

Programowanie w Logice

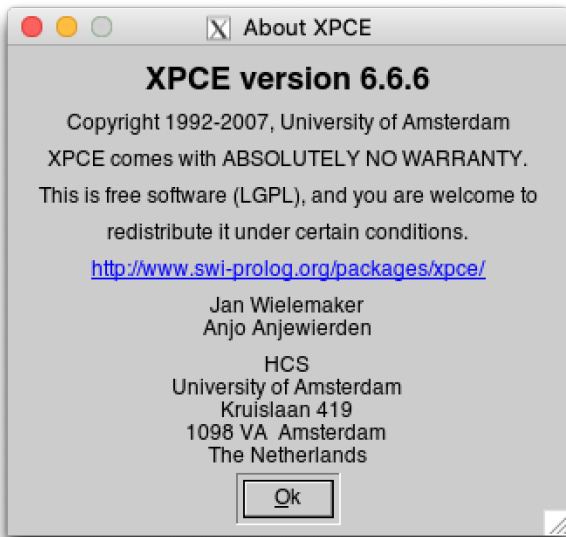
Interfejs graficzny

Przemysław Kobylański



Interfejs graficzny

Biblioteka XPCE



Interfejs graficzny

Biblioteka XPCE

- ▶ Autorami biblioteki są Jan Wielemaker i Anjo Anjewierden.
- ▶ Pierwsza wersja biblioteki powstała w roku 1985.
- ▶ Biblioteka napisana jest w języku C.
- ▶ Wykorzystuje funkcje systemu okienkowego na najniższym poziomie (biblioteka libX11).
- ▶ Korzystanie z biblioteki odbywa się z wykorzystaniem paradygmatu obiektowego (predykaty `new/2`, `send/[2-8]` i `get/[3-8]`).
- ▶ Bogata kolekcja klas oraz możliwość definiowania nowych klas (nawet w programie prologowym).
- ▶ Podręcznik: Jan Wielemaker, Anjo Anjewierden. **Programming in XPCE/Prolog**



Interfejs graficzny

Biblioteka XPCE

Hierarchia 202 klas zdefiniowanych w XPCE:



Interfejs graficzny

Tworzenie, wyświetlanie i usuwanie obiektów graficznych

```
?- use_module(library(pce)).
true.
?- new(B, box(10, 10)).      % zmienna
B = @11846215313815/box.
?- new(@box, box(10, 10)).  % globalna referencja obiektu
true.
?- get(@box, width, X).    % pobranie szerokości obiektu
X = 10.
?- send(@box, width, 100). % zmiana szerokości prostokąta
true
?- get(@box, width, X).
X = 100.
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Tworzenie, wyświetlanie i usuwanie obiektów graficznych

```
?- new(@window, picture).  % utworzenie okna
true.
?- send(@window, open).    % wyświetlenie okna
true.
?- send(@window, display, @box). % wyświetlenie obiektu
true.
?- send(@box, x, 50).      % przesunięcie obiektu
true.
?- send(@box, y, 75).     % przesunięcie obiektu
true.
?- free(@box).            % usunięcie obiektu
true.
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Tworzenie, wyświetlanie i usuwanie obiektów graficznych

```
?- send(@window, display, new(@box, box(40, 20)),
        point(20, 30)).

true.
?- send(@box, fill_pattern, colour(red)).
true.
?- send(@box, radius, 10).
true.
?- send(@window, destroy).
true.
?- get(@box, width, X).      % obiekt nadal istnieje
X = 40.
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Wybrane obiekty graficzne

Klasy obiektów dialogowych umieszczanych w obiekcie z klasy **dialog**:

text_item pole tekstowe

menu menu

slider suwak do wprowadzania wartości całkowitych

button przycisk

- ▶ `new(Obj, Class)` utworzenie obiektu
- ▶ `send(Dialog, append, Obj)` dodanie obiektu do dialogu
- ▶ `send(Obj1, below, Obj2)` dodanie obiektu **Obj1** pod obiektem **Obj2**
- ▶ `send(Obj1, right, Obj2)` dodanie obiektu **Obj1** na prawo od obiektu **Obj2**
- ▶ `send(Obj, Attribute, Value)` ustawienie i
- ▶ `get(Obj, Attribute, Variable)` pobranie atrybutu

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Wybrane obiekty interfejsu

Klasa: **text_item**



Wybrane atrybuty:

label etykieta

selection wpisany tekst

active czy można wpisywać

```
new(TI, text_item(label, selection) )
get(TI, selection, Variable)
send(TI, selection, Atom )
send(TI, active, on/off )
```



Interfejs graficzny

Wybrane obiekty interfejsu

Klasa: **menu**



Wybrane atrybuty:

label etykieta

kind rodzaj: cycle, marked, choice, ...

selection wybrana opcja

active czy można wybierać

```
new(M, menu(label, kind) )
send(M, append, Atom )
get(M, selection, Variable)
send(M, active, on/off )
```



Interfejs graficzny

Wybrane obiekty interfejsu

Klasa: **slider**



Wybrane atrybuty:

label etykieta

low zakres dolny

high zakres górny

selection wybrana wartość z zakresu

active czy można wybierać

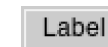
```
new(S, slider(label, low, high, selection) )
get(S, selection, Variable)
send(S, active, on/off )
```



Interfejs graficzny

Wybrane obiekty interfejsu

Klasa: **button**



Wybrane atrybuty:

label etykieta

message akcja po kliknięciu

active czy można kliknąć

```
new(B, button(label) )
send(B, message, Action)
new(B, button(label, message) )
send(B, active, on/off)
```



Interfejs graficzny

Rysowanie drzew

```
?- use_module(library(pce)).  
?- new(@window, picture).      % utworzenie okna  
?- send(@window, open).       % wyświetlenie okna  
?- new(@n1, node(text(a))).    % utworzenie węzła  
?- new(@tree, tree(@n1)).     % utworzenie drzewa  
?- send(@window, display, @tree). % narysowanie drzewa
```



Interfejs graficzny

Rysowanie drzew



Interfejs graficzny

Rysowanie drzew

```
?- new(@n2, node(text(b))).    % nowy węzeł  
?- send(@n1, son, @n2).       % podwiązanie syna  
?- new(@n3, node(text(c))).    % kolejny węzeł  
?- send(@n1, son, @n3).       % drugi syn
```



Interfejs graficzny

Rysowanie drzew



Interfejs graficzny

Rysowanie drzew

```
?- send(@tree, direction, vertical). % zmiana kierunku
```



Interfejs graficzny

Rysowanie drzew



Interfejs graficzny

Rysowanie drzew

```
?- send(@tree, neighbour_gap, 20). % zwiększenie odstępu
```



Interfejs graficzny

Rysowanie drzew



Interfejs graficzny

Definiowanie własnej klasy

Example (Ułamki)

```
:- pce_begin_class(fraction, object, "Ułamek").

variable(numerator, int, get, "Licznik").
variable(denominator, int, get, "Mianownik").

initialise(F, Num:int, Den:int) :->
    send(F, slot, numerator, Num),
    send(F, slot, denominator, Den).

numerator(F, Num:int) :->
    "Ustalenie wartości licznika"::
    send(F, slot, numerator, Num).
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Definiowanie własnej klasy

Example (Ułamki cd.)

```
value(F, Value:real) :-<-
    "Pobranie wartości"::
    get(F, numerator, Num),
    get(F, denominator, Den),
    Value is Num/Den.

:- pce_end_class.
```

Przykładowy dialog:

```
?- new(@u1, fraction(1, 3)).
?- new(@u2, fraction(2, 3)).
?- send(@u1, add, @u2).
?- get(@u1, value, X).
X = 1.
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Definiowanie własnej klasy

Example (Ułamki cd.)

```
denominator(F, Num:int) :->
    "Ustalenie wartości mianownika"::
    send(F, slot, denominator, Num).

add(F, Arg:fraction) :->
    "Dodanie drugiego ułamka (można lepiej)"::
    send(F, numerator,
        F?numerator*Arg?denominator+
        F?denominator*Arg?numerator),
    send(F, denominator,
        F?denominator*Arg?denominator).
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Samodzielna aplikacja

- ▶ Aplikacja otwiera okno zawierające dwa przyciski.
- ▶ Najeżdżenie myszką na przycisk powoduje pojawienie się opisu działania danego przycisku.
- ▶ Kliknięcie na przycisku **Hello** powoduje wydrukowanie napisu *Hello!*.
- ▶ Kliknięcie na przycisku **Quit** powoduje zakończenie pracy aplikacji.



◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Interfejs graficzny

Samodzielna aplikacja

```
% main.pl
:- use_module(library(pce)).
:- use_module(library(help_message)).

main :-
    pce_main_loop(open_window).

open_window(_Argv) :-
    new(Window, dialog('')),
    new>Hello, button(hello,
                    message(@prolog, writeln, 'Hello!'))),
    new(Quit, button(quit,
                    message(Window, destroy))),
    send>Hello, help_message, tag, 'kliknij aby sie przywitać'),
    send(Quit, help_message, tag, 'kliknij aby zakończyć'),
    send(Window, append, Hello),
    send>Hello, left, Quit),
    send(Window, open).
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↻

Interfejs graficzny

Samodzielna aplikacja

Zapisanie stanu maszyny Prologu:

```
$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free softw
Please run ?- license. for legal details.
```

For online help and background, visit <http://www.swi-prolog.org>
For built-in help, use ?- help(Topic). or ?- apropos(Word).

```
?- [main].
true.
```

```
?- qsave_program(main, [goal(main)]).
true.
```

```
?- halt.
$ ./main
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↻

Interfejs graficzny

Samodzielna aplikacja

Kompilacja poleceniem swipl:

```
# Makefile
all: main

main: main.pl
    swipl --goal=main -o main -c main.pl

clean:
    rm -f main *~

$ make
swipl --goal=main -o main -c main.pl
$ ./main
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↻

Interfejs graficzny

Przykładowa aplikacja

- ▶ W repozytorium znajdziecie program **TabSem** dla metody tablic semantycznych (metoda dowodzenia twierdzeń).
- ▶ O metodzie tabel semantycznych można poczytać między innymi w książce Raymond M. Smullyan "First-Order Logic".
- ▶ Pliki źródłowe:
 - `tabsem.pl` metoda tabeli semantycznej
 - `gui.pl` interfejs graficzny
 - `main.pl` załadowanie definicji i uruchomienie aplikacji
- ▶ Kompilacja poleceniem `make`
- ▶ Uruchomienie poleceniem `./main`

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↻

Interfejs graficzny

Przykładowa aplikacja

