



Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

# Erweiterung von SECONDO

## Attributdatentypen

Thomas Behr

Fakultät für Mathematik und Informatik  
Datenbanksysteme für neue Anwendungen



**FernUniversität in Hagen**

2014

©2014 FernUniversität in Hagen



# Attributdatentypen

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- Werte können in Relationen gespeichert werden
- sind von der Klasse `Attribute` abgeleitet
- haben bestimmte Anforderungen an die Klasse
- müssen verschiedene Funktionen implementieren
- werden der Kind `DATA` hinzugefügt
- können undefiniert sein
- variabel große Attributklassen verwenden FLOBs



- Dokument beschreibt den Umbau eines bestehenden Typs in einen Attributdatentyp
- Wenn Typ von Anfang an als Attributdatentyp geplant ist:  
Verwendung generischer Typkonstruktoren (Vortrag Relationale Algebra)



# Anforderungen an die Klasse

Erweiterung  
von `SECONDO`

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- ist von `Attribute` abgeleitet
- darf keine Zeiger verwenden (dies schließt `Member`, die Zeiger enthalten (z.B. `string`) ebenfalls aus)
- Standardkonstruktor (der ohne Argumente) darf nichts tun
- weiterer Konstruktor, der Initialisierungen durchführt, ist notwendig
- alle Constructoren (außer dem Standardkonstruktor) rufen den `Attribute(bool)`-Konstruktor auf<sup>1</sup>
- es ist stets auf undefinierte Werte zu prüfen

---

<sup>1</sup>der boolesche Parameter gibt an, ob das Attribut definiert ist



# Klassenfunktionen

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

Zur Unterstützung von Operationen der relationalen Algebra, sind folgende Funktionen zu implementieren:

- `virtual int NumOfFLOBs() const`  
gibt die Anzahl der FLOB-Member zurück
- `virtual Flob* getFLOB(const int i)`  
gibt einen Zeiger auf den i-ten Flob zurück
- `int Compare( Attribute* arg) const`  
Vergleicht das `this`-Objekt mit `arg`
- `bool Adjacent(const Attribute* arg) const`  
Prüft auf Nachbarschaft zwischen `this` und `arg`
- `size_t Sizeof() const`  
Gibt die Speichergröße zurück
- `size_t HashValue() const`  
berechnet einen Hashwert
- `void CopyFrom(const Attribute* arg)`  
übernimmt die Werte von `arg`
- `Attribute* Clone() const`  
erzeugt eine Tiefenkopie



# Umbau eines vorhandenen Typs zum Attributdatentypen

Erweiterung  
von `SECONDO`

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- keine Pointer in der Klasse!
- leite von `Attribute` ab
- Standardkonstruktor darf nichts tun
- Standardkonstruktor niemals (außer in Cast-Funktion) aufrufen
- andere Konstruktoren müssen `Attribute (bool)` aufrufen
- **alle** notwendigen Funktionen definieren
- überall auf undefinierte Werte achten



# Beispiel: `SCircle` $\rightarrow$ `ACircle`

Erweiterung  
von `SECONDO`

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

**Beispiel**

Analyse

Änderungen

Funktionen

Einbindung

**Beschreibung, wie aus einem normalen `SECONDO`-Typ (`scircle`, siehe Vortrag Algebraimplementierung) ein Attributdatentyp wird.**



# Analyse der vorhandenen Klasse

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- Standard-Konstruktor tut nichts ✓
  - Anderer Konstruktor vorhanden ✓
  - Klasse verwendet keine Pointer ✓
  - bislang keine eigene `defined`-Umgebung ✓
- ⇒ Gute Voraussetzungen ☺





# Notwendige Änderungen an der Klasse

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

```
class ACircle: public Attribute{
public:
    ACircle() {} // Standardkonstruktor tut nichts
    ACircle(const double _x, const double _y, const double _r):
        Attribute(true),
            x(_x), y(_y), r(_r) {}
    ~ACircle(){}
    static const string BasicType(){ return "acircle"; }
    static const bool checkType(const ListExpr list) {
        return listutils::isSymbol(list, BasicType());
    }

    double perimeter() const{
        return 2*M_PI*r;
    }

    double getX() const{ return x; }
    double getY() const{ return y; }
    double getR() const{ return r; }
    virtual int NumOfFLOBs () const;
    virtual Flob * GetFLOB ( const int i );
    int Compare ( const Attribute * arg ) const;
    bool Adjacent ( const Attribute * arg ) const;
    size_t Sizeof () const;
    size_t HashValue () const;
    void CopyFrom ( const Attribute * arg );
    Attribute * Clone () const;
private:
    double x;
    double y;
    double r;
};
```



# Implementierung der Attribut-Funktionen

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

**Funktionen**

Einbindung

Viele der nachfolgenden Funktionen sind sehr kurz und können somit auch direkt in der Klassen-Deklaration implementiert werden.



# NumOfFLOBs und GetFLOB

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

Die Klasse `ACircle` besitzt keine `FLOB`-Member. Daher sollte die Funktion `NumOfFLOBs` auch `0` zurückliefern. Damit darf die `GetFLOB`-Funktion nicht aufgerufen werden. Die Implementierungen sind daher:

```
int ACircle::NumOfFLOBs() const{
    return 0;
}

Flob* ACircle::GetFLOB(const int i){
    assert(false);
    return 0;
}
```



Diese Funktion soll es ermöglichen, Bereichsdatentypen anzulegen. Für Typen, für die Bereichsdefinitionen nicht sinnvoll sind, gibt diese Funktion stets `false` zurück.

```
bool ACircle::Adjacent(const Attribute* arg) const{  
    return false;  
}
```



- wird z.B. beim Sortieren von Relationen verwendet
- **muss** eine eindeutige Ordnung definieren
- undefinierte Werte sind stets kleiner als definierte
- das Argument ist stets vom gleichen Typ wie `this`
- Ergebnis ist
  - `< 0`, falls `*this < *arg`
  - `= 0`, falls `*this == *arg`
  - `> 0`, falls `*this > *arg`



# Compare - Implementierung

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

Wir verwenden die lexikographische Ordnung auf den einzelnen Mitgliedern der Klasse:

```
int ACircle::Compare(const Attribute* arg) const{
    // first compare defined flags
    if(!IsDefined()){
        return arg->IsDefined()?-1:0;
    }
    if(!arg->IsDefined()){
        return 1;
    }
    ACircle* c = (ACircle*) arg;
    if(x < c->x) return -1;
    if(x > c->x) return 1;
    if(y < c->y) return -1;
    if(y > c->y) return 1;
    if(r < c->r) return -1;
    if(r > c->r) return 1;
    return 0;
}
```



# Sizeof

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

Attribute verwenden generische `Open-` und `Save-`Funktionen. Die `Sizeof`-Implementierung sieht daher für Attributdatentypen stets gleich aus:

```
size_t ACircle::Sizeof() const{  
    return sizeof(*this);  
}
```



# HashValue

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

Die `HashValue`-Funktion wird von den verschiedenen Hash-Joins sowie einer Variante zur Duplikate-Eliminierung aus einem Tupelstrom verwendet. Bei einer ungenügenden Implementierung dieser Funktion (z.B. Rückgabe eines konstanten Werts), werden solche Operationen extrem langsam. Die Hashfunktion muss auf undefinierte Werte achten, sollte schnell sein und muss für gleiche Werte die gleiche Rückgabe liefern (möglichst auch für unterschiedliche Werte eine unterschiedliche Rückgabe).

```
size_t ACircle::HashValue() const{
    if(!IsDefined()){
        return 0;
    }
    return (size_t) (x + y + r);
}
```





Bei Aufruf dieser Funktion wird der Wert des Arguments übernommen. Wie üblich ist auf undefinierte Werte zu achten.

```
void ACircle::CopyFrom(const Attribute* arg){
    ACircle* a = (ACircle*) arg;
    if(!a->IsDefined()){
        SetDefined(false);
        return;
    }
    SetDefined(true);
    x = a->getX();
    y = a->getY();
    r = a->getR();
}
```



# Clone

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

```
Attribute* ACircle::Clone() const{
    ACircle* res = new ACircle(x,y,r);
    if(!this->IsDefined()){
        res->SetDefined(false);
    }
    return res;
}
```



Für die Einbindung an Secondo müssen die gleichen Funktionen wie für Nicht-Attribut-Datentypen implementiert werden. Besonders zu beachten sind hierbei:

- Alle Funktionen müssen auf undefinierte Werte achten (z.B. IN, OUT)
- OPEN und SAVE müssen nicht implementiert werden
- SIZE gibt die Größe der Klasse zurück



# Property-Funktion

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

```
ListExpr ACircleProperty () {  
  return ( nl -> TwoElemList (  
    nl->FourElemList (  
      nl->StringAtom ( " Signature " ) ,  
      nl->StringAtom ( " Example Type List " ) ,  
      nl->StringAtom ( " List Rep " ) ,  
      nl->StringAtom ( " Example List " ) ) ,  
    nl->FourElemList (  
      nl->StringAtom ( " -> DATA " ) , //Neue Kind beachten  
      nl->StringAtom ( ACircle::BasicType () ) ,  
      nl->StringAtom ( " ( real real real ) = ( x , y , r ) " ) ,  
      nl->StringAtom ( " (13.5 -76.0 1.0) " )  
    ) ) ) ;  
}
```



## ■ Auf undefinierte Werte achten

```
Word InACircle( const ListExpr typeInfo, const ListExpr instance,
               const int errorPos, ListExpr& errorInfo, bool& correct ){
    // create a result with addr pointing to 0
    Word res((void*)0);
    // assume an incorrect list
    correct = false;

    // check for undefined
    if(listutils::isSymbolUndefined(instance)){
        correct = true;
        ACircle* c = new ACircle(1,2,3);
        c->SetDefined(false);
        res.addr = c;
        return res;
    }

    // check whether the list has three elements
    if(!nl->HasLength(instance,3){
        msg.inFunError("expected three numbers");
        return res;
    }
    ... // same as for SCircle
}
```



# Out-Funktion

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

```
ListExpr OutACircle( ListExpr typeInfo, Word value ) {  
    ACircle* k = (ACircle*) value.addr;  
    if(!k->IsDefined()){  
        return listutils::getUndefined();  
    }  
    return nl->ThreeElemList(  
        nl->RealAtom(k->getX()),  
        nl->RealAtom(k->getY()),  
        nl->RealAtom(k->getR()));  
}
```



# Create, Delete, Close Funktionen

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

## ■ keine Unterschiede zu den einfachen Typen

```
Word CreateACircle( const ListExpr typeInfo ) {
    Word w;
    w.addr = (new ACircle(0,0,1.0));
    return w;
}

void DeleteACircle ( const ListExpr typeInfo , Word & w ) {
    ACircle * k = ( ACircle *) w.addr ;
    delete k ;
    w.addr = 0;
}

void CloseACircle ( const ListExpr typeInfo , Word & w ) {
    ACircle * k = ( ACircle *) w.addr ;
    delete k ;
    w.addr = 0;
}
```



# Open- und Save-Funktionen

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- werden von der Klasse Attribute zur Verfügung gestellt
- müssen nicht implementiert werden





# Clone, Cast und TypeCheck

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

## ■ Clone stützt sich auf das klasseninterne Clone ab

```
Word CloneACircle ( const ListExpr typeInfo , const Word & w ){
    ACircle * k = ( ACircle *) w . addr ;
    Word res(k->Clone());
    return res;
}
```

## ■ Cast und TypeCheck wie für einfache Typen

```
void * CastACircle ( void * addr ) {
    return ( new ( addr ) ACircle );
}
bool ACircleTypeCheck ( ListExpr type , ListExpr & errorInfo ){
    return ACircle::checkType(type);
}
```



# sizeof

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- gibt die Größe der Klasse zurück

```
int SizeOfACircle () {  
    return sizeof ( ACircle );  
}
```



## ■ Verwendet generische Open- und Save-Funktionen

```
TypeConstructor ACircleTC(  
    ACircle::BasicType(),           // name of the type  
    ACircleProperty,               // property function  
    OutACircle, InACircle,         // out and in function  
    0, 0,                           // deprecated, don't think about it  
    CreateACircle, DeleteACircle, // creation and deletion  
    OpenAttribute<ACircle>,        // open function  
    SaveAttribute<ACircle>,        // save functions  
    CloseACircle, CloneACircle,    // close and clone functions  
    CastACircle,                   // cast function  
    SizeOfACircle,                 // sizeof function  
    ACircleTypeCheck);             // type checking function
```



# Hinzufügen zur Algebra

Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

- wie bei einfachen Typen
- Zuordnung zur Kind `DATA`

```
...  
AddTypeConstructor( &ACircleTC );  
ACircleTC.AssociateKind( Kind::DATA() );  
...
```



Erweiterung  
von SECONDO

Thomas Behr

Attribute

Klassenstruktur

Funktionen

Der Umbau

Beispiel

Analyse

Änderungen

Funktionen

Einbindung

