# VMADL

## An Architecture Definition Language for Variability and Composition of Virtual Machines

Stefan Marr

9 August 2008

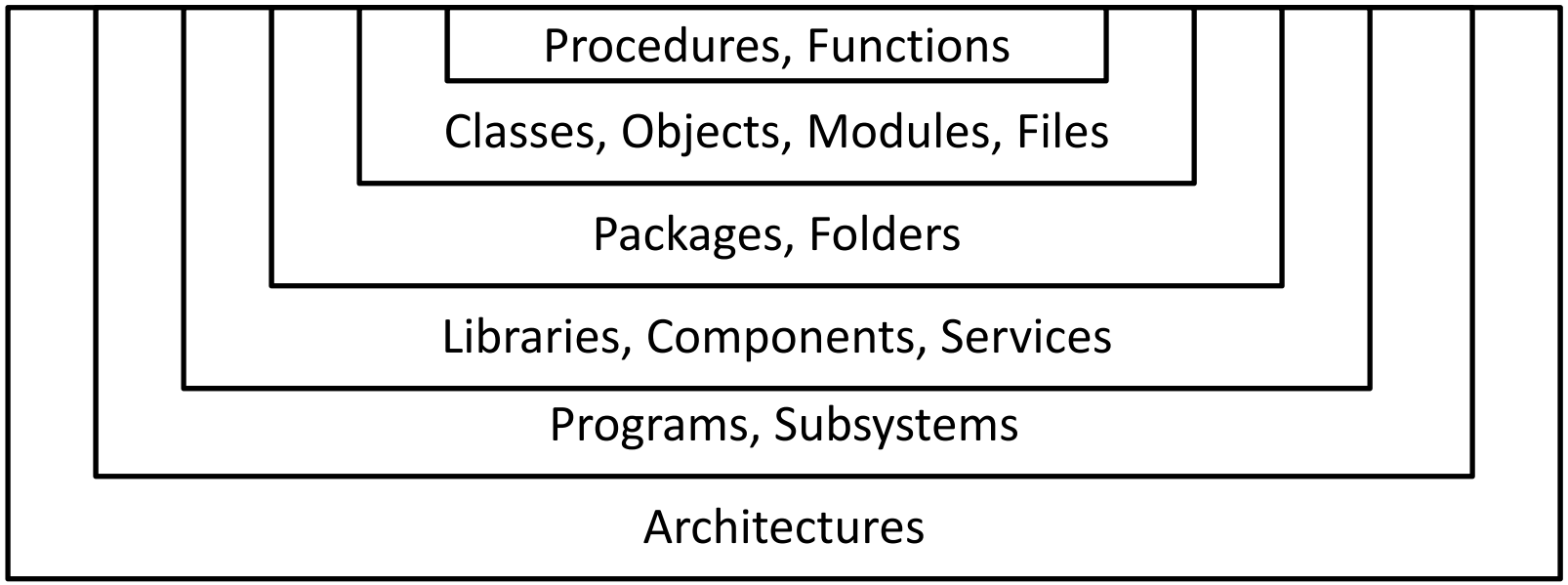
SVPP 08

HPI Hasso Plattner Institut

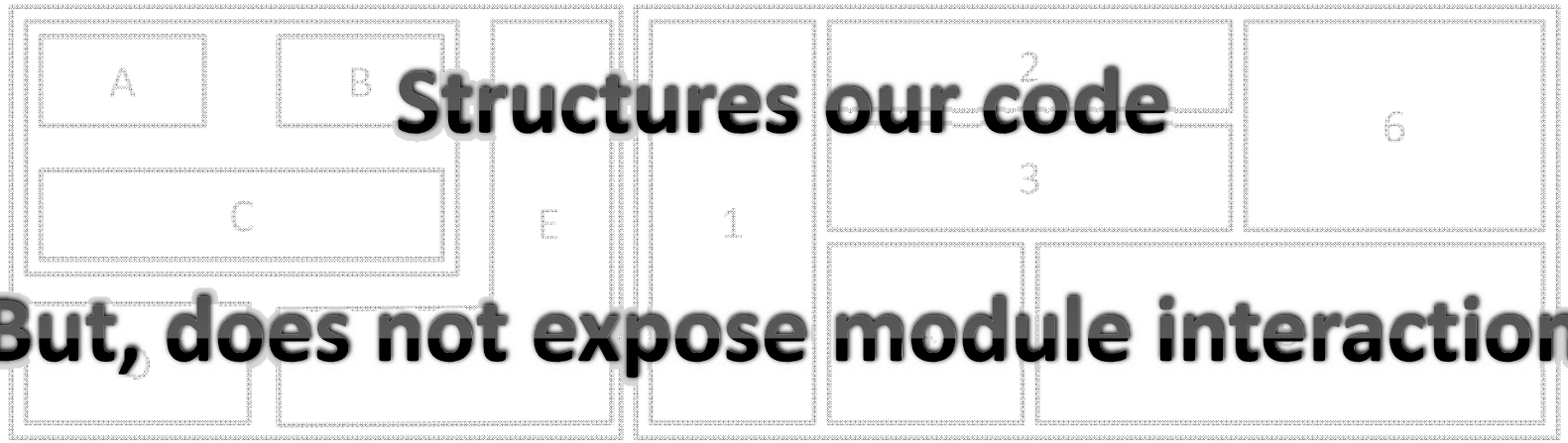
# Agenda

1. Problems with Modularization
2. VM Architecture Definition Language
3. CSOM and VMADL
4. Summary

# Modularization



# Modularization




- Semantic grouping
- Encapsulation
- Separation of Concerns
- Divide and Conquer

# Interaction?
# Read the Code!

```
void initialize_system_class(pVMClass system_class,
    pVMClass super_class, const char* name
) {
    pString s_name = String_new(name);

    // Initialize the superclass hierarchy
    if(super_class != NULL) {
        SEND(system_class, set_super_class, super_class);
        pVMClass sys_class_class = SEND((pVMObject)system_class, get_class);
        pVMClass super_class_class = SEND((pVMObject) super_class, get_class);
        SEND(sys_class_class, set_super_class, super_class_class);
    } else {
        pVMClass sys_class_class = SEND((pVMObject)system_class, get_class);
        SEND(sys_class_class, set_super_class, class_class);
    }

    pVMClass sys_class_class = SEND((pVMObject)system_class, get_class);

    // Initialize the array of instance fields
    SEND(system_class, set_instance_fields, VMArray_new(0));
    SEND(sys_class_class, set_instance_fields, VMArray_new(0));

    // Initialize the array of instance invokables
    SEND(system_class, set_instance_invokables, VMArray_new(0));
    SEND(sys_class_class, set_instance_invokables, VMArray_new(0));

    // Initialize the name of the system class
    SEND(system_class, set_name, Universe_symbol_for(s_name));
    pString class_class_name =String_new_from(s_name);
    SEND(class_class_name, concatChars, " class");
    SEND(sys_class_class, set_name, Universe_symbol_for(class_class_name));

    // Insert the system class into the dictionary of globals
    Universe_set_global(SEND(system_class, get_name), (pVMObject)system_class);
}
```

```
void Universe_initialize(int argc, pString* argv) {
    /*
     * affected globals:
     * affected file globals: globals_dictionary
     */

    // setup the Hashmap for all globals
    globals_dictionary = Hashmap_new();
    // init the Symboltable
    Symbol_table_init();
    ////////////////////////////////
    //     allocate the nil object
    nil_object = VMObject_new();
    // allocate the Metaclass classes
    metaclass_class = new_metaclass_class();
    // allocate the rest of the system classes
    object_class    = new_system_class();
    nil_class       = new_system_class();
    class_class     = new_system_class();
    array_class     = new_system_class();
    symbol_class    = new_system_class();
    method_class    = new_system_class();
    integer_class   = new_system_class();
    biginteger_class= new_system_class();
    frame_class     = new_system_class();
    primitive_class = new_system_class();
    string_class    = new_system_class();
    double_class    = new_system_class();

    // setup the class reference for the nil object
    SEND(nil_object, set_class, nil_class);

    // initialize the system classes.
    initialize_system_class(object_class, NULL, "Object");
    initialize_system_class(class_class, object_class, "Class");
    initialize_system_class(metaclass_class, class_class, "Metaclass");
    initialize_system_class(nil_class, object_class, "Nil");
    initialize_system_class(array_class, object_class, "Array");
    initialize_system_class(method_class, array_class, "Method");
    initialize_system_class(symbol_class, object_class, "Symbol");
```

```
// load methods and fields into the system classes
    Universe_load_system_class(object_class);
    Universe_load_system_class(class_class);
    Universe_load_system_class(metaclass_class);
    Universe_load_system_class(nil_class);
    Universe_load_system_class(array_class);
    Universe_load_system_class(method_class);
    Universe_load_system_class(symbol_class);
    Universe_load_system_class(integer_class);
    Universe_load_system_class(biginteger_class);
    Universe_load_system_class(frame_class);
    Universe_load_system_class(primitive_class);
    Universe_load_system_class(string_class);
    Universe_load_system_class(double_class);

    // load the generic block class
    block_class =
            Universe_load_class(Universe_symbol_for_chars("Block"));

    // setup the true and false objects
    true_object = VMObject_new_instance(
                Universe_load_class(Universe_symbol_for_chars("True"))
            );
    false_object = VMObject_new_instance(
                Universe_load_class(Universe_symbol_for_chars("False"))
            );

    // load the system class and create an instance of it
    system_class =
            Universe_load_class(Universe_symbol_for_chars("System"));
    pVMObject system_object = VMObject_new_instance(system_class);

    // put special objects and classes into the dictionary of globals
    Universe_set_global(Universe_symbol_for_chars("nil"), nil_object);
    Universe_set_global(Universe_symbol_for_chars("true"),
            true_object);
    Universe_set_global(Universe_symbol_for_chars("false"),
            false_object);
    Universe_set_global(Universe_symbol_for_chars("system"),
            system_object);
    Universe_set_global(Universe_symbol_for_chars("System"),
                (pVMObject)system_class);
    …
```

# Architecture and Modification

# Problems with Modularization in Virtual Machines

- Complex problem domain
- Highly tangled module dependencies
- Common modularization is insufficient
- Portability issues
  - Different operating systems
  - CPU architectures
- Adaptation needs for specific environments or requirements

Virtual Machine Architecture Definition Language

# VMADL

# Virtual Machine Architecture Definition Language

- Language to define service modules
  - Proposed by M. Haupt et. al. [8]
  - Module on architectural level
  - Consists of several implementation modules
- Definition of bidirectional interfaces
  - Typical functions
  - And “points of interests”, i.e., pointcuts
- Independent of a concrete implementation language

# Service Module Definition

```
service GCMarkSweep {
  require Memory;
  require VM;
  require VMObjects;

  // implicit sections for interface definitions
  void gc_collect();

  expose {
    pointcut mark_object(void* _self) =
      execution("void gc_mark_object(...)")
        && args(_self);
  }

  refine VMObject {
    int gc_field { before fields[0] }
  }
}
```

# Startup And Shutdown Sections

```
service GCMarkSweep {
  require Memory;
  require VM;
  require VMObjects;
  ...

  // support for initialization and cleanup phases
  startup {
    void initialize();
    expose { ... }
  }

  shutdown {
    void cleanup();
    expose { ... }
  }
}
```

# Service Module Combination

```
combine GCMarkSweep, Memory {
  advice execution("void* memory_allocate(unsigned int)")
   && args(size) : around(unsigned int size) {
    *tjp->result() = gc_allocate(size);
  }

  advice execution("void memory_free(void*)")
   && args(ptr) : around(void* ptr) {
    gc_free(ptr);
  }
}

combine GCMarkSweep, VMObjects {
  advice execution(VMObjects::initializer()) : around() {
    gc_start_uninterruptable_allocation();
    tjp->proceed();
    gc_end_uninterruptable_allocation();
  }
}
```

# Contribution to Modules

```
service ObjectModel {

  Basics {
    /* Named part of a module definition */

    #define SEND(O,M,...) \
       ({ typeof(O) _O = (O); \
       (_O->_vtable->M(_O , ##__VA_ARGS__)); \
       })

    #define IS_A(object,class) \
       (((class *)object)->_vtable
          == class##_vtable())

    ...
  }
}
```

# Contribution to Modules

```
service TaggedIntOne {
  replace ObjectModel.Basics {
    #define INSTANCE_POINTER_ACCESS(O) \
     ({ (VMINTEGER_IS_TAGGED(O) ? (void*)VMInteger_Global_Box() : O); })

    #define SEND(O,M,...) \
      ({ typeof(O) _Org = (typeof(O))(O); \
         typeof(_Org) _O =
              (typeof(_Org))(INSTANCE_POINTER_ACCESS(_Org));\
         (_O->_vtable->M(_Org, ##__VA_ARGS__)); \
      })

    #define IS_A(object,class) \
        (((class *)INSTANCE_POINTER_ACCESS(object))->_vtable
            == class##_vtable())
    ...
  }
}
```

# Class Definition Language

```
service VMObjects {
  class VMObject : OOObject {
        size_t    num_of_fields
        pVMObject fields[0]
        pVMClass  clazz

        pVMClass  get_class()
        void      set_class(pVMClass clazz)
  }
}

service GCMarkSweep {
  refine VMObject {
        int   gc_field { before fields[0] }
  }
}
```

# Current VMADL Implementation

- Almost independent of implementation languages
- Actually, C is the VM implementation language
  - Header files are generated from VMADL
- AspectC++ is used for AOP
  - Aspect files (*.ah) are generated, too
- Compiler uses templates
- Used ANTLR grammar is based on the assumption of balanced braces {}

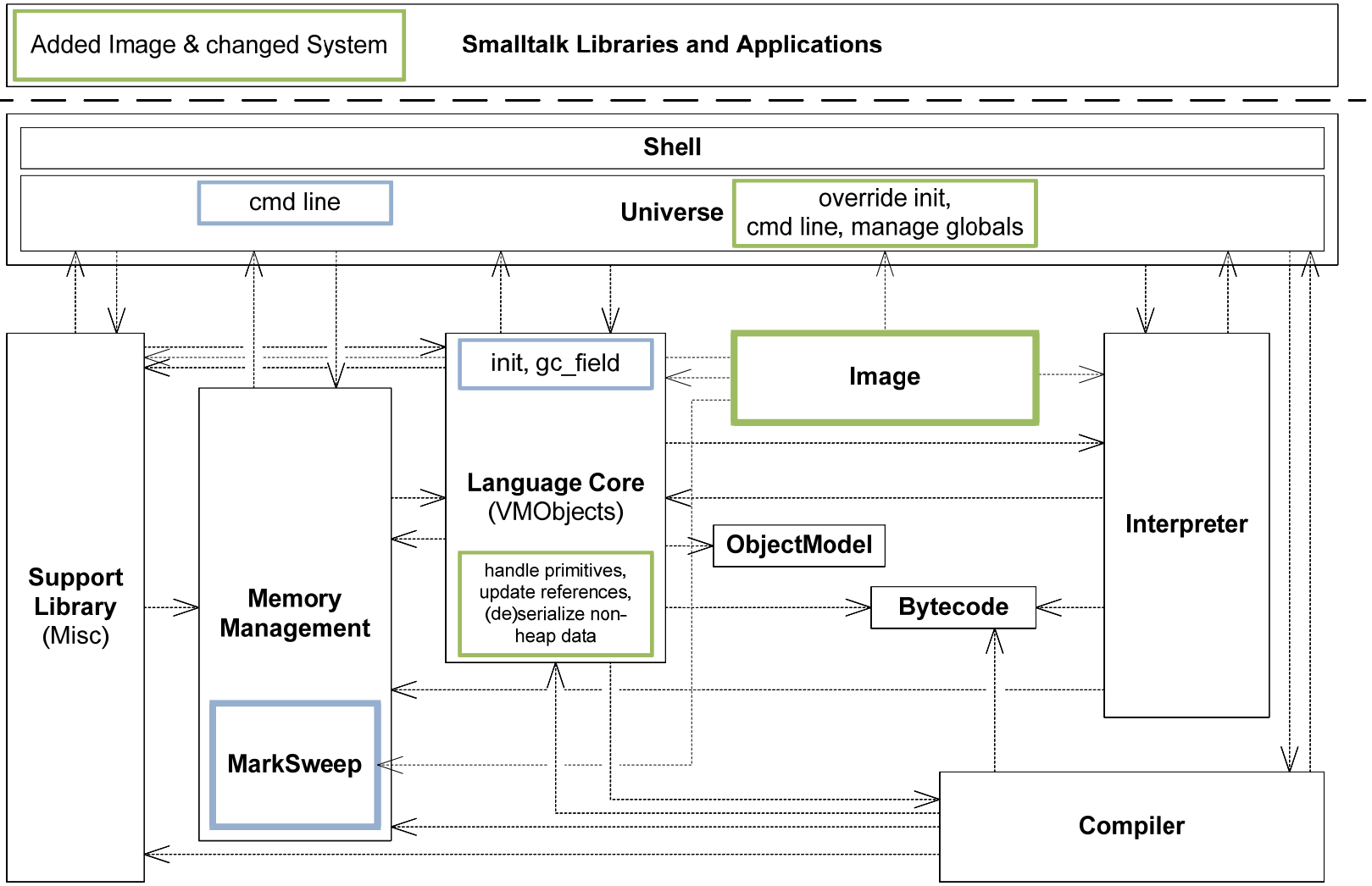
Case Study to Demonstrate Benefits of AOP and VMADL for Modularization

# CSOM AND VMADL

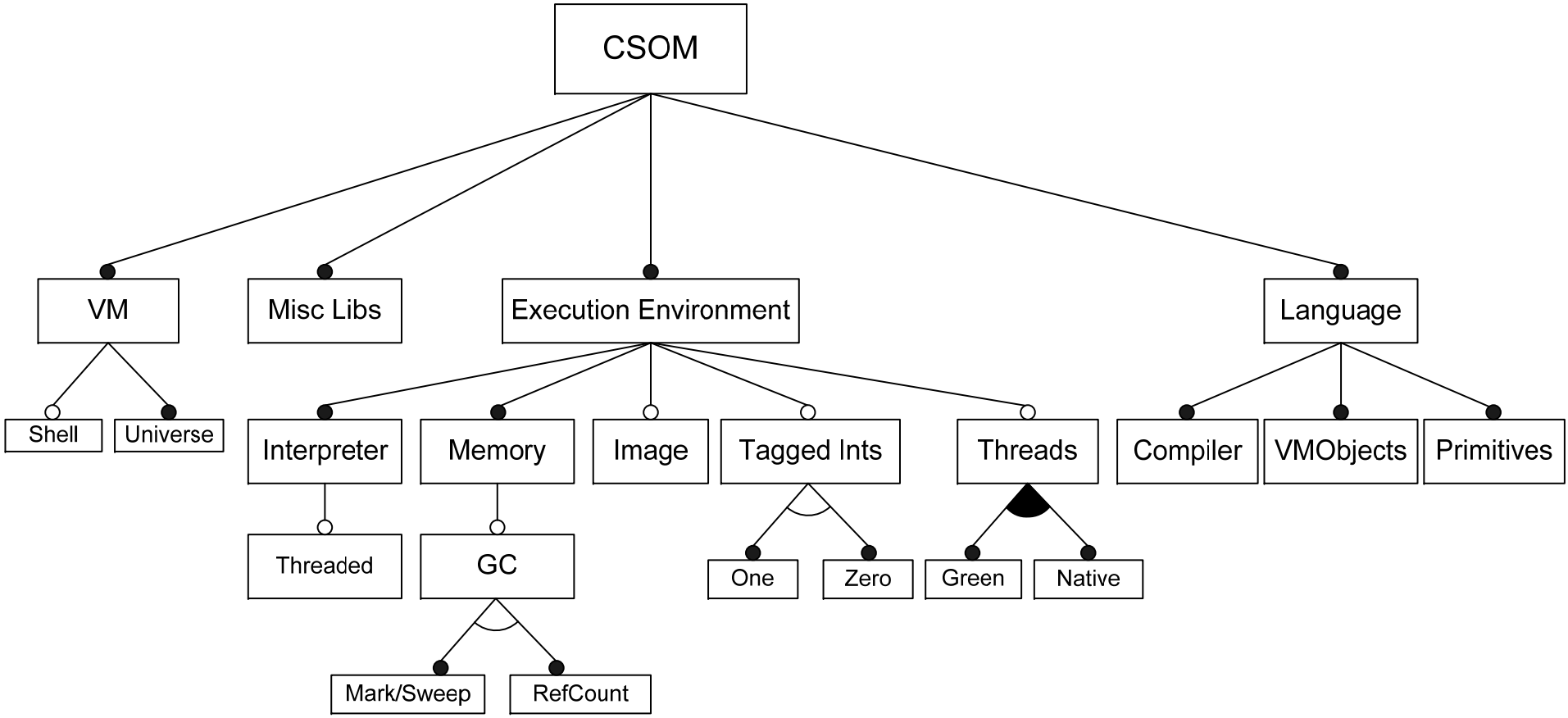
# CSOM: A Simple Smalltalk VM

- Derived from SOM by M. Haupt and T. Pape
  - Originally written in Java at the University of Århus
- Reengineered with VMADL
  - Code now reflects architecture directly
  - Variability by service modules
    - Threaded interpretation, green/native threading, reference counting GC, Smalltalk images, one/zero tagged integers
  - VM product line

# Architecture of CSOM



# CSOM Product Line



Summary and Outlook

# SUMMARY

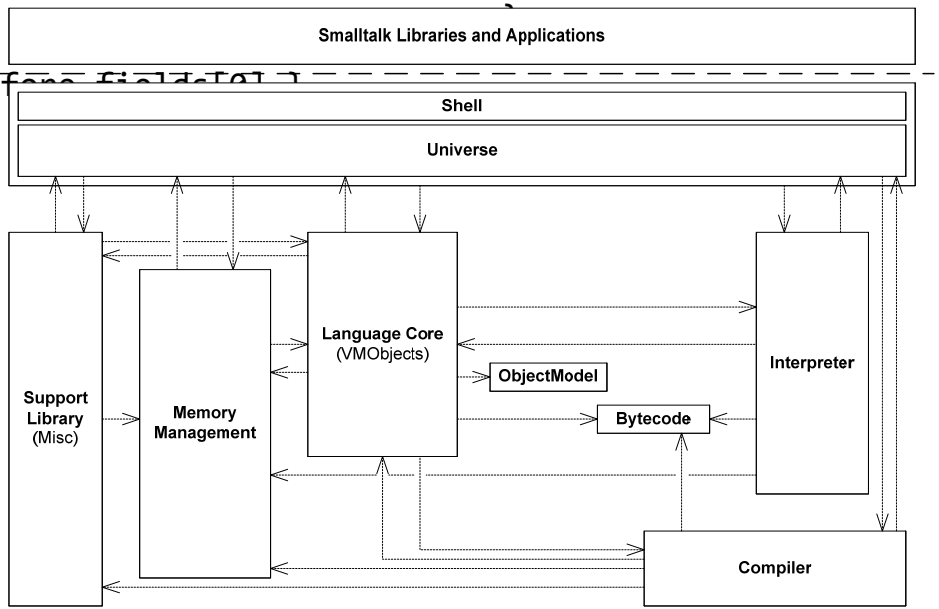
# Summary

- VMADL and bidirectional interfaces
  - Help developers to recognize module relationships
  - Explicit description of architecture
- Case study using CSOM
  - Very promising results
- Evaluation, still to be done
  - Experiment with students

# Q & A

```
service GCMarkSweep {
  require Memory;
  require VM;
  require VMObjects;

  // implicit sections for
  // interface definitions
  void gc_collect();

  expose {
    pointcut mark_object(void* _self) =
     execution("% gc_mark_object(...)")
     && args(_self);
  }

  refine VMObject {
    int gc_field { be
  }
}
```

```
combine GCMarkSweep, Memory {
  advice execution("void*
    memory_allocate(unsigned int)")
  : around(unsigned int size) {
    *tjp->result()= gc_allocate(size);
  }

  advice execution("void
    memory_free(void*)"):around(void* ptr) {
    gc_free(ptr);
  }
}
```



# Basic Literature

## Foundation

[8] M. HAUPT, B. ADAMS, S. TIMBERMONT, C. GIBBS, Y. COADY, AND R. HIRSCHFELD, *Disentangling Virtual Machine Architecture*. Under review for IET Journal special issue on Domain-Specific Aspect Languages, 2008.

### AOP and Interfaces

[2] J. ALDRICH, *Open Modules: Modular Reasoning About Advice*, in ECOOP, A. P. Black, ed., vol. 3586 of Lecture Notes in Computer Science, Springer, 2005, pp. 144–168.

[7] W. G. GRISWOLD, K. SULLIVAN, Y. SONG, M. SHONLE, N. TEWARI, Y. CAI, AND H. RAJAN, *Modular Software Design with Crosscutting Interfaces*, IEEE Software, 23 (2006), pp. 51–60.

### AOP and Operating Systems

[4] M. Y. COADY, *Improving evolvability of operating systems with AspectC*, PhD thesis, 2003. Adviser-Gregor Kiczales.

# Literature


[1] F. AFONSO, C. SILVA, S. MONTENEGRO, AND A. TAVARES, *Applying aspects to a real-time embedded operating system*, in ACP4IS '07: Proceedings of the 6th workshop on Aspects, components, and patterns for infrastructure software, New York, NY, USA, 2007, ACM, p. 1.

[3] D. BEUCHE, A. GUERROUAT, H. PAPAJEWSKI, W. SCHRODER-PREIKSCHAT, O. SPINCZYK, AND U. SPINCZYK, *The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems*, Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC?99), pp. 45–53.

[5] C. DRIVER, *Evaluation of aspect-oriented software development for distributed systems*, master's thesis, University Of Dublin, September 2002.

[6] L. K. FERRETT AND J. OFFUTT, *An empirical comparison of modularity of procedural and object-oriented software*, in ICECCS '02: Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems, Washington, DC, USA, 2002, IEEE Computer Society, p. 173.

[9] V. ISSARNY, M. CAPORUSCIO, AND N. GEORGANTAS, *A perspective on the future of middleware-based software engineering*, in FOSE '07: 2007 Future of Software Engineering, Washington, DC, USA, 2007, IEEE Computer Society, pp. 244–258.

# Literature


[10] D. LOHMANN, F. SCHELER, W. SCHRODER-PREIKSCHAT, AND O. SPINCZYK, *PURE Embedded Operating Systems – CiAO*, IEEE Intel Workshop on Operating System Platforms for Embedded Real-Time Applications, (2006).

[11] D. LOHMANN, F. SCHELER, R. TARTLER, O. SPINCZYK, AND W. SCHRÖDER-PREIKSCHAT, *A quantitative analysis of aspects in the ecos kernel*, SIGOPS Oper. Syst. Rev., 40 (2006), pp. 191–204.

[12] D. MAHRENHOLZ, O. SPINCZYK, A. GAL, AND W. SCHRÖDER-PREIKSCHAT, *An aspect-orientied implementation of interrupt synchronization in the PURE operating system family*, Proceedings of the 5th ECOOP Workshop on Object Orientation and Operating Systems, (2002).

[13] N. PATAKI, ÁDÁM SIPOS, AND Z. PORKOLÁB, *Measuring the Complexity of Aspect-Oriented Programs with Multiparadigm Metric*. 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2006.

[14] O. SPINCZYK AND D. LOHMANN, *Using aop to develop architectural-neutral operating system components*, in EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop, New York, NY, USA, 2004, ACM, p. 34.

[15] S. TIMBERMONT, B. ADAMS, AND M. HAUPT, *Towards a dsal for object layout in virtual machines*. Position Paper, 2008.