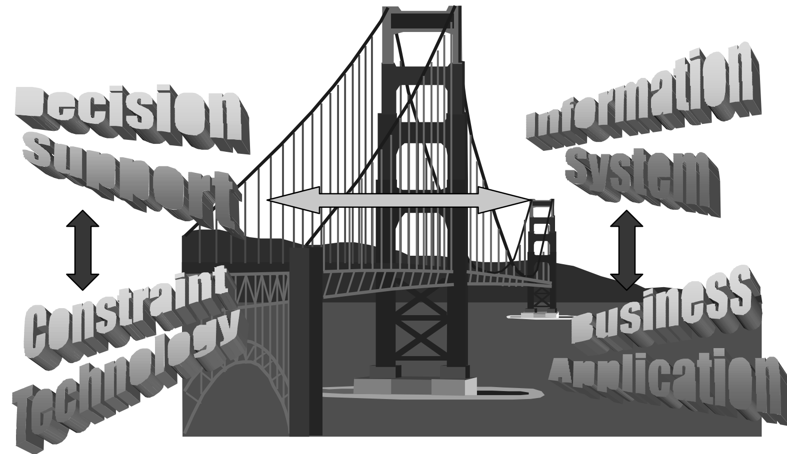
Practical Patterns for Constraint Programming

Jacob Feldman, IntelEngine
Didier Vergamini, ILOG

Overview

- ✓ Constraint-based Decision Support
- ✓ Patterns for Constraint Programming
  - – How to plug in a constraint-based engine
  - – How to define a constrained problem
  - – How to resolve a constraint problem
  - – How a user interacts with a constraint-based system

Constraint-based Decision Support

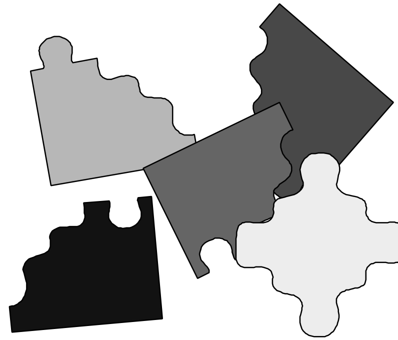


The Challenges of Integrated Decision Support

- ✓Optimization
- ✓Integration
- ✓Interaction

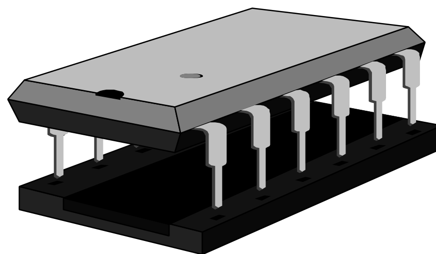
Optimization

- ✓ Intrinsic complexity
- ✓ Moving target
- ✓ End-user support



Integration

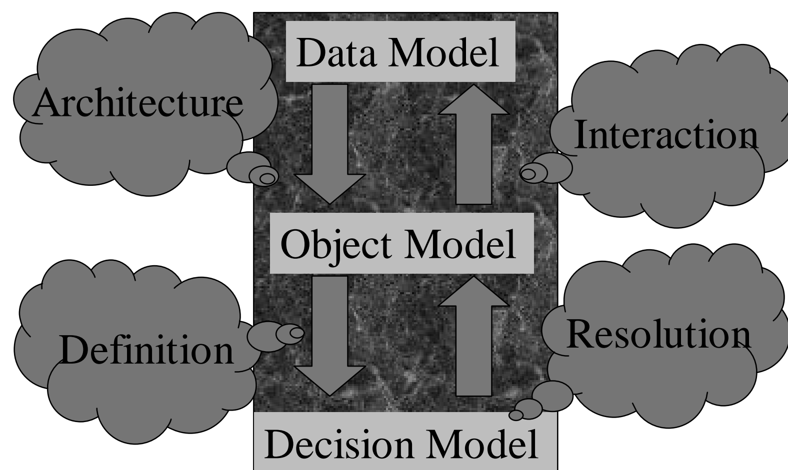
- ✓ Business Application
- ✓ Constraint-based Intelligent Engines
- ✓ Full duplex



Interaction

- ✓ Dynamic costs
- ✓ Trade-off quality versus time
- ✓ Heuristics parameters
- ✓ What-if analysis
- ✓ Self-explanation

Three Classical Layers



Data Model / Decision Model

- ✓ Entities
- ✓ Relations
- ✓ Tables

- ✓ Variables
- ✓ Constraints
- ✓ Objectives
- ✓ Heuristics

Object Model

- ✓ Bridge between the data model and the decision model(s)
- ✓ Interface piloted by the end-user and the data flow

The hottest topic today

- ✓ We already have a technology and supported tools.
- ✓ The hottest topic today is how to use them.

What experts say

- ✓ "A growing number of us feel we have misplaced our collective attention for some time. We no longer need to focus on tools, techniques, notations and even code. We already have in our hands the machinery to build great programs. When we fail we fail because we lack experience."
  Ward Cunningham

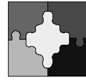
Design Patterns

- ✓ Design Patterns is the newest and most practical approach to give us what we need: the way to share development experience.
- ✓ Pattern is an idea that has been useful in practical context and will probably be useful in others
- ✓ Book "Design Patterns: Elements of Reusable Object-Oriented Software": Gamma-Helm-Johnson-Vlissides, 1995.

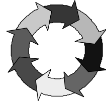
Design Patterns Format

- ✓ Name - a good name is vital
- ✓ Intent - what does it do?
- ✓ Also Known As - other well-known names
- ✓ Motivation - a typical scenario
- ✓ Applicability - when to be applied
- ✓ How Does It Work - participants, structure, implementation, sample code
- ✓ Known Uses - examples found in real systems

Generic Patterns for Constraint Programming



Architectural patterns

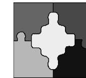


Problem Definition patterns

Problem Resolution patterns

User Involvement patterns

Architectural Patterns



✓ Batch Constraint Satisfaction
- Pattern “Batch Engine”
- Pattern “Pluggable Engine”
- Pattern “Engine Factory”

✓ Interactive Constraint Satisfaction
- Pattern “Interactive Engine”
- Pattern “Consistent Constrained Core”

Applying Constraint Technology to Business Application

- ✓Three layers of a constraint-based system:
  - – Client Application
  - – Constraint-based Intelligent Engine(s)
  - – Interface Between the Application and the Engine
- ✓The architectural patterns present the Engines insights and different ways of plugging them into an Application

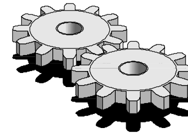
Batch Constraint Satisfaction

Pattern “Batch Engine”

Pattern “Pluggable Engine”

Pattern “Engine Factory”

Pattern “Engine”



✓Intent
- solve a complex constraint satisfaction problem

✓Also Known As
- Solver, Planner, Scheduler

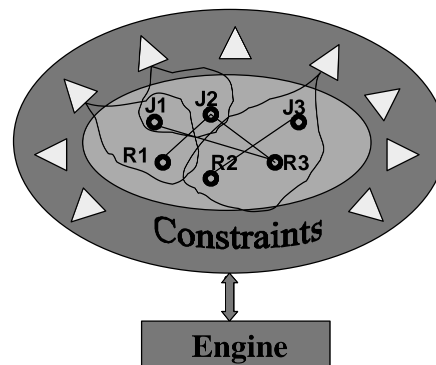
✓Motivation
- Find a “good” solution for job scheduling and resource allocation problem:
  - A set of jobs with unknown starts and set of resources with known capacities.
  - A set of business constraints defined on jobs and resources

Pattern “Engine”: Structure

❺“Intuitive” structure of the constrained environment

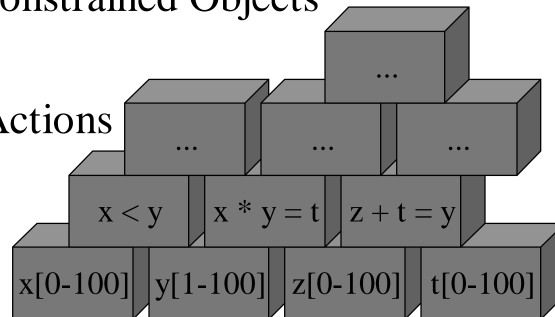
❺Different constraints (demons △) have different views of the constrained objects

❺Engine creates and modifies the state of this environment



Basic Participants for CP Patterns

- ✓ Domains
- ✓ Variables, Constrained Objects
- ✓ Constraints
- ✓ Reversible Actions
- ✓ Goals



Constraints

- ✓ Extension of event-driven paradigm:
  - event handlers for "non-GUI" events
  - events defined on constrained objects: construction, modification, destruction
- ✓ Basic functions
  - Constructor
  - Posting
  - Initial Consistency Check
  - Demons

Constraint sample: ILOG Template (1)

```
class MyConstraintI: public IlcConstraintI
{
public:
   // Constructor
   MyConstraintI(IlcIntVar x, IlcIntVar y);
   // post/propagate
   void post();
   void propagate() { xDomain(); yDomain(); }
   // demons
   void xDomain();
   void yDomain();
private:
…
};
```

Constraint sample: ILOG Template (2)

```
IlcConstraint
MyConstraint(IlcIntVar x, IlcIntVar y)
{
   return
      new (x.getManager().getHeap())
          MyConstraint(x, y);
}
```

Pattern "Batch Engine": Implementation

- ✓ Clear Separation between Problem Definition and Problem Resolution
- ✓ Problem Definition
  - – Constrained Objects
  - – Static Constraints
- ✓ Problem Resolution
  - – Dynamic Constraints
  - – Search Algorithms (Goals)

Typical Problem Definition

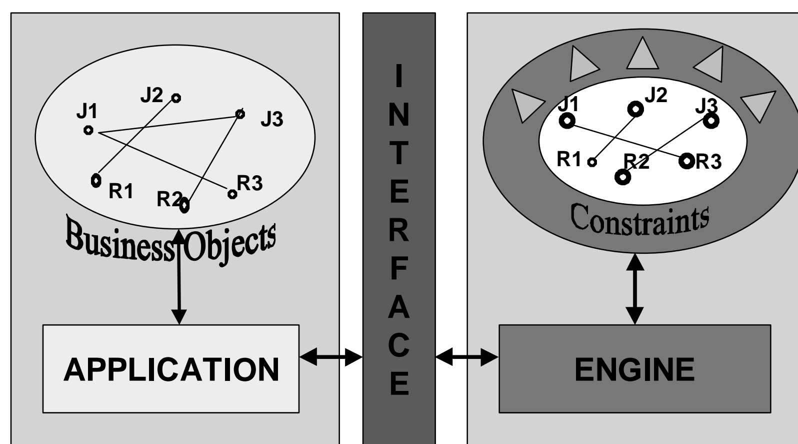
- ✓ Creates a partition of independent sub-problems
- ✓ Defines constrained objects with decision variables
- ✓ Defines static constraints (usually in object constructors)
- ✓ Defines objectives
- ✓ Detects symmetries

Typical Problem Resolution

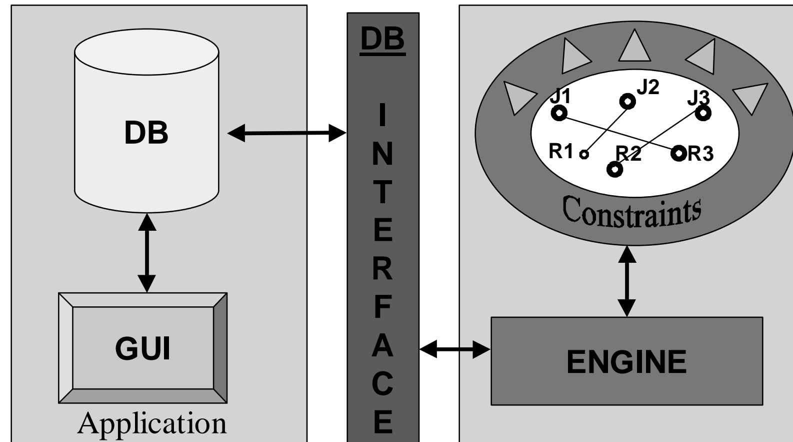
- ✓ Builds Search Goals as building blocks for different resolution strategies
- ✓ Adds additional decision variables
- ✓ Builds dynamic constraints
- ✓ Selects and executes resolution strategy

(See problem resolution patterns)

Pattern "Batch Engine": Integration with Application



Pattern "Batch Engine": Integration Sample

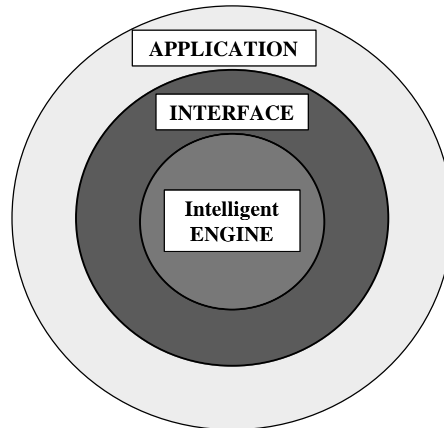


Batch Constraint Satisfaction: Pros and Cons

✓ Pros:
- Simplified development
- Clear demarcation between the engine's developer and actual customer problems (pros?)

✓ Cons:
- Inconsistency (uncontrolled manual overrides)
- Inefficiency (schedule "all")
- Redundant Functionality (for GUI and Engine)
- Difficulties to interpret scheduling results

How to Plug In a Batch Engine

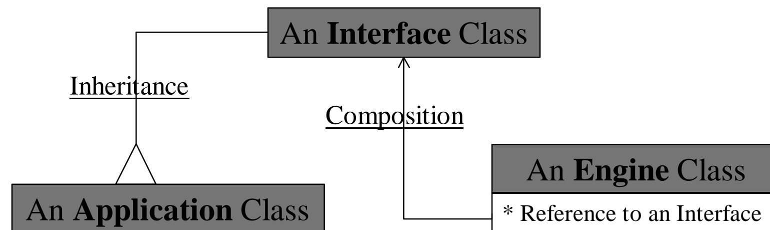


Pattern “Pluggable Engine”

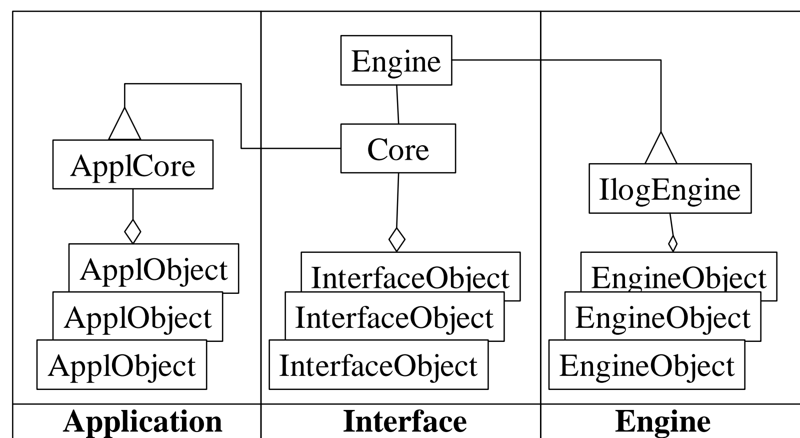
- ✓Intent
  - Plugging the Engine into the existing or a new application
- ✓Also Known As
  - Abstract Interface
- ✓Motivation
  - The engine should be independent of the implementation details of the client application
  - The same engine should be pluggable in different system environments

Pattern “Pluggable Engine” : structure

- ✓ Interface classes: virtual methods
- ✓ Application classes: concrete implementation of interface classes
- ✓ Engine Classes: concrete engine implementation



Pattern “Pluggable Engine”: Participants



Pattern “Pluggable Engine”: Sample of Interface Classes

✓Each interface class specifies mainly virtual accessors & modifiers to application objects (usually no data members). Sample:

```
class Resource {
    public:
    virtual Skill* getSkill() const = 0;
    virtual void assignJob(Job* job) = 0;
    virtual int getSelectionCost() = 0;
    …..
};
```

Pattern “Pluggable Engine” : Sample of Application Classes

✓<u>Concrete Subclasses of the Interface Classes</u>:

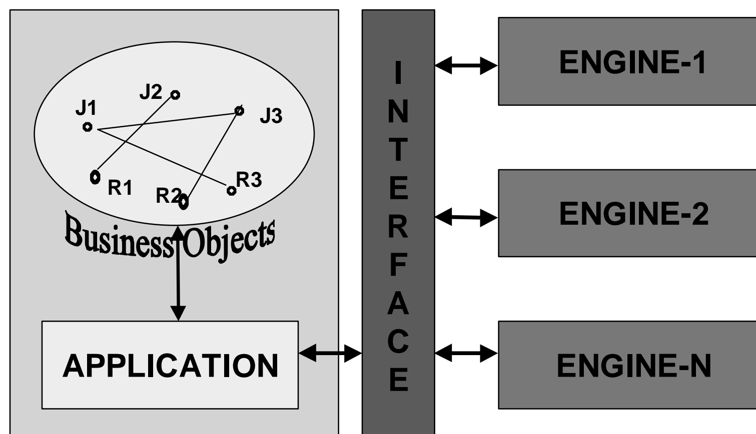
```
class applResource : public Resource {
    public:
    Skill* getSkill() const { return _skill; }
    void assignJob(Job* job);
    private:
    applSkill* _skill;
    …..
};
```

Pattern “Pluggable Engine”: Sample of Engine Classes

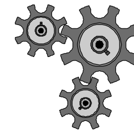
- ✓Engine based on ILOG Scheduler™
- ✓<u>Concrete ILOG-aware Classes</u>

```
class ilogResource  {
      public:
      Resource*  appl() const { return _resource; }
      bool  does(ilogJob* job);   // may fail
      ….
      private:
      Resource*              _resource; // -> interface
      IlcUnaryResource   _unary_resource;
      ….
};
```

Architecture with Multiple Engines

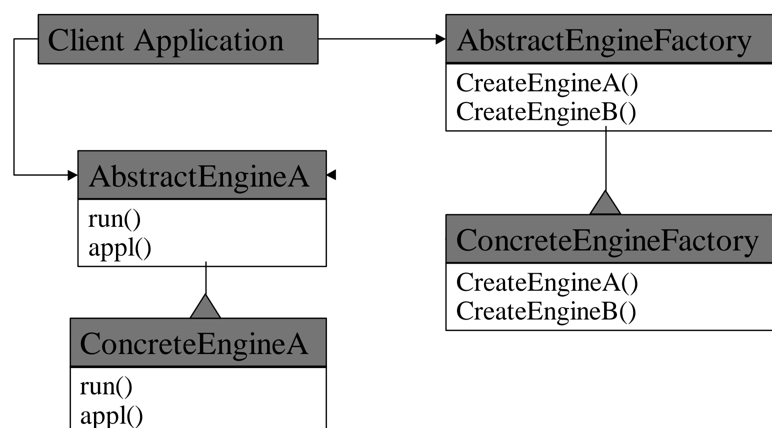


Pattern "Engine Factory"



✓ Intent
  – Provide an interface for creating families of related engines
✓ Also Known As
  – Abstract Factory
✓ Motivation
  – Application can select an engine from the family without specifying its concrete classes

Pattern "Engine Factory": Structure



Interactive Constraint Satisfaction

Pattern “Interactive Engine”

Pattern “Consistent Constrained Core”

Scheduling Reality means Instant Changes

✓ When it comes to managing jobs and resources, ***change*** is the name of the game

✓ Users want to:
  - Make changes quickly and easily
  - Update and fine-tune schedule in a flash, whether they’re altering jobs’ start, duration or adjusting resources.
  - Being warned by the system when they make “impossible” assignments

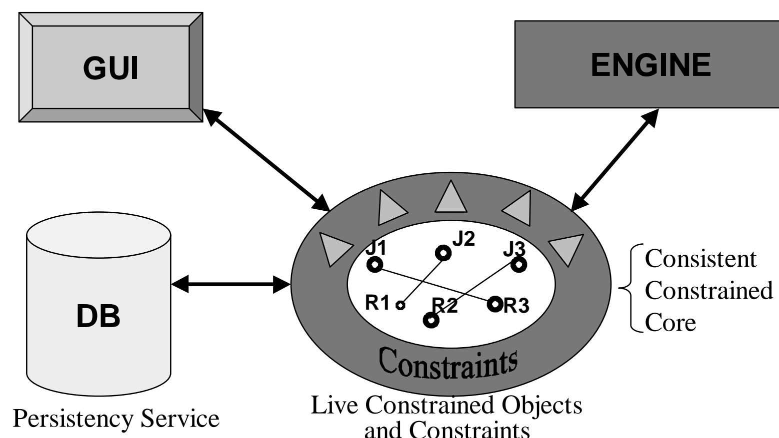
Pattern "Interactive Engine"

✓Intent
- The end user must be able to cooperate with the constraint-based software in developing solutions

✓Also Known As
- Interactive Constraint Solver
- Constraint-Based Graphical Interface

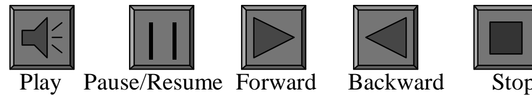
✓Motivation
- Add a user expertise while searching for a better solution of the complex CSP

Pattern "Interactive Engine": A Typical Architecture



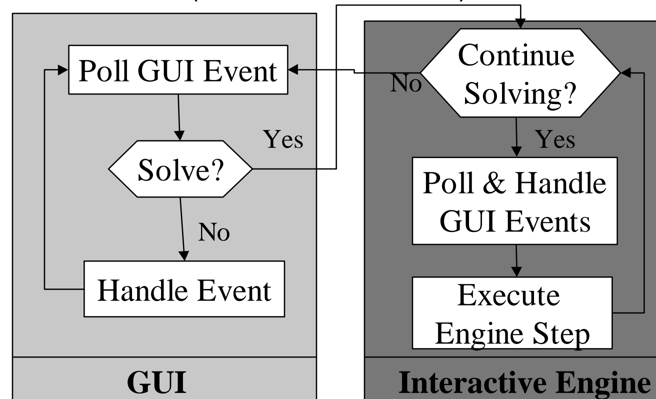
Pattern "Interactive Engine": Motivation

- ✓ Users want to interrupt a Constraint Solver after one or several search steps
- ✓ A Constraint Solver should make its choices explicit to user with an opportunity to reject, accept or modify the choice
- ✓ VCR-like paradigm:



Pattern "Interactive Engine": Sample - Timetabling Engine

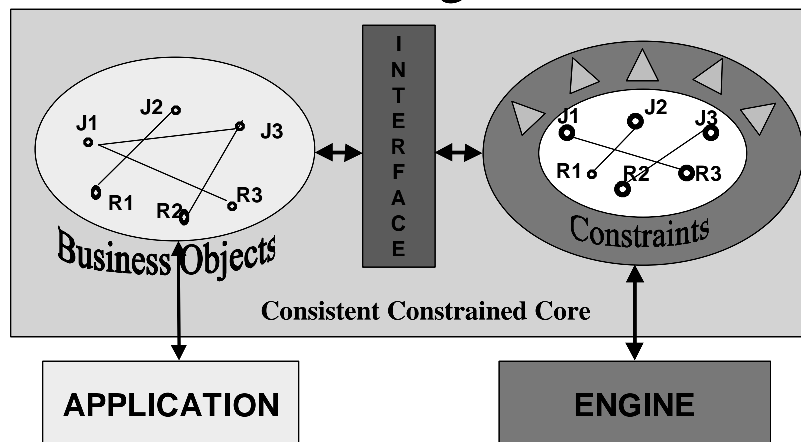
- ✓ Staff Planner (BBL, Alain Dresse)



Pattern "Interactive Engine": Code Sample (from BBL)

```
ILCGOAL1(GoalSolve, InteractiveEngine*,engine) {
  engine->pollAndHandleUserActions();
  IlcGoal next_goal = engine->getUserActionGoal();
  if (!next_goal) next_goal = engine->getNextStep();
  return IlcAnd(next_goal,this); // recursion
}
void InteractiveEngine::solve() {
  IlcSolve(GoalSolve(this));
}
```

How To Plug In An Interactive Engine



Interface as a two-way road

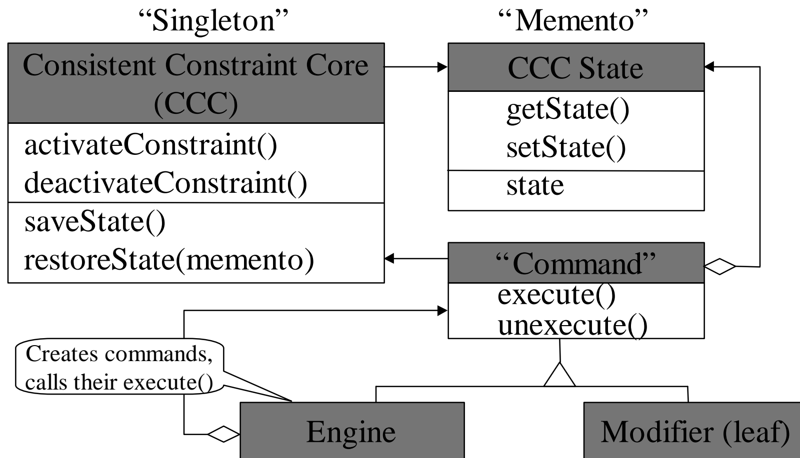
✓ Actions (events) from GUI to CCC
✓ Actions (events) from CCC to GUI

Pattern "Consistent Constrained Core"

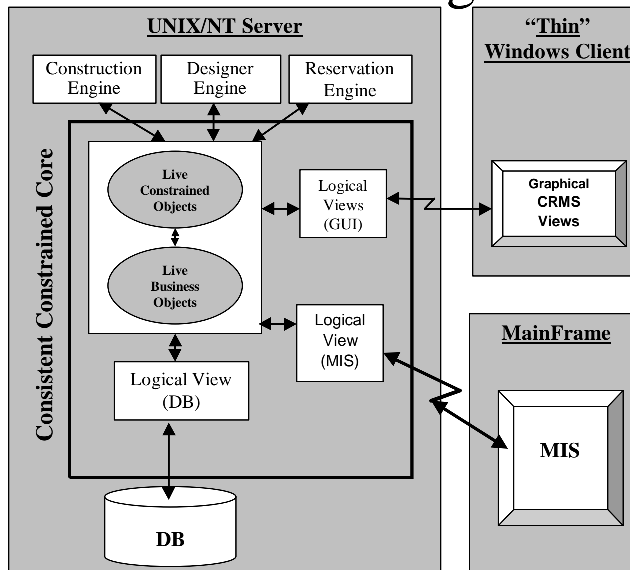
✓ Intent
- Create a constraint-based object-oriented environment to support an interactive constraint satisfaction

✓ Motivation
- Support different user views (GUI) in a consistent state
- Allow a user to add/remove constrained objects
- Allow a user to activate/deactivate constraints
- Warn a user about possible inconsistencies in his/her actions

Consistent Constrained Core: Structure with known patterns



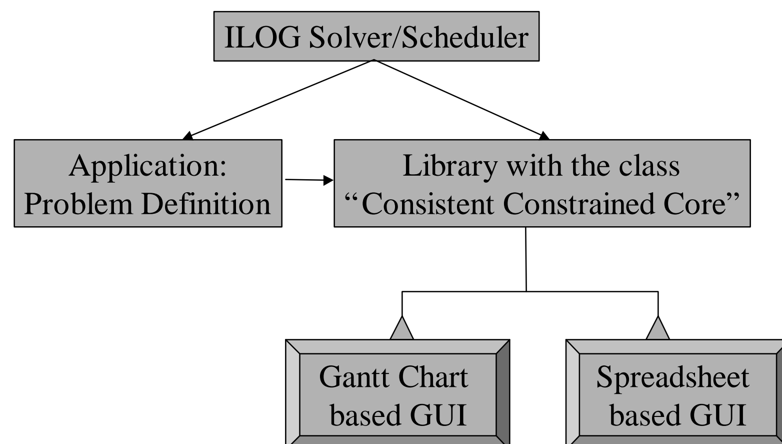
Interactive Scheduling in LILCO



Consistent Constrained Core: Integration Principles

- ✓ Consistent Constrained Core is built on top of a concrete Constraint Solver (for example, ILOG Solver/Scheduler)
- ✓ There are no universal constraint-based GUI, but different GUI's could be inherited from the same Consistent Constrained Core
- ✓ Problem Definition uses the same Constraint Solver

Consistent Constrained Core: Integration Sample



Interactive Constraint Satisfaction: Pros and Cons

- ✓Pros:
  - Tight integration of GUI and Engines
  - Efficiency
  - What-if analysis support
  - Ability of manual scheduling with controlled constraint propagation
  - Simplified interpretation of scheduling results
- ✓Cons:
  - Complex development

DEMO "CONSTRAINER"

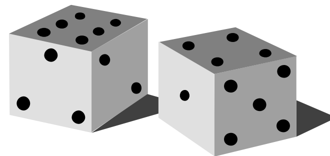
- ✓Demonstrates an implementation of a simple Consistent Constrained Core
  - small arithmetic problems
  - small logical problems
- ✓Uses two C++ libraries
  - ILOG Solver™
  - IntelEngine Constrainer™
- ✓Acknowledgement
  - Authors would like to thank Vince Moshkevich for help in the GUI development

CONSTRAINER: Main Features

- ✓ C++ as a parser for a constraint programming language
- ✓ An end user can activate / deactivate (!) constraints
- ✓ Search goals as constraints
- ✓ Interactive constraint propagation

DEMO: Simple Arithmetic Problem

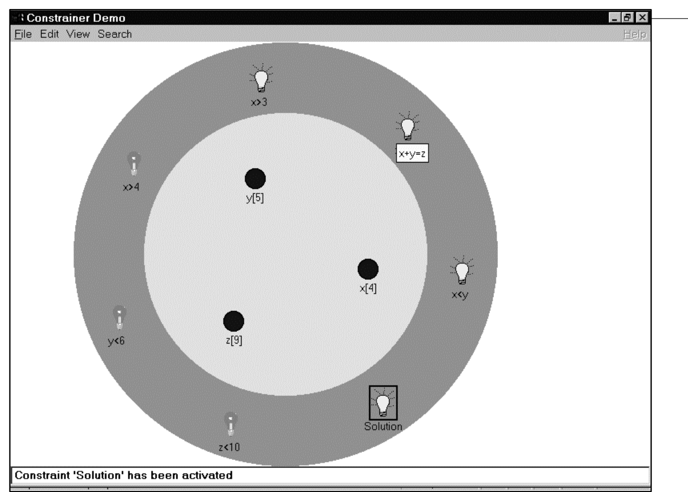
- ✓ Integer variables X, Y, Z defined from 0 to 10
- ✓ Constraints: X<Y and X+Y=Z
- ✓ New constraints may be added/removed later



Sample of the solution code

```
main() {
   IlcInit();
   ViewCore core(“Demo”);
   IlcIntVar x(0,10,"x"), y(0,10,"y"),
   z(0,10,"z");
   core.add(x); core.add(y); core.add(z);
   core.add( "x<y ",  x<y );
   core.add( "x+y=z", x+y==z );
   core.add( "x>3",   x>3 );
   core.add( "x>4",   x>4 );
   core.add( "y<6",   y<6 );
   core.add( "z<10", z<10 );
   core.addDefaultGoals();
   core.mainLoop();
    IlcEnd();
}
```

Live Demo: Arithmetic Problem



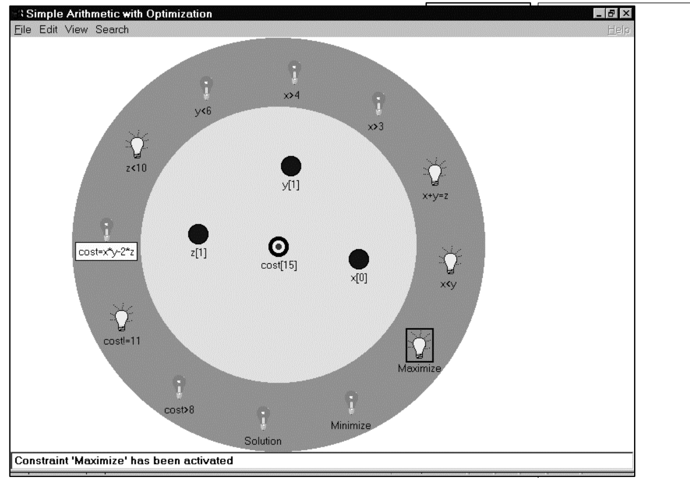
DEMO: Simple Arithmetic Problem with Optimization

- ✓ Add a cost variable defined from 2 to 15
- ✓ Find a solution with a minimal cost
- ✓ Find a solution with a maximal cost
- ✓ Add/Remove different cost constraints

Sample of the solution code

```
…….
IlcIntVar cost(2,15,”cost");
core.addObjective(cost);
core.add(”cost=x*y-2*z",cost==x*y-2*z);
core.add(”cost!=11",cost!=11);
core.add(”cost>8",cost>8);
…..
```

Live Demo: Extended Arithmetic Problem

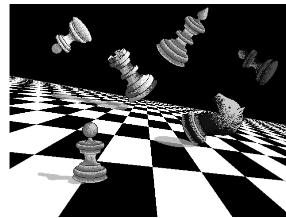


Problem:World Chess Champion?

- ✓ **Kasparov**
- ✓ **Karpov**
- ✓ **Fisher**

DEMO: Logical Problem "Virtual World Chess Match"

- ✓ **Kasparov, Karpov and Fisher played 7 games against each other.**
- ✓ **Kasparov won the most games.**
- ✓ **Karpov lost the least games.**
- ✓ **Fisher became a champion.**
- ✓ **Find a final score.**



Chess tournament: solution (1)

```
// Define mutual Victories, Losses and Draws
IlcIntVar
V12(0,7,"1 won 2"), L12(0,7,"1 lost 2"), D12=7-V12-L12,
V13(0,7,"1 won 3"), L13(0,7,"1 lost 3"), D13=7-V13-L13,
V23(0,7,"2 won 3"), L23(0,7,"2 lost 3"), D23=7-V23-L23;

core.add(V12); core.add(L12);
core.add(V13); core.add(L13);
core.add(V23); core.add(L23);
```

Chess tournament: solution (2)

```
// Define personal Victories, Draws, Losses
IlcIntVar
V1 = V12 + V13,  D1 = D12 + D13,  L1 = L12 + L13,
V2 = L12 + V23,  D2 = D12 + D23,  L2 = V12 + L23,
V3 = L13 + L23,  D3 = D13 + D23,  L3 = V13 + V23;

// The first player won the most games
core.add("1 won most", V1>V2 && V1>V3);
// The second player lost the least games
core.add("2 lost least", L2<L1 && L2<L3);
```

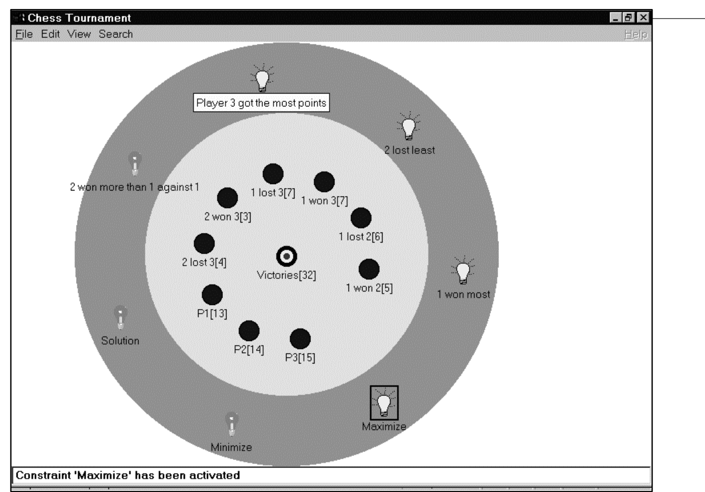
Chess tournament: solution (3)

```
// Define Points
IlcIntVar
P1 = 2*V1 + D1,  P2 = 2*V2 + D2,  P3 = 2*V3 + D3;
core.add(P1); core.add(P2); core.add(P3);

// The third player became a champion
IlcConstraint champion = P3>P1 && P3>P2;
champion.setName("Player 3 got the most points");
core.add("3 is champion", champion );
```

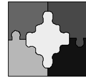
Chess tournament: Minimize/Maximize Victories

```
// Define Total number of Victories
IlcIntVar Victories = V1 + V2 + V3;
Victories.setName("Victories");
core.addObjective(Victories);

// Define default goals: Solution, Minimize, Maximize
core.addDefaultGoals();
core.mainLoop();
```

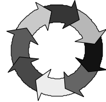
Live Demo: Chess Tournament



Generic Patterns for Constraint Programming



Architectural patterns



Problem Definition patterns

Problem Resolution patterns

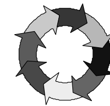
User Involvement patterns

Object Model with Relations

- ✓ Data model
  - Entities
  - Relations
- ✓ Object model
  - Direct navigational model
  - Template classes implementing relations
  - Automated consistency management

Decision Models as Modal Views

- ✓ Different decisions need different models
- ✓ Potential many to many relations between decision concepts (variables, constraints) and objects
- ✓ Navigation in the object model induces navigation in the decision model
- ✓ Results are stored back in the object model



Problem Definition Patterns

- ✓ How to Organize the Problem Definition
- ✓ Pattern "Data Validator"
- ✓ Pattern "Hard Constraints (rules)"
- ✓ Pattern "Frozen Constraints"

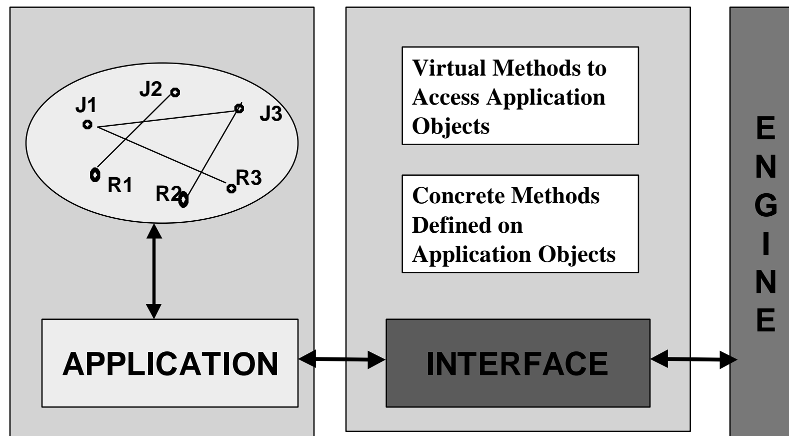
Separation of Problem Definition and Problem Resolution

- ✓ The separation is a classical advantage of CT. The question is "How to separate"?
- ✓ Where to keep knowledge about the separated problem definition?
- ✓ How to distribute the problem definition between 3 major layers:
  - – Application
  - – Interface
  - – Engine?

How to Organize the Problem Definition

- ✓ The Interface via abstract methods defines main Application's objects the Engine operates with.
- ✓ It is natural to ask the Interface to keep the relationships between these objects too.
- ✓ Usually such relationships are defined via concrete (non-virtual) methods inside the Interface

Problem Definition using the Interface



Example: define a machine setup time for different products

✓The Interface has virtual methods *machine(id)* and *product(id)* to get concrete Application's machines and products

✓The Interface has a concrete method *Machine::setupTime(Product p1,Product p2)* that can (and should!) be defined directly in the Interface using other Machine's and Product's virtual methods.

Separation of PD and PR: pros and cons

- ✓ Advantages: well known, but today they are considered as a common place
- ✓ Disadvantages:
  - – "fighting" with constraint propagation
  - – inefficiency
  - – dynamically posted constraints
- ✓ Alternatives/Complements:
  - – constraints as resolution select-methods

Pattern "Data Validator"

- ✓ Intent
  - – Checking data integrity versus different criteria
- ✓ Also Known As
  - – Filter
- ✓ Motivation:
  - – Before performing any kind of optimization, you want to filter at any level all possible "easy" inconsistencies.

Pattern “Data Validator” (2)

✓ Applicability
- Database level data validation
- Interface level data validation
- Engine level data validation
  - Initial constraint propagation
  - Quick search to prove a solution existence

Pattern “Hard Constraint”

✓ Intent
- Express mandatory characteristics of a solution

✓ Also Known as
- Rule

✓ Motivation
- A problem definition usually contains a core of constraints that must be satisfied.
- In some cases, some properties have to be satisfied as consequence of characteristics of the problem.

Pattern “Hard Constraint” (2)

- ✓ Applicability
  - Structural Constraints
  - Relational Constraints
  - Intrinsic Constraints
- ✓ Consequences
  - Search tree pruning: faster solutions
  - Search space very sparse: slower solutions

Pattern “Frozen Constraint”

- ✓ Intent
  - Freeze possibilities for some part of the problem
- ✓ Also Known As
  - Manual Override
- ✓ Motivation
  - An end-user always wants the possibility of “freezing” some part of the problem, and search for a solution, with additional constraints.

Pattern "Frozen Constraint" (2)

✓ Applicability
  - Manual Overrides
  - Consistency Checker
  - Solution Checker

✓ Engine provides 0-100% automation
  - In practical constrained systems, a "good" engine does about 60-80% of job, and a user through "frozen constraints" improves the solution "optimality" and "practicality"

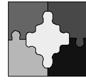
Pattern "Frozen Constraint" (3)

✓ Consequences: Search Goals & Constraints
  - "Batch" View:

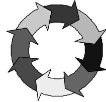
    Constraint is a Goal
  - "Interactive" View:

    Goal is a Constraint
  - Engine as a "heavy" constraint

Generic Patterns for Constraint Programming



Architectural patterns



Problem Definition patterns

Problem Resolution patterns

User Involvement patterns

Problem Resolution Patterns

- ✓Pattern “Hybrid Solvers”
- ✓Pattern “Strategy”
- ✓Pattern “Greed”
- ✓Pattern “Slack”
- ✓Pattern “Prioritization”
- ✓Pattern “ Constraint Relaxation”

Pattern “Hybrid Solvers” (1)

✓Intent

– taking advantage of complete resolution of relaxed versions of a given problem

✓Also Known As

– Cooperative Solvers

– Collaborative Solvers

– Redundant Solvers

– Constraint Programming!!!!

Pattern “Hybrid Solvers” (2)

✓Motivation

– A problem is decomposed into sub-problems that can be efficiently solved independently

– A solution to the global problem has to satisfy all the sub-problems

Pattern “Hybrid Solvers” (3)

✓ Applicability
  - Incremental resolution
  - Trade-off between quality of resolution and global heuristic
  - Heuristic based on optimizing a given sub-problem to find quicker good solutions
  - Introduction of redundant constraints

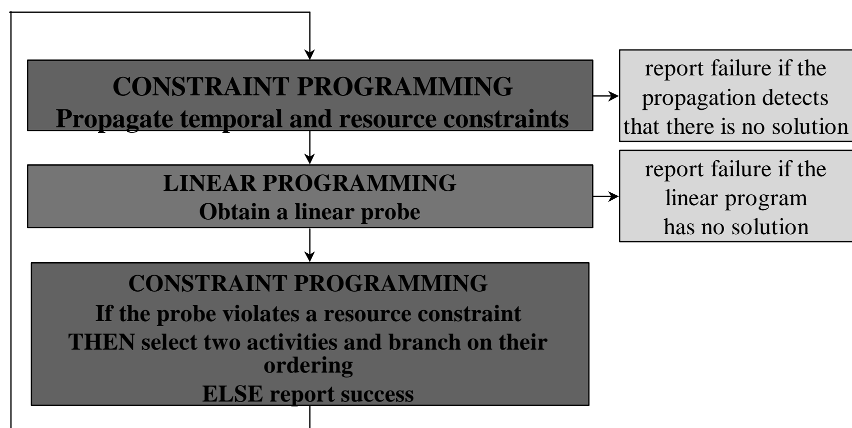
Pattern “Hybrid Solvers” (4)

✓ Known Uses
  - Ilog Solver and related components!
  - Resource allocation: redundant constraint of aggregated resources
  - MILP: heuristic based on the “current solution” of an optimization based on a SIMPLEX algorithm

Integration with linear programming

- ✓ Exploit the power of linear programming through the resolution of a relaxed or modified problem
  - To prune the search space
  - To obtain lower bounds on cost variables
  - To guide the search
- ✓ Use constraint programming
  - To take specific constraints into account
  - To solve the overall problem

Application: reactive scheduling



Confer Claude Le Pape

Pattern “Strategy” (1)

✓Intent
– definition of family of algorithms

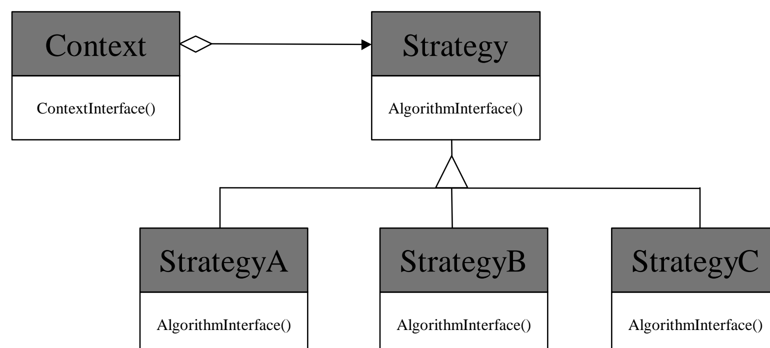
✓Also Known As
– Policy

✓Motivation
– Common encapsulation tackling different situations

✓Confer Gamma’s Strategy

Pattern “Strategy” (2)

✓Structure



Pattern “Strategy” (3)

✓Implementation:

- Search goals as inner classes, generated by virtual member functions
- Selectors as virtual member functions

Pattern “Strategy” (4)

✓Known Uses

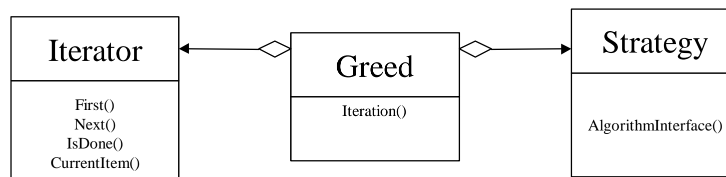
- Generic Enumeration (Generate, Best Generate)
- Exhaustive Column Generation
- Directed Search for Scheduling and Resource Allocation
  - Job Selector
  - Resource Selector

Pattern "Greed" (1)

- Intent
  - Reduction or elimination of backtracking
- Also Known As
  - Myopia
- Motivation
  - Fast solution
  - Local optimization

Pattern "Greed" (2)

- Structure



Pattern “Greed” (3)

✓ Applicability

- Grouping related decision
- Time phased reasoning
- Sequencing

Pattern “Greed” (4)

✓ Consequences

- Non exhaustive search, but can be locally exhaustive
- Beware of greedy failures
- Optimization by local improvements

Pattern “Slack” (1)

- ✓ Intent
  - – Relaxation of an over constrained problem
- ✓ Also Known As
  - – Soft Constraint
- ✓ Motivation
  - – Failing in finding a solution to a given problem is usually not enough; you want to be able to propose solutions that respect “most of it”!

Pattern “Slack” (2)

- ✓ Applicability
  - – Very hard constraint satisfaction problem
  - – Real problems usually contains some flexibility, because they are handled by imperfect creatures

Pattern “Slack” (3)

✓ Known Uses
- Introduction of additional resources
- Relaxation of due dates
- Constraints as Boolean

✓ Consequences
- Very often specific solution search
- Additional optimization heuristics

Pattern “Slack” (4)

✓ Example: Map coloring (Ilog Solver User’s Manual)
- Color the map of Europe with the constraint that neighbors use different colors
- Use no more than 3 colors!!!
  - Relax the constraints on Luxembourg borders
  - Set the objective function on these slacks

Pattern “Prioritization” (1)

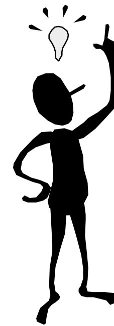
- ✓Intent
  - – Prioritize problem-specific sequence of constrained objects
- ✓Also Known As
  - – Preference
- ✓Motivation
  - – Need to take into account job’s priorities, resource-to-resource preferences, resource-to-job preferences, etc.

Pattern “Prioritization” (2)

- ✓Applicability
  - – Different quality criteria
  - – Partitioning of the problem into different layers
- ✓Known Uses
  - – Priority escalation: the closer the job’s due date, the higher its priority
  - – Greedy use of the priorities
  - – Required Resource Sequencing

Pattern “Prioritization” (3)

✓ Deterministic Search based on a static or a dynamic order
  – Job Selectors
  – Resource Selectors
  – Sorting of Object Containers



Pattern “Constraint Relaxation” (1)

✓ Intent
  – Relax a non-satisfiable constraint

✓ Also Known As
  – Valve

✓ Motivation
  – Instead of optimizing by improving a solution, you want to relax an over-constrained problem until you find a solution

Pattern "Constraint Relaxation" (2)

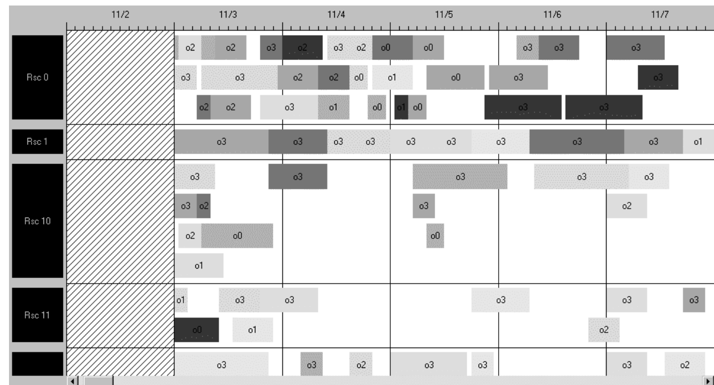
✓ Applicability

- Bounding a Counting constraint
- Bounding due date relaxation
- Adding additional resources

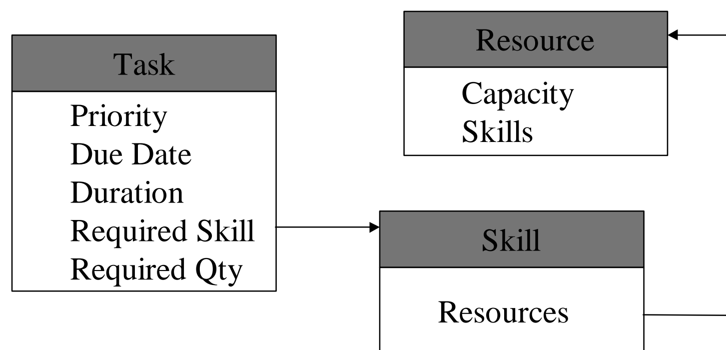
Pattern "Constraint Relaxation": Sample

```
IlcIntVar objective = …;
IlcInt lowerBound = objective.getMin();
IlcInt upperBound = objective.getMax();
IlcInt bound;
for (bound = lowerBound; bound <= upperBound; bound++)
{
   // we add a constraint on the objective
   IlcGoal goal = (objective == bound);
   manager.add(goal);
   if (manager.nextSolution())
      break; // we have found a solution, we stop here
   // we did not find a solution
   // we remove the constraint on the objective
   manager.restart();
   manager.remove(goal);
}
```

Example: Multiple Assignment



Example: Multiple Assignment



Multiple Assignment

- ✓ A task needs a resource to be scheduled
- ✓ A task cannot be scheduled after its due date
- ✓ A task can remain unscheduled
- ✓ Resource capacities are hard constraints

Multiple Assignment: Objectives

- ✓ Minimize the number of unscheduled tasks per priority level
- ✓ Maximize the use of resources
- ✓ Minimize the global “earliness” of the solution

Multiple Assignment: Simple Strategies

✓Greedy search, no backtrack
  - for each task, try all possibilities, and keep the best

✓Greedy by priority level
  - no backtrack between priorities
  - exhaustive search (with limit on the number of backtrack) on each priority

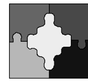
Multiple Assignment: Sample Source Code

✓The source code illustrated some mentioned above patterns is attached

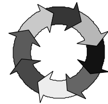
✓The latest version of the source code could be requested from: vergamini@ilog.com

✓You are encouraged to suggest improvements and extensions

Generic Patterns for Constraint Programming



Architectural patterns



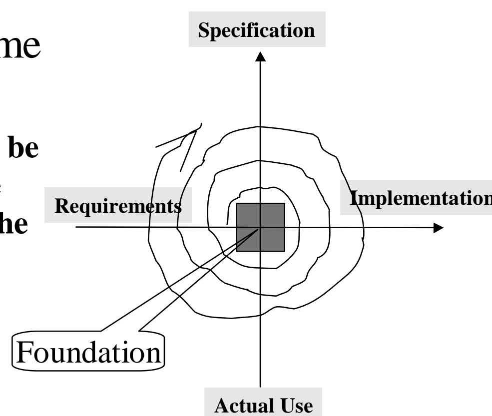
Problem Definition patterns

Problem Resolution patterns

User Involvement patterns

User Involvement Patterns

- ✓ Development-time involvement
  - **An end user should be involved during the entire life-cycle of the constraint-based system**
- ✓ Run-time involvement



User Involvement Patterns

- ✓ Pattern “Configurator”
- ✓ Pattern “Time versus Quality”
- ✓ Pattern “Multiple Objectives”
- ✓ Pattern “Snapshot”
- ✓ Pattern “What if”

Pattern “Configurator” (1)

- ✓ Intent
  - Handling data not specific to the problem but to the way it is going to be solved or optimized
- ✓ Also Known As
  - Tweaker, Fine-Tuner
- ✓ Motivation
  - Need to take into account user specific, and data specific information. As well, the end-user may help choosing the strategy.

Pattern "Configurator" (2)

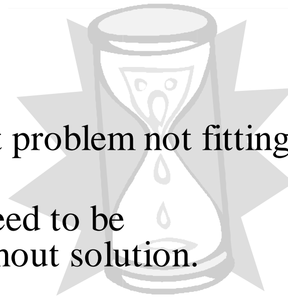
✓ Applicability
- Iterative improvements
- Options of specific constraints
- Alternative search or improvement strategies
- Alternative objective
- Fine tuning
- What-if analysis

Pattern "Configurator" (3)

✓ Configuration parameters definition
- User profile
- Environment variables
- Configuration file (ini-file)
- Run-time parameters
- GUI

Pattern “Time versus Quality” (1)

✓ Intent
  – Setting an artificial limit to stop a search
✓ Also Known As
  – Watchdog
✓ Motivation
  – Need to be protected against problem not fitting the tried heuristic,
  – Often, a problem too hard need to be considered as a problem without solution.



Pattern “Time versus Quality” (2)

✓ Applicability
  – Iterative improvements
  – Optimization by iterative relaxation
✓ Implementation
  – Based on CPU time
  – Based on backtrack count
  – Based of the “tried” percentage
  – Prorated enumeration
  – User interrupt

Pattern “Multiple Objectives”(1)

- ✓Intent
  - Giving the respective importance of different optimization objectives
- ✓Also Known As
  - Weighted Cost
- ✓Motivation
  - When your objective is clearly decomposed into comparable/incomparable sub-objective, you want to give the user an opportunity of tuning what his/her real objective is.

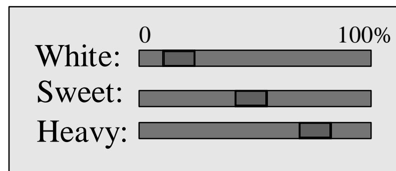
Pattern “Multiple Objectives”(2)

- ✓Applicability
  - Aggregation of the quality of the solution
  - Several objectives without clear prioritization
- ✓Sample Implementations
  - GUI Sliders (input)
  - Pie Chart, Bar Chart (output)

Incomparable Objectives

✓ Applicability
- How to compare "White" and "Sweet"

✓ Objective Weights
- Cost = $\sum_{i}$ w[I] * cost[I]



Pattern "Snapshot" (1)

✓ Intent
- Save all the data necessary to characterize a solution.

✓ Also Known As
- Solution

✓ Motivation
- Need to keep track of one or several solutions to a given problem together with the condition of the search (parameters) to allow improvement heuristics (semi-manual what-if, local improvement techniques)

Pattern “Snapshot” (2)

✓ Applicability
- Elaborate comparison between different solutions
- Heuristics based on perturbation of a solution
- Persistent mechanism
- Model validation on provided valid and invalid solutions

Pattern “What if”

✓ Intent
- Allow the user helping in the solution search

✓ Also Known As
- Driver

✓ Motivation
- Combine the end-user expertise with the computation power
- Limit exhaustive search, add “determinism”
- Take into account preferences not expressed in the model

Pattern “What if”

- ✓ Implementation
  - – change weights and re-run the Engine
  - – set frozen assignments and re-run the Engine
  - – request a “different” solution

A Few Research Directions

- ✓ Dynamic Weighting
- ✓ Over-Constrained Problems
- ✓ Self-Explanatory Engines
- ✓ A User in A Choice Point

Dynamic Weighting

- ✓ Provocative conclusion:
  - – each data set requires a customized search strategy?!
- ✓ Computing problem metrics up front
  - – Data Analyzer
  - – Dynamic selection of scheduling weights
  - – Dynamic selections of scheduling strategies

Over-Constrained Problems

- ✓ "..rather than searching for a solution to a problem, we must, in a sense, search for a problem we can solve"

  Eugene Freuder
- ✓ Partial Constraint Satisfaction
- ✓ A lot of useful patterns could be found in the book "Over-Constrained System", ISBN 3-540-61479-6

Self-Explanatory Engines

✓ Engine Logs
- Commented “tries” and “failures”
- Configurable printing

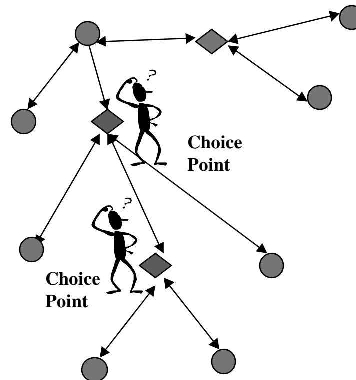
✓ Explanation of failures
- where
- why
- when

Self-Explanatory Engines (2)

✓ Propagation trace

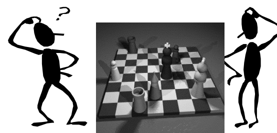
✓ Interactive experiments
- Freeze “everything”
- Deactivate a selected constraint
- Activate a selected constraint

A User At a Choice Point

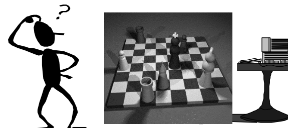


Chess as a model for the Interactive CSP

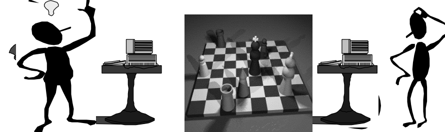
✓Man vs Man



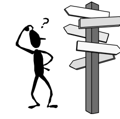
✓Man vs Computer



✓**(Man+Computer) vs (Man+Computer)**



Choice Point Visualization

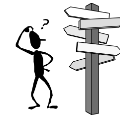


✓ A man with a computer in a choice point

✓ What to present to a human about the current search situation:

  – Search Size Estimate
  – Search Time Estimate
  – Search Complexity Estimate
  – Preferable resulting positions (no knights, at least one rook, a “boring” position, etc.)

Choice Point Visualization



✓ Dynamically configurable strategies

  – “They tuned it to play against me personally” (Kasparov about the “Deep Blue”)

✓ Any chess position (choice point) has a “shape” favorable or unfavorable at this particular moment to this particular player

✓ “I can hear the moves” (V. Nabokov. “The defence”)

CP Pattern Development

- ✓ We just attempted to initiate CP patterns development
- ✓ Horizontal (generic) CP Patterns
- ✓ Vertical (industry-specific) CP Patterns
- ✓ Other Patterns
  - Tabu, GA, Lagrangian Relaxation, and much more

Conclusion

- ✓ Consider not one but a Family of multi-objective constraint-based engines
- ✓ Keep users involved during the entire system life-cycle.
- ✓ Use common CP design patterns
- ✓ Join us in our efforts to build a library of design patterns for CP