Combean/Grooml - a simple modelling language for the Java platform

Thomas Schickinger
combean@sourceforge.net

Optimization for everybody?

- O.R. Technology should be for everybody:
  - Modelling languages provide simple access to optimization technology.
  - An increasing set of problems can be solved without expert knowledge by applying powerful, generic optimization codes.
  - Possible applications can be found everywhere...

- Why has O.R. Technology not yet made it into the 'ordinary' enterprise application developer's toolkit?

The integration challenge

	General purpose O.R. Systems	Enterprise Applications
Modelling gap	Mathematical models	Business object models
Technical gap	(Often) special modelling languages or native language APIs (C/C++/Fortran/...)	Dominated by Java and .Net

- The integration of O.R. Technology into Enterprise Applications ...
  - ... requires expert knowledge and
  - ... only gets some support by expensive high-end, commercial systems.
- → Applications with simple optimization requirements need a simple solution!

Objectives of Grooml

- A mathematical modelling language that is
  - Expressive
  - Simple to learn
  - Easy to integrate with enterprise applications written in Java
- Use cases:
  - Applications where constructing the model is no performance bottleneck
  - Rapid prototyping
  - Teaching

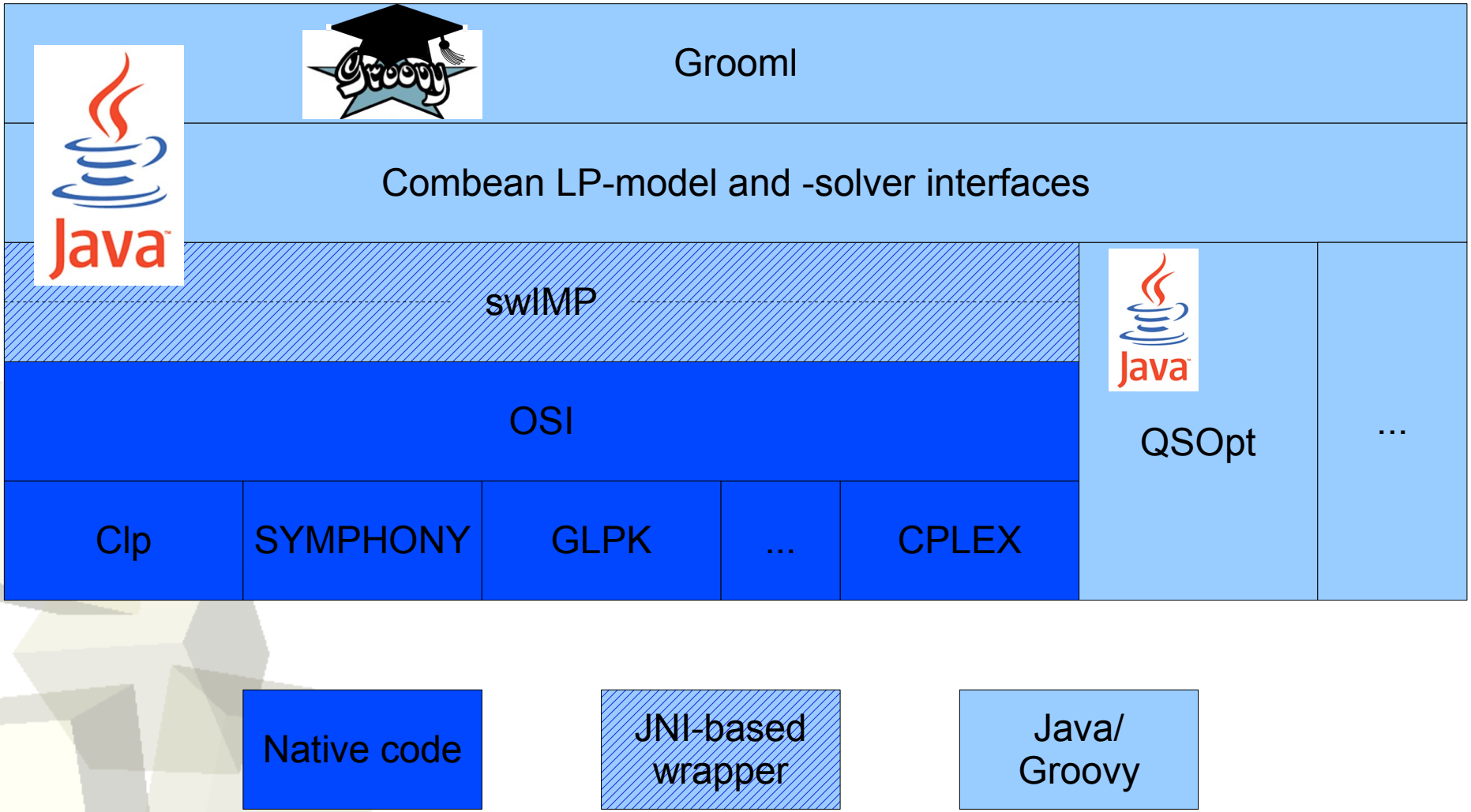
Grooml model = Groovy code

- Grooml models are real Groovy code:
  - Leverage expressiveness of Groovy
  - Language easy to learn for Java/Groovy developers
  - Direct integration of Java or Groovy business objects
  - Access to numerous tools and libraries available in the Java+Groovy world
    (graph drawing, interfaces to office applications, XML import/export, networking, s...)

Why Groovy?

- Dynamic scripting language
  → supports flexible definition of domain-specific languages:
  - 'Pretended' methods and data members
  - Builder pattern
  - Categories
- Full integration with JVM (compiles to Java bytecode)
- Rich built-in data structures
- Simple syntax for closures
- Operator overloading

System architecture



swIMP

- swIMP = SWIG-based Interfaces for Mathematical Programming
  - Bridges gap between native code and Java through auto-generated wrappers.
  - Based on SWIG (Simple Wrapper and Interface Generator)
  - Provides access to OSI-compatible solvers
  - Fast performance through special wrapper templates (~30% overhead in model construction compared to pure C++ code)
  - Platforms: Unix* (Cygwin-port for Windows in preparation)
- More information: swimp.sourceforge.net

Combean

- Combean = combinatorial optimization + JavaBeans
  - Grooml is part of Combean
  - Defines a set of interfaces for standard optimization problems
  - Facilitates integration of optimization codes
  - Focus for this talk: Contains an abstraction layer for IP/LP-models and solvers
  - Supports swIMP + some Java-solvers
  - 100% Java
- More information: combean.sourceforge.net

Open Source modelling languages

- GNU MathProg
- Zimpl
  - Modelling language similar to AMPL
  - No API for integration in applications

- FlopC++
  - Based on C++
  - Language style somewhere between 'real' modelling languages and C++ API

A simple LP model: Knapsack

```
intp = new GroomlInterpreter();
intp.load {
    def  items = ["ring", "money", "diamond", "painting", "statue"]
    def  value =
         [ring:4, money:2, diamond:10, painting:10, statue:20]
    def  weight =
         [ring:1, money:2, diamond:1, painting:5, statue: 20]

    max()
    // Indicator variables for the chosen items
    intvars("x", items) { [value[it], 0..1] }
    // Maximum weight to be carried away
    row("maxweight") {
        sum([i:items]) { weight[i] * x[i] } << 4
    }
    // Choose solver: Coin Cbc
    solver("cbc")
    // Run the MIP solver
    solveMIP()

    print "Solution: ${x()}"
}
```

Grooml interpreter

- interpreter.load { ... } executes the closure { ... } in a special context
  - Redirect method calls to the interpreter object (approach known as 'builder pattern' is Groovy)
  - Maintain 'environment' where variables are bound to specific values
  - Redirect access to variables to the environment (pretended method calls and data elements)
  - Add behavior to some built-in types (mainly: provide additional operators)

Sets

- Set of values
- Typically used to bind the values to a variable

- Based on Groovy built-in types List and Range

- Examples of elementary sets:
  - 1 .. 10
  - [„mon“, „tue“, „wed“, „thu“, „fri“]

Product Sets

- Sets support operator '*'
- Examples:
  - (1..2) * [„a“, „b“] is equivalent to [[1,“a“], [2, „a“], [1, „b“], [2,“b“]]
- Implemented by:
  - Operator overloading
  - Categories (technique which allows to add methods to predefined classes; here: add multiply() to java.util.List)

Variable Bindings

- Variable binding =
  set + variable to which the elements shall be bound
- Used for indexing sums, families of rows in the LP, ...

- Examples:
  - [i : 1..2] bind i to the values 1,2
  - [i : 1..2] * [j : [„a“,„b“]] bind i,j to (1,a),(1,b),(2,a),(2,b)

- Based on built-in type Map
  (key: variable name, value: set definition)

Dynamic sets

- Dynamic sets can be defined by closures
- Definition may refer to variables

- Example:
  - [i : 1..3] * { 1 .. i } is equivalent to
    (1,1), (2,1), (2,2), (3,1), (3,2), (3,3)

- Implemented by:
  - Closures
  - Pretended data members
    (for dynamically defining variable 'i' in the interpreter environment)

Definition of LP-variables

- Command:
  vars(<name>, <binding>) {
  definition with coeff, domain
  }
- Analogous commands: var, intvar, intvars

- Examples:
  - vars(„x“, 1..10) { it % 2 + 1 }
    defines variables x[1], x[2], x[3], x[4] ...
    with coefficients 2, 1, 2, 1, ...
  - intvars(„y“, [„high“, „low“]) { [1, 0..1] }
    defines binary variables y[high] and y[low]

LP-variables and expressions

- After their definition LP-variables are visible through the environment and can be referenced in expressions
- Example: 2*x[1] – y[2, „foo“]
- Implemented by:
  - Pretended data members (of the interpreter)
  - Operator overloading:
    - Operators '+' and '-' for expressions
    - Operator '[]' for indexing of variables

Sums

- Command: sum(<binding>) { <expression> }
- Examples:
  - sum(i : [„alice“, „bob“]) { x[i] }
  - sum(i : 1..10, j : -1..1) { (i+j) * q[i,j+1] }
- Uses: index bindings, expressions of LP-variables
- Implemented by:
  - Pretended data elements (for index binding)
  - Dynamic expressions defined by closure

Definition of LP-rows

- Command: rows(<name>, <binding>) { <row definition> }
- Analogous command: row

- Examples:
  - row(„foo“) {
    sum(i : 1..2) { x[i] } << 10
    }
  - rows(„coverdays“), [d:weekdays]) {
    sum(w:workers) { x[w,d] } | 1
    }
- Note: individual overloading of operators <=, == and >= is not possible (constraint in Groovy).

Solving LPs

- Set direction of optimization: min() or max()
- Run solver: solve()

- Access objective value: solutionValue()
- Access values of variables: by variable name (pretended method call)
  Examples: x(), y(„mon“), z(1,2)

Knapsack revisited

```
intp = new GroomlInterpreter();
intp.load {
    def  items = ["ring", "money", "diamond", "painting", "statue"]
    def  value =
         [ring:4, money:2, diamond:10, painting:10, statue:20]
    def  weight =
         [ring:1, money:2, diamond:1, painting:5, statue: 20]

    max()
    // Indicator variables for the chosen items
    intvars("x", items) { [value[it], 0..1] }
    // Maximum weight to be carried away
    row("maxweight") {
        sum([i:items]) { weight[i] * x[i] } << 4
    }
    // Choose solver: Coin Cbc
    ssolver("cbc")
    // Run the MIP solver
    solveMIP()

    print "Solution: ${x()}"
}
```

Code maturity

- Alpha-stage:
  - Usable for non-critical applications after thorough testing
  - Feedback is highly welcome
  - No experience with big problems and real world applications yet. No dedicated performance tuning done.
- Combean and Grooml are intensively tested through Junit-based automatic test suite:
  - High coverage: 86% lines of code
- Comprehensive documentation:
  - Detailed Javadoc and Groovydoc (www.ohloh.net: „well commented“)
  - Grooml user manual

Getting started

- Install Coin (OSI + solvers) and swIMP:
  - Easiest variant: use package CoinAll 1.0.0 and latest version of swIMP (available at swimp.sourceforge.net)
- Get Grooml at combean.sourceforge.net
- Get documentation
  - User manual and example code
  - These slides are available as well...
- Have fun!