerowe (LAN - Local Area Network),
- sieci rozległe (WAN - Wide Area Network).

Serwery

W systemach automatyki wykorzystywane są różne rodzaje serwerów:

- Serwery do realizacji zaawansowanych obliczeń zwiększające moc obliczeniową systemu. Są stosowane do realizacji zaawansowanych algorytmów sterowania i optymalizacji.
- Serwery archiwizujące nazywane historianami. Archiwizacji mogą podlegać wartości zmiennych procesowych wraz ze stemplami czasowymi, a także alarmy i zdarzenia. Dane archiwizowane wykorzystywane są do analizy przebiegu procesu, w tym szczególnie stanów awaryjnych, do opracowywania różnorodnych raportów, do budowy modeli itp.
- Serwery WWW udostępniają dane aktualne i historyczne w sieci Internet/Intranet. Stosowane są do tego standardowe przeglądarki. Odbiorcami danych są kierownictwo zakładu, nadzór dyspozytorski i służby techniczne.

10. Charakterystyka i różnice między modelem koncepcyjnym, logicznym i fizycznym bazy danych.

Konstruowanie bazy danych jest procesem modelowania. Jest to proces kolejnych transformacji poprzez trzy poziomy modelowania:

- koncepcyjny,
- logiczny i
- fizyczny.

Te trzy modele łączą się luźno ze sobą i mogą służyć jako różne punkty widzenia pojedynczego procesu biznesowego.

Model koncepcyjny powstaje na podstawie specyfikacji wymagań tworzonej w początkowej fazie projektowania bazy danych. Jest to model stosunkowo wysokiego poziomu abstrakcji opisujący strukturę oraz wzajemne powiązania obiektów biorących udział w realizacji procesu biznesowego.

Koncepcyjny model danych stanowi pomost między wymaganiami stawianymi przez klientów, a projektantem systemu odpowiedzialnym za opracowanie struktury systemu bazodanowego (sposobu przechowywania i dostępu do danych). Pozwala na porozumienie pomiędzy klientem, analitykiem, a projektantem systemu i jednocześnie jest jednym z dokumentów formalnych definiujących zakres realizacji systemu oraz zasady jego działania.

Tworzony jest w prostej formie, beż użycia elementów informatycznych, dlatego jest zrozumiały przez wszystkie zainteresowane osoby - stanowi uniwersalną metodę porozumiewania między klientem, a zespołem opracowującym system bazodanowy.

Często model ten nie występuje jawnie, ale jest wyrażony w innych elementach tworzonej dokumentacji np. formalnie spisanej dokumentacji wymagań.

Także forma fizyczna opracowywanego modelu może być bardzo różna. Może być on opracowany w formie opisowej albo w postaci diagramu z elementami objaśniającymi. Istnieje także kilka rodzajów modeli koncepcyjnym. Jednym z powszechniej stosowanych jest model związków encji ER.

Model koncepcyjny opisuje strukturę danych, łączące je związki oraz założone na nie ograniczenia, w sposób niezależny od przyjętego modelu danych oraz systemu zarządzania bazą danych (SZBD).

Encje koncepcyjne reprezentują idee i nie podlegają fizycznemu wdrożeniu. Posiadają najważniejsze atrybuty, niezbędne do zrozumienia procesu biznesowego. Głównie są wykorzystywane w celu powiadamienia zespołów o obszarach zainteresowania oraz rozległych pojęciach, które mają być poddawane dalszej analizie w ramach projektu. Encje koncepcyjne mają zazwyczaj ogólny charakter.

Związki są to wyrażenia czasownikowe łączące encje. Mogą być bardziej złożone niż pojedynczy czasownik. Każdemu związkowi odpowiada linia obrazująca odrębne reguły biznesowe. Związki posiadają liczność, w modelach koncepcyjnych mogą być wykorzystywane dowolne liczności, ale najczęściej stosowane są te w ramach notacji "jeden" lub "wiele"

Model logiczny (inaczej implementacyjny) - jest implementacją modelu koncepcyjnego w przyjętym modelu danych, jaki będzie wykorzystywać opracowywana aplikacja np. relacyjnym lub obiektowym. Model logiczny ciągle pozostaje niezależny od konkretnego Systemu Zarządzania Bazą Danych (przynajmniej w ramach grupy SZBD wykorzystujących ten sam model danych). Najbardziej rozpowszechniony jest relacyjny model danych.

Podstawowe modele danych:

- **Sieciowy** - w modelu tym dane reprezentowane są w postaci odpowiednich typów rekordów. Umożliwia on odzworowanie jedynie związków typu "1 do n". Jest to model praktycznie niewykorzystywany. Może być zastosowany do lokalnych aplikacji wykorzystujących bardzo prosty zapis danych np. w plikach tekstowych.
- **Hierarchiczny** - dane reprezentowane są w hierarchicznej strukturze drzewiastej. Jest to także rozwiązanie nieco archaiczne aczkolwiek nadal spotykane. Pewnego rodzaju rozwinięciem tego modelu są XML'owe struktury danych.
- **Relacyjny** - model danych, w którym określone byty (obiekty, zjawiska) świata rzeczywistego opisywane są w postaci tabel reprezentujących atrybuty opisujące te byty. Tabele stanowiące reprezentację typu bytów, łączone są określonymi relacjami reprezentującymi związki występujące pomiędzy rzeczywistymi obiektami i zjawiskami. Jest to najpopularniejszy i najbardziej rozpowszechniony model danych. Obejmuje on grubo ponad 90% zastosować systemów bazodanowych .
- **Obiektowy** - model danych wywodzący się z obiektowej analizy systemów oraz programowania obiektowego. Podstawowe elementy danych reprezentowane są w tym modelu w postaci obiektów. Obiekty te mają zdefiniowane atrybuty oraz metody. Tego typu model danych obsługuje bezpośrednio pojęcia klas (typów obiektów) oraz dziedziczenia, czyli łączenia klas w hierarchię. Jest to bardzo ciekawy model danych o rosnącej liczbie aplikacji. Cały czas ustępuje on jednak znacznie (w implementacjach) modelowi relacyjnemu.
- **Obiektowo - relacyjny** - rozwinięcie hybrydowe, łączące elementy modelu relacyjnego oraz obiektowego. Zwykle jest to model relacyjny wzbogacony o elementy obsługi obiektów.
- **XML** - specyficzny model danych oparty na specyfikacji języka XML. Model ten zawiera elementy modelu obiektowego w połączeniu z hierarchiczną strukturą drzewiastą (poszczególne obiekty w XML'u zawierają się w sobie jako elementy składowe, brak jest jawnej obsługi jednoczesnej przynależności do kilku rodziców, czyli wielodziedziczenia). Jest to model zdobywający coraz większą popularność, szczególnie w zakresie wymiany danych pomiędzy różnymi systemami informatycznymi,. Spotykane rozwiązania SZBD implementujące model oparty na XML, są zazwyczaj w postaci specjalistycznej nakładki na model relacyjny.

Encje logiczne muszą opisywać elementy procesu biznesowego w sposób umożliwiający zaimplementowanie bazy danych. Analiza logiczna wykorzystuje model koncepcyjny jako punkt wyjścia i zapewnia przejścia do w pełni udokumentowanych "zbiorów" lub encji, które muszą być tworzone i wiązane ze sobą w celu zapewnienia obsługi zakresu projektu.

Związki w modelu logicznym stanowią ścieżki, za pomocą których klucze główne łączą ze sobą zbiory danych. Na tym etapie musimy jednak określić czy związki między poszczególnymi pojęciami są identyfikujące, czy nieidentyfikujące, wyrażenia czasownikowe zachować w jak najprostszej postaci, dokładnie określić liczność stron związków oraz ich opcjonalność.

Logiczny model danych operuje na obiektach bazodanowych:

- encje przekształcane są na tabele,
- atrybuty danych stają się kolumnami w tabelach,
- związki między danymi przekształcane są na relacje między tabelami tworzone za pomocą kluczy głównych i kluczy obcych,

Model fizyczny jest implementacją logicznego modelu w konkretnym systemie zarządzania bazą danych. Na fizyczną implementację modelu danych mają wpływ wymagania dostępu do danych oraz możliwości funkcjonowania SZBD. Czynniki te powodują, że ten sam fizyczny model danych może być implementowany na różne sposoby.

Różnice:

Różnice między tymi modelami wynikają ze stopnia zaawansowania projektu i ich różnych poziomów abstrakcji. Na początku tworzona jest koncepcja bazy, oparta na analizie potrzeb klienta. W modelu koncepcyjnym są więc wyróżnione dane, które będą gromadzone oraz przedstawiona ich organizacja. Musi on być zaprezentowany w sposób zrozumiały zarówno dla projektantów bazy danych jak i dla osób nie posiadających specjalistycznej wiedzy technicznej. Następnie model koncepcyjny przekształcany jest w logiczny, opisujący szczegółowo wszystkie gromadzone dane, powiązania między nimi, atrybuty będące kluczami relacji itd. Najbardziej zaawansowanym modelem jest model fizyczny. Uzupełnia on model logiczny o specyficzne zagadnienia związane z fizyczną realizacją bazy (konkretny typ danych, język programowania, system zarządzania bazą danych, sposób przechowywania danych, realizacja różnych perspektyw itd.).

11. Różne formy reprezentacji związków pomiędzy obiektami danych w relacyjnej bazie danych

W relacyjnej bazie danych jej schemat jest zbiorem schematów relacji. **Relacja**, zwana tabelą, jest postrzegana jako dwuwymiarowa tablica, której kolumny są nazywane atrybutami, a wiersze krotkami, bądź rekordami. W modelu relacyjnym każda krotka reprezentuje wystąpienie encji.

Kluczem podstawowym relacji nazywamy atrybut lub zbiór atrybutów jednoznacznie identyfikujący krotkę relacji.

Kluczem obcym nazywamy atrybut lub zbiór atrybutów wskazujący na klucz podstawowy innej relacji. Innymi słowy jest to atrybut lub zbiór atrybutów w relacji B, będący jednocześnie kluczem podstawowym w relacji A, przy czym klucz obcy może odnosić się do klucza podstawowego tej samej relacji, w której został zadeklarowany.

Związki - określają wzajemne powiązania (logiczne lub fizyczne) typów encji, a w konsekwencji poszczególnych egzemplarzy encji.

Ograniczenia związków.

Wyróżniamy dwa rodzaje ograniczeń: **współczynnika liczności** oraz **udziału**. Obie te grupy tworzą razem **ograniczenia strukturalne**.

Ograniczenie współczynnika liczności - dotyczy maksymalnej liczby egzemplarzy danego związku r_j, w których może uczestniczyć pojedyncza encja e_j. Inaczej mówiąc maksymalną liczbę encji jednego typu, powiązanych z daną encją drugiego typu.

- (1) - każda encja typu E_i może być powiązana tylko z jedną encją typu E_j związkiem R_k
- (n) - każda necja typu E_i może być powiązana z nieograniczoną liczbą encji typu E_j związkiem R_k

W modelowaniu mamy zwykle do czynienia ze związkami drugiego stopnia łączącymi dwa typy encji. Ponieważ ograniczenie współczynnika liczności nakładane jest na każdy typ encji występujący w związku oddzielnie, dlatego ostatecznie mówimy o związkach typu:

- (1:1) - każda encja typu E_i może być powiązana z tylko jedną encją typu E_j i na odwrót
- (1:n) - każda encja typu E_i może być powiązana z kilkoma encjami typu E_j, ale każda encja typu E_j może być powiązana tylko z jedną encją typu E_i,
- (n:m) - Wiele do wielu. każda encja typu E_i może być powiązana z kilkoma encjami typu E_j i na odwrót

Ograniczenie udziału - mówi o tym, czy dana encja musi uczestniczyć w danym związku. Inaczej mówię jest to dolne ograniczenie udziału w związku. Ograniczenie udziału nakładane jest oddzielnie na każdy z typów encji występujący w danym związku. Wyróżnia się związki o udziale:

- całkowitym (1) - każda encja typu E_i musi być powiązana z przynajmniej jedną encją typu E_j, aby przynależeć do związku R_k,
- częściowym (0) - nie każda encja typu E_i musi być powiązana z jakąkolwiek encją typu E_j, aby przynależeć do związku R_k,

Słaby typ encji - każdy typ encji, dla którego nie można określić żadnego klucza.

Silny typ encji - każdy typ encji, dla którego można określić klucz.

Podział związków ze względu na istnienie relacji:

- opcjonalny - wartość klucza obcego jest opcjonalna tzn. dopuszcza wartość NULL.
- wymagany - wartość klucza obcego jest wymagana tzn. nie dopuszcza wartości NULL.

Występują 3 **typy związków** według podziału pod względem liczebności**:**

- **1:1,**
- **1:N,**
- **N:M**

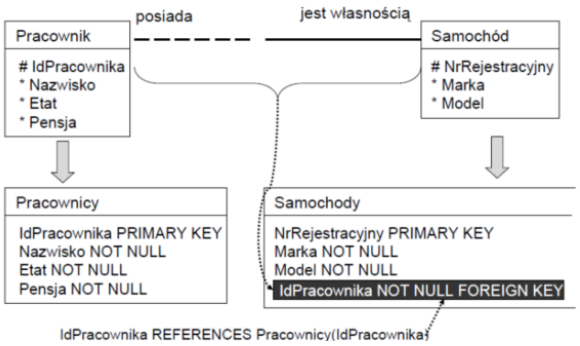
Związek binarny 1:1

Każda encja typu E_i może być powiązana z tylko jedną encją typu E_j i na odwrót

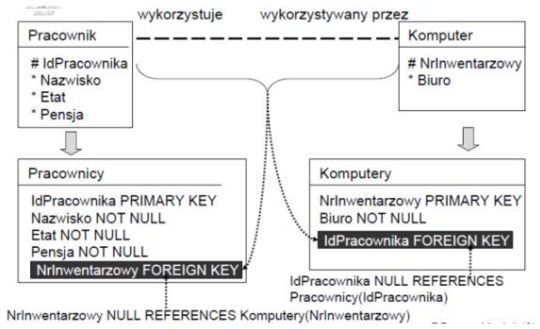
Sposoby odwzorowania:

- *sposób 1*: realizacja związku przez dodanie do jednej z relacji związku atrybutu (-ów) **klucza obcego** odpowiadających kluczowi głównemu drugiej relacji,
- *sposób 2*: **scalenie** relacji występujących w związku w jedną relację,
- *sposób 3*: utworzenie oddzielnej relacji reprezentującej związek tzw. relacji związku

Przykłady:



Jest to związek binarny 1:1 jednostronnie obowiązkowy, reprezentowany przez klucz obcy w tabeli po obowiązkowej stronie związku. Ograniczenie integralności jest definiowane dla atrybutu klucza obcego. Klucz ten nie może przyjmować wartości pustych.



Jest to związek binarny 1:1 obustronnie opcjonalny.

Związek binarny 1:N

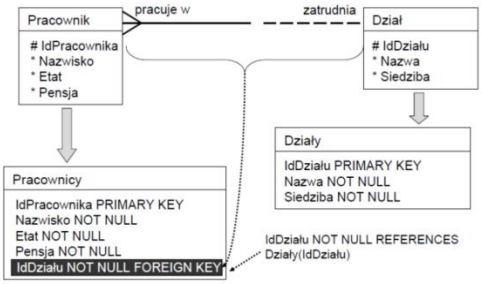
Każda encja typu E_i może być powiązana z kilkoma encjami typu E_j, ale każda encja typu E_j może być powiązana tylko z jedną encją typu E_i.

Sposoby odwzorowania:

- *sposób 1:* relacja związku przez dodanie do relacji występującej po stronie *N* związku atrybutu (-ów) **klucza obcego** odpowiadających kluczowi głównemu relacji występującej po stronie *1*.

- *sposób 2:* utworzenie **oddzielnej relacji** reprezentującej związek tzw. relacji związku (klucz główny takiej relacji może odpowiadać kluczowi głównemu relacji po stronie *N* związku).

Przykład:



Przykład ilustruje sposób transformacji binarnego związku 1:N jednostronnie obowiązkowego. Z encji Pracownik postaje tabela Pracownicy, a z encji dział tabela Działy. Klucz obcy jest dodawany do tabeli Pracownicy (strona "wiele") i wskazuje on na klucz podstawowy tabeli Działy.

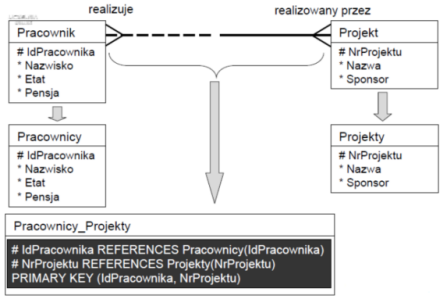
Związek binarny N:M

Każda encja typu E_i może być powiązana z wieloma encjami typu E_j i na odwrót

Sposoby odwzorowania:

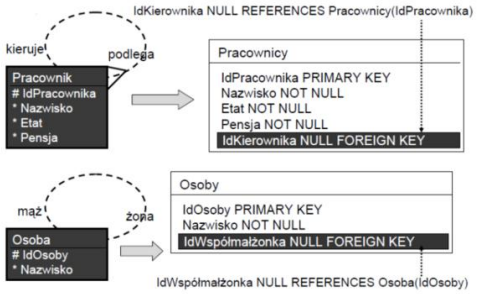
- utworzenie **oddzielnej relacji** reprezentującej związek,

Przykład:

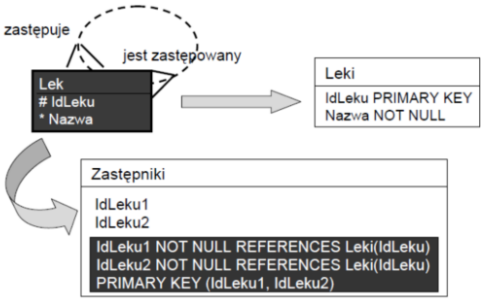


Związek unarny

Inaczej rekursywny - łączy encję samą ze sobą.



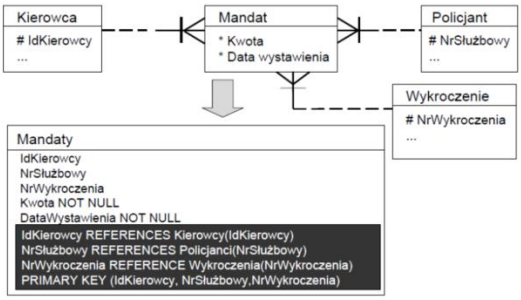
Związek unarny 1:1 lub 1:N transformuje się do klucza obcego w tej samej tabeli. W pierwszym przykładzie z encji Pracownik powstaje tabela Pracownicy. Zawiera ona klucz obcy IdKierownika wskazujący na IdPracownika w tej samej tabeli.



Związek unarny N:M jest transformowany do tabeli pośredniej. W przykładzie z encji Lek powstaje tabela Leki, a związek jest transformowany do tabeli o nazwie Zastępniki. Tabela ta posiada dwa klucze obce (IdLeku1, IdLeku2) i oba wskazują na klucz podstawowy tabeli Leki, czyli na atrybut IdLeku. Oba klucze wchodzą w skład klucza podstawowego tabeli Zastępniki.

Związki ternarne

Są to związki łączące trzy encje.



Związek ternarny transformuje się w sposób identyczny jak związek 1:M. W przykładzie z encji mandat powstaje tabela Mandaty, zawierające 3 klucze obce: IdKierowcy, NrSluzbowy, NeWykroczenia.Te trzy klucze obce wchodzą w skład klucza podstawowego.

Związnki n-arne

Rodzaj związku łączącego n encji.

Powstaje tak samo jak związek ternarny.

12. Podstawowe człony składniowe zapytania wybierającego w języku SQL

SQL (Structured Query Language) - strukturalny język zapytań używany do tworzenia i modyfikowania baz danych oraz do umieszczania i pobierania danych z bazy. Język SQL jest językiem deklaratywnym. Decyzję o sposobie przechowywania i pobierania danych pozostawia się systemowi zarządzania bazą danych. Język SQL jest znormalizowany przez ISO i ANSI, jednak istnieją jego różne wersje w zależności od używanego SZBD. Wielkość znaków nie jest rozróżnialna.

Instrukcja SELECT

Instrukcja wyszukiwania SELECT służy do generowania zapytań do bazy danych. Po jej wykonaniu powstaje tablica wynikowa zawierająca żądane atrybuty pobrane z wierszy spełniających podane warunki.

SELECT atrybuty FROM tabela [WHERE warunek];

Nazwy kolumn zapytania nie muszą być identyczne z nazwami atrybutów bazy. Można zdefiniować własne nazwy (aliasy) za pomocą fraz AS.

SELECT p.NumerZesp AS Zespół, Sum(p.Pensja) AS Zarobki FROM Pracownicki p GROUP BY p.NumerZesp

Ogólna skladnia:

```
SELECT <column>
FROM <table1>
JOIN <table2>
ON <table1.col1> = <table2.col2>
WHERE <criteria for row>
GROUP BY <column>
HAVING <criteria>
ORDER BY <column> ASC
```

Wartości unikalne

Aby w zapytaniu uzyskać jedynie wartości unikalne danego atrybutu lub grupy atrybutów należy użyć słowa kluczowego *DISTINCT*.

Umieszczenie frazy bezpośrednio za słowem *SELECT*:

SELECT DISTINCT Nazwisko FROM Pracownicy

Zapytanie zwróci zbiór nazwisk pracowników bez powtórzeń.

SELECT DISTINCT Imie, Nazwisko FROM Pracownicy

Powyższe zapytanie zwróci zbiór unikalnych par imię - nazwisko. Dla tego samego nazwiska i różnych imion pojawią się dwa rekordy np. Jan Kowalski i Piotr Kowalski.

Inne użycie frazy *DISTINCT* to użycie jej wewnątrz funkcji agregującej:

SELECT SUM(DISTINCT Pensja) FROM Pracownicy;

Zapytanie zwróci sumę rożnych pensji tzn. dla zbioru (100,200,300,200,300) zwróci wartość 600, a bez frazy *DISTINCT* zwróciłoby wartość 1100.

Zawężanie wyników - klauzula WHERE

W celu pobrania tylko rekordów spełniających danych wynik wykorzystuje się frazę *WHERE:*

*SELECT * FROM Pracownicy WHERE NumerPrac>10*

- Operatory porównujące:
    - *LIKE/NOT LIKE* - służy do porównywania wartości tekstowych z podanym wzorem ('_' - jeden dowolny znak, '%' - dowolny ciąg znaków)

      *SELECT * FROM Pracownicy WHERE Nazwisko LIKE "Kowal%"*
    - *IN*

      *SELECT * FROM Pracownicy WHERE NumerPrac IN (100,101,200)*
    - *BETWEEN x AND y*

      *SELECT * FROM Pracownicy WHERE PensjaPrac BETWEEN 1000 AND 2000*
    - *IS NULL - IS NOT NULL*

      *SELECT * FROM Pracownicy WHERE IdDzialuPrac IS NOT NULL*

Sortowanie

Do sortowania danych używa się frazy ORDER BY. Występuje ona zawsze na końcu zapytania.

*SELECT * FROM Pracownicy WHERE NumerPrac > 10 ORDER BY Nazwisko;*

Zapytanie zwróci tablicę wartości posortowaną alfabetycznie według nazwisk.

Można sortować tablicę wartości według więcej niż jednego atrybutu.

*SELECT * FROM Pracownicy WHERE NumerPrac > 10*

ORDER BY NumerZesp, Nazwisko;

W tym przypadku tablica posortowana zostanie według numerów zespołów, a w ramach jednego zespołu według nazwisk. Do sortowania w odwrotnej kolejności służy znacznik *DESC;*

Łączenie tabel

W zapytaniu dane można pobierać z więcej niż jednej tabeli jednocześnie poprzez zdefiniowanie warunku łączącego tablicę.

SELECT p.Nazwisko, z.NazwaZespolu FROM Pracownicy p, Zespoly z

WHERE p.NumerZesp = Z.NumerZesp

lub

SELECT p.Nazwisko, z.NazwaZesp FROM Pracownicy p

JOIN Zespoly z ON p.NumerZesp = z.NumerZesp

Takie zapytanie tworzy najpierw złączenie dwóch tablic według podanego warunku, a następnie dokonuje odpowiedniej selekcji i projekcji. Ponieważ pole NumerZesp występuje w obu tablicach dla każdego wystąpiena w zapytaniu należy określi, z której tablicy ma być pobrane.

Rodzaje JOIN'ów:

- INNER JOIN - zwraca wszystkie rekordy z obu tabel mające wzajemnie część wspólną.
- LEFT JOIN - zwraca wszystkie rekordy z tablicy lewej i pasujące z tablicy prawej
- RIGHT JOIN - zwraca wszystkie rekordy z tablicy prawej i pasujące z tablicy lewej
- FULL JOIN - zwraca wszystkie rekordy z obu tablic, jeśli to możliwe złączone w jednym wierszu

Przykład:

SELECT p.NumerPrac, Nazwisko, d.Imie FROM Pracownicy p

JOIN Dzieci d ON p.NumerPrac=d.NumerPrac

Zapytanie takie zwróci jedynie numery pracowników mających dzieci podczas gdy zapytanie:

SELECT p.NumerPrac, Nazwisko, d.Imie FROM Pracownicy p

LEFT (OUTER) JOIN Dzieci d ON p.NumerPrac=d.NumerPrac

zwróci wszystkich pracownikow niezależnie od tego, czy posiadają dzieci, czy nie.

Funkcje agregujące

To funkcje działające na grupie wartości zwracanych przez zapytanie, a nie na pojedynczej wartości rekordu.

Przykłady: *SUM(), AVG(), COUNT(), MAX(), MIN()*

Grupowanie - *GROUP BY*

Aby uzyskać informację na temat sum dochodów pracowników poszczegolnych zespołach można pogrupować wynik zapytania agregującego:

SELECT SUM(Pensja) FROM Pracownicy GROUP BY NumerZesp;

Powyższe zapytanie zwróci jeden rekord dla każdej wartości pola NumerZeso.

SELECT p.NumerZesp, NazwaZesp, Sum(Pensja) FROM Pracownicy p

JOIN Zespoly z ON p.NumerZesp=z.NumerZesp

WHERE Nazwa Zesp <>"Piece"

GROUP BY p.NumerZesp, NazwaZesp

To zapytanie zwróci tablicę z trzema kolumnami: numer zespołu, nazwa zespołu, suma zarobków w zespole, przy czym wszystkie atrybuty występujące bezpośrednio po frazie *SELECT*, na ktorych nie dokonuje się agregacji muszą znaleźć się we frazie *GROUP BY*.

Nałożenie warunku na całą grupę lub wynik zapytania agregującego jest możliwe poprzez klauzulę *HAVING*.

SELECT p.NumerZesp, Sum(Pensja) FROM Pracownicy p

GROUP BY p.NumerZesp

HAVING p.NumerZesp>10

Zagnieżdżanie instrukcji

Instrukcja *SELECT* może byc zagnieżdżona w obrębie innej instrukcji *SELECT* np.

SELECT Nazwisko FROM Pracownicy WHERE NumerZesp

IN (SELECT NumerZesp FROM Zespoly WHERE Kierownik = "Misiura");

Zapytanie zagnieżdżone ujęte w nawiasy może być traktowane jako zbiór wartości. W takim przypadku powinno zwracać tylko jedną kolumnę danych. Odwoływać się do niego można poprzez porównanie, z którymś z atrybutów zapytania zewnętrznego.

- *atrybut IN (zapytanie)* - wartość znajduje się w tablicy wyników zapytania,
- *atrybut > ALL (zapytanie)* - wartość jest większa od wszystkich elementow tablicy wyników zapytania zagnieżdżonego
- *atrybut > SOME|ANY (zapytanie)* - wartość jest większa od przynajmniej jednego elementu tablicy wyników zapytania zagnieżdżonego
- *EXISTS/NOT EXISTS (zapytanie)* - zwróci wynik jeśli odpowiednio istnieje lub nie istnieje chociaż jeden rekord w tabeli wynikowej zapytania zagnieżdżonego np.:

SELECT Nazwisko FROM Pracownicy p

WHERE EXISTS (SELECT NumerZesp FROM Zespoly z WHERE z.NumerZesp>p.NumerZesp);

Zapytanie to zwraca nazwiska pracowników, dla których istnieją zespoły o numerach większych niż numer zespołu, do którego należą.

Perspektywy

Istnieje możliwość przekształcenia zapytania w wirtualną tablicę, tak zwaną perspektywę lub widok (ang. VIEW). Perspektywę możemy stworzyć za pomocą frazy:

CREATE VIEW Nazwa_perskeptywy [(astrybuty)] AS (zapytanie)

Od tego momentu identyfikator Nazwa_perspektywy będzie traktowany w kolejnych instrukcjach SQL jak zwykła tablica.

Przykład:

CREATE VIEW DanePersonelu (Nazwisko, Imię, Zespół) AS

(SELECT Nazwisko, Imię, NazwaZesp FROM Pracownicy p, Zespoly z WHERE p.NumerZesp = z.NumerZesp)

Następnie do tak stworzonej perspektywy można się odwoływać zapytanem:

*SELECT * ROM DanePersonelu WHERE Imię Like 'Adam'*

Usunięcie perspektywy realizuje się instrukcją: *DROP VIEW Nazwa;*

Perspektywy mogą służyć do modyfikacji danych w bazie za pomocą instrukcji *INSERT* lub *UPDATE*. Aby jednak było to mozliwe perspektywa nie może odwoływać się do więcej niż jednej tabeli (nie jest prawdą dla niektórych serwerów), nie może zawierać funkcji agregujących, frazy *GROUP BY* oraz *DISTINCT*.

Użycie perspektyw upraszcza dostęp do danych i pozwala na lepszą kontrolę i ograniczenie uprawnień użytkownika.

Przykładowe zapytanie do omówienia:

```
SELECT u.user_id, ua.firstName, ua.lastName, COUNT(d.assignedDrone_id) as 'Przypisane drony'
FROM ClientUser u
LEFT JOIN UserAccount ua ON u.account_id = ua.account_id
LEFT JOIN User_AssignedDrones d ON d.user_id=u.user_id
GROUP BY u.user_id, ua.firstName, ua.lastName
HAVING COUNT(d.assignedDrone_id) > 3
ORDER BY user_id DESC;
```