Let's Modularize the Data Model Specifications of the ObjectLens in VisualWorks/Smalltalk

Dr. Michael Prasse

European Smalltalk User Group Conference

Prague, 4. September 2006

1. Introduction

Aims of the presentation

- ObjectLens Database Access Layer
- declarative definitions, design patterns, refactoring, product families
- monolithic ⇔ modular design
- adaptation of system components of VisualWorks

2. Context

Collogia Unternehmensberatung AG

- management consulting and software development
- > 50 employees
- 3 business fields: SAP consulting, project services, pension management

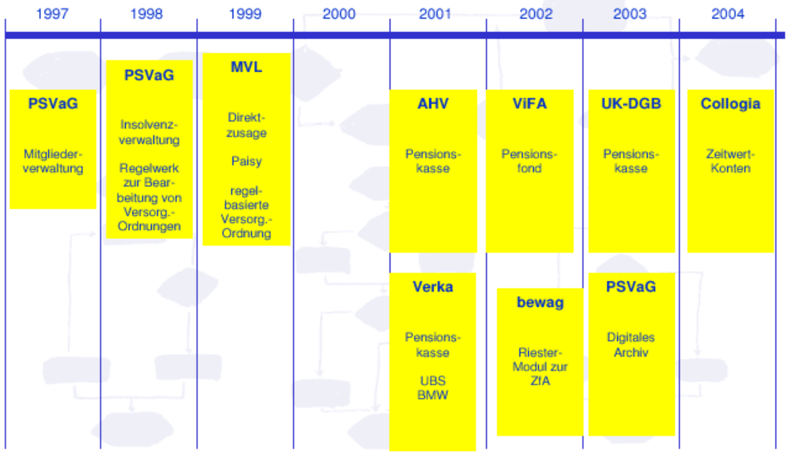
Pension Management

- VisualWorks since 1997
- Collphir Product Family: application software in the domain of pension schemes
- over 100 man years (including analysis, design, testing, maintenance)

We are not:

- a research organisation
- a framework or software tools vendor

Collphir - Software-Standard of management of pension schemes



1997

1998

1999

2000

2001

2002

2003

2004

PSVaG

Mitglieder-verwaltung

PSVaG

Insolvenz-verwaltung

Regelwerk zur Bearbeitung von Versorg.-Ordnungen

MVL

Direkt-zusage

Paisy

regel-basierte Versorg.-Ordnung

AHV

Pensions-kasse

Verka

Pensions-kasse

UBS
BMW

ViFA

Pensions-fond

bewag

Riester-Modul zur ZfA

UK-DGB

Pensions-kasse

PSVaG

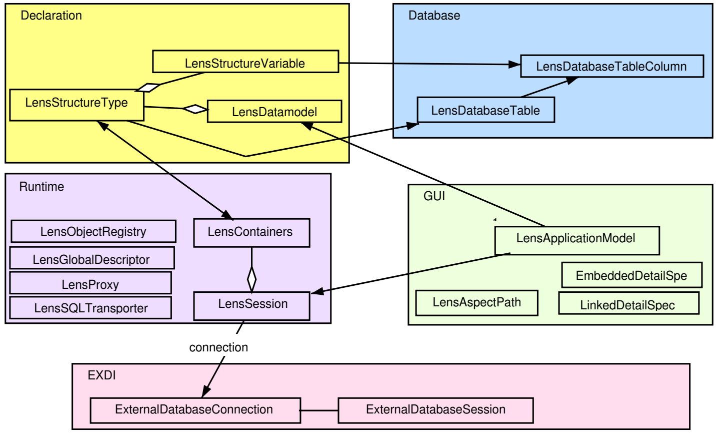
Digitales Archiv

Collogia

Zeitwert-Konten

3. ObjectLens

Architecture



Conceptual Mapping from Classes to Tables

Mapping Support	Concept	Mapping
	calculus level	Φ: object calculus → relational calculus (static semantics)
directly	class level	$\Phi_{classes}$: classes → tables Each class is unambiguously mapped to one table. You can't store objects of different subclasses in the same table.
directly	instance variable level	$\Phi_{instance\ variables}$: variables → columns
	simple data types	are mapped directly to one column
	object references (1:1 relationship)	are realized as a foreign key relationship
indirectly	1:n and n:m relationships	additional tables and select-statements (association classes)
no support	inheritance	A table contains all instance variables of the class including inherited variables.
no support	polymorphism	own support for untyped object references foreign key = (classID, objectID).

Programming Metaphor

programmer

- explicit persistence
- The ObjectLens is interpreted as a persistent collection.
  - to make an object persistent: *aLensSession add: anObject*
  - to remove an object from the database: *aLensSession remove: anObject*
- Database queries can be written in Smalltalk (select:).

internal

- automatic „isDirty“ detection of all persistent objects in the Lens
- flat transactions (begin, rollback, commit)
- proxy objects

➡ Smalltalk syntax can be used for persistent objects.
➡ This reduces the impedance mismatch.

4. DataModelSpec

Conception

- declarative description of a LensDataModel
- LiteralArray (Array of Arrays)
- encoding:

  LensDataModel>>literalArrayEncoding
- decoding:

  LensDataModel>>fromLiteralArrayEncoding:anArray

- using the methods *literalArrayEncoding* and *fromLiteralArrayEncoding:* then you can switch between the data model level and the data model specification level.

➡ You can choose the language level for the specification.

☛ windowSpec of the GUI-Framework

Example

```
literal array
     ^#(<Class>
          <aspect> <value> ...)

lens literal array
^#(#{Lens.LensDataModel}
          #setDatabaseContext: #(...)
          #structureTypes: #(
                    #(#{Lens.LensStructureType}
                               #memberClass: <memberClass>
                               #setVariables: #(
                                    #(#{Lens.LensStructureVariable}
                                              #name: 'angelegtAm'
                                              #column:    <Database Column>
                                              #privateIsMapped: true )
                                         ...)
                               #table:  <Database Table> )
                         ...)
          #lensPolicyName: #Mixed
          #lensTransactionPolicyName: #PessimisticRR
          #validity: #installed )
```

Properties

- One monolithic *datamodelSpec* describes the data model of an application.
- The data model has to contain all entity classes of the application.
- The lens structure type of a class defines all instance variables of a class including inherited variables.
- Instance variables of a class can be mapped (persistent) or unmapped (transient).

Maintenance Problems
Class hierarchy problems

An instance variable is specified in each LensStructureType of a subclass.

You have to adapt all LensStructureTypes of subclasses:

➡ when you define a new instance variable of a super class.

➡ when you change the specification of an instance variable of a super class

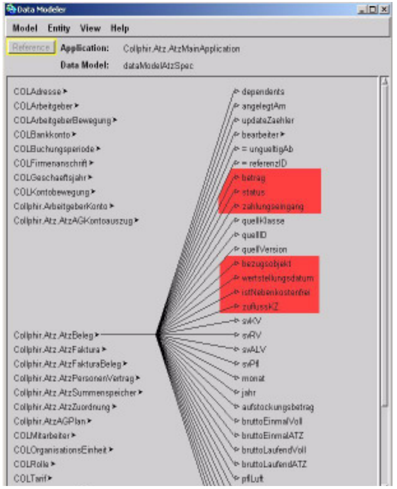
➡ when you rename an instance variable of a superclass

➡ when you change the class hierarchy

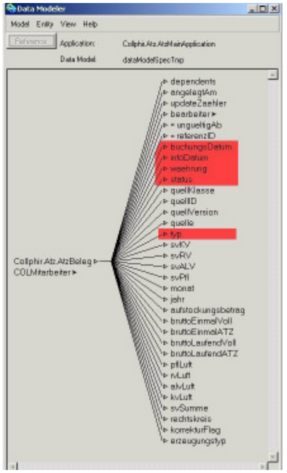
An instance variable can be specified differently in several subclasses.

- geschlecht (engl. gender) : Boolean
- geschlecht (engl. gender) : {'m', 'w'}

Example: Changing the class hierarchy

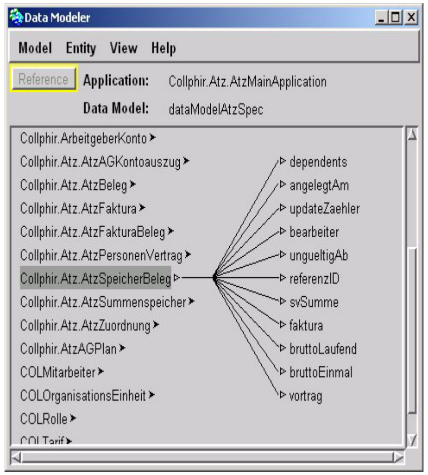


Application: Collphir.Atz.AtzMainApplication
Data Model: dataModelAtzSpec



Application: Collphia.Atz.AtzMainApplication
Data Model: dataModelSpecTrip

Example: Definition of a new class

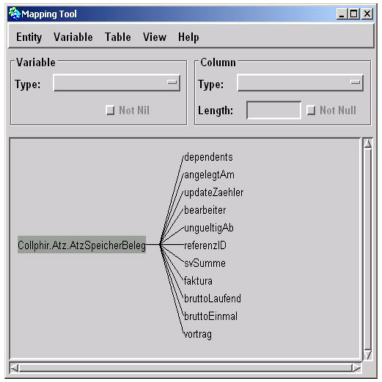


Application: Collphir.Atz.AtzMainApplication

Data Model: dataModelAtzSpec

Collphir.ArbeitgeberKonto
Collphir.Atz.AtzAGKontoauszug
Collphir.Atz.AtzBeleg
Collphir.Atz.AtzFaktura
Collphir.Atz.AtzFakturaBeleg
Collphir.Atz.AtzPersonenVertrag
Collphir.Atz.AtzSpeicherBeleg
Collphir.Atz.AtzSummenspeicher
Collphir.Atz.AtzZuordnung
Collphir.AtzAGPlan
COLMitarbeiter
COLOrganisationsEinheit
COLRolle
COLTarif

dependents
angelegtAm
updateZaehler
bearbeiter
ungueltigAb
referenzID
svSumme
faktura
bruttoLaufend
bruttoEinmal
vortrag



Collphir.Atz.AtzSpeicherBeleg

dependents
angelegtAm
updateZaehler
bearbeiter
ungueltigAb
referenzID
svSumme
faktura
bruttoLaufend
bruttoEinmal
vortrag

Multiple datamodelSpec Problems

- One monolithic *dataModelSpec* is used to describe the data model of an application.
- Collphir:
  - begin: one application ➡ one dataModelSpec
  - today: a product family with common core ➡ multiple dataModelSpecs
  - copy & paste an existing *dataModelSpec* and adapt this to the new requirements.
  - overlapping parts in all dataModelSpecs concerning the common data basis
  - synchronizing several dataModelSpecs

➡ The origin of all these problems is the redundant specification of instance variable mappings in subclasses and *datamodelSpecs*.

➡ There is no single source principle for specifications of the ObjectLens.

5. Modularization of the ObjectLens

General Ideas

- general aspects: modularization and the use of inheritance
- The *datamodelSpec* is one aspect of the class and is organized by the class itself.
  - We break up the monolithic specification in several pieces.
  - Each piece describes the mapping of one class without inherited variables.
- The single class data specifications are the pieces from which the whole data model specification is constructed.
- Instead of changing a central monolithic definition, we change only the modular definitions of the concerned classes.

➡ The result is a normal but generated monolithic data model specification.

➡ We change only the definition and construction process.

➡ All other aspects of the ObjectLens are unchanged.

Solution

a). We store the mappings in the domain classes.

b). We construct the *datamodelSpec* from these mapping fragments.

c). We support the common development tools.

d). We support the migration of our existing data model specifications.

Data Model Mappings of Classes

```
dataModelDefinitionSpec
    " You should not override this message. "
    ^ self dataModelDefinition literalArrayEncoding

dataModelDefinition
    " You should not override this message. You can adapt primDataModelDefinition"
    | type |
    type := self primDataModelDefinition.
    self primLocalDataModelDefinitionChanges: type.
    type variables: (List withAll: type variables ).
    type resolveStandalone.
    ^type
```

- The method *primDataModelDefinition* provides the standard implementation. It will usually be automatically generated.
- The method *primLocalDataModelDefinitionChanges*: gives each class the opportunity to override the inherited definitions. It is created by hand and describes changes, which should not be overridden by further generation steps.

```
COLPersistentModel>>primDataModelDefinition
    | type |
    type := LensStructureType new.
    type memberClass: self.
    type table: ((Oracle7Table new) name: self name; owner: 'COLBAV').
    type idGeneratorType: #userDefinedId.
    ^type

 primDataModelDefinition
    | type |
    type := super primDataModelDefinition.
    type variables add: #(#{Lens.LensStructureVariable} #name: 'name' #setValueType: #String
#fieldType: #String #column: #(#{Oracle7TableColumn} #name: 'name' #dataType: 'varchar2' #maxColumnConstraint: 100) #generatesAccessor: false #generatesMutator: false #privateIsMapped:
true) decodeAsLiteralArray.
    self addSummenspeicherVariableIn: type.
    type table name: 'kontoZuordnung' .
    ^type

primLocalDataModelDefinitionChanges:type
    | var |
    super primLocalDataModelDefinitionChanges:type.
    (type variableNamed: 'speicherBeleg') setValueType: #AtzSummenspeicherBeleg.
```

LensApplication datamodelSpec

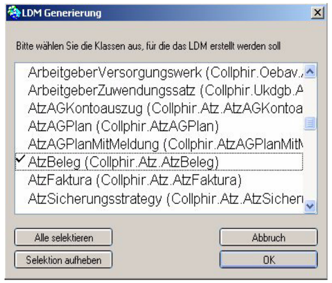
```
dataModelSpecGenerated
    | ldm |
    (ldm := LensDataModel new)
        application: self;
        fromLiteralArrayEncoding: (self dataModelSpecForStructureTypeSpecs:
                                            self dataModelStructureTypeSpecs).
    self adaptDataModel: ldm.
    ldm compile.
    ^ldm literalArrayEncoding
```

- The *LensDataModel* is created by *"self dataModelSpecForStructureTypeSpecs: self dataModelStructureTypeSpecs"*.
- The method *adaptDataModel* permits adaptations, which are only valid for this special application.

```
dataModelStructureTypeSpecs
    ^ self dataModelStructureTypeSpecsFor: self dataModelClasses

 dataModelStructureTypeSpecsFor: classColl
    ^ (classColl collect:[ :cl | cl dataModelDefinitionSpec]) asArray

dataModelClasses
    „Returns a set of all classes which contain to the data model“

dataModelSpecForStructureTypeSpecs: aColl
    | res |
    res := self dataModelTemplate copy.
    res at: 5 put: aColl.
    ^res

dataModelTemplate
    ^#(#{Lens.LensDataModel}
        #setDatabaseContext:
        #(#{Oracle7Context} ... )
        #structureTypes: #()
        #lensPolicyName: #Mixed
        #lensTransactionPolicyName: #PessimisticRR
        #validity: #installed )
```

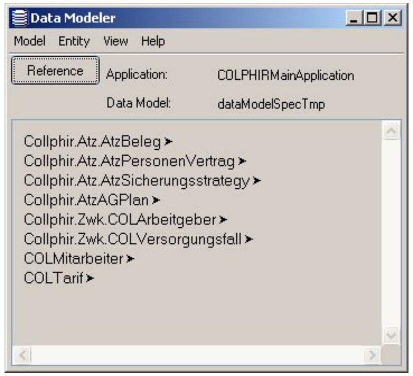
Integration into the Lens Modeling Tools

LensEditor for Class Data Models



Bitte wählen Sie die Klassen aus, für die das LDM erstellt werden soll

- ArbeitgeberVersorgungswerk (Collphir.Oebav.
- ArbeitgeberZuwendungssatz (Collphir.Ukdgb.A
- AtzAGKontoauszug (Collphir.Atz.AtzAGKontoa
- AtzAGPlan (Collphir.AtzAGPlan)
- AtzAGPlanMitMeldung (Collphir.AtzAGPlanMitN
- ✓ AtzBeleg (Collphir.Atz.AtzBeleg)
- AtzFaktura (Collphir.Atz.AtzFaktura)
- AtzSicherungsstrategy (Collphir.Atz.AtzSicher



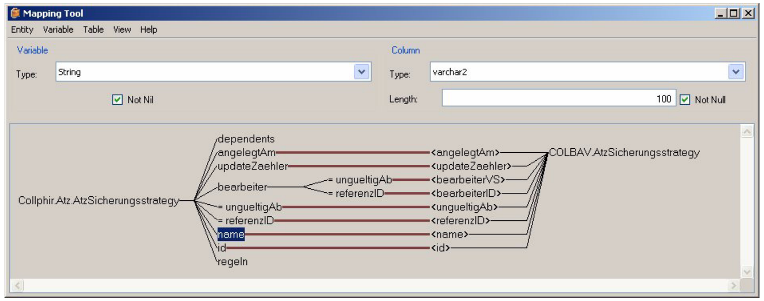
Application: COLPHIRMainApplication

Data Model: dataModelSpecTmp

- Collphir.Atz.AtzBeleg ➤
- Collphir.Atz.AtzPersonenVertrag ➤
- Collphir.Atz.AtzSicherungsstrategy ➤
- Collphir.AtzAGPlan ➤
- Collphir.Zwk.COLArbeitgeber ➤
- Collphir.Zwk.COLVersorgungsfall ➤
- COLMitarbeiter ➤
- COLTarif ➤

```
LensMainApplication
     openLensEditorFor:CollphirMainApplication
     with: (Set new add: AtzBeleg; yourself)
```

Lens Mapping Tool for Class Data Models



Variable

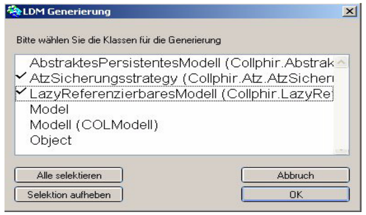
Type: String

☑ Not Nil

Column

Type: varchar2

Length: 100 ☑ Not Null



Bitte wählen Sie die Klassen für die Generierung

- AbstraktesPersistentesModell (Collphir.Abstrak
- ✓ AtzSicherungsstrategy (Collphir.Atz.AtzSicher
- ✓ LazyReferenzierbaresModell (Collphir.LazyRe
- Model
- Modell (COLModell)
- Object

```
DataModelDefinitionGenerator new
     generateLensSpecsFrom: self ldm
     for: selectedClasses
```

Migration Process

I). transformation T

```
generator := DataModelDefinitionGenerator new
   add: AtzMainApplication dataSpec: #dataModelSpec;
   add: ZwkMainApplication dataSpec: #dataModelSpec;
   yourself.
```

II). conflict reports

```
generator report
```

III). data model classes

```
generator
   generateDataModelClassesFor: AtzMainApplication dataSpec: #dataModelSpec .
```

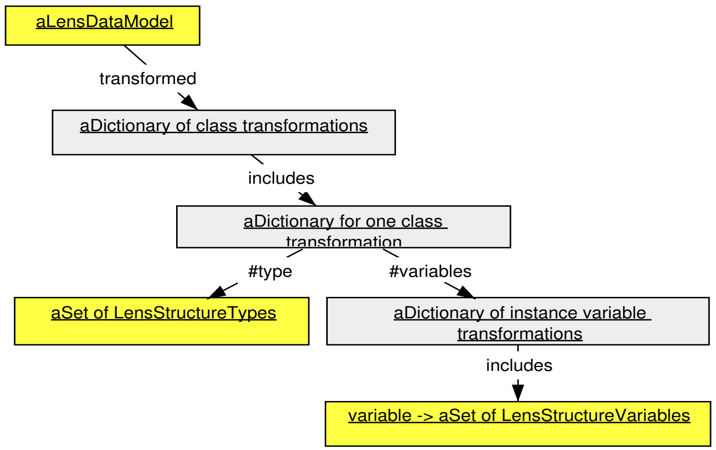
IV).generating of all classes

```
generator generate
```

generating of a subset of classes

```
generator generateLensSpecsFrom: ldm
               for: (Set new add: COLRente;add: COLAZ03 ;add: COLAZRR ;yourself)
```

Dictionary structure of transformation T


aLensDataModel
transformed
aDictionary of class transformations
includes
aDictionary for one class transformation
#type
#variables
aSet of LensStructureTypes
aDictionary of instance variable transformations
includes
variable -> aSet of LensStructureVariables

DataModelDefinitionGenerator - Generation and Migration

- migration of old monolithic *dataModelSpecs*
- generating class data models in the mapping tool

Data Model Spec Migration

- Computing the conflicts between different definitions of an entity (transformation T).
- conflict solving: two-step strategy
  - trivial cases: different max column constraints ➡ weakest condition.
  - complicated cases: pair reviews, which mapping should be become the standard.

Support of different mappings by using the methods *primLocalDataModelDefinitionChanges* and *adaptDataModel*.

Code Generation

- Generation of method *dataModelClasses* for the application
- Generation of method *primDataModelDefinition* from the corresponding *classDict*.
  - Simple data mappings are inlined.
  - Complicated mappings for foreign key relationships are extracted in separate methods.

- Code generation uses common Smalltalk techniques.
- Methods:
  - for invariant code fragments.
  - which provides a string representation for related parts of the mapping like table name, primary key, or variables.
- a stream to merge this fragments.
- Result: a source string of a Smalltalk method that we compile in the metaclass of the considered class in the protocol *’lens data model specs’*.

Testing and V&V

Generation of dataModelSpecs

- simple test data for old dataModelSpecs
- simple test data for modular dataModelSpecs (same data)
- comparison of the datamodels

Migration

- reviews (code and dataModel)
- application test
- stepwise introduction

Editing

- development use