IMPLEMENTATION OF EXPRESSIONS IN ADOXX 1.0: INTRODUCTION

OPEN | MODEL Initiative

www.openmodels.at

universität wien

DKE

o.Univ.-Prof. Dr. Dimitris Karagiannis

Modelling Method Implementation Instruments

Modelling Method / Conceptualisation	Modelling Language			Modelling Procedure	Mechanisms and Algorithms		
	Notation	Syntax	Semantic		Basis Functions	Script Functions	External Access
Particularisation							
Implementation	Model Type: Method-GraphRep, Icons	Model Type: ADOxx Library Language			Functional Description Table		
	Object: GraphRep	Object: ADOxx Library Language	Object: Class Hierarchy		Configuration Description	Scripting and Expression	ADO Web-Service ADOscript API, Java API)
	Attribute: GraphRep	Attribute: ADOxx Library Language					

PART - 2

Platform Specific Development Environment for the Implementation Process

1. ADOxx Platform
2. ADOxx Method Development Environment
   1. Implementation in ADOxx Library Language (ALL)
   2. Implementation in ADOxx Method Development Tool
3. ADOxx Modelling Language Implementation
   1. ADOxx Model Hierarchy
   2. ADOxx Class Hierarchy
   3. ADOxx GraphRep
4. ADOxx Mechanisms and Algorithms
   1. Basis Functionality
   2. **ADOscript**
   3. External Access
      1. ADOscript
      2. ADO Web-Service

Expressions
Gegenüberstellung AdoScript vs. Expressions

AdoScript	Expressions
• Erlaubt Einbindung externer Funktionalität	• Keine Einbindung externer Funktionalität
• Lese- und Schreibrechte auf die meisten Attribute	• Können auf die meisten Attribute lesend zugreifen, schreiben können Sie nur den eigenen Attributwert
• Müssen vom Benutzer explizit ausgelöst werden	• Werden vom Client automatisch ausgewertet
• Können Expressions einbetten	• N/A
• Können nicht vom Modellierer verändert werden	• Können vom Benutzer editiert werden sofern sie nicht „fix“ sind
• Beliebige Komplexität	• Meistens einfacher als AdoScripts

Expressions
Operatoren (1)

Logische Op.	**AND, OR, NOT**	Boolsche Ausdrücke
Vergleichsop.	**< > <= >= = <> !=**	Größer, kleiner, gleich, ungleich
Arithmetische Op.	**+ - * / - (unär)**	
String Op.	**s + t**	Konkatenieren der Strings s und t
	n * s	Replizierung: String s wird n-mal repliziert
	s / t	Anzahl: wie oft kommt String s in String t vor
	s SUB i	Das i-te Zeichen in String s
	LEN s	Länge des Strings s

Expressions
Operatoren (2)

Konversions-op.	**STR val**	Die Stringrepräsentation des Werts val
	VAL str	Numerische Repräsentation des Strings str
	CMS measure **PTS measure**	Konvertierung einer Maßeinheit (in cm oder points) in eine reelle Zahl (z.B.: CMS 3.5cm → 3.5).
	CM real **PT real**	Konvertierung einer reellen Zahl in eine Maßeinheit (in cm oder points; z.B.: CM 3.5 → 3.5cm).
	uistr(val, n)	Konvertierung einer reellen Zahl in einen String im lokalen Format (OS) mit n Ziffern.
	uival(str)	Konvertierung eines Strings im lokalen Format (OS) in eine reelle Zahl.
Sequenzop.	**,**	Der Kommaoperator wird verwendet um eine Sequenz von Expressions zu definieren. Das Resultat ist immer der Wert der letzten Expression.

Expressions
Vordefinierte Funktionen (1)

Arithmetische Fkt.	`abs(x)`
	`max(x, y) min(x, y)`
	`pow(x, y) sqrt(x)`
	`exp(x)`
	`log(x) log10(x)`
		Arithmetische Funktionen

	`sin(x) cos(x) tan(x)`
	`asin(x) acos(x) atan(x)`
	`sinh(x) cosh(x) tanh(x)`
		Trigonometrische Funktionen

	`random()`	Zufallszahl 0 >= n < 1
	`round(x)`	Kaufmännisches Runden, d.h. wenn Nachkommazahl >= 0.5
	`floor(x) ceil(x)`	Auf- bzw. abrunden

Expressions
Vordefinierte Funktionen (2)

String-fkt.	`search(source, pattern, start)`	Der String source wird nach string pattern durchsucht, beginnend bei start (0-basiert), returniert index oder -1
	`bsearch(source,pattern,start)`	Suche beginnend beim Ende des Suchstrings (rückwärts)
	`copy(source, from, count)`	Kopiert count Symbole aus Source beginnend bei from (0-basiert)
	`replall(source, pattern, new)`	Alle Vorkommen von pattern in source werden durch new ersetzt
	`lower(source)`	Alles wird in kleine Buchstaben konvertiert
	`upper(source)`	Alles wird in große Buchstaben konvertiert
	`mstr(string)`	Der String wird in „“ gesetzt und Sonderzeichen werden maskiert

Expressions
Vordefinierte Funktionen (3)

Listen Fkt.	**tokcnt(source[,sep])**	Anzahl der Token in source die durch sep (default Leerzeichen) getrennt sind
	tokcat(source1, source2 [,separator])	Konkatenieren zweier Listen
	tokunion(source1, source2[, separator])	Union zweier Listen
	tokisect(source1, source2 [, separator])	Durchschnitt zweier Listen
	tokdiff(source1, source2 [, separator])	Differenz zweier Listen
Farb-fkt.	**rgbval(colorname)**	24bit RGB-Wert des Farbnamens.
	rgbval(r, g, b)	Berechnung des RGB-Werts für die angegebenen Farbwerte.

Expressions
Kontrollstrukturen

Anweisungen	**`set(var, expr)`**	Expr soll in var gespeichert werden. Variable var wird implizit angelegt.
	`cond(cond1, expr1, ..., expr_else)`	Evaluiere cond1, falls wahr retourniere expr1, falls falsch evaluiere die nächste Bedingung bzw. retourniere expr_else
	`while(cond, loopexpr[, resultexpr])`	Während cond wahr ist, evaluiere loopexpr. Retourniere resultexpr.
	`fortok(varname, source, sep, loopexpr [, resultexpr])`	Für jedes Element in der Liste source evaluiere loopexpr. Das derzeitige Element wird in varname gespeichert. Die Listenelemente werden durch sep getrennt. Retourniere resultexpr.

Expressions
Fehlerbehandlung, Typprüfung

Fehler-behandlung	`try(expr, failexpr)`	Retourniert expr, falls die Expression ausgeführt werden kann, andernfalls wird failexpr zurückgegeben
Typprüfung	`type(expr)`	Retourniert den Typ der angegeben Expression expr. Mögliche Werte: "string", "integer", "real", "measure", "time", "expression„ oder "undefined„.

Ausdrücke (=Expressions) in AdoScript

Typen von Ausdrücken

- **Core Expressions:**
  - Werden verwendet um Attribute vom Typ EXPRESSION zu definieren
  - Können auf Funktionen für Core Expressions zugreifen
- **AdoScript Expressions:**
  - Werden innerhalb von AdoScript Code verwendet
  - Können in Funktionen ausgelagert werden
  - Können auf ausgelagerte Funktionen zugreifen (definiert durch keyword FUNCTION

Core Expressions
Funktionen für Core Expressions

- Die aufgelisteten Funktionen können in Core Expressions verwendet werden

```
aval()
avalf()
maval()
paval()
pavalf()
irtmodels()
irtobjs()
profile()
ctobj()
cfobj()
conn()
```

```
rcount()
row()
rasum()
prasum()
allobjs()
aql()
prevsl()
nextsl()
```

```
asum()
amax()
awsum()
pmf()
class()
mtype()
mtclasses()
mtrelns()
allcattrs()
alliattrs()
allrattrs()
```

- Weiters können auch alle LEO Operatoren und Funktionen verwendet werden

Core Expressions
Attribute vom Typ EXPRESSION

- Ein Expression Attribut beinhaltet sowohl eine Formel als auch den berechneten Wert
- Es gibt zwei Formen der Verwendung: fixe- und editierbare Expressions
- Für fixe Expressions wird die Formel im default Attributwert gespeichert
- Wenn bei der Berechnung einer Formel ein Fehler passiert wird eine Fehlernachricht zurückgegeben
- Wenn eine inter-Modell Expression nicht evaluiert werden kann (es muss auf ein Modell zugegriffen werden, das nicht geladen worden ist), wird das letzte gültige Resultat angezeigt.
- Expression Attribute werden immer dann evaluiert, wenn ein Ereignis eintritt, das den Wert ändern könnte. Die Änderungen werden sofort im Userinterface angezeigt.

Core Expressions
Attribute vom Typ EXPRESSION

Definition von Expressions als Attribut

Syntax

```
ExprDefinition:    EXPR type:ResultType
                        [format:FormatString]
                        expr:[fixed:]CoreExpression
        ResultType :            double | integer | string |
          time
```

Beispiel

```
EXPR type:string expr:("\"Name = \" + aval(\"Name\")")
```

AdoScript Expressions
Einsatz

- Expressions können direkt als Argumente bei Aufrufen verwendet und so direkt in AdoScript Code eingebettet werden.
- Runde Klammern () werden verwendet um die Argumente einer Expression abzugrenzen.

Beispiel

```
SET n:(copy (vn, 0, 1) + ". " + nn)

IF ( cond( type( n ) = "integer", n = 1, 0 ) )
{
      ...
}

EXECUTE ("SET n:(" + n + ")")
```

- Expressions können auch in dedizierte Funktionen ausgelagert werden um Wiederverwendbarkeit zu erreichen

AdoScript Expressions
Funktionen in AdoScript

Mit dem keyword FUNCTION ist es möglich LEO Ausdrücke als wiederverwendbare Funktionen zu definieren und anderswo aufzurufen.

Syntax

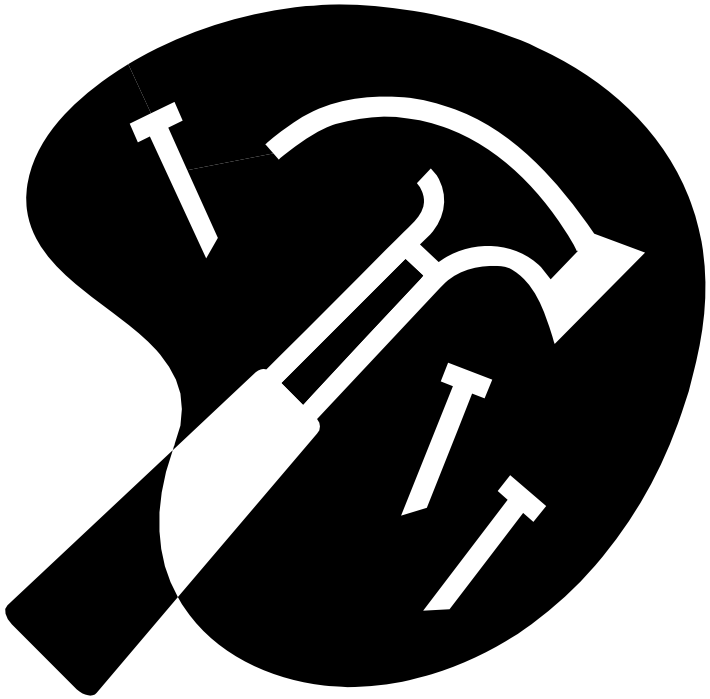
```
FunctionDefinition  ::= FUNCTION functionName[:global]
                        { FormalFuncParameter }
                        return:expression .
FormalFuncParameter ::= paramName:TypeName .
TypeName            ::= string | integer | real | measure |
                        time | expression | undefined .
```

Beispiel

```
FUNCTION helloWorld world:string
    return:("Hello " + world + "!")

SET hello:(helloWorld("world"))
CC "AdoScript" INFOBOX (hello)
```

Übung



Core Expressions
Übungen

- Erstellen Sie ein Expression Attribut für die Klasse „Task“
  - Name des Attributs: EXP
  - Binden Sie das Attribut in das Notebook für die Klasse ein
  - Die Expression soll den Wert des Attributs „Name“ auslesen und anzeigen
- Ändern Sie die Expression:
  - Die Expression soll den Namen x-mal ausgeben wobei x eine Zahl ist die im Attribut „Beschreibung“ eingetragen wird
  - Ändern Sie die Expression so, dass diese zwei Integerwerte miteinander multiplizieren kann
  - Erstellen Sie ein weiteres Objekt. Schreiben Sie hier eine Expression die 1000 mal den Namen des Objektes ausgibt.
- Exportieren Sie das Modell als ADL. Sehen Sie sich an wie Expression Attribute gespeichert werden

AdoScript Expressions
Übungen

- Schreiben Sie eine Funktion die für eine ganze Zahl n die Fakultät berechnet und den Wert als Integer retourniert

- Machen Sie die Funktion global verfügbar und rufen Sie sie auf

- Schreiben Sie ein AdoScript das:
  - für ein ausgewähltes Objekt das Attribut „Beschreibung“ ausliest
  - für den Inhalt der „Beschreibung“ die Fakultät berechnet und ausgibt
  - mögliche Fehler abfängt (im Attribut „Beschreibung“ könnte auch etwas anderes stehen als eine ganze Zahl)
  - das Ergebnis ausgibt

Aufgaben - Expression

Erstellen Sie ein neues Expression Attribut „Anzahl der verbundenen Objekte“ in der Klasse „Task“.

a) Zeigen Sie das Attribut im Notebook an

b) Erstellen Sie eine Expression, die die Anzahl der ein- und ausgehenden Beziehungen der Klasse „verbindet“ zählt.

Erstellen Sie ein neues Expression Attribut „Farbe der Aggregation“ in der Klasse „Task“.

a) Zeigen Sie das Attribut im Notebook an

b) Erstellen Sie eine Expression, die die Beziehungsklasse „ist innerhalb“ auswertet.

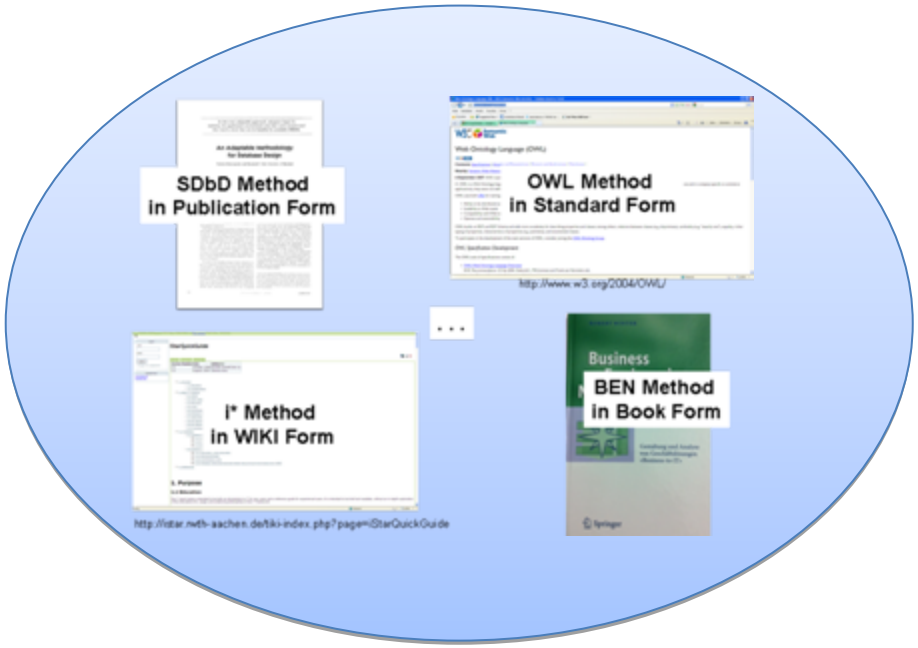
c) Ermitteln Sie die über die Beziehung „ist innerhalb“ verbundene Instanz der Klasse „Aggregation“.

d) Ermitteln Sie den Wert des Attributes „Farbe“

e) Adaptieren Sie das GraphRep der Klasse Task so, dass die Füllfarbe der ermittelten Farbe entspricht.

Aufgaben – Meine Methode

Startpunkt



Aufgabe:

- Anlegen von Expression Attributen
- Verwenden und Auswerten von Expression Attributen