Strategic Programming by Model Interpretation and Partial Evaluation

William R. Cook
UT Austin

Presented at ACL2 Meeting
Oct 2, 2008

Web Applications

web(UI, Schema, db, request) : HTML

UI : description of user interface (pages, sections)

schema: description of data (constraints, etc)

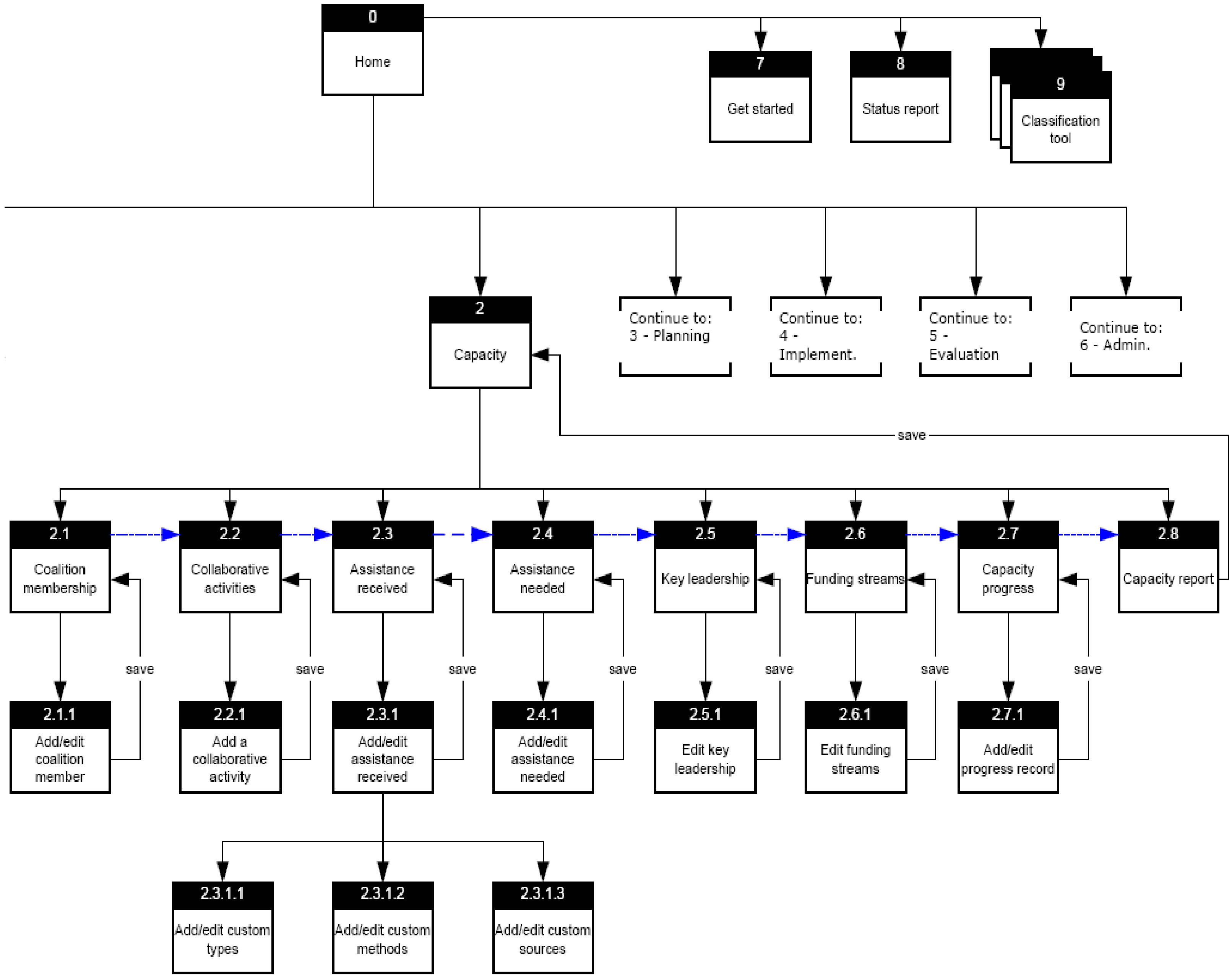
db : data store (described by schema)

request : an HTTP request

web : interpreter, with design knowledge

Addresses key problem: **integrating multiple models**

Web Site Wireframe



Web Page Wireframe

Drug Free Communities Program Monitoring and Management System	SAMHSA Logo	ONDCP Logo	Prev Platform

Home | Assessment | Capacity | Planning | Implementation | Evaluation | Administration

Home > Assessment > Community needs assessment > Edit community needs assessment

User: Status | Logout

Edit community needs assessment

Target community name

Target geographic areas:

(Hold down Ctrl to select multiple)

Specific targeted geographic areas

(Hold down Ctrl to select multiple)

Further describe the geographic areas selected above (e.g., names of cities, counties, etc.).

Assessment summary

Targeted gender

Targeted grade

(Hold down Ctrl to select multiple)

Partial Evaluation of Web Interpreter

web$_{[UI,\ Schema]}$ (db, request) : **HTML**
static *dynamic*

web$_{[UI,\ Schema]}$ is partial evaluation of web with respect to UI model and data schema

- Dynamic web site (wiki-style) or compiled
- Possible to generate both web and GUI

Partial Evaluation

Given a function of two arguments

```
(define (pow n m)   ;; computes m^n
   (cond ((= n 0) 1)
         (t (* m (pow (- n 1) m)))))
```

If the first argument is known statically, then

```
(pow 3 x) == (* x x x)
(pow 6 x) == (* x x x x x x)
```

Eliminates the recursive and conditional overhead

Partial Evaluation

A better strategy

```
(define (pow2 n m)   ;; computes m^n
  (cond ((= n 0) 1)
        ((= (mod n 2) 0) (square (pow2 (/ n 2) m))))
     (t (* m (pow2 (- n 1) m)))))
```

Gives a faster *residual* program

```
(pow2  8  n) == (square (square (square n)))
(pow2  6  n) == (square (* n (square n)))
```

Partial Evaluation Function

Traditionally called **mix**

```
(mix  pow  3) = (lambda (n) (* n n n))
(mix  pow2  3) = (lambda (n) (* n (square n)))
```

mix : (A × B → C) × A → (B → C)

Looks like uncurried version of the curry function

curry: (A × B → C) → (A → (B → C))

Specification

```
(defthm mix-behavior
     (equal ((mix f a) b) (f a b)))
```

oops, its higher-order

List Comprehensions in Scheme (first-order map)

Borrowed from Haskell

(**for** var items op result)

op [result | var ∈ items]

(**for** var items op (**if** cond result))

op [result | var ∈ items, cond]

Examples

(for x ’(1 2 3) + x)) ➔ 6

(for x ’(1 2 3) list (* x x))) ➔ (1 4 9)

(for x ’(1 2 3) and (even? x))) ➔ #f

(for x ’(1 2 3) list (if (odd? x)))) ➔ (1 3)

(for x ’(1 2 3) list (if (odd? x) x (- x)))) ➔ (1 -2 3)

A Normal Scheme Evaluator (first-order)

x : Variable   v : Value

e : Expression = x | v | if e e e | f(e, …, e) | op(e, …, e)

ρ : Environment = Variable → Value

$\mathcal{E}$: Expression → Environment → Value

$\mathcal{E}[\![v]\!]\rho = v$

$\mathcal{E}[\![x]\!]\rho = \rho(x)$

$\mathcal{E}[\![\text{if } e_1\ e_2\ e_3]\!] = \textbf{if } \mathcal{E}[\![e_1]\!] \textbf{ then } \mathcal{E}[\![e_2]\!] \textbf{ else } \mathcal{E}[\![e_3]\!]$

$\mathcal{E}[\![f(e_1, \ldots, e_n)]\!]\rho = \mathcal{E}[\![e]\!]\rho'$

  lookup definition of f: $f(x_1, \ldots, x_n) = e$

  $\rho' = [x_1 = \mathcal{E}[\![e_1]\!]\rho, \ldots, x_n = \mathcal{E}[\![e_n]\!]\rho]$

$\mathcal{E}[\![op(e_1, \ldots, e_n)]\!]\rho = \text{apply}(op, \mathcal{E}[\![e_1]\!]\rho, \ldots, \mathcal{E}[\![e_n]\!]\rho)$

From Full Evaluation to Partial Evaluation

The type of eval

eval : Expression → Environment → Value

Environment = Variable → Value

FreeVars(e) $\subseteq$ Domain(v)

All variables are bound

What about a partial evaluator?

Environment gives values to *some* variables

mix : Expression → Environment → Expression

Result might not be a complete value

(mix ′(+ x y) ′(x=3 y=2)) ➔ 5^S

(mix ′(+ x y) ′(x=3)) ➔ $(+\ 3\ y)^D$

Online Partial Evaluator (first-order)

$\mathcal{P}$: Expression $\to$ Environment $\to$ Expression

$\mathcal{P}[\![\mathrm{v}]\!]\rho = \mathrm{v}$

$\mathcal{P}[\![\mathrm{x}]\!]\rho =$ **if** $\mathrm{x} \in \mathrm{dom}(\rho)$ **then** $\rho(\mathrm{x})$ **else** $[\![\mathrm{x}]\!]$

$\mathcal{P}[\![\mathrm{if}\ \mathrm{e}_1\ \mathrm{e}_2\ \mathrm{e}_3\ ]\!]\rho =$ **case** $\mathcal{P}[\![\mathrm{e}_1]\!]\rho$ **of**

$\mathrm{v} \to$ **if** v **then** $\mathcal{P}[\![\mathrm{e}_2]\!]\rho$ **else** $\mathcal{P}[\![\mathrm{e}_3]\!]\rho$

$\mathrm{e} \to [\![\mathrm{if}\ \ \mathrm{e}\ \ \mathcal{P}[\![\mathrm{e}_2]\!]\rho\ \ \mathcal{P}[\![\mathrm{e}_3]\!]\rho\ ]\!]$

$\mathcal{P}[\![\mathrm{f}(\mathrm{e}_1,\ldots,\mathrm{e}_\mathrm{n})]\!]\,\rho = [\![\,\langle \mathrm{f},\ \rho_\mathrm{S}.\mathrm{vals}\rangle(\rho_\mathrm{D}.\mathrm{vals})\ ]\!]$

lookup definition of f: $\mathrm{f}(\mathrm{x}_1, \ldots, \mathrm{x}_n)\text{=}\mathrm{e}$

$\rho_\mathrm{S} = [\ (\mathrm{x}_i,\ \mathrm{v}_i)\ |\ i \in 1..n,\ \mathrm{v}_i = \mathcal{P}[\![\mathrm{e}_i]\!]\rho\ ]$

$\rho_\mathrm{D} = [\ (\mathrm{x}_i,\ \mathrm{e}'_i)\ |\ i \in 1..n,\ \mathrm{e}'_i = \mathcal{P}[\![\mathrm{e}_i]\!]\rho,\ \ \mathrm{e} \notin \mathrm{Value}\ ]$

create function: $\langle \mathrm{f},\ \rho_\mathrm{S}.\mathrm{vals}\rangle(\rho_\mathrm{D}.\mathrm{vars}) = \mathcal{P}[\![\mathrm{e}]\!]\,\rho_\mathrm{S}$

$\mathcal{P}[\![\mathrm{op}(\mathrm{e}_1,\ \ldots,\ \mathrm{e}_\mathrm{n})]\!]\rho =$

$$\begin{cases} \mathrm{apply}(\mathrm{op},\ \mathrm{v}_1,\ \ldots,\ \mathrm{v}_n) & \text{if}\ \ \mathrm{v}_i = \mathcal{P}[\![\mathrm{e}_i]\!]\rho \\ [\![\mathrm{op}(\mathcal{P}[\![\mathrm{e}_1]\!]\rho,\ \ldots,\ \mathcal{P}[\![\mathrm{e}_n]\!]\rho)]\!] & \text{otherwise} \end{cases}$$

Specialization Example

Specialization

```
(mix '(pow 2 *))
```

Residual code

```
(define (pow-0-* m) 1)
(define (pow-1-* m) (* m (pow-0-* m)))
(define (pow-2-* m) (* m (pow-1-* m))))
```

Residual code (with actual post-processing)

```
(define (pow-2-* m)  (* m m))
```

Issues

- In general does not terminate
- Might fail to specialize (more in a minute)

Correctness

```
(defthm ev-pe
      (implies (and (= (free-vars e) (bound-vars v))
                    (= v (append v1 v2)))
               (= (S (eval e v))
                  (mix (lift (mix e v1)) v2)
```

Evaluation in a complete environment gives the same result as partial evaluation with part of the environment, followed by partial evaluation with the result of the environment

Not proven!

Partial Evaluation of Interpreters

Consider an interpreter

<u>interpret</u>(<u>program</u>, data)

Use the same program for many data inputs: program is static

First Futamura Projection

<u>mix</u>(<u>interpret</u>, program) = compiled

Such that compiled(data) = interpret(program, data)

Second Futamura Projection

<u>mix</u>(<u>mix</u>, interpet) = compiler

Such that compiler(program) = compiled

Third Futamura Projection

mix(mix, mix) = compilerGenerator

Such that compilerGenerator(interpret) = compiler

Example: Web Applications

web(UI, Schema, db, request) : HTML

UI : description of user interface (pages, sections)

schema: description of data (constraints, etc)

db : data store (described by schema)

request : an HTTP request

web : interpreter, with design knowledge

Addresses key problem: **integrating multiple models**

Partial Evaluation of Web Interpreter

$\mathbf{web}_{[UI,\ Schema]}$ (db, request) : **HTML**

static *dynamic*

$\mathbf{web}_{[UI,\ Schema]}$ is partial evaluation of web with respect to UI model and data schema

Supports both dynamic interpretation and compiled execution in same framework

Strategic Programming

A different form of modularity

- Factor programs into:
  - General strategies
  - Specifics of problem at hand ➔ *model*

Examples (no particular order)

- Parsers (Yacc)
- Semantics Data Models
- Dependency models (Make)
- State machines (statecharts)
- Security models
- User interface models (web)
- Workflow models

Traditional Modularity: Procedural Decomposition

- Why not just use a library?
  - Grammars
    - First/follow set computation
  - Regular expressions in Java
    - variable binding problems
  - User interfaces
    - instantiate components, then run
- Use general-purpose language?
  - More difficult to analyze, optimize
  - Static becomes dynamic
- Fairly clean in Haskell combinator libraries

Models

"A schematic *description* of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics." American Heritage Dictionary

A map of Texas
Molecular model of DNA
Economic model
Regular expression
SAP Database = model of a business
Excel model of finances (project, home, etc)

Meta-Models

Description of the structure of a model

- A map of Texas
  - Planar boundaries, points, labels, elevation, vegetation, etc
- Molecular model of DNA
  - Sticks and balls, connectivity
- Economic model
  - Math (equations)
- Regular expression
  - Grammar: e ::= c | ee | e|e | e*
- Database = model of a business
  - Named tables with columns, constraints, etc
- Excel model of finances (project, home, etc)
  - Grid of values and equations, formatting

Model

Person: name=“Bob” manager=*Sue*

Person: name=“Sue” employees={*Bob*, …}

Meta-Model

Type: name=“Person” fields={

Field: name=“name” type=*String*

Field: name=“manager” type=*Person* optional inverse=*employees*

Field: name=“employees” type=*Person* many inverse=*manager* }

Meta-Meta Model

Type: name=“Type” fields={

Field: name=“name” type=*String* key

Field: name=“fields” type=*Field* many inverse=*owner* }

Type: name=“Field” fields={

Field: name=“name” Field: name=“owner” type=*Type*

Field: name=“type” type=*Type*

Field: name=“inverse” optional type=*Field*

Field: name=“cardinality” type=one,optional,many }

What Kind of Models?

User Interface

Mapping to/from data for web & GUI

Security/Authorization

Enforced as checks, provide metadata to UI

Triggers and Workflow

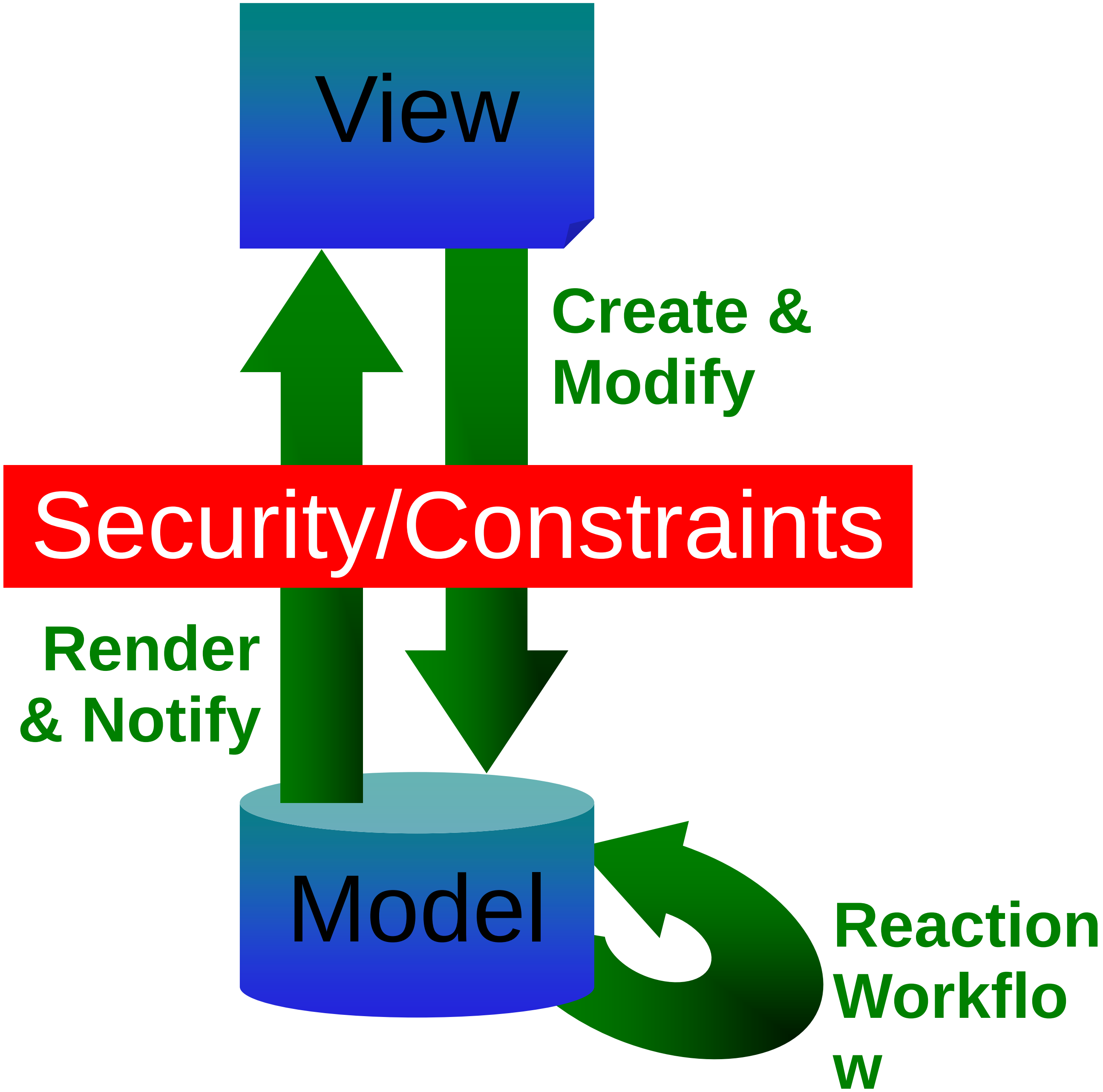
React to conditions, concurrent workflow

Semantic Data Models (graphs)

Graphs: constraints, computations, relationships

Generic Operations

Read, write, parse, compare, diff, merge, …


View
Create & Modify
Security/Constraints
Render & Notify
Model
Reaction Workflow

One Example of Potential Benefit

Acrobat API

- 6000 pages of documentation (+ 1310 PDF)
- 395 structures, 1293 functions, 89 enumerations

Microsoft Word Automation API

- 638 interfaces, 6621 functions, 325/3127 enums

SAP

- 20,000 to 50,000 tables

Each with UI, security, workflow, consistency…

Semantic Data Model (1978)
(model = description of something of interest)

BoundedValue
    x : Integer
        enforce $min \leq x \wedge x \leq max$
    min: read-only Integer
    max: Integer

Java interpretation

```
class BoundedValue {
 private int x ;
 private int min;
 private int max;

  // read-only values
 public BoundedValue(int min )  {
    this.min = min ;
  }

  // x : Integer
  int getX()   { return x ; }
  void setX( int val ) {
    if (!( min ≤ val ∧ val ≤ max))
        error ("Constraint Violation");
    x = val ;
  }
```

```
// min: read-only Integer
 int getMin()   { return min; }

 // max: Integer
 int getMax()   { return max; }
 void setMax(int val )  {
   if ( val > max)
      error("Constraint Violation");
   max = val;
}}
```

Model → Java code

```
TransJava(Type T) {
  write (" class " + T.name + " {");
  // omit generation of member variables
  for ( Field F in T. fields ) {
    write (" public " + F.type + " get" + capitalize(F.name) + "() {");
    write ("     return " + t.name + ";");
    write (" }");
    if (! F. read- only ) {
      write (" public " + F.type + " set" + capitalize(F.name));
      write (                                                "(" + F.type + " val) {");
      for ( Field S in T. Fields )
        if ( affects (F. name, S. constraint ))
          ... // generate constraint code with x replaced by val
      write (" " + t.name + " = val;");
      write ("}");
   }
   write ("}");
}}
```

Can also be done with code quoting

Strategic Programming with Interpreters

Translate : Model → Code

- Run the code to create objects

Define data abstraction using an interpreter

Factory : Model → (Message → Value)

Generic Object Factory

```
Factory (Type T) {
  D = new Hashtable();
  return lambda (Message m) {
    // iterate over fields in type description
    for (Field F in T.fields)
      if (m.match(“get” + capitalize(F.name) )
        return D.get(F.name)
      if (!F.read-only) {
        if (m.match(“set” + capitalize(F.name), val )) {
          for (Field S in T.Fields)
            if (affects(F.name, S.constraint))
              if (! eval (S. constraint , {F.name = val}))
                 error ("Constraint violation")
       D.set(F.name, val );
       return;
  }}
```

Residual Code

```
BoundedValue_Factory() {
  D = new Hashtable();
  return lambda (Message m) {
      if (m.match("getX"))
        return D.get("x")

      if (m.match("setX", val))
         if (!(D.get("min") ≤ val
                ∧ val ≤ D.get("max")))
            error ("Constraint violation")
        D.set("x", val)

      if (m.match("getMin"))
        return D.get("min")

      if (m.match("getMax"))
        return D.get("max")

      if (m.match("setMax", val))
         if (!(D.get("x") ≤ val))
            error ("Constraint violation")
        D.set("max", val)
}}
```

Generic Operations

Model

Tree
  value : Integer
  left : Tree
  right : Tree

Tree Equality

$\text{Equal}_{\text{Tree}}$(a, b)
  return $\text{Equal}_{\text{Integer}}$(a.value, b.value)
      && $\text{Equal}_{\text{Tree}}$(a.left, b.left)
      && $\text{Equal}_{\text{Tree}}$(a.rght, b.right)

32

Generic Equality

```
Equal(M, a, b)
  if (M.primitive)
    cast(M.type, a) == cast(M.type, b)
  else
    for all F in M.fields
      Equal(F.type, a.(F.name), b.(F.name))
```


33

Generic Equality

```
Equal(M, a, b)
  if (M.primitive)
    cast(M.type, a) == cast(M.type, b)
  else
    for all F in M.fields
      Equal(F.type, a.(F.name), b.(F.name))
```

Partially Evaluate Equal(Tree, a, b)...

Specialized Equality

$Equal_{Tree}(a, b)$
  return $Equal_{Integer}$(a.value, b.value)
      && $Equal_{Tree}$(a.left, b.left)
      && $Equal_{Tree}$(a.rght, b.right)

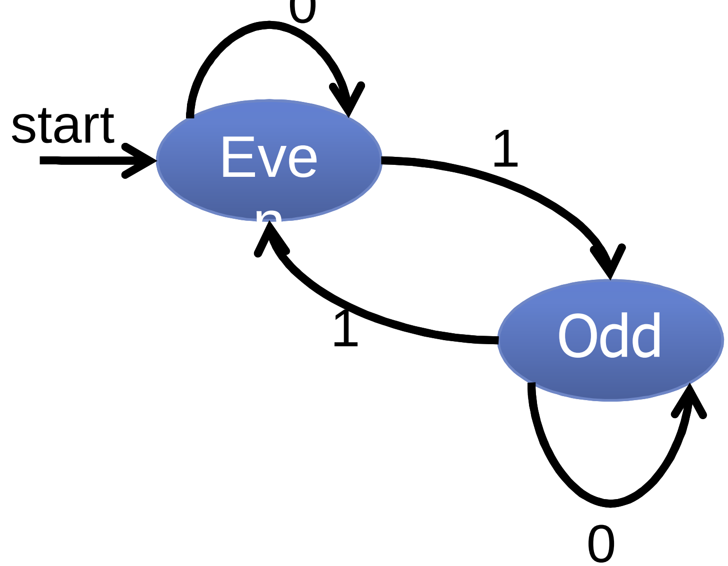
Issues with Partial Evaluation

Mixing dynamic and static values

evaluate all calls where *all arguments* are static

Example: Finite State Machine interpreter

```
(define EvenOdd
  '( ((even 1)  odd)
     ((even 0)  even)
     ((odd  1)  even)
     ((odd  0)  odd)))
```



Issues with Partial Evaluation

Mixing dynamic and static values

```
(define (run FSM state input)
  (if (null? input) state
      (let ((new (assoc (list state (car input)) FSM)))
        (if new   ;; new is dynamic, not static
            (run FSM (cadr new) (cdr input))
            `(error ,state ,(car input))))))
```

Assoc is called with a dynamic and a static value

Static table: A → B and a dynamic key a

b = lookup(table, a)

Result is *dynamic,* so little specialization takes place

The "Trick"

Convert lookup to *iteration*

```
(define (run FSM state input)
  (if (null? input) state
    (for trans FSM first
      (if (and (equal? (caar trans) state)
              (equal? (cadar trans) (car input)))
        (run FSM (cadr trans) (cdr input)))
      `(error ,state ,(car input)))))
```

Unrolls the static table into a series of conditions in effect, a case statement

Allows specialization of the table results

Residual Code

```
(define (run-even input)
  (if (null? input)
      'even
  (if (equal? 1 (car input))
      (run-odd (cdr input))
  (if (equal? 0 (car input))
      (run-even (cdr input))
      (list 'error 'even (car input))))))
```

```
(define (run-odd input)
  (if (null? input)
      'odd
  (if (equal? 1 (car input))
      (run-even (cdr input))
  (if (equal? 0 (car input))
      (run-odd (cdr input))
      (list 'error 'odd (car input))))))
```

Deforestation

Separate logical structure from rendering

- web – creates S-Expression representation of web page
- html – renders S-Expression as text

Web server composes these functions

- server(request) = html(web(UI, DM, db, request))

Need deforestation to remove intermediate structures

Partially Static Values

Consider

```
web:  (....   (list ’TABLE ’((BORDER 1)) (web body...))  )
html:  (... (print “<“) (print (car x))
                  (for attr (cadr x) ...) (print “>”)
                (html (caddr x))
              (print “</“) (print (car x)) (print “>”))
```

If we can bring web and html together:

```
html-web: (... (print “<TABLE BORDER=1>“)
                 (html-web-body ...)
                  (print “</TABLE>”))
```

Note that (list ...) is partially static

The top level of the list is static

Deforestation

```
;; rendering to HTML text
(define (html x)
 (if (not (pair? x))
     (print x)
     (begin
      (print "<") (print (car x))
      (for attr (cadr x) begin
         (print " ")
         (print (car attr))
         (print "=\"")
         (print (cadr attr))
         (print "\""))) 
      (print ">")
      (for c (cddr x) begin (html c))
      (print "</")
      (print (car x))
      (print ">\n")))
```

```
;; logical structure of table
`((TABLE ((padding 0))
   (TR ()  ,@(for sub (@ layout 'contents) cons
                   `(TD () ,(@ sub 'label))))
  ,@(for item part cons
    `(TR () ,@(for sub (@ layout 'contents) cons
                   `(TD () ,@(render sub type item
request)))
```

```
;; supercompiled version
(print "<TABLE padding=0><TR>")
(for sub (@ layout 'contents) begin
     (print "<TD>")
     (print (@ sub 'label))
     (print "</TD>\n"))
(print "</TR>\n")
(for item (@ part 'items) begin
     (print '"<TR>")
     (for sub (@ layout 'contents) begin
          (print '"<TD>")
          (html-render sub type item request)
          (print '"</TD>\n")))
     (print '"</TR>\n")))
(print '"</TABLE>\n"))
```

Deforestation with Partial Evaluation

Integrate with partial evaluation

Merge: (f a.. (g b..) c..) ==> (f-g a.. b.. c..)
Driving: (f (if a b c)) ==> (if a (f b) (f c))

Specialize on partially static values

(cons x y) is partially-static; use axioms

(null? (cons x y)) ➔ #f

(car (cons x y)) ➔ x

(cdr (cons x y)) ➔ y

Environment = Variable $\rightarrow$ (Value^S + Expr^D)

Generic Reader

Read(Model, Data, Factory) : Object

Specialized reader

- $\text{Read}_{\text{Model}}$(Data, Factory)
  - Read then patch up circularity

Object graph instantiator

- $\text{Read}_{\text{Model, Data}}$(Factory)
  - Fixup table is static when data is static
  - But dynamic objects must be inserted into it

Operators to control binding time

(dynamic e [test])

- Force e to be dynamic
  - If test is not given, or is dynamic
- (dynamic (make-table) data)
  - Make a static table if data is static

(indirect (var e) body)

- Bind dynamic e to a static identifier within body

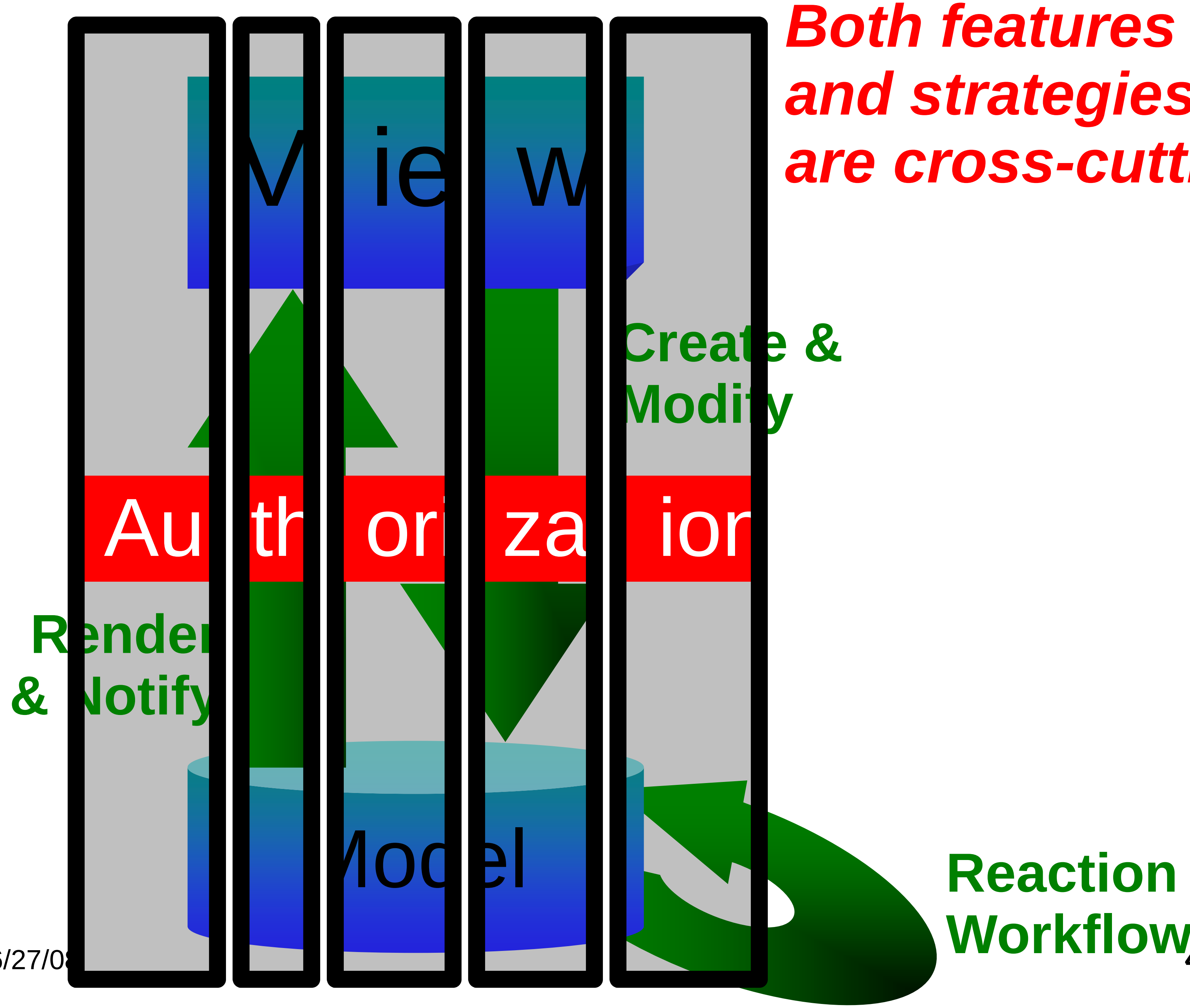
(plugin e)

- Evaluate a piece of code from the model

Aspects and Features

- Aspect = change to an interpreter
  - AspectJ = Modifying residual code!
- Change on multiple levels
  - Models, Interpreters, Metamodels
- Evolution of models
  - Tracking differences, Refactoring, Upgrades
- Feature Modularity
  - Schema is partitioned by “features”
  - General composition operators (Batory)

Both features and strategies are cross-cutting



Pummel Language and System

Interpreters

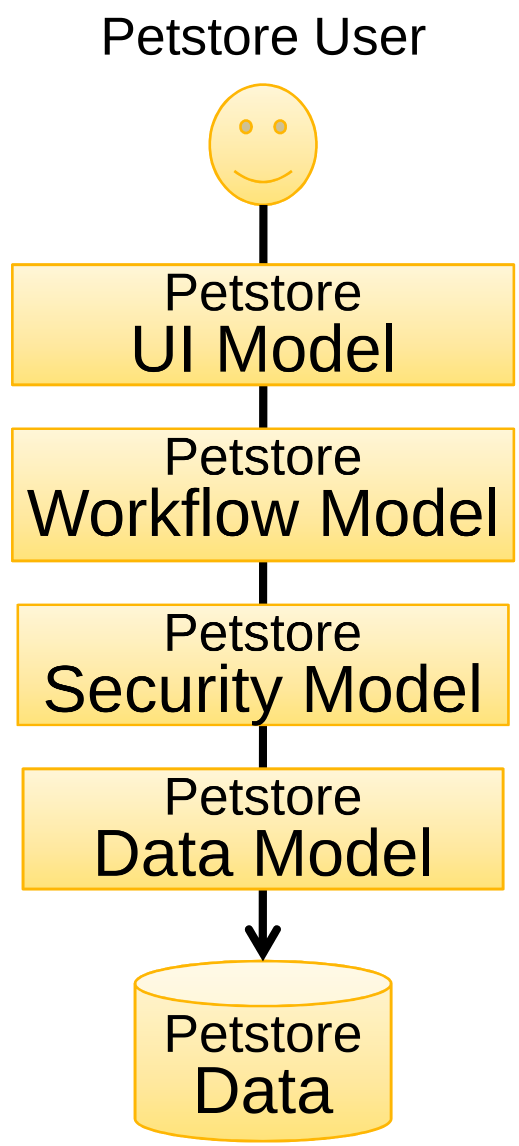
- First-order imperative fragment of Scheme with objects

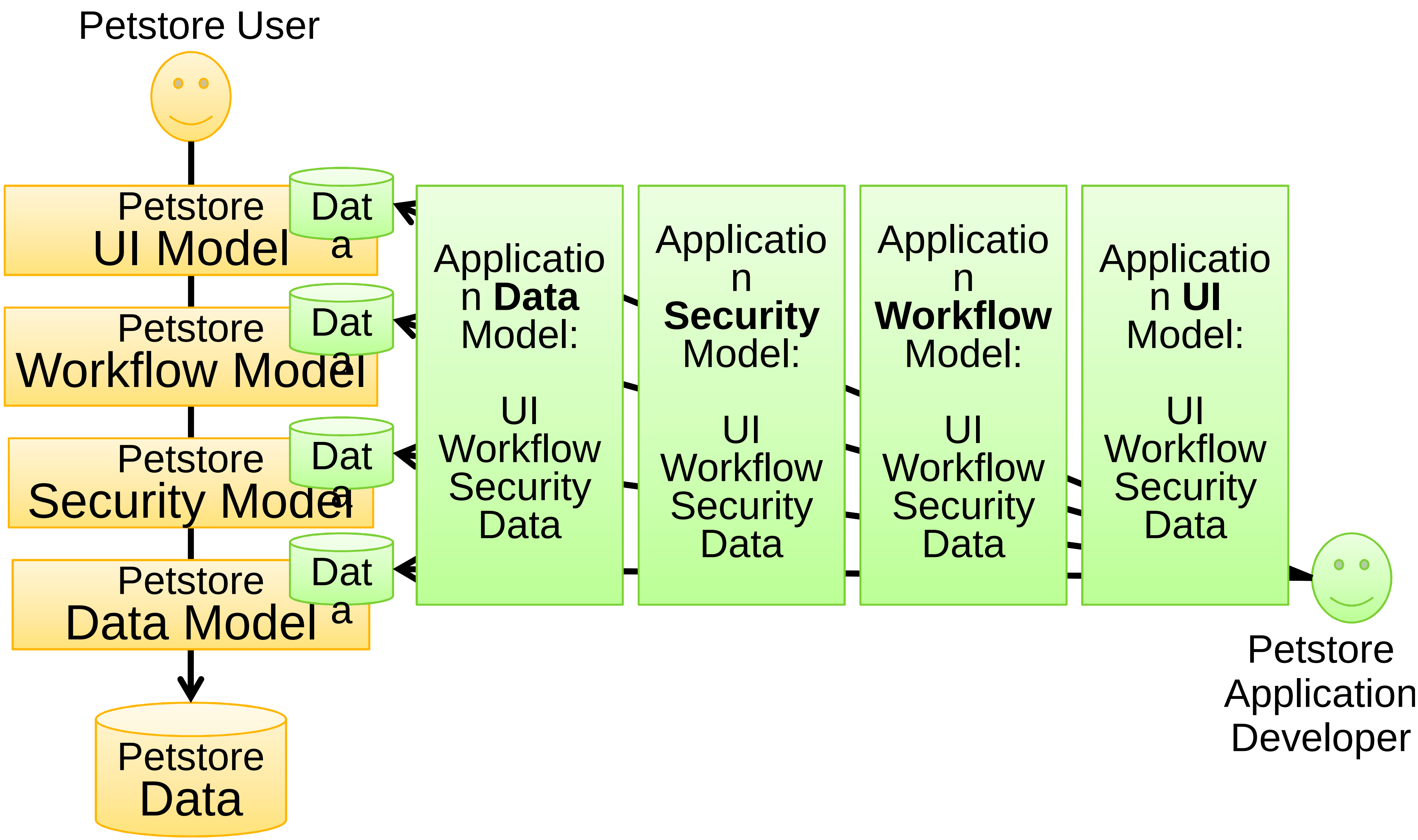
Partial Evaluation “sweet spot?”

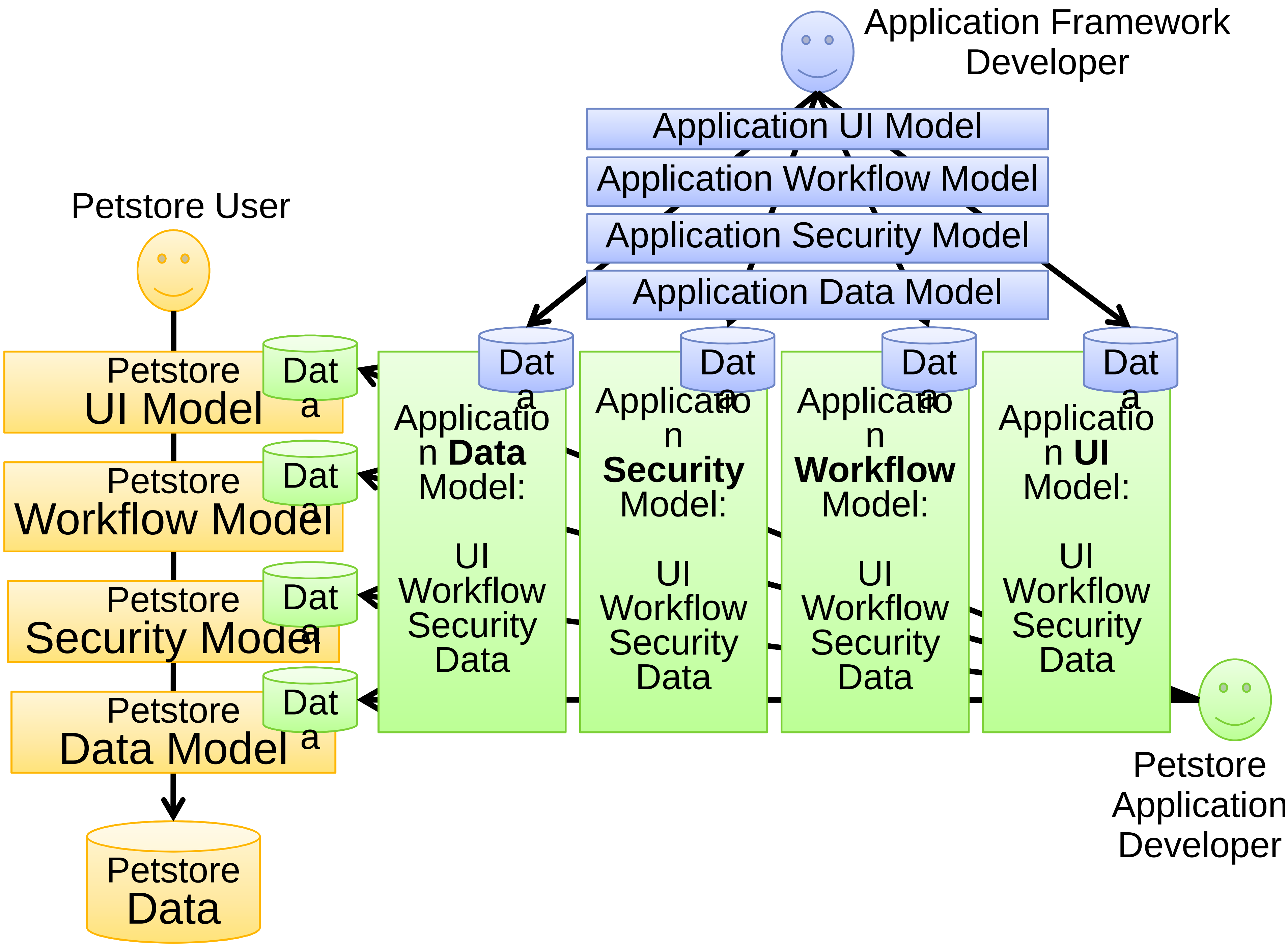
- The models are static, fully eliminated
- Non-Turing complete language

Bootstrapping and self-hosting

- Change fundamental model/interpretation
- Rebuild new version of system


Petstore User
Petstore
UI Model
Petstore
Workflow Model
Petstore
Security Model
Petstore
Data Model
Petstore
Data


Petstore User
Petstore UI Model
Data
Petstore Workflow Model
Data
Petstore Security Model
Data
Petstore Data Model
Data
Petstore Data
Application **Data** Model: UI Workflow Security Data
Application **Security** Model: UI Workflow Security Data
Application **Workflow** Model: UI Workflow Security Data
Application **UI** Model: UI Workflow Security Data
Petstore Application Developer



Application Framework Developer
Application UI Model
Application Workflow Model
Application Security Model
Application Data Model
Petstore User
Petstore UI Model
Petstore Workflow Model
Petstore Security Model
Petstore Data Model
Data
Data
Data
Data
Petstore Data
Data
Data
Data
Data
Application Data Model: UI Workflow Security Data
Application Security Model: UI Workflow Security Data
Application Workflow Model: UI Workflow Security Data
Application UI Model: UI Workflow Security Data
Petstore Application Developer

Conclusion

Strategic Programming

- Interpretation of descriptions
- Aspects as interpreter extensions
- Compilation by partial evaluation + deforestation
- Self-implemented system (Scheme)

Not for *all* kinds of programming

- Not for unique programs

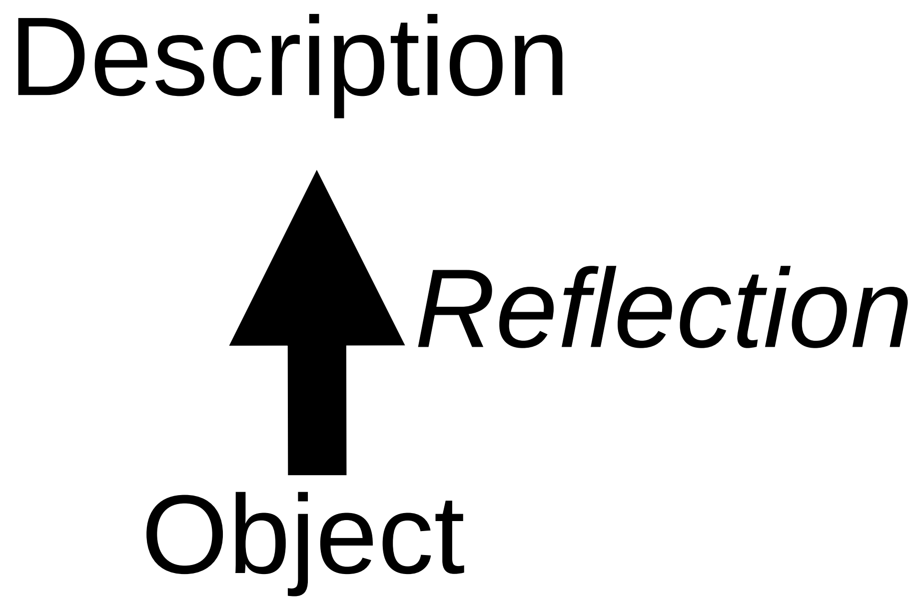
Good when problem is *scale*, not intricacy

51

Related Work/Concepts

Aspects : modify the interpreter
Reflection inversion : description→code
Features : compose models, modify interpreters
Polytypic programming : models generalize types
Template metaprogramming : use partial eval instead
Design patterns : elicit in meta-level interpreter
Debugging : debug the interpreters
Adaptive Programming: meta-strategies?
Types : not sure... untyped meta, typed object ?
Applications : eliminate boilerplate
Evolution : compare and transform, rebuild

52



53

Description

*Interpretation**

Object

**and partial evaluation*

54

Do not Design Your Programs

55

Program Your Designs!

56