Mini-Kursus Objekte mit XPCE

Lidia Khmylko, Wolfgang Menzel

21.2.2011

Zauberwort Objektorientierung

Objektorientierung ist ein Sammelbegriff für ganz verschiedene Konzepte.

- Polymorphie
- Modularisierung
- Kapselung
- message passing
- Vererbung
- späte Bindung

(Alles über Objekte: http://www.faqs.org/faqs/object-faq/part1/)

Objektorientierung entzaubert

- Für uns bedeutet Objektorientierung im wesentlichen:
  Programmierung mit Objekten!
- Was unterscheidet ein Objekt vom ‘normalen’ Prolog?
- Prolog bietet bereits
  - Polymorphie (generische Prädikate)
  - Modularisierung (Schlüsselwort `module`)
  - Kapselung (Schlüsselwort `import`, `export`)
- Und noch?

“It’s Prolog, but not as we know it”

1. Die Erweiterung XPCE stellt echte Objekte zur Verfügung.
2. Objekte haben einen internen Zustand.
3. There is no point 3.

(Alles über XPCE: http://gollem.swi.psy.uva.nl:8080/)

Was bedeutet das?

- Zur Erinnerung:
  Der Gültigkeitsbereich einer Variablen ist die Klausel.
  Variablen werden durch Unifikation gebunden.
  Die Variablenbindung endet an der Klauselgrenze, oder wenn sie durch Backtracking rückgängig gemacht wird.
- Ab jetzt gilt:
  Ein Objekt lebt, solange es nicht ausdrücklich gelöscht wird.
  Objekte können sich verändern, wenn man entsprechende Botschaften schickt.
  Diese Veränderungen werden *nicht* durch Backtracking aufgehoben.

```
X = hugo ;
No
```

Zustandsorientiertes Programmieren mit XPCE

- Unsere bisherigen Programme waren zustandsfrei.
- Zwei Aufrufe derselben Anfrage liefern daher dasselbe Ergebnis:

```
?- vater(X,klaus).
  X = hugo ;
  No
?- vater(X,klaus).
  X = hugo ;
  No
```

Zustandsorientiertes Programmieren mit XPCE

- Bei der Arbeit mit Objekten kann das ganz anders sein:

```
?- get(@counter, value, X).
  X = 23
?- send(@counter, step).
  Yes
?- get(@counter, value, X).
  X = 24
```

- Vergleichbar ist im herkömmlichen Prolog nur das umständliche `assert/retract`.

Und so wird's gemacht:

XPCE erweitert Prolog um vier Mechanismen:

- `new` erzeugt ein neues Objekt
- `get` fragt den Zustand des Objektes ab
- `send` sendet dem Objekt eine Botschaft
- `free` löscht ein Objekt

Probieren Sie es aus!

Wir verwenden das klassische Einführungsbeispiel: Programmierobjekte werden verwendet, um GUI-Objekte zu programmieren.

```
?- new(@dialog, dialog('Mein Dialog')).
   Yes
?- send(@dialog,open).
   Yes
?- new(@line,text_item('Eingabe')).
   Yes
?- send(@dialog,append(@line)).
   Yes
```

Unangekündigte Klassenarbeit!

Beantworten Sie die folgenden Fragen:

- Wozu dient der Aufruf `get(@textitem,selection,S).` ?
- Beweisen Sie anhand dieses Aufrufes, daß das XPCE-System zustandsorientiert ist!
- Erzeugen Sie einen Button durch `new(@button,button('OK')).` Wie kann man diesen Button dem Dialog hinzufügen?
- Löschen Sie das Button-Objekt. Verschwindet es auch aus der graphischen Darstellung?
- Wie programmiert man diesen Button?

(Alles über XPCE: http://gollem.swi.psy.uva.nl:8080/)

Ein neuer, verbesserter Button

- Versuchen Sie dieses:

```
new(B,
    button('Press me!',
           message(@pce, write_ln, 'Click.'))),
send(B, open).
```

- Die Eigenschaft ‘message’ eines Buttons bestimmt, was er tut.
- Die ‘message’ ist eine ganz normale Botschaft, wie man sie an ein Objekt schicken kann.
- In diesem Fall schicken wir XPCE persönlich die Botschaft “Schreibe!”

Das sehen wir uns noch einmal in Zeitlupe an. . .

- Im letzten Beispiel haben wir allerlei Aufrufe ineinander geschachtelt.
- Eigentlich muß es heißen:

```
new(B, button('Bite me!')).
  B = @535833/button
new(M, message(@prolog, write_ln, 'Ouch!')).
  M = @543286/message
send(@535833, message, @543286).
  Yes.
send(@535833, open).
  Yes.
```

- Die tatsächlichen Objektnamen können hiervon geringfügig abweichen.

Dynamische Dialoge

- Erzeugen Sie einen Button, der neue Buttons erzeugt!
- Hinweis: `@prolog` erwartet normale Prolog-Anfragen als Botschaften.
- Hinweis: Die Klasse `dialog` besitzt auch eine Methode `display`, die auch Positionsangaben erlaubt.
- Hinweis: all dies und sehr viel mehr erfahren Sie unter `http://gollem.swi.psy.uva.nl:8080/`
- Hinweis: Bookmarken Sie diese Seite. Sie werden Sie noch 2487mal brauchen.

Dynamische Dialoge

Hier ist eine mögliche Lösung:

```
new_button :-
  new(B, button('Press me!')),
  send(B, message, message(@prolog, new_button)),
  X is random(100),
  Y is random(100),
  send(@d, display(B, point(X,Y))).

new(@d, dialog), send(@d, open), new_button.
```

Dynamische Dialoge

Was tut dieses Programm?

```
new_button(Old) :-
  free(Old),
  new(B, button('Catch me if you can!')),
  send(B, message, message(@prolog, new_button, B)),
  X is random(100),
  Y is random(100),
  send(@d, display(B, point(X,Y))).

go:-new(@d, dialog), send(@d, open), new_button(_).

:-go.
```

Dialoge und mehr

Finden Sie anhand der Klassenreferenz heraus

- wie man zwei Buttons automatisch anordnet
- wie man ein Fenster mit buntem Hintergrund erhält
- was Dialoge alles enthalten können
- wie man beliebige Figuren (z.B. Kreise) darstellt
- wie man beliebige Objekte auf Mausklick reagieren läßt
- ...