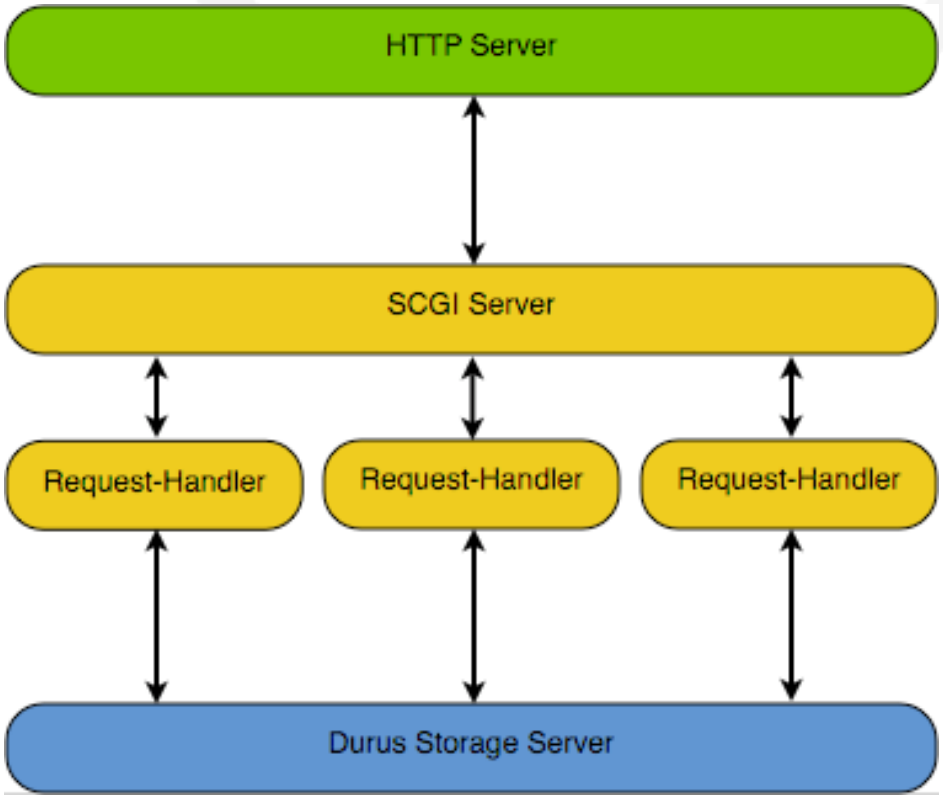
moellus
a rich and soft layer on Durus

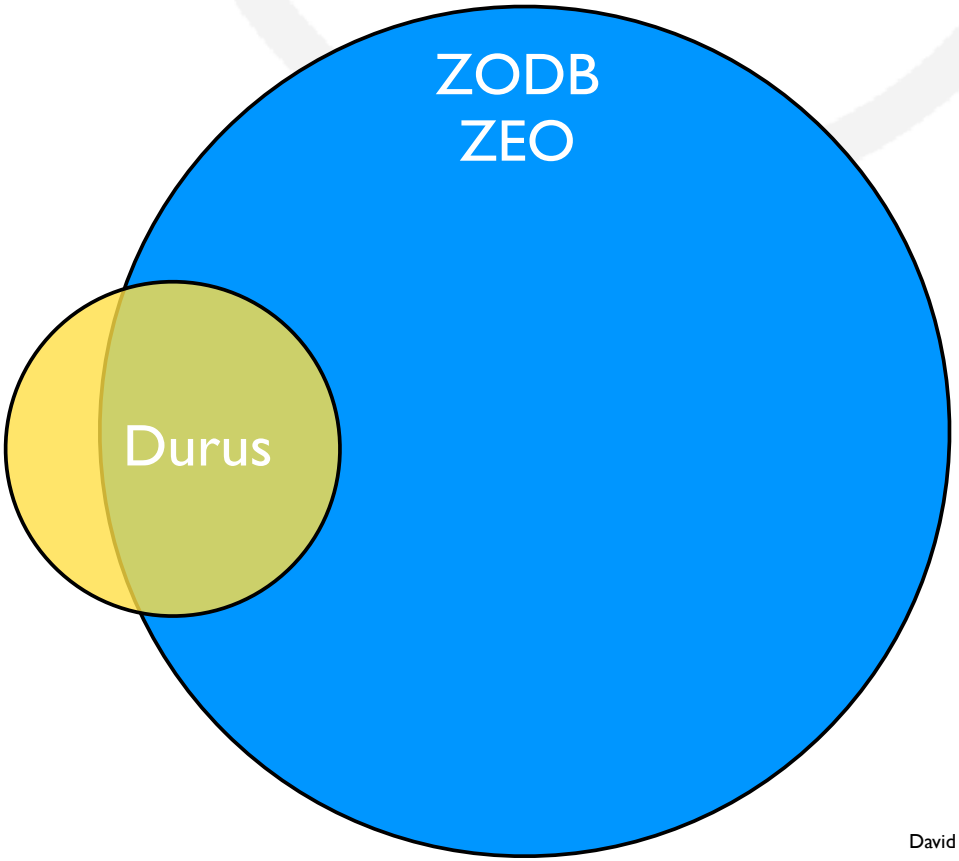
Mario Ruggier

Durus target environment



credit
David Binger

Durus simplicity



credit
David Binger

Durus provides

- Persistent class
- Transparent persistence of subclasses of Persistent
- Transactions
  changes to object graph ➟ commit() or abort()
- PersistentDict, PersistentList, BTree
- ComputedAttribute

ComputedAttribute

- Container for derived information
- Designed *not* to save any state changes
  - access is cached ➟ get(compute)
  - changes in sources ➟ invalidate()
- Avoids duplication
- Maintains consistency

persistent object graph

- Durus takes care of persistence
  - a root PersistentDict instance
  - values may be any object (persistent or not)
  - or other persistent containers
- Free-form object graph
  - Improve usability of our object graph by adding some shape
  - Better accessibility of our objects by organizing how objects are stored

moellus

a soft and rich chocolate cake or a luxuriously mellow red wine are said to be moelleux

conceptual
object schema

__plus__

conceptual
tabular storage

Containers and Items

- a moellus "table" is made up of:
  - instance of a subclass of PersistentContainer
  - instances of a subclass of PersistentItem
- root PersistentDict keys are container class names
- database instance ➟ get/set container
- container ➟ methods to manage items, indices, ...
- items ➟ properties

moellus magic attributes

```
class Container(PersistentContainer):
    __m_item_classname__ = 'Item'

class Item(PersistentItem):
    __m_container_classname__ = 'Container'
    __m_attrs__ = {}
    __m_cattrs__ = {}
    __m_keybindings__ = ()
    __m_enums__ = {}



makeup_m_classes(model)
```

attrs

- automatically generated properties, unless provided
- inherited by all items from PersistentItem
  - id
  - container
- item.get_m_attr(name)
  item.set_m_attr(name, value)
  - handle enums
  - invalidate derived attributes
  - update indices if necessary

cattrs

- Custom cattrs must supply own property definition
- Predefinitions for common cases, e.g. relationships
  - inverse of one-to-one
  - inverse of one-to-many

```
class Employee(PersistentItem):
    __m_attrs__ = { 'holds': None }

class Position(PersistentItem):
    __m_cattrs__ = { 'employees': { 'src':('Employee.holds',), 'typ':'inv/1:*' } }
```

keybindings

```
__m_keybindings__ = ('attr1', 'attr2', ...)
```

- names used may for attributes of any type
  - attr, enum attr, cattr
- corresponding tuple of values identifies item uniquely
- values must be valid when item is added to container
- an index is automatically maintained
- may be modified like any other attribute

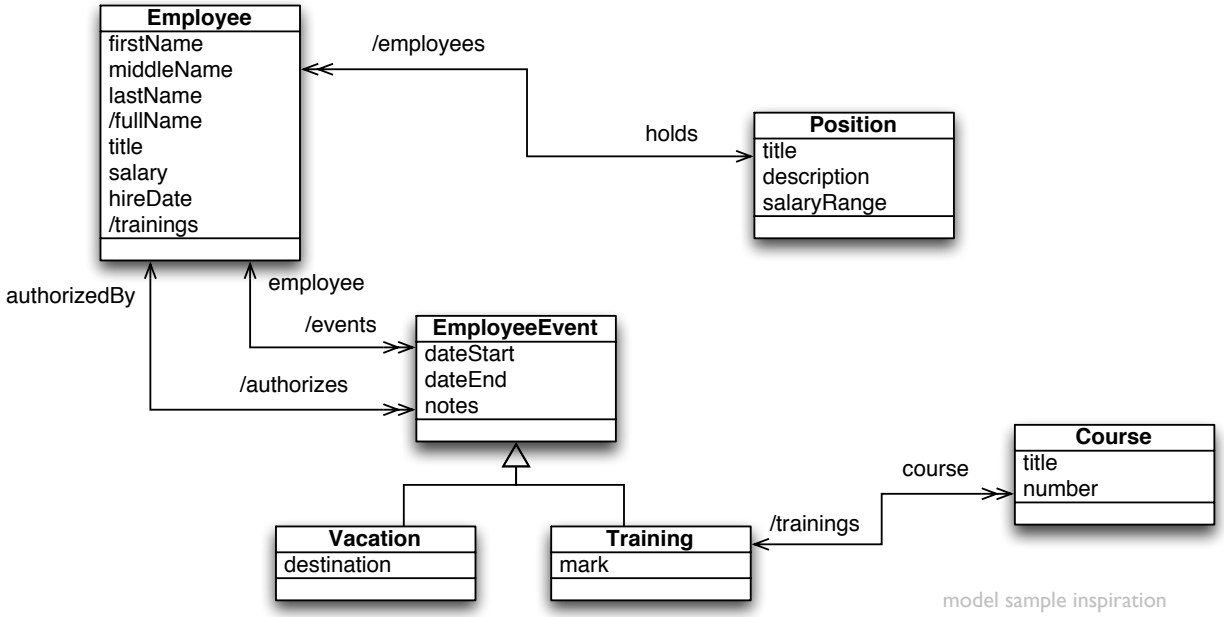
indices

```
class PersistentContainer(Persistent):
    def __init__(self):
        self._items = PersistentList()
        self._indices = PersistentDict()
        ...
        self.add_index('id')
        self.add_index(self.item_cls.__m_keybindings__)
        ...

    def get_item(self, item_key, index_key):
        ...


# adding an index to a container c results in:
c._index[('attr1', 'attr2', ...)] = PersistentDict()

# and each item in the container generates an entry:
c._index[('attr1', 'attr2', ...)][(item.attr1, item.attr2, ...)] = item
```

usefulness of keybindings / indices

- no need to depend on arbitrary ids
- no "duplicate" items
- automatically maintained keybindings index
  - Person.__m_keybindings__ = ('first','last')
  - persons.gikb(('Jack','Smack'))
- performance advantages
  - looking for an item considers only the index keys
  - can guess "key" for an item: persons.get_index().has_key(('Jack','Smack'))
  - not necessary to actually unpickle item

a sample model



model sample inspiration
http://www.agiledata.org/essays/umlDataModelingProfile.html

demo

future considerations

- Consider BTree instead of PersistentList as the internal container of items in a PersistentContainer
- Non-intrusive mechanism to do type and referential integrity checking
- Evolve web application to create/modify/delete

Released under LGPL

http://scoopics.com/software/moellus/

gracias