



Hasso
Plattner
Institut

IT Systems Engineering | Universität Potsdam



Virtual Machines

3th reading exercise:

Open, Reusable Object Model

(I. Piumarta, A. Warth, Los Angeles, CA, 2006/2007)

Robert Schuppenies, Stefan Hüttenrauch, Tobias Queck

Outline

2

- Definitions
- Problems with common Object Models (OMs)
- Details of the implementation of the presented OM
- Extensibility – Flexibility – Usability

Definitions

3

- *Object Model*
 - “Description of the structural relationships among components of a [...] object including its metadata.” [1]
 - “Defines the structural relationships and dynamic interaction between a group of [...] objects.” [2]

- *Open*
 - Changeable and adaptable by the end user
 - Only few syntax and restrictions

- *Reusable*
 - Code reuse... what we all want

Problems with common Object Models

4

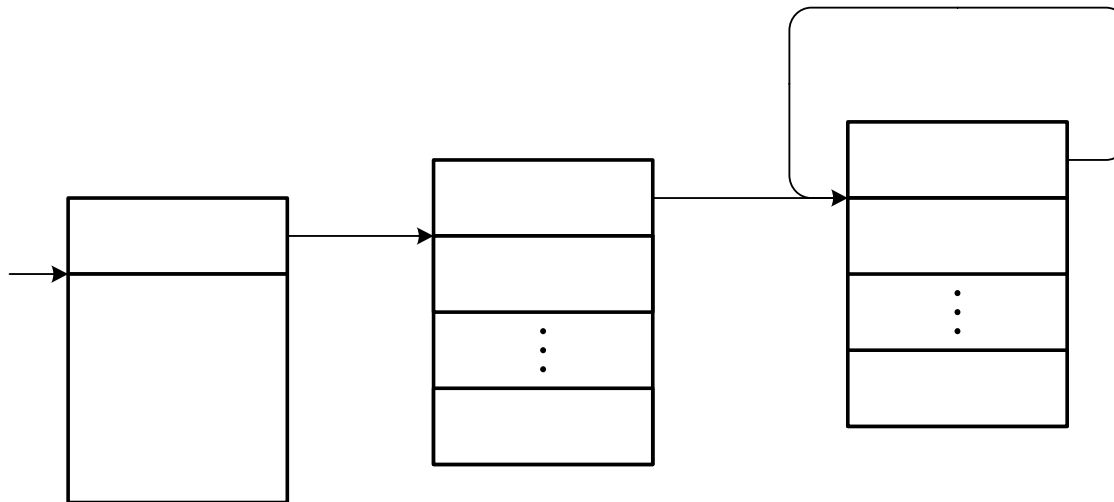
- Problems with OMs
 - Implemented in language on lower level of abstraction
 - No customizing
 - Execution mechanisms cannot be changed
 - Static

- A way out
 - A more generic OM → that adds complexity
 - A different implementation of the Object Model

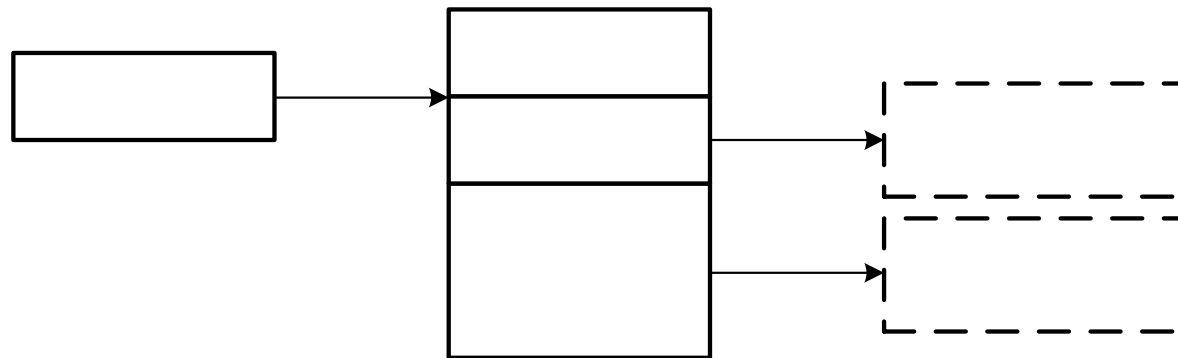
Objects = state + behaviour

5

- Nothing but objects
- Object = state + behaviour



Objects = state + behaviour, cont.



Closure object

behaviour

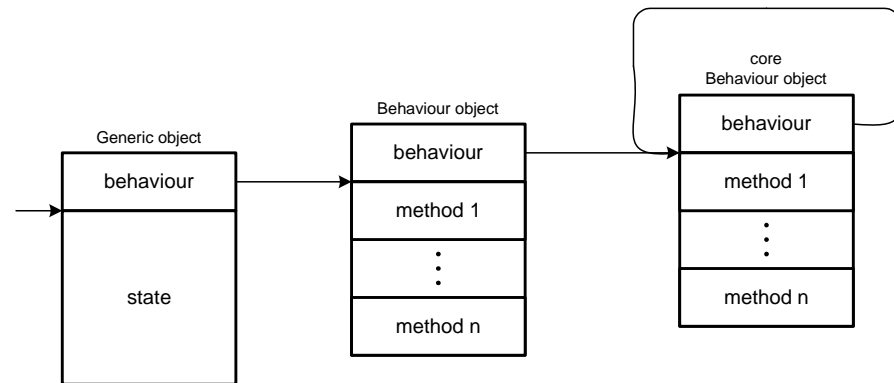
method name

method
implementation

Let's see some code: prototypes

```
class struct_object(object):
    ...
    self._vt = {}
```

```
class struct_vtable(object):
    ...
    self._vt = {}
    ...
    self.keys = []
    self.values = []
    self.parent = None
```

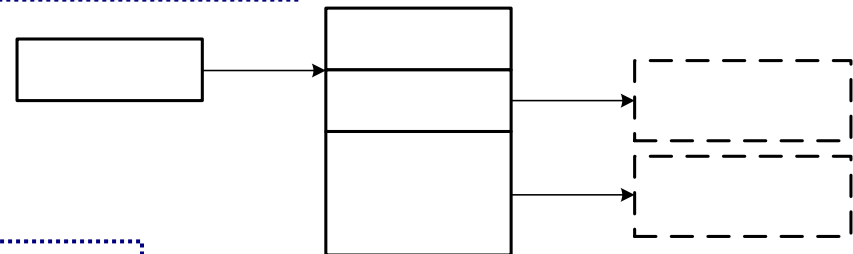
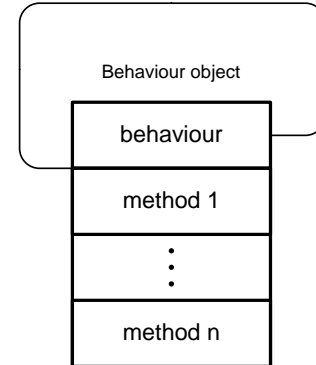


```
class struct_closure(object):
    self._vt = {}
    self.method = None
    self.data = None
```

```
class struct_symbol(object):
    ..
    self._vt = {}
    self.string = None
```

.. code: methods

```
def vtable_addMethod(closure, self, symbol, method):
    ...
def vtable_lookup(closure, self, symbol):
    ...
def vtable_allocate(closure, self, size):
    ...
def vtable_delegated(closure, self):
    ...
```



```
def symbol_intern(closure, self, string):
    ...
```

```
def closure_new(method, data):
    ...
```

.. code: bindings

```
def send(object, messageName, *args):  
    ...  
def bind(object, messageName):  
    ...
```

.. code: bootstrapping

```
global SymbolList
global vtable_vt, object_vt, symbol_vt, closure_vt
global s_lookup, s_addMethod, s_allocate, s_delegated

vtable_vt = vtable_delegated(0,0)
vtable_vt._vt["-1"] = vtable_vt

object_vt = vtable_delegated(0,0)
object_vt._vt["-1"] = vtable_vt
vtable_vt.parent = object_vt

symbol_vt = vtable_delegated(0,object_vt)
closure_vt = vtable_delegated(0,object_vt)

SymbolList = vtable_delegated(0,0)

s_lookup    = symbol_intern(0, 0, "lookup")
s_addMethod = symbol_intern(0, 0, "addMethod")
s_allocate   = symbol_intern(0, 0, "allocate")
s_delegated  = symbol_intern(0, 0, "delegated")

vtable_addMethod(0, vtable_vt, s_lookup, vtable_lookup)
vtable_addMethod(0, vtable_vt, s_addMethod, vtable_addMethod)

send(vtable_vt, s_addMethod, s_allocate, vtable_allocate)
send(vtable_vt, s_addMethod, s_delegated, vtable_delegated)
```

Flexibility examples

11

- Adding data types
 - Creating a vtable
 - Putting functionality into the vtable
- Converting single inheritance to multiple inheritance
 - Modify the semantics of message sending
 - Redefine lookup
 - Convert single parent vtable to parent list
 - Look for method in all parents
 - Taking state of parents into account

Evaluation

12

- Compact implementation
- Benchmarks
 - Overhead of dynamic dispatch nearly factor 2
 - Object based primitives
 - Slower without cache (0.6) but faster with cache (2)
- Traits
- Limitations
 - Model relies on method cache
 - Implementation of bind and send

Questions & Answers



References

14

- [1] <http://www.cs.cornell.edu/wya/DigLib/MS1999/glossary.html>
- [2] <http://www.haley.com/helpcenter/glossary.html>
- [3] www.st.informatik.tu-darmstadt.de/database/seminars/data/SWTrends_Reflection_Radic.pdf?id=179