Systemy ekspertowe

Część szósta

Realizacja systemów ekspertowych z wykorzystaniem pakietu Sphinx

Autor
Roman Simiński

Kontakt
siminski@us.edu.pl
www.us.edu.pl/~siminski

System ekspertowy jako system z bazą wiedzy

System ekspertowy i inżynieria wiedzy

- *Systemy ekspertowe* są programami umożliwiającymi rozwiązywania problemów w sposób przypominający postępowanie eksperta lub specjalisty z pewnej, zwykle wąsko określonej dziedziny.
- Pod pojęciem *systemu ekspertowego* należy rozumieć zatem taki program komputerowy, który, działając w oparciu o posiadaną szczegółową wiedzę, potrafi wyciągać wnioski oraz sugerować decyzje, rozwiązując specjalistyczne problemy w sposób zbliżony do tego, w jaki czynią to istoty inteligentne.
- Problem konstruowania systemów ekspertowych należy do dziedziny nauki, jaką jest *inżynieria wiedzy*. Obejmuje ona również takie zagadnienia, jak pozyskiwanie wiedzy, jej strukturalizacja oraz kodyfikację i weryfikację, dopasowywanie odpowiednich metod wnioskowania oraz mechanizmów wyjaśnień.

System ekspertowy to nazwa określająca jego funkcjonalność i zakres stosowalności. Systemy ekspertowe są najczęściej realizowane w technologii systemów z *bazą wiedzy*.

System ekspertowy jako system z bazą wiedzy

Ogólne właściwości systemów ekspertowych

Systemy ekspertowe:

- są narzędziem kodyfikacji wiedzy eksperckiej,
- mają zdolność rozwiązywania problemów specjalistycznych, w których duża rolę odgrywa doświadczenie a wiedza ekspercka jest dobrem rzadkim i kosztownym.
- zwiększają dostępność ekspertyzy,
- zapewniają możliwość prowadzenia jednolitej polityki przez centralę firm mających wiele oddziałów,
- poziom ekspertyzy jest stabilny - jej jakość nie zależy od warunków zewnętrznych i czasu pracy systemu,
- jawna reprezentacja wiedzy w postaci zrozumiałej dla użytkownika końcowego,
- zdolność do objaśniania znalezionych przez system rozwiązań,
- możliwość przyrostowej budowy i pielęgnacji bazy wiedzy.

System ekspertowy jako system z bazą wiedzy

Inżynieria wiedzy i akwizycja wiedzy

Proces realizacji systemów ekspertowych różni się od procesu realizacji klasycznych systemów informatycznych. W przypadku tych ostatnich, kompleksowo rozumiany proces realizacji systemu jest przedmiotem *inżynierii programowania*. W przypadku systemów ekspertowych mówi się o *inżynierii wiedzy*:

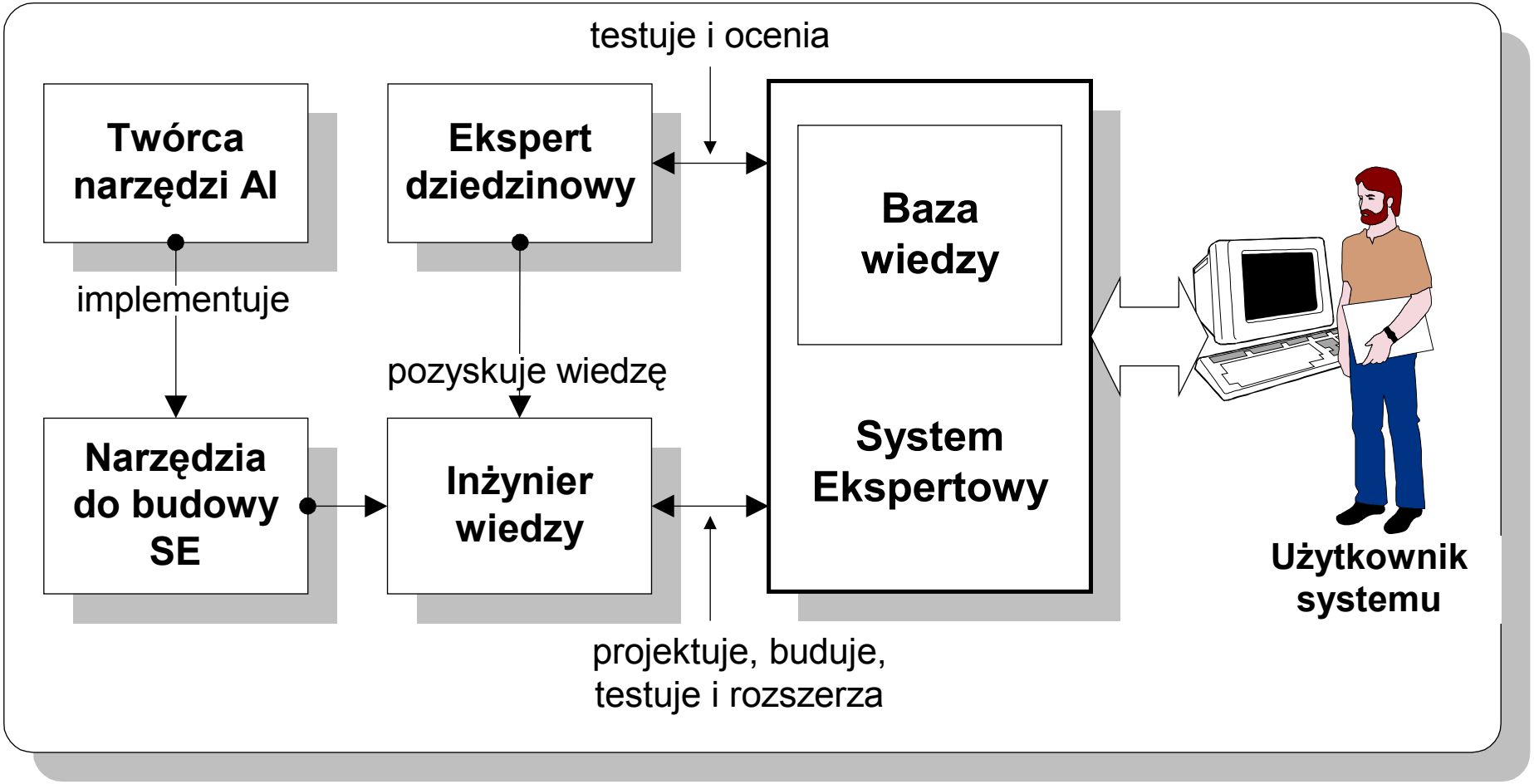
- *Inżynieria wiedzy* (ang. *knowledge engineering*) to dziedzina sztucznej inteligencji zajmująca się projektowaniem i realizacją systemów ekspertowych.

Jednym z kluczowych elementów inżynierii wiedzy jest akwizycja wiedzy. Przyjmijmy następującą jej definicję:

- *Akwizycja wiedzy* (ang. *knowledge acquisition*, *knowledge elicitation*) to proces pozyskiwania, gromadzenia i strukturalizowania wiedzy dziedzinowej niezbędnej do realizacji baz wiedzy systemu ekspertowego.

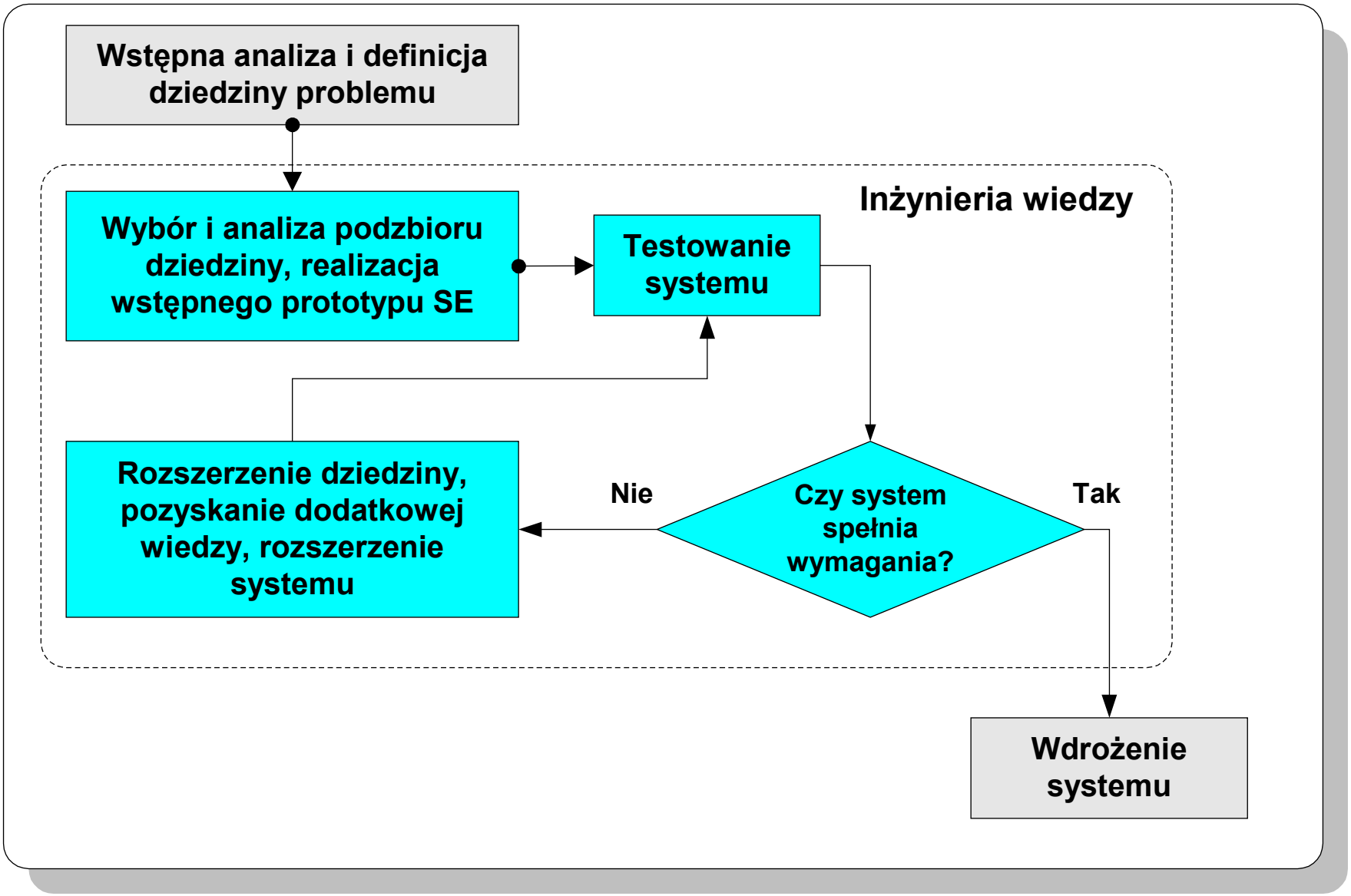
System ekspertowy jako system z bazą wiedzy

Narzędzia i role uczestników procesu realizacji systemu ekspertowego



System ekspertowy jako system z bazą wiedzy

Inkrementacyjny model cyklu rozwojowego systemu ekspertowego



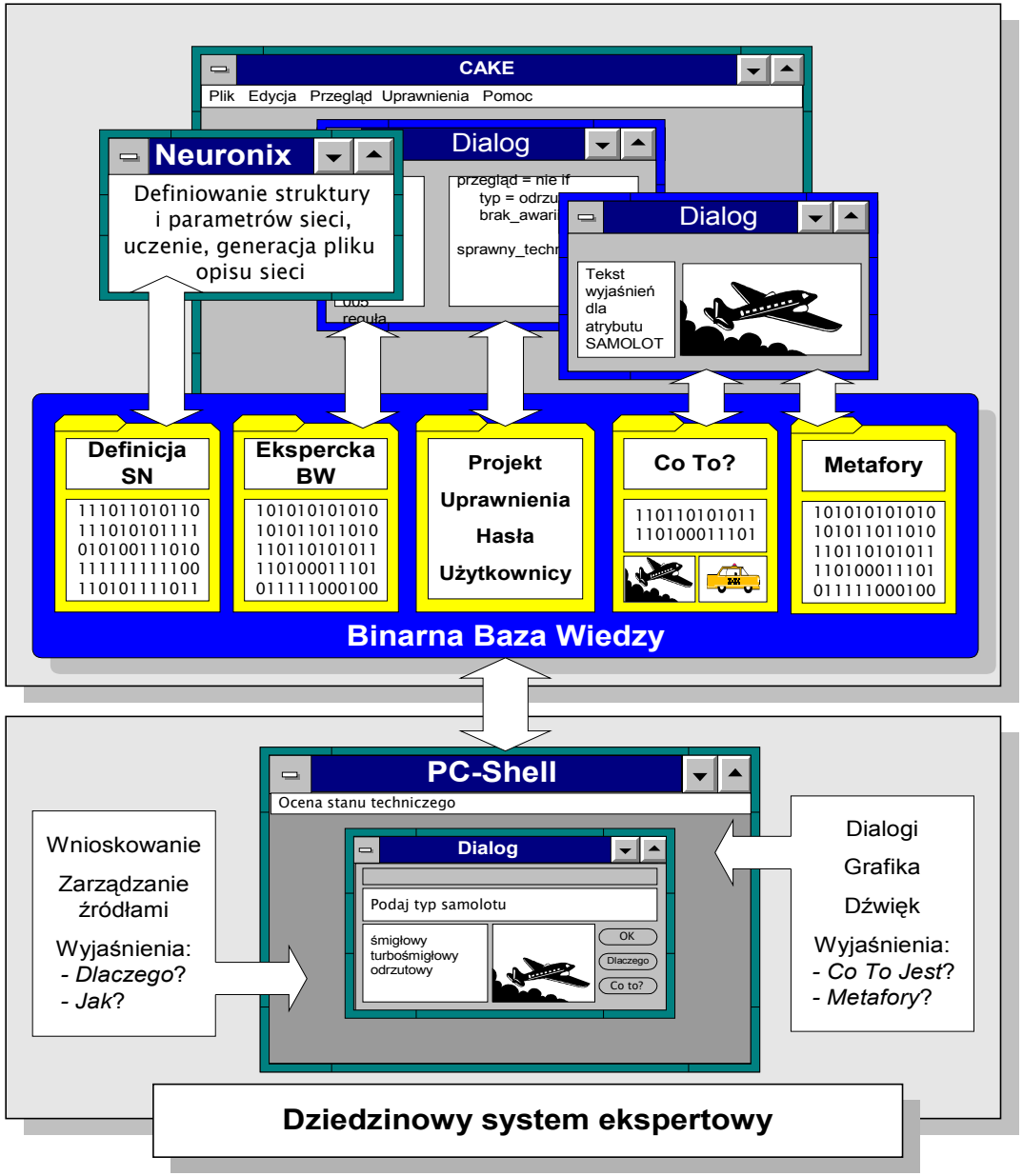
Pakiet Sphinx i system szkieletowy PC-Shell

Ogólne właściwości pakietu Sphinx

- Pakiet Sphinx jest zintegrowanym pakietem oprogramowania z zakresu sztucznej inteligencji.
- Przeznaczony jest do realizacji inteligentnych aplikacji wykorzystujących technologię systemów ekspertowych oraz sieci neuronowych.
- Pakiet jest dziedzinowo–niezależny, typowe zastosowania ukierunkowane są na zagadnienia związane ze wspomaganiem podejmowania decyzji, klasyfikacją, diagnostyką, analizą danych.
- Pakiet składa się z trzech podstawowych systemów:
  - *PC–Shell* — szkieletowy system ekspertowy,
  - *CAKE* — system spomagający realizację baz wiedzy,
  - *Neuronix* — symulator sieci neuronowych.

Pakiet Sphinx i system szkieletowy PC-Shell

Rola elementów pakietu Sphinx



Pakiet Sphinx i system szkieletowy PC-Shell

Właściwości systemu PC-Shell

- Podstawowym elementem pakietu Sphinx jest szkieletowy system ekspertowy PC–Shell, posiadający właściwości hybrydowe, wykorzystujący elementy architektury tablicowej.
- Bazy wiedzy systemu zapisywane są przy użyciu języka opisu bazy wiedzy Sphinx, integrującego w sobie deklaratywny język reprezentacji wiedzy oraz imperatywny język programowania strukturalnego.
- Baza wiedzy zapisywana jest w postaci pliku (lub plików) tekstowych poddawanych procesowi translacji na początku każdej sesji konsultacyjnej

Pakiet Sphinx i system szkieletowy PC-Shell

Właściwości systemu Neuronix

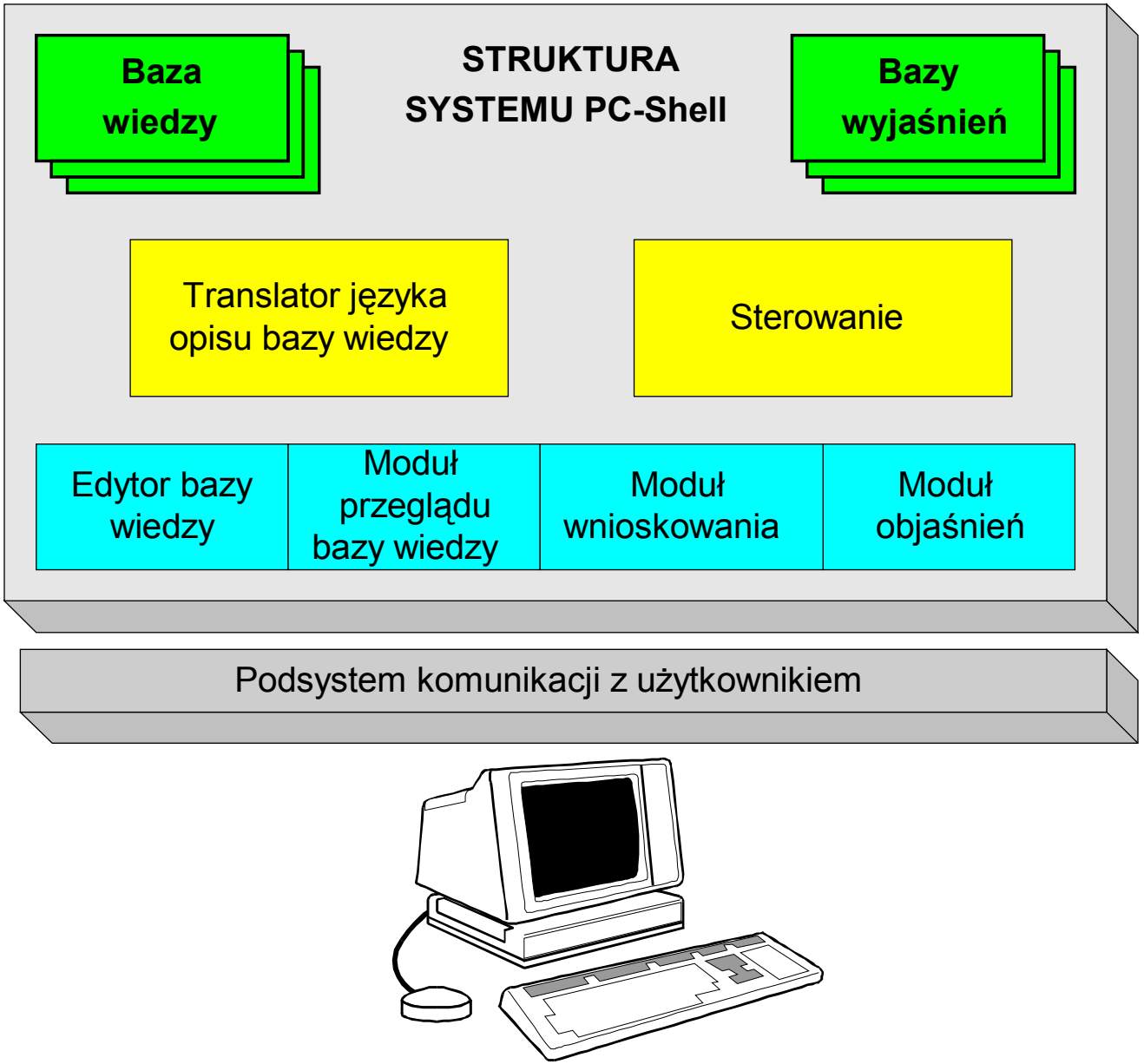
- System *Neuronix* przeznaczony jest do tworzenia samodzielnych aplikacji neuronowych lub neuronowych źródeł wiedzy, które mogą wchodzić w skład hybrydowych aplikacji systemu *PC-Shell*.
- W środowisku systemu Neuronix realizuje się proces definiowania sieci – struktury, parametrów oraz przeprowadza się proces trenowania sieci neuronowej.

Właściwości systemu CAKE

- Zadaniem system CAKE jest wspomaganie procesu realizacji dziedzinowych baz wiedzy dla systemu PC Shell.
- System CAKE oferuje możliwość edycji baz wiedzy bez konieczności bezpośredniego pisania w języku Sphinx, zapewniając wygodny podsystem komunikacji z użytkownikiem, oferując mechanizmy kontroli i weryfikacji wprowadzanych informacji.

System PC-Shell

Architektura



System PC-Shell

Struktura bazy wiedzy systemu PC Shell

Struktura bazy wiedzy	Opis funkcji bloków bazy wiedzy
knowledge base *nazwa*	
sources *opis_plików* **end;**	Definicja plików zawierających źródła wiedzy: ekspercka baza wiedzy, definicja sieci neuronowej, baza wyjaśnień.
facets *opis_faset* **end;**	Definicja atrybutów - ich typów i właściwości, ustalenie wartości przełączników sterujących wnioskowaniem.
rules *opis_reguł* **end;**	Blok opisu reguł zapisanych w postaci klauzul Horna.
facts *opis_faktów* **end;**	Blok opisu faktów zapisanych w postaci trójek Obiekt-Atrybut-Wartość.
control *program* **end;**	Blok programu - sterowanie wnioskowaniem i aktywacją źródeł, pozyskiwanie i wstępne przetwarzanie danych, dostęp do plików baz danych, dynamiczna wymiana danych itp.
end;	

Blok deklaracji źródeł wiedzy

Format opisu źródła:

```
nazwa_źródła :
    type { kb | metaphor | what_is | neural_net };
    file łańcuch_znaków;
```

gdzie *type* służy do specyfikacji typu źródła, jednego z poniższych:

- *kb* - eksperckie bazy wiedzy,
- *neural_net* - sieci neuronowe,
- *metaphor* - bazy danych zawierające wyjaśnienia typu *metafory*,
- *what_is* - bazy danych zawierające wyjaśnienia typu *co to jest*,

natomiast file łańcuch_znaków określa plik, w którym przechowywane jest źródło wiedzy.

System PC-Shell

Przykładowa deklaracja bloku źródeł

```
sources
    decyzja_kredytowa :
        type kb
        file "c:\\bazy\\bw\\decyzja.zw";
    prognoza_finansowa :
        type neural_net
        file "c:\\bazy\\sieci\\prognoza.def";
    metafory :
        type metaphor
        file "c: \\bazy\\bw\\kredyt.dbm";
    coto :
        type what_is
        file "c: \\bazy\\bw\\kredyt.dbw";
end;
```

Ogólna struktura bloku faset

Fasetami określa się tu zbiór deklaracji odnoszących się do wybranych *atrybutów*. Blok faset zawiera wykaz wszystkich atrybutów używanych w bazie wiedzy, wraz z przypisanymi do nich fasetami.

```
facets
  opis_faset
end;

opis_faset:
  atrybut1 [ deklaracje_faset1 ];
  -
  atrybutn [ deklaracje_fasetn ];
```

Przykład definicji atrybutu z fasetami

```
facets

  temperatura_ciała :
    query "Podaj temperaturę ciała:";
    unit "st. C";
    val range < 36, 42 >;
    param { NORMALNA = 36.7, STAN_PODGOR = 37.5 };

end
```

Rodzaje faset opisujących atrybuty

- ***ask***- określa czy system może stawiać pytania dotyczące danego atrybutu. System zadaje pytania jedynie w sytuacji, gdy nie potrafi potwierdzić warunku reguły lub hipotezy wykorzystując fakty i reguły zawarte w bazie wiedzy.
- ***query*** - umożliwia zdefiniowanie przez użytkownika własnej treści zapytań o wartość atrybutu, generowanych przez system.
- ***unit*** - umożliwia zadeklarowanie jednostki miary, w której wyrażane są wartości danego atrybutu, podczas wyświetlania informacji zawierającej dany atrybut, dodatkowo - po wartości - będzie pojawiał się tekst zadeklarowany jako *jednostka_miary*.

Rodzaje faset opisujących atrybuty

- ***val*** - określa zbiór dopuszczalnych wartości danego atrybutu. Wartości mogą być numeryczne lub symboliczne. Do określenia dozwolonych lub niedozwolonych wartości służą następujące deklaracje związane z fasetą *val* : *oneof*, *someof*, *range*, *except*;
- ***param*** - faseta ta umożliwia zadeklarowanie tzw. zmiennych parametrycznych i przypisanie im wartości domyślnych;
- ***picture***, ***sound***, ***video*** - fasety te umożliwiają związanie plików multimedialnych z atrybutem lub jego wartościami. Rysunek jest automatycznie pokazywany, np. gdy pojawia się zapytanie dotyczące atrybutu z którym związany jest rysunek. Dźwięk i animację można odtworzyć po wybraniu odpowiedniego przycisku.

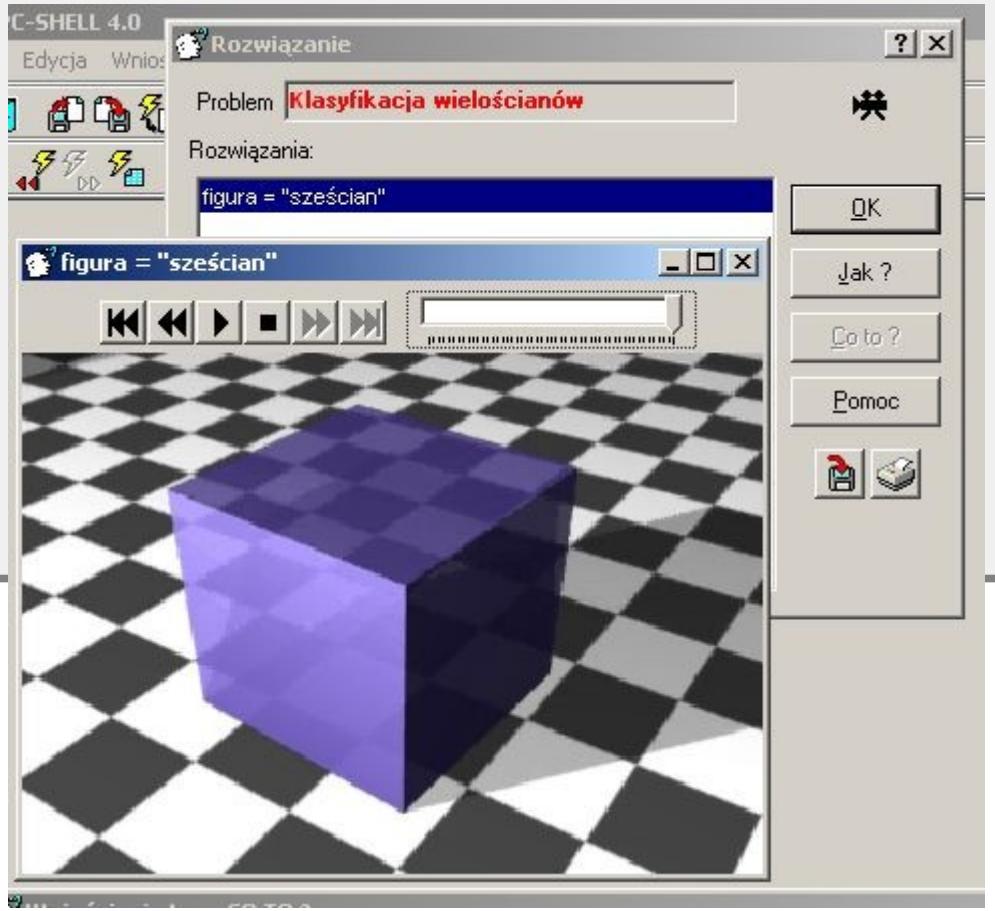
Przykład wykorzystania faset query i picture

```
matowa_powierzchnia_kapelusza:
   query   "Czy powierzchnia skórki kapelusza jest matowa?"
   picture "grzyb22.bmp";
```

System PC-Shell

Przykład wykorzystania faset val i video

```
figura :
  val oneof
  {
    "sześcian",
    "prostopadłościan",
    "ostrosłup",
    "graniastosłup",
    "nieprawidłowa"
  }
  video
  {
    "sześcian.avi",
    "prostopadłościan.avi",
    "ostrosłup.avi",
    "",
    "nieprawidłowa.avi"
  };
```



System PC-Shell

Przykłady definicji atrybutów z użyciem faset

```
kolor_nadwozia :
  query "Proszę podać kolor nadwozia samochodu:" ;
  val oneof { "biały", "czerwony", "niebieski" };

pojemność_silnika :
  query "Proszę podać pojemność silnika:" ;
  unit "cm sześć.";
  val except { < MIN, 600 ), ( 5000, MAX > };

grzyb :
  val oneof { "pieczarka", "muchomor",     "maślak" }
  picture   { "piecz.bmp", "muchomor.bmp","maslak.bmp"};
  sound     { "ok.bmp",    "alarm.bmp",    "ok.bmp" };

ilość_pamięci_RAM :
  query "Podaj przewidywaną ilość pamięci RAM:";
  unit "MB";
  val range < 128, 2048 >
```

Bloki opisu faktów

Blok opisu faktów pozwala na wprowadzenie do treści bazy wiedzy zbioru faktów.

```
facts
  opis_faktów
end;

opis_faktu: trójka_OAW; lub not trójka_OAW;
```

Przykłady faktów:

```
facts
  temperatura_ciała( kowalski ) = 37.5;
  kolor_nadwozia = "biały";
  pojemnosc_silnika = 2500;
  grzyb = "maślak";
end;
```

Bloki opisu faktów

- Blok opisu reguł zawiera listę reguł. Każda z reguł formalnie stanowi odpowiednik klauzuli Horna.
- Konkluzje reguł zapisywane są w postaci trójek Obiekt-Atrybut-Wartość, warunki dodatkowo mogą przyjąć formę wyrażenia relacyjnego lub instrukcji przypisania (więcej informacji na temat dwóch ostatnich postaci warunków zawiera dokumentacja systemu).
- W aktualnej wersji języka, reguły mogą posiadać warunki koniunkcyjne jak i połączone operatorem alternatywy, dodatkowo warunki mogą być grupowane w warunki złożone za pomocą notacji nawiasowej.

System PC-Shell

Bloki opisu reguł

```
rules
 opis_reguł
end;

opis_reguły_prostej (klauzula Horna):
  konkluzja if warunek1, warunek2, ... , warunekN;

opis_reguły_złożonej:
  konkluzja if warunek1, warunek2, ... , warunekN;

[ numer_reguły : ]   konkluzja1  if
                          warunek1 & warunek2 &...& warunekn;

[ numer_reguły : ]   konkluzja2  if
                          warunek1 | warunek2 &...& warunekn;

Operatorowi logicznemu AND odpowiada znak , lub &.
Operatorowi logicznemu OR odpowiada znak |
```

Przykłady reguł prostych

```
typModemu = "Zoltrix 56000 Cobra Voice V.90 PCI" if
    producentModemu = "Zoltrix" &
    jakiModem = "modem wewnętrzny" &
    szybkośćModemu = "56000";

typDrukarki = "Hewlett-Packard Desk Jet 1120C" if
    rodzajDrukarki = "drukarka atramentowa",
    producentDrukarki = "Hewlett-Packard",
    zastosowanieDrukarki = "wydruk tekstu i rysunków",
    jakośćWydruku = "jak najlepsza";

ryzyko_rozwoju_choroby_wieńcowej = "wysokie" if
    skłonności_dziedziczne_choroby_wieńcowej = "występują" &
    miażdżyca_tętnic_wieńcowych = "występuje";

  gatunek = "DOOM METAL" if
    instrument = "gitara elektryczna",
    rytm = "umiarkowany",
    rodzaj_wokalu = "mroczny, ponury",
    klimat = "ponury",
    tempo = "wolne";
```

Przykłady reguł złożonych

```
ryzyko_ubezpieczeniowe = "umiarkowane" if
    ( rodzaj_zabezpieczenia = "autoalarm" |
      rodzaj_zabezpieczenia = "blokada skrzyni biegów" ) &
    ( miejsce_postoju = "garaż" |
      miejsce_postoju = "parking strzeżony" );

rodzaj_sterownika = "SCSI" if
    ( przeznaczenie = "serwer sieciowy" |
      przeznaczenie = "stacja graficzna" |
      przeznaczenie = "wspomaganie projektowania" ) &
      dozwolony_koszt = "wysoki" &
      ilość_współpracujących_urządzań = "duża";
```

System PC-Shell

Numeracja reguł

- System PC-Shell udostępnia dwa rodzaje numeracji reguł: użytkownika (jawna) i automatyczną (niejawna).
- Numeracja użytkownika tworzona jest przez inżyniera wiedzy w opisie bazy wiedzy. Każda reguła powinna otrzymać numer, będący jej jednoznacznym identyfikatorem w obrębie całej bazy wiedzy, uwzględniając również ewentualne źródła wiedzy. Numery reguł muszą być liczbami z przedziału 0-9999.
- System zakłada, że jeśli pierwsza w kolejności reguła ma numer jawny, to pozostałe reguły muszą mieć również przypisane numery. I odwrotnie, jeśli pierwsza reguła nie ma numeru, to żadna z reguł w danej bazie wiedzy nie może mieć przypisanego przez użytkownika numeru. Złamanie którejś z tych zasad spowoduje błąd w czasie translacji bazy wiedzy.
- Jeśli inżynier wiedzy nie nada jawnej numeracji regułom to system *automatycznie przypisze wszystkim* regułom w bazie wiedzy numery, zgodne z ich kolejnością w tekście źródłowym bazy wiedzy.
- Zaleca się stosowanie jawnej numeracji. W przypadku aplikacji z bazami wiedzy ujętymi w formie *źródeł wiedzy*, jawna numeracja jest *obowiązkowa*.

Blok sterowania

- Blok sterujący zawierać może klasyczny program algorytmiczny. Programista ma do dyspozycji bogaty zestaw typów danych oraz instrukcji. Można również definiować własne podprogramy, również rekurencyjne.
- Program składa się ze zbioru instrukcji zawartych w bloku *control*. W ten sposób zachowana została zasada wyraźnego rozdzielenia wiedzy eksperckiej oraz tzw. sterowania.

Blok sterowania - typy danych

- Język oferuje następujące typy danych :
  - całkowite (int, longint),
  - rzeczywiste (float, double),
  - tablicowe (jedno i dwuwymiarowe),
  - rekordowe.

Przykłady deklaracji zmiennych:

```
int Zmienna1;
char C1, C2, STR, Lancuch_znakowy;
float X, Y,Z, Srednia;
float TAB1[10], TAB2[5,10];
```

Blok sterowania - ogólny podział instrukcji

Dostępny zestaw instrukcji programowania obejmuje zarówno klasyczne instrukcje takie jak instrukcje iteracyjne, warunkowe, dostępu do plików, obsługi ekranu, aż do wyspecjalizowanych instrukcji sterujących wnioskowaniem, operujących na bazach wiedzy, zarządzających źródłami.

Instrukcje można podzielić na następujące grupy:

- instrukcje sterujące wykonaniem programu,
- instrukcje inicjujące i sterujące procesem wnioskowania,
- instrukcje operujące na bazie wiedzy,
- instrukcje podsystemu komunikacji z użytkownikiem,
- instrukcje symulatora sieci neuronowej,
- instrukcje związane z parametryzacją baz wiedzy,
- instrukcje obsługujące dostęp do plików,
- instrukcje obsługujące dostęp do baz danych,
- instrukcje dotyczące dynamicznej wymiany danych.

System PC-Shell

Blok instrukcji – przykład prostej sekwencji uruchamiającej wnioskowanie

```
char Text1, Text2, Text3;

run;

createAppWindow;

Text1 := "Nazwa aplikacji";
Text2 := "Informacje dodatkowe\nAutor programu\nWydawca";
Text3 := "© 2006, Autor/Firma";

vignette( Text1, Text2, Text3 );

goal( ”atrybut_celu=X” );
```

System PC-Shell

Blok instrukcji – przykład iteracyjnej sesji wnioskowania

```
int Odp;
char Text1, Text2, Text3;

run;

createAppWindow;

Text1 := "Nazwa aplikacji";
Text2 := "Informacje dodatkowe\nAutor programu\nWydawca";
Text3 := "© 2006, Autor/Firma";

vignette( Text1, Text2, Text3 );

Odp := 1;
while( Odp == 1 )
  begin
    delNewFacts;
    goal( ”atrybut_celu=X” );
    confirmBox( 0, 0, "", "Kontynuować konsultacje?", Odp );
  end;
```

System PC-Shell

Blok instrukcji – przykład iteracyjnej sesji wnioskowania

```
char Text1, Text2, Text3;

run;

Text1 := "Nazwa aplikacji";
Text2 := "Informacje dodatkowe\nAutor programu\nWydawca";
Text3 := "© 2006, Autor/Firma";

createAppWindow;
setAppWinTitle( Text1 );
vignette( Text1, Text2, Text3 );

menu "Menu"
  1. "Wspomaganie doboru procesora"
  2. "Wspomaganie doboru ilości pamięci"
  3. "Koniec"
  case 1:
    delNewFacts;
    goal( ”procesor=X” );
  case 2:
    delNewFacts;
    goal( ”pamiec=X” );
  case 3:
    exit;
end;
```

System PC-Shell

Blok instrukcji — przykład programowania imperatywnego

```
knowledge base silnia
  control
    long K, S, N;
    int Odp;
    char STR, Tekst;
    run;
    createAppWindow;
    precision( 1, 0 );
    Odp := 1;
    while( Odp == 1 )
       begin
           neditBox( 0, 0, 0, 12, "Podaj liczbę n dla n!", N );
           S := 1;
           for K := 1 to N step 1
               begin
                 S := S * K;
               end;
           ntos( S, STR );
           Tekst := "Wartość n! = ";
           strcat( Tekst, STR );
           messageBox( 0, 0, "Komunikat", Tekst );
           confirmBox( 0, 0, "KONTYNUOWAĆ?", "", Odp );
       end;
  end;
end;
```

System PC-Shell

Blok instrukcji — przykład programowania imperatywnego

```
knowledge base nazwa
 control

   record Complex
   begin
      double R, U;  // r - część rzeczywista, u - urojona
   end;

   function addComplex( record Complex L1, record Complex L2, record
                        Complex &Result )
   begin
      Result.R := L1.R + L2.R;
      Result.U := L1.U + L2.U;
   end;

   run;

   record Complex R1, R2, R3;
   R1.R := 1;  R1.U := 2;
   R2.R := 1;  R2.U := -1;

   addComplex( R1, R2, R3 );
 end;
end;
```