Koalog Constraint Solver: fast constraint solving in Java

Yan Georget

yan.georget@koalog.com

2004/10/25

Summary

- Koalog in one slide,
- Koalog Constraint Solver in one slide,
- “Why a Java library?”,
- genericity of the core engine,
- backtrackable references, variables,
- contraints, problems, constraint scheduling,
- choice points,
- optimization,
- performances,
- software quality,
- current research.

Koalog in one slide

- founded in May 2002, headquartered in Paris,
- business model: software and services,
- software for combinatorial optimization: constraint solver, configurator, journey planner,
- clients: universities, small and large businesses (General Motors),
- looking for partnerships.

Koalog Constraint Solver in one slide

- Java library for constraint solving,
- development started in January 2002,
- current version: 2.2 stable (2.3 alpha),
- 45 000 lines of code (including tests),
- used by a dozen of universities and research centers (DFKI, Uppsala, 4C, KAIST, ...).

Objectives

- Java library for constraint solving,
- fast,
- easy-to-use, extensible,
- good software quality (doc, tests, ...).

Why Java?

- Java is portable (in theory, see support matrix),
- very active community (Apache, JUnit, Trove, ...),
- good industrial choice (used more and more).

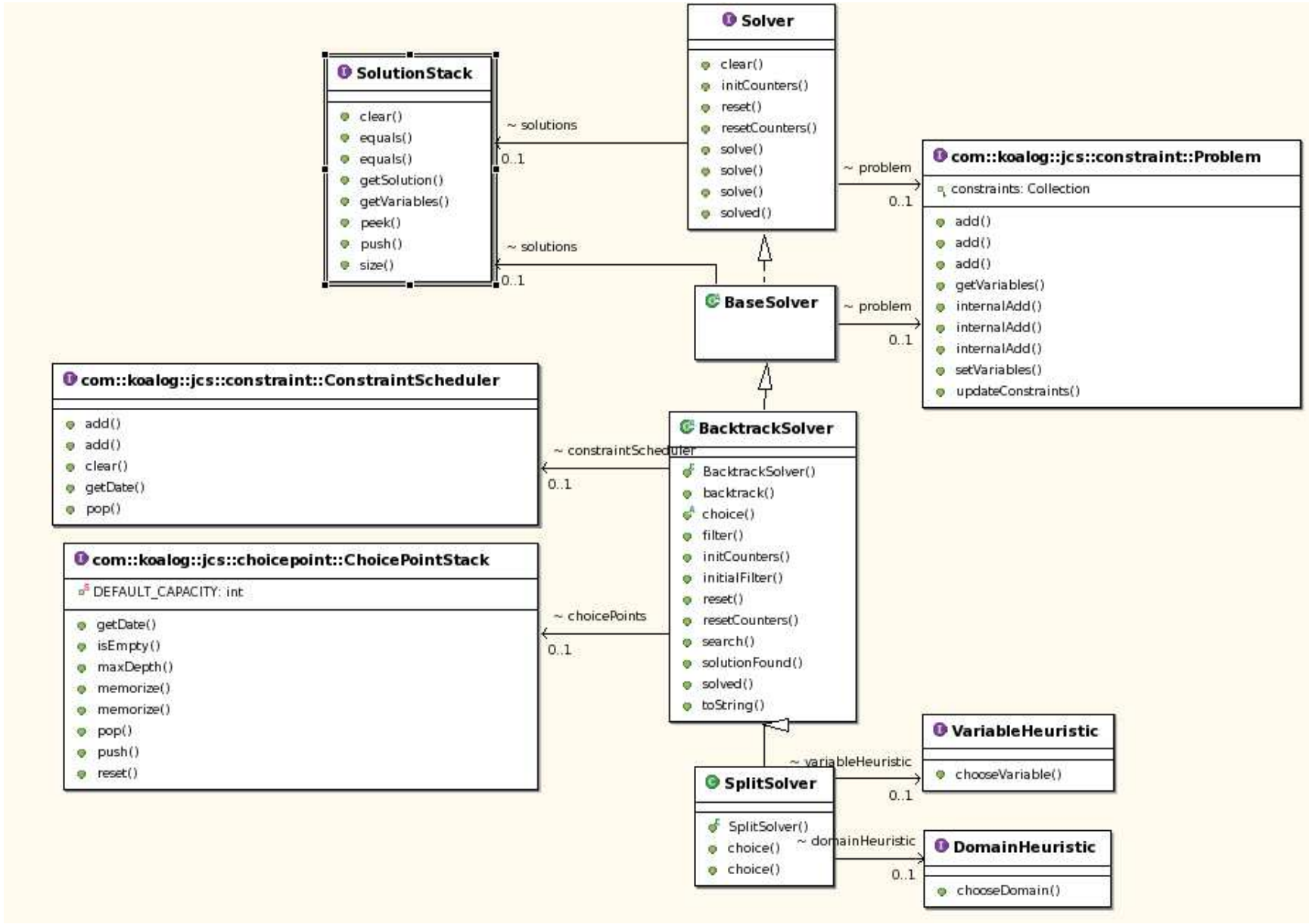
Why a library (and not a language)?

- easier to integrate in existing applications,
- easier to understand for end-users,
- faster to develop.

But a language would be interesting for modelization and control.

Genericity of the core engine

- no hypothesis on the method used for solving (exact or local search),
- no hypothesis on the domains (integer, sets, ...),
- no hypothesis on the underlying data structures.



Genericity of the core engine

3 types of extension are possible (in increasing order of complexity):

- parameters,
- hooks,
- more generally method overrides.

Genericity of the core engine

- set domains: prototype of a "solver on sets" in less than a week,
- local search (Adaptive): prototype of a local search solver in less than a week,
- other possible extensions: solver on floats, soft constraint solving.

Backtrackable references

- generalization of logical variables,
- useful to write global constraints (for example `Cycle`).

Variables

- references on domains,
- syntactic sugar for easier writing of constraints.

Constraints

- encapsulation of a filtering algorithm,
- don't adapt themselves to the domain types,
- in the future: modelization layer?

Problems

- collection of constraints,
- some “relations” are defined as problems.

Less constraint

```
public class Less extends BinaryConstraint {
    private IntegerVariable x, y;

    /**
     * Sole constructor.
     * @param x an integer variable
     * @param y an integer variable
     */
    public Less(IntegerVariable x, IntegerVariable y) {
        super(x, y);
        this.x = x;
        this.y = y;
        name = "x<y";
        idempotent = true;
    }
```

```
    /** @see com.koalog.jcs.constraint.BaseConstraint */
    public void updateConstraints() {
        ((IntegerVariable) variables[0]).addMinConstraint(this);
        ((IntegerVariable) variables[1]).addMaxConstraint(this);
    }

    /** @see com.koalog.jcs.constraint.Constraint */
    public void filter(ChoicePointStack cp,
                       ConstraintScheduler cs)
        throws InconsistencyException {
        x.adjustMax(cp, cs, this, y.getMax()-1);
        y.adjustMin(cp, cs, this, x.getMin()+1);
        if (x.getMax() < y.getMin()) {
            entailed(cp);
        }
    }
}
```

Constraints

- arithmetic constraints,
- boolean constraints,
- set constraints,
- meta-constraints,
- "global constraints": `AllDifferent`, `Cumulative`, `Cycle`, `Disjunctive`, `GCC`, `Inverse`, `Sort`.

Constraints scheduling

- filtering generates events sent to the scheduler,
- sort based on computational complexity,
- optimizations: idempotence, entailment.

Choice point handling

- choice points memorize backtrackable references,
- choice points are handled through a pool.

Optimization

- parameterized by a `Solver` object,
- 2 modes: RESTART and CONTINUE (modification of the search tree).

Performances (easy problems)

Problem	**KCS**	**Choco**	**Cream**
SEND+MORE=MONEY	3	34	
Queens8 (all sol.)	52	528	300
Steiner7	4	72	
MagicSquare4 (all sol.)	2000		11000
Golomb8	81		9000
MT06	233		6000

Performances (hard problems)

- TSP GR17: 3.5s,
- TSP GR21: 1.3s,
- Golomb 14: 58h,
- MT10 (optimality proof): 2mn,
- MT10: 11mn,
- MT20 (optimality proof): 38ms,
- LA01 (optimality proof): 6ms,
- LA31 (optimality proof): 64ms.

Performances

- cost of OOP: it is the price to pay for genericity,
- cost of Java (GC, ...): JVMs are faster and faster,
- our approach works better for large problems with global constraints,
- possible improvements: ‘lazy‘ constraints scheduler, replace “Collections” API by custom data structures well suited for CP.

Software quality

- I18N support,
- logging through Log4J,
- fully tested (JUnit, KCC).

Research

- new global constraints (`LexLeq`, ...),
- soft constraints,
- solver on floats.

Documentation

`http://www.koalog.com/php/jcs.php`:

- tutorial,
- Javadoc,
- examples.