Georg Heeg eK

Eurydike
(Εὐρυδίκη)

Schemaless
Object Relational
SQL Mapper

Eurydike

- **E**asily
- **U**sable

- **R**elational
- **Y**obbish
- **D**atabase

- **I**ntuitive
- **K**eymappingless
- **E**nvironment

Objects

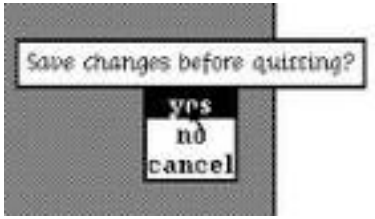
- Smalltalk Objects live
- Objects respond to messages
- Objects have identity

Persistence

- Long time
- Sharing
  - multiple users
  - concurrent changes

Objects and Persistence

- Save the image
  - long term
  - multiple users have copies
  - no concurrent changes
- Serialize Objects
- Use GemStone/S
  - Huge image on disk
  - multi-user
  - transaction bases synchronization



GemStone/S

Serialize Objects

- Timestamp now storeString
  - '(Timestamp readFrom: ''06/09/2013 14:38:53.432'' readStream)'
  - Issue: cyclic structures
    - solved by Hubert Baumeister 1988 using keys and IdentityDictionaries
- BOSS
  - Binary Object Storage System
- Parcels
  - Special serializer for code
- Others
  - VOSS, XML, JSON, ODMG

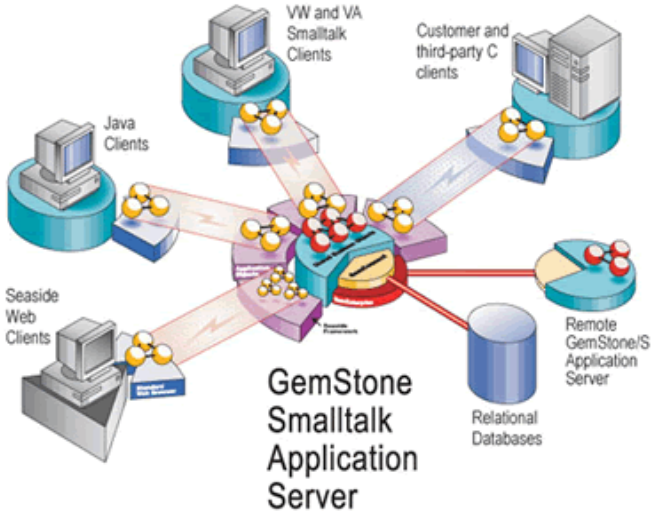
Serialize Objects

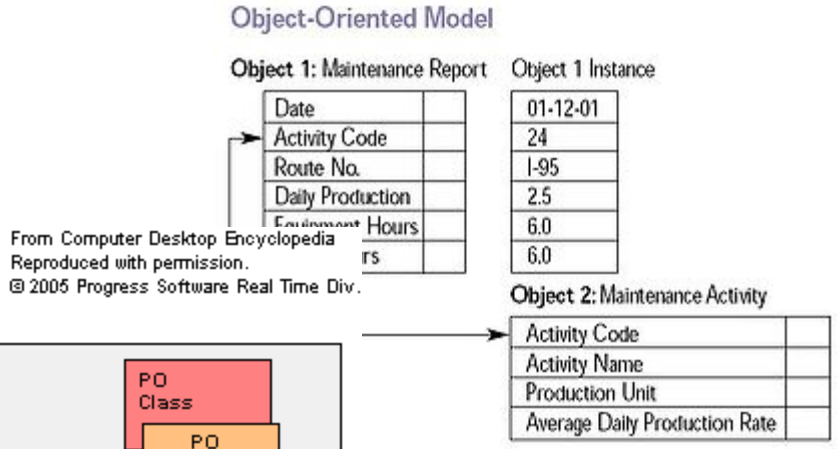
- Store all or nothing
- No concurrency

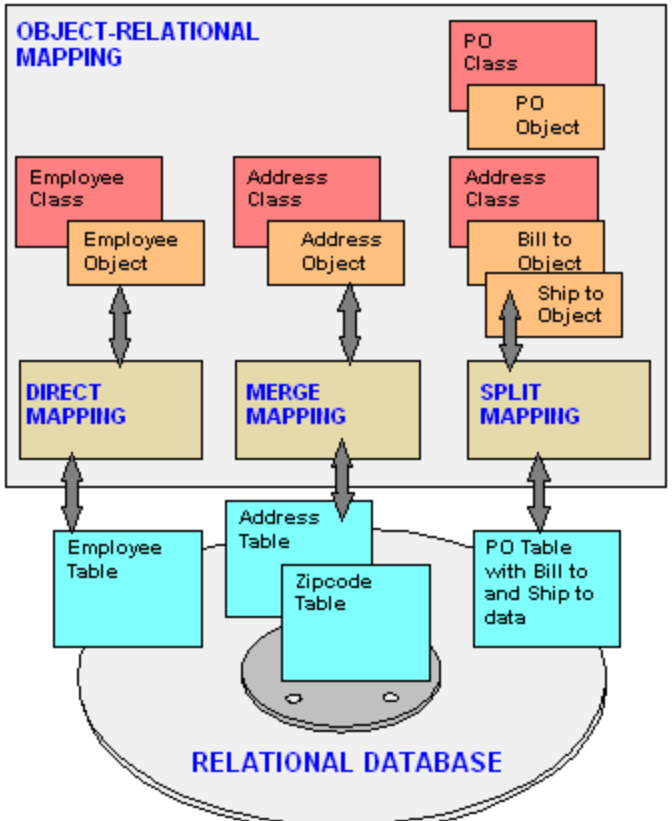
➔

- Desire for databases

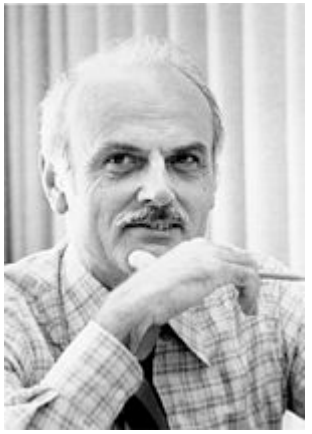
A few Pictures







Relational databases



- Finite sets of typed values
  - Numbers, strings, …
- Equality
  - of values
- Cross products
  - called tables

Relations → Objects

- Table rows as objects
  - easy as Objects can describe everyting
  - tables are mapped onto classes
  - columns are mapped onto instance variables
    - ObjectStudio database support
    - EXDI (VisualWorks)
    - GemConnect
    - VA Database support

Objects ➔ Relations

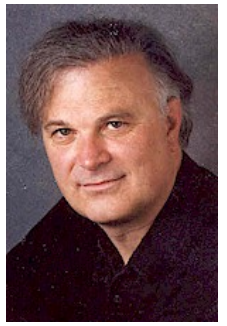
- Much more interesting
- Much more difficult
  - due to lack of flexibilty on the relation side
- Typical solution
  - OR Mapping
  - Objects Networks map onto Entity Relationship Models

Object Lens

- 'the industry's first Object Relational Mapping (ORM) technology, providing seamless integration between Smalltalk-80 applications and RDB
  - Richard Steiger
    - Parc, ParcPlace and Ensemble
    - Invented Object Relational Mapping (ORM)
    - Smalltalk Type System, providing strong typing, data entry validation, and program transformation inferencing

TOPLink

THE OBJECT PEOPLE
Your Source for e-Solutions™



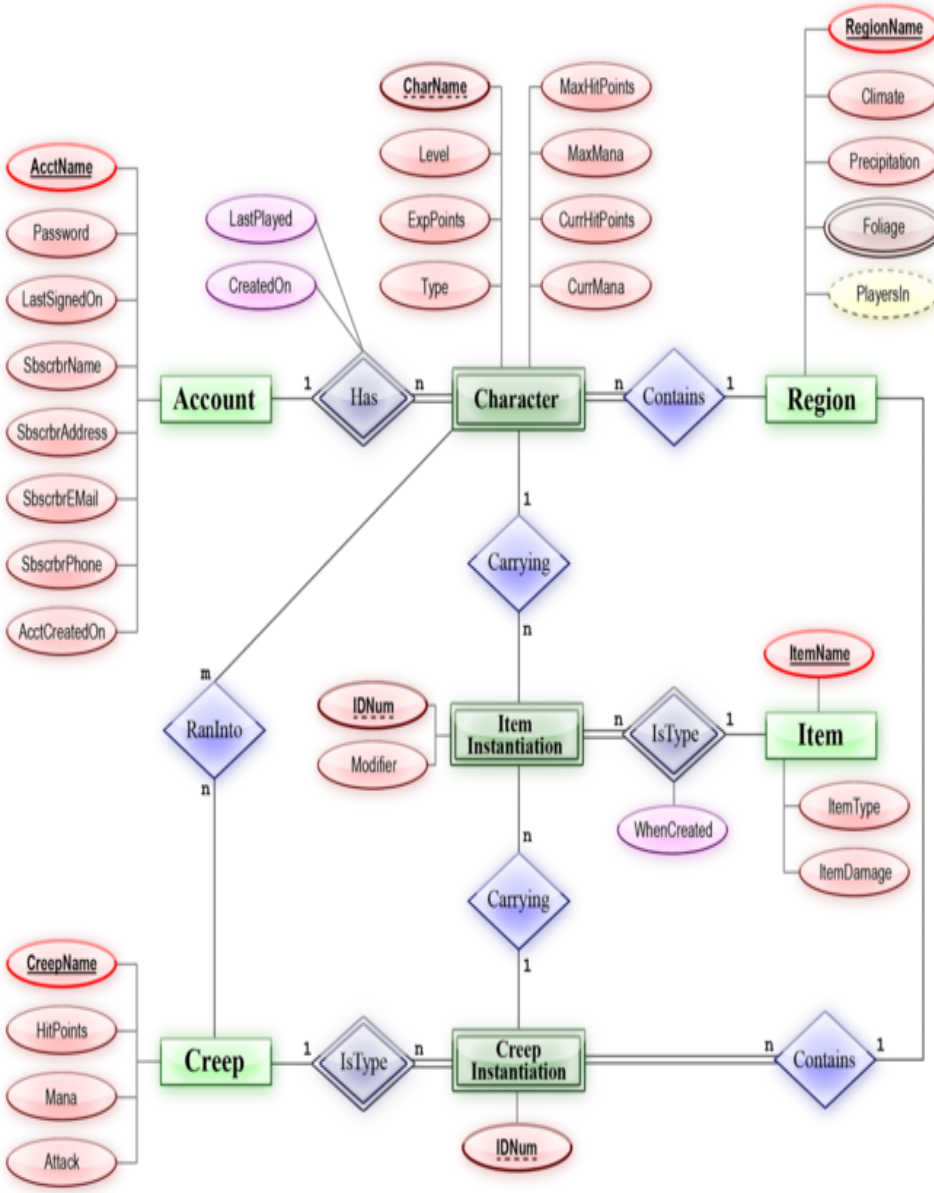
- TOPLink is an application framework
- allows Smalltalk applications to access relational databases
- provides a high degree of integration between the Smalltalk objects and the relational database tables
- relational database can be made to behave like an object database

GLORP

- Generic Lightweight Object-Relational Persistence
  - open-source
  - object-relational mapping layer for Smalltalk
- Camp Smalltalk Project
  - led by Alan Knight
- goal is
  - to provide a simple, powerful framework for reading and writing objects from relational databases
- originally sponsored by The Object People
- concepts are reminiscent of TOPLink
  - though noticably different in some respects.

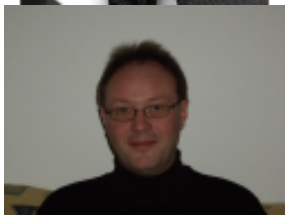
Properties

- Class structures map to ER models
- Change is difficult
  - change classes
  - change tables
    - schema migration
    - data migration
  - change mapping



Georg Heeg eK
www.heeg.de

Orpheus



- Goal: Easy change
  - No more database schema migrations
  - No more data migrations
- Designed and developed for Component Studio (ERP)
- Way: universal database schema
  - one schema for all objects
- Based upon Object Lens
  - Mapping is reduced to typing

Orpheus object table

```
CREATE TABLE domainmodel(
    domainmodelclass character varying(64),
    objectid integer)
```

Georg Heeg eK
www.heeg.de

Object attribute table

```
CREATE TABLE objectreference(
    attribute character varying(40),
    basevalue numeric,
    myinstance integer)
```

Georg Heeg eK
www.heeg.de

Integer attributes table

```
CREATE TABLE integervalue(
    attribute character varying(40),
    basevalue integer,
    myinstance integer)
```

Many more basic type tables

- Examples
  - ARBITRARYSTRING
  - BINARYVALUE
  - BOOLEANVALUE
  - LARGESTRING
  - TINYSTRING
  - TIMESTAMPVALUE
  - …

Our Dream

- Persistance without pain
- Intuitive for Smalltalkers
- Easy to use

- A Smalltalk …
  - as Arden Thomas would say

Can we use the existing ORMs?

- Let us look at
  - Glorp
  - Orpheus

GLORP

- Explicit description in methods
  - #classModelForXXX:
  - #tableForXXX:
  - #descriptorForXXX:
- Subclasses need special considerations

Orpheus

- Every object class needs
  - a Lens description method
- Relationship classes required for N:M relationships

All OR-Mappings

- require typed Smalltalk

Eurydike – Wishlist

- EASY
  - We want to store any object
  - We do not want to write any mapping

Georg Heeg eK
www.heeg.de

Eurydike – Timeline

- Start of the research project
  - 28 February 2013
- Project task
  - "Build schema-less system for storage of any object in a SQL DB"
- STIC 2013: Presentation
- Since STIC 2013:
  - Used internally at Heeg
  - Used in GH seaMS
  - Experience starts floats back

Eurydike – Tasks

- Which tables in the database?
- How to do the (invisibe) mapping object access to SQL?
- What about (invisible) proxies?
- What about concurrency?
- What about garbage collection in the database?

Research Strategy

- Bottom up
  - Database Tables and Mappings first
  - Garbage Collection last
- Try out multiple ideas

Idea A

- Take the Orpheus Model and make it flexible
- One table per type
  - makes queries unworkable
- Thus
  - One table for all the types

Idea A - Table

```
CREATE TABLE fields(
    objectid integer,
    rowid integer,

    objectclass character varying,

    stringfieldkey character varying,
    stringfieldvalue character varying,

    integerfieldkey character varying,
    integerfieldvalue integer,

    referencefieldkey character varying,
    referencefieldvalue integer)
```

Georg Heeg eK
www.heeg.de

Eurydike – Idea A – Example 1

objectid	rowid	owner class	string field key	string field value	integer field key	integer field value	reference field key	reference field value
1	1	Association	'key'	'KeyA'	'value'	1		

```
association := 'KeyA' -> 1.
```

Eurydike – Idea A – Example 2

objectid	rowid	owner class	string field key	string field value	integer field key	integer field value	reference field key	reference field value
1	1	Association	'key'	'KeyA'	'value'	1		
2	1	Association	'key'	'KeyB'				

```
association := 'KeyB' -> nil.
```

Eurydike – Idea A – Example 3

objectid	rowid	owner class	string field key	string field value	integer field key	integer field value	reference field key	reference field value
1	1	Association	'key'	'KeyA'	'value'	1		
2	1	Association	'key'	'KeyB'				
3	1	Association	'key'	'KeyC'				
3	2	Association	'value'	'ValueC'				

```
association := 'KeyC' -> 'ValueC'.
```

Eurydike – Idea A – Self References

objectid	rowid	owner class	string field key	string field value	integer field key	integer field value	reference field key	reference field value
1	1	Association	'key'	'KeyA'	'value'	1		
2	1	Association	'key'	'KeyB'				
3	1	Association	'key'	'KeyC'				
3	2	Association	'value'	'ValueC'				
4	1	Association	'key'	'KeyD'			'value'	4

```
association := 'KeyD' -> nil.
assocation value: association

association := session detect: [:each | each value = each]
```

Eurydike – Idea A – Sequenceables

groupid	rowid	owner class	string field key	string field value	integer field key	integer field value	…
1	1	Array	'1'	'elementA'	'2'	200	
1	2	Array	'3'	'elementC'	'4'	400	

array := Array with: 'elementA' with: 200 with: 'elementC' with: 400.

array := session detect: [:each | (each at: 1) = 'elementA']]
array := session detect: [:each | (each at: 2) = 200]

The same happens for all other sequenceable collections.

Unordered collections like Set are stored with fictive index as key.

Eurydike – Idea A – Dictionaries

- Dictionaries
  - are collections of associations
  - are stored as collections of associations

Eurydike – Idea A – SQL

```
[:base | base name = 'hello']

SELECT fields.groupid 'baseid', embedded.filterid 'filterid'
    FROM
        fields,
        (SELECT base.groupid 'baseid', base.groupid 'filterid'
                FROM fields base) embedded
    WHERE
        fields.groupid = embedded.filterid AND
        fields.stringfieldkey = 'name' AND
        fields.stringfieldvalue = 'hello'
```

Eurydike – Idea A - SQL

- SQL generation is complicated
  - double dispatching
    - for each possible datatype in table
    - New expressions need new code paths for all datatypes

Eurydike – Idea A - Summary

- Easy to use
  - Almost any Object can be stored
  - Our world is a sequence of 'Object allInstances'
  - we can search with #select:, #detect: and #reject:

- Not quite excellent enough
  - SQL generation is complex because of primitive datatypes and unknown object type
  - One column pair per type makes queries complicated

Idea B

- Take Idea A and make it simpler
- Thus
  - Unifiy all types as string pairs
  - Store every object as a dictionary

Georg Heeg eK
www.heeg.de

Idea B - Table

- CREATE TABLE fields(
  objectid character varying,

  keybasetype character varying,
  keytype character varying,
  keymajor character varying,
  keyminor character varying,

  valuebasetype character varying,
  valuetype character varying,
  valuemajor character varying,
  valueminor character varying)

Eurydike – Idea B

- #baseType is like a class hierarchy filter for comparing. Only things with same #baseType are compared in SQL queries
- #type is the real class of object for instantiation
- #major is the primary value, e.g. the nominator for Number
- #minor is the secondary value, e.g. the denominator for Number

Eurydike – Idea B – Example 1

objectid	keybasetype	keytype	Keymajor	keyminor	valuebasetype	valuetype	valuemajor	valueminor
1	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
1	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyA'	
1	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.Number'	'Core.SmallInteger'	'1'	'1'

association := 'KeyA' -> 1.

Eurydike – Idea B – Example 2

objectid	keybasetype	keytype	keymajor	keyminor	valuebasetype	valuetype	valuemajor	valueminor
1	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
1	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyA'	
1	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.Number'	'Core.SmallInteger'	'1'	'1'
2	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
2	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyB'	

association := 'KeyB' -> nil.

Eurydike – Idea B – Example 3

objectid	keybasetype	keytype	keymajor	keyminor	valuebasetype	valuetype	valuemajor	valueminor
1	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
1	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyA'	
1	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.Number'	'Core.SmallInteger'	'1'	'1'
2	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
2	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyB'	
3	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
3	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyC'	
3	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.CharacterArray'	,Core.ByteString'	'ValueC'	

association := 'KeyC' -> 'ValueC'.

Eurydike – Idea B – Self References

objectid	keybasetype	keytype	keymajor	keyminor	valuebasetype	valuetype	valuemajor	valueminor
1	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
1	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyA'	
1	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.Number'	'Core.SmallInteger'	'1'	'1'
2	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
2	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyB'	
3	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
3	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyC'	
3	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.CharacterArray'	,Core.ByteString'	'ValueC'	
4	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Association'	
4	'Core.Symbol'	'Core.ByteSymbol'	'key'		'Core.CharacterArray'	,Core.ByteString'	'KeyD'	
4	'Core.Symbol'	'Core.ByteSymbol'	'value'		'Core.Object'	'Core.Object'	'4'	

```
association := 'KeyD' -> nil.
assocation value: association
association := session detect: [:each | each value = each]
```

Eurydike – Idea B - Dictionaries

- In Idea A Dictionaries are collections of Associations
- In Idea B there is native implemantation of Dictionaries

objectid	keybasetype	keytype	keymajor	keyminor	valuebasetype	valuetype	valuemajor	valueminor
5	'Core.Symbol'	'Core.ByteSymbol'	'class'		'Core.CharacterArray'	'Core.ByteString'	'Core.Dictionary'	
5	'Core.CharacterArray	'Core.ByteString'	'KeyA'		'Core.Number'	'Core.SmallInteger'	'1'	'1'
5	'Core.CharacterArray'	'Core.ByteString'	'KeyC'		'Core.CharacterArray'	'Core.ByteString'	'ValueC'	

```
d := Dictionary new
d at: 'KeyA' put: 1.
d at: 'KeyC' put: 'ValueC'
```

- As we store dictionaries we need only expressions for the primitive dictionary access like:

  #at:, #keyAtValue:, #keys, #values, #size, #select:

  The rest can be compound by these primitives.

```
includes: aValue
        ^self values select: [:each | each = aValue]

includesKey: aKey
        ^self keys select: [:each | each = aKey]
```

Eurydike – Idea B - Expressions

- As we return uniform data
  - leads to simpler SQL-generation
  - no need for context information
  - Special expression combinations can be optimized later
    - when “make it fast” gets important

Special Objects

- ByteArrays
  - Base 64 encoded strings
- Compiled Methods
  - as source code or
  - as byte array of compiled code
- BlockContexts
  - same consirations needed as for OpenTalk STST

Georg Heeg eK
www.heeg.de

Eurydike – Idea B - Summary

- Same advantages as Idea A
- Simple SQL generation
  - Expressions can be compound by primitives
  - Uniform data -> easier expression and SQL generation
  - Uniform data -> different Numbers like double, fraction, integer can be compared in queries
- Most of the collection protocol is possible like #includes:, #includesKey:, #anySatisfy:, #allSatisfy:

- But
  - …

Eurydike - ObjectSpace

- It is not the Smalltalk-way to store and retrieve big unstructured collections of instances
- Smalltalk wants a root objects and navigates from object to object
- Eurydike implements abstract class ObjectSpace

ObjectSpace – Example

- create a class for the objects

```
Smalltalk.Eurydike defineClass: #Person
        superclass: #{Core.Object}
        instanceVariableNames: 'name lastName address '
```

One ObjectSpace Subclass

```
Smalltalk.Eurydike defineClass: #PersonSpace
        superclass: #{Eurydike.ObjectSpace}
        instanceVariableNames: 'persons '
```

One instance method

initialize

```
super initialize.
persons := OrderedCollection new
```

One class method

connectionProfile

```
^(ConnectionProfile new)
        platform: PostgreSQLPlatform new;
        environment: 'localhost:5432_postgres';
        username: 'postgres';
        password: 'postgres';
        yourself
```

Create the objects

```
p := PersonSpace default.
p persons add: ((Person new)
        name: 'Jörg'; lastName: 'Belger'; address: 'Köthen').
p persons add: ((Person new)
        name: 'Karsten'; lastName: 'Kusche'; address: 'Köthen').
p persons add: ((Person new)
        name: 'Magnus'; lastName: 'Schwarz'; address: 'Dortmund').
p commit
```

Georg Heeg eK
www.heeg.de

PGAdmin shows the result

Daten editieren - localhost (localhost:5432) - postgres - fields

Datei Bearbeiten Anzeigen Werkzeuge Hilfe

Keine Begrei

	id character var	keybasetype character varying	keytype character varying	keymajor character va	keyminor character v	valuebasetype character varying	valuetype character varying	valuemajor character varying	valueminor character v
1	1	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
2	1	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Jörg	
3	1	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Belger	
4	1	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
5	2	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.PersonSpace	
6	2	Core.Symbol	Core.ByteSymbol	persons		Core.Object	Core.OrderedColle	5	
7	3	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
8	3	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Karsten	
9	3	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Kusche	
10	3	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
11	4	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
12	4	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Magnus	
13	4	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Schwarz	
14	4	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Dortmund	
15	5	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Core.OrderedCollection	
16	5	Core.Number	Core.SmallInteger	1	1	Core.Object	Eurydike.Person	1	
17	5	Core.Number	Core.SmallInteger	2	1	Core.Object	Eurydike.Person	3	
18	5	Core.Number	Core.SmallInteger	3	1	Core.Object	Eurydike.Person	4	

Notizblock

18 Zeilen.

Georg Heeg eK
www.heeg.de

Eurydike – Schema migration

```
p := PersonSpace default.
d := Dictionary new.
p persons do: [:person |
        d at: person lastName put: person].
p persons: d.
p commit
```

Georg Heeg eK
www.heeg.de

Result

Daten editieren - localhost (localhost:5432) - postgres - fields

Datei Bearbeiten Anzeigen Werkzeuge Hilfe

Keine Begrei

	id character var	keybasetype character varying	keytype character varying	keymajor character va	keyminor character v	valuebasetype character varying	valuetype character varying	valuemajor character varying	valueminor character v
1	5	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Core.OrderedCollection	
2	5	Core.Number	Core.SmallInteger	1	1	Core.Object	Eurydike.Person	1	
3	5	Core.Number	Core.SmallInteger	2	1	Core.Object	Eurydike.Person	3	
4	5	Core.Number	Core.SmallInteger	3	1	Core.Object	Eurydike.Person	4	
5	3	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
6	3	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Karsten	
7	3	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Kusche	
8	3	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
9	1	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
10	1	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Jörg	
11	1	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Belger	
12	1	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
13	4	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
14	4	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Magnus	
15	4	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Schwarz	
16	4	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Dortmund	
17	6	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Core.Dictionary	
18	6	Core.Character?	Core.ByteString	Belger		Core.Object	Eurydike.Person	1	
19	6	Core.Character?	Core.ByteString	Kusche		Core.Object	Eurydike.Person	3	
20	6	Core.Character?	Core.ByteString	Schwarz		Core.Object	Eurydike.Person	4	
21	2	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.PersonSpace	
22	2	Core.Symbol	Core.ByteSymbol	persons		Core.Object	Core.Dictionary	6	

Notizblock

22 Zeilen.

Getting rid of garbage

PersonSpace default garbageCollect

Georg Heeg eK
www.heeg.de

Cleaned up database

Daten editieren - localhost (localhost:5432) - postgres - fields

Datei Bearbeiten Anzeigen Werkzeuge Hilfe

Keine Begrei

	id character var	keybasetype character varying	keytype character varying	keymajor character va	keyminor character v	valuebasetype character varying	valuetype character varying	valuemajor character varying	valueminor character v
1	3	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
2	3	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Karsten	
3	3	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Kusche	
4	3	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
5	1	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
6	1	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Jörg	
7	1	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Belger	
8	1	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Köthen	
9	4	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.Person	
10	4	Core.Symbol	Core.ByteSymbol	name		Core.CharacterArray	Core.ByteString	Magnus	
11	4	Core.Symbol	Core.ByteSymbol	lastName		Core.CharacterArray	Core.ByteString	Schwarz	
12	4	Core.Symbol	Core.ByteSymbol	address		Core.CharacterArray	Core.ByteString	Dortmund	
13	6	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Core.Dictionary	
14	6	Core.CharacterA	Core.ByteString	Belger		Core.Object	Eurydike.Person	1	
15	6	Core.CharacterA	Core.ByteString	Kusche		Core.Object	Eurydike.Person	3	
16	6	Core.CharacterA	Core.ByteString	Schwarz		Core.Object	Eurydike.Person	4	
17	2	Core.Symbol	Core.ByteSymbol	class		Core.CharacterArray	Core.ByteString	Eurydike.PersonSpace	
18	2	Core.Symbol	Core.ByteSymbol	persons		Core.Object	Core.Dictionary	6	

Notizblock

18 Zeilen.

Incremental GarbageCollect

```
DELETE FROM fields
   WHERE fields.id NOT IN (
         SELECT DISTINCT reffields.id FROM fields "reffields", fields
      WHERE
         (reffields.keymajor = 'class' AND
         reffields.valuemajor = 'Eurydike.PersonSpace')
      OR (fields.keybasetype = 'Core.Object' AND
         fields.keymajor = reffields.id)
      OR (fields.valuebasetype = 'Core.Object' AND
         fields.valuemajor = reffields.id))
```

Georg Heeg eK
www.heeg.de

Global GarbageCollect

```
DELETE FROM fields
    WHERE
      fields.id NOT IN
        ((WITH RECURSIVE refs(id) AS
          (
          SELECT fields.id FROM fields
            WHERE fields.keymajor = 'class' AND fields.valuemajor = 'Eurydike.PersonSpace'
          UNION ALL
          SELECT fields.valuemajor FROM fields, refs
            WHERE fields.valuebasetype = 'Core.Object' AND fields.id = refs.id
          )
        SELECT * FROM refs)

        UNION ALL

        (WITH RECURSIVE refs(id) AS
          (
          SELECT fields.id FROM fields
            WHERE fields.keymajor = 'class' AND fields.valuemajor = 'Eurydike.PersonSpace'
          UNION ALL
          SELECT fields.keymajor FROM fields, refs
            WHERE fields.keybasetype = 'Core.Object' AND fields.id = refs.id
          )
        SELECT * FROM refs))
```

Eurydike – Current Status

- Used in GH seaMS
  - new content management system based on seaBreeze, seaBreeze Mobile and Seaside
  - currently in initial use internally
  - planned to be available on Cincom Marketplace when released

Georg Heeg eK
www.heeg.de

Eurydike – Future – Multi User

- Life connections
- GlobalLock
  - semaphore like object
  - SELECT … FOR UPDATE
- single object lock

- ongoing research

Georg Heeg eK
www.heeg.de

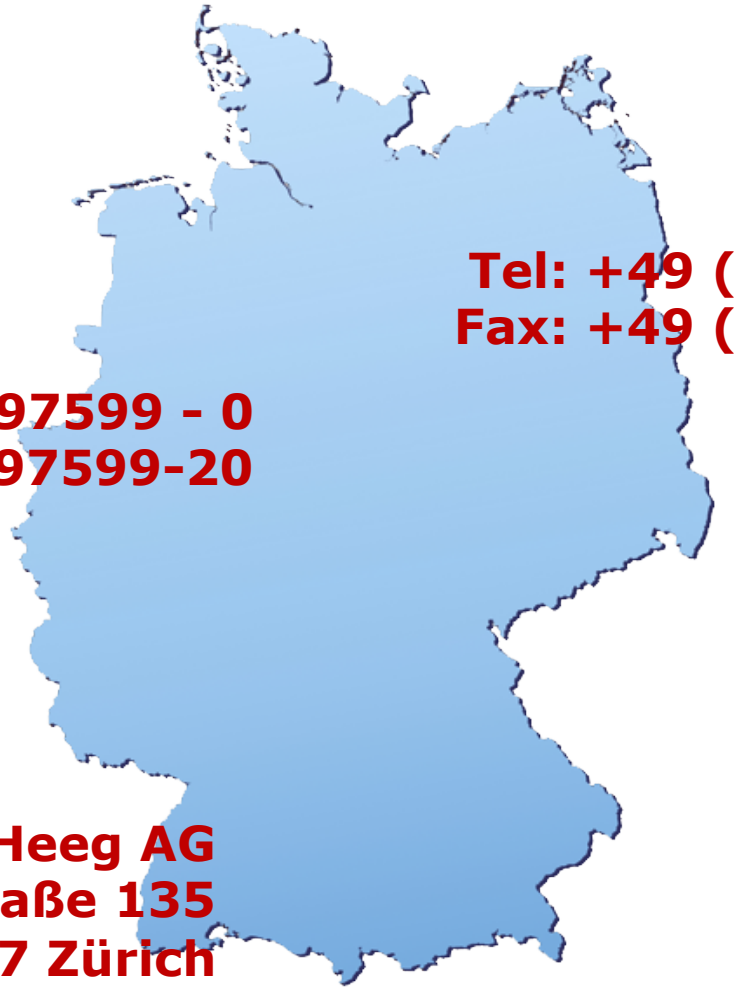
Market of Eurydike

- Easy object based persistancy in medium size projects

Georg Heeg eK
www.heeg.de

Our Team made the dream come true

Georg Heeg eK



Georg Heeg eK
Baroper Str. 337
44227 Dortmund
Germany
Tel: +49 (0)231 - 97599 - 0
Fax: +49 (0)231 - 97599-20

Georg Heeg eK
Wallstraße 22
06366 Köthen
Germany
Tel: +49 (0)3496 - 214 328
Fax: +49 (0)3496 - 214 712

Georg Heeg AG
Seestraße 135
8027 Zürich
Switzerland
Tel: +41 (848) 43 34 24

Email: georg@heeg.de
http://www.heeg.de