Helmut Simonis[1], Paul Davern[1], Jacob Feldman[1],
Deepak Mehta[1], Luis Quesada[1], and Mats Carlsson[2,⋆]

[1] Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
[2] Swedish Institute of Computer Science
SICS AB,Uppsala Science Park, SE-751 83 Uppsala, Sweden
h.simonis@4c.ucc.ie

Abstract. In this paper we describe the design and implementation of CP-V
a generic visualization platform for constraint programming. It provides multi
views to show the search tree, and the state of constraints and variables for a po
mortem analysis of a constraint program. Different to most previous visualizat
tools, it is system independent, using a light-weight, intermediate XML for
to exchange information between solvers and the visualization tools. CP-VI
available under an open-source licence, and has already been interfaced to f
different constraint systems.

1 Introduction

Visualization[1] is one of the best techniques for understanding the behavior of
programs, allowing us to directly observe the impact of changes by visual
instead of using tedious debugging. So far, most constraint visualization tools
closely linked to specific solvers, making it difficult to compare alternative s
to reuse development effort spent on other systems. Previous attempts [4]
tools did not find widespread use largely due to the complexity of the sp
and the level of detail captured. The new, light-weight CP-VIZ system provi
ple XML based interface for solvers, and can be easily extended for new sy
constraints. In CP-VIZ, we try to visualize the search tree and the state of
and (global) constraints in parallel views. The search tree shows choices, as
and failures, modeled on the tree display in the Oz Explorer [14] and later in C
Constraints and variables are shown in a 2D layout defined by the user, individ
constraints are shown in custom visualizations similar to [17]. A new constra
added to the package by simply deriving a new class with a custom drawin

⋆ This work was supported by Science Foundation Ireland (Grant Number 05/IN/
support of Cisco Systems and of the Silicon Valley Community Foundation is gr
knowledged.

[1] Visualization relies heavily on the use of colors, with a potential loss of informat
in black&white only. An on-line version of the paper with colored diagrams ca
loaded from the URL `http://4c.ucc.ie/~hsimonis/cpviz.pdf`. Also
the electronic version you can zoom into all SVG diagrams, revealing additional in

D. Cohen (Ed.): CP 2010, LNCS 6308, pp. 460–474, 2010.

programming, and helping to develop successful applications. Systems like
relied on Prolog-based coroutines to visualize the assignment of variables
throughout a search process. Visualizations were written as application spe
which were co-developed with the constraint model. This approach restricted
components and was tightly linked to a logic-programming host language. N
was the first to abstract visualization types based on collections of variabl
propose different views for them. The visualization of the search tree was
in the Oz Explorer [14], its interactive use tightly linked to the generalized
possibilities of the Oz environment. The DISCiPl project [5] produced a mu
results for constraint debugging and visualization, the ones most relevant for
are the search tree tool for CHIP [16] and the idea of specialized visualizers
constraints [17]. The French OADymPPaC project [4] considered a system in
view of visualization. But the XML-based specification for post-mortem t
quite complex and achieved limited acceptance, and seems no longer to b
maintained. The main design aim for the OADymPPaC trace format was
all possible information about program execution. The visualization tools w
extract those pieces which were of interest to them. While this allowed diffe
to work at different abstraction levels, it also required rather deep integration
supported CP solver to generate the trace, and led to very large trace files
relatively small problems. The visualization tools for Comet [7] provide an en
for developing visualizations for constraint-based local search, which mix ge
application specific aspects of the visualization inside the modeling language

2 Design Aims

The design of CP-VIZ was largely driven by the development of an ECLiPSe
course [15], for which we wanted to be able to show and explain the solving
models for application programmers. We did not want to restrict the use of the
to ECLiPSe only, but rather tried to design a constraint system independent ar
This led to a number of key design decisions:

- We decided to concentrate on post-mortem analysis, which minimizes th
ments of interaction between the constraint solver and the visualization
ment, but still provides most of the information required for analysis.
- The output of the visualization can be studied on-screen, but can also be p
high-quality, colored, vector based print output. Data feeds for other vis
tools are also provided.
- The tools are solver independent, written in a general purpose language
can be easily extended and specialized by deriving new classes from exis
alization classes.
- We added invariant checking at each search node to the functionality, thi
solver independent validation of the results, and can highlight missing pr
in individual constraints.

At the same time, these choices restricted some functionality that is provide
visualization tools [14,16].

– The tool currently is not designed for interactive problem solving, contr search from within the visualization by making manual decisions on how should progress. To allow this in a system independent way seems quite and would require rather deep interaction with each solver. At the same t is limited evidence that such an interactive use helps application progra developing strategies which solve problems automatically.
– We are not considering individual propagation steps, showing in wh constraints are woken and how they make detailed domain restrictions. application programmers, this level of abstraction is too detailed, and time consuming to follow the execution of constraint propagators throug search process.
– We don't collect and display the constraint graph. Many existing tools fo ing constraint graphs [9,8] seem to work only for binary constraints, an rely on graph layout algorithms, without finding usable displays for large sizes.

Which type of design choices can be improved when using visualization? shows the well-known example of a Sudoku puzzle expressed in constraint ming, which compares three different consistency levels for the ALLDIFFER straints in the model. The same problem is modeled using forward checkin consistency and domain consistency; the pictures show the state after the init before search is started. The variables are shown in form of a two-dimens trix, each cell corresponds to a variable, which shows the current values in th (small, in green) or the assigned value (large, in red). For this carefully select tic example, different consistency levels lead to different amounts of propag this is not universally true. In many applications the visualization can help

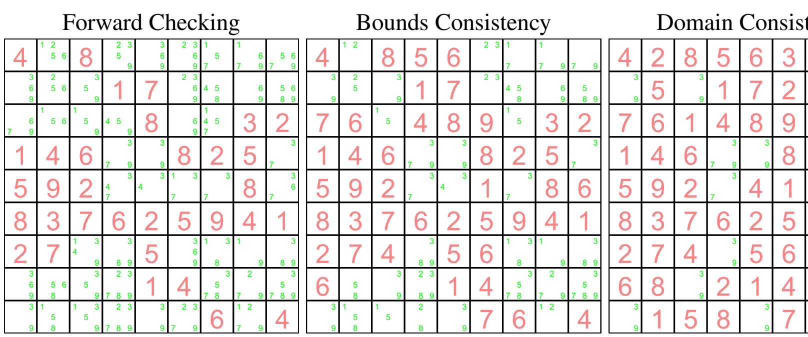

Fig. 1. Sudoku: Consistency Level Comparison

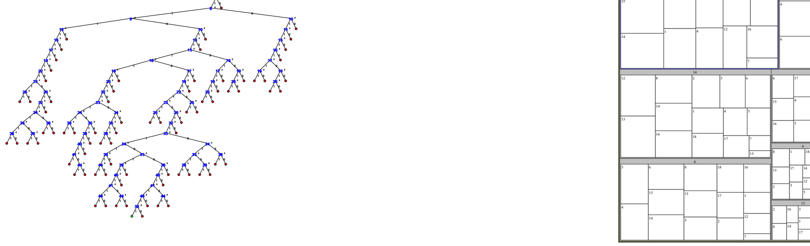

Fig. 2. Search Tree Analysis - Different Views of Search Tree Data

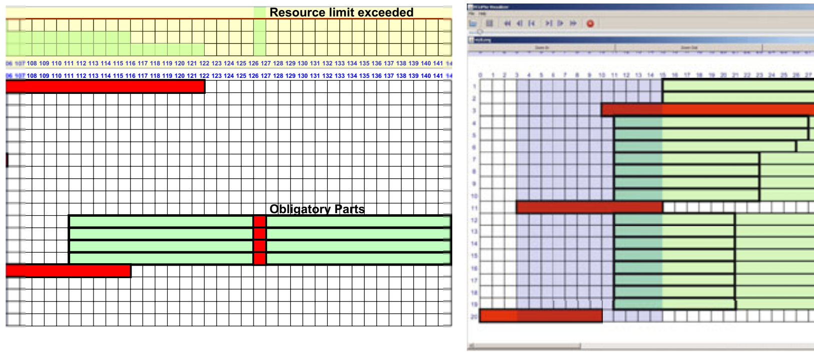

Fig. 3. Invariant Checks for Cumulative Scheduling Problem

which consistency level to use in order to find the right compromise betwe
propagation and problem solving stability.

Figure 2 shows different diagrams for visualization of the search tree. If
space is small, the full tree can be shown (on the left). For more complex
this is no longer possible, and a more compact form, originally proposed in [1
abstracts failed sub-trees, can be displayed (see Figure 7 for an example). But
detailed analysis is not required, it suffices to have a simple quantitative an
shown in the middle part of Figure 2. It plots the number of success and fail
with the depth of the search tree. The shape of the plot often is enough to u
how well a model is able to explore the search space. On the right we show
visualization which indicates the size of the generated subtree below a top-le
in the search. This can help to understand more clearly if the search strategy
making the right choices.

Finally, the diagrams in Figure 3 show an example where invariant chec
used to detect nodes in the search where the constraint propagation was not
The pictures are from a cumulative scheduling problem proposed b

atory parts (dark, in red) are generated. The sum of these obligatory parts ex
resource limit, which is not detected by the propagator. Invariant checking
the constraint and has also marked the problem in the search tree. On the rig
ber of tasks have been assigned, and their resource profile reaches the resou
but the start times of unassigned tasks are not updated properly. This not o
some missing propagation, but affects the search routine as well, as the he
task selection will pick the wrong tasks to be assigned next. The problem wa
by developing another propagator for CUMULATIVE based on obligatory pa

3 Architecture

Figure 4 shows the basic architecture of the CP-VIZ system. The visualizatio
by annotations in the constraint program. When run in the solver, two XM
(one for the search tree, the other for the constraint and variable visualizatio
duced. These files are then parsed in the main CP-VIZ application, producing
output as SVG, or as input for other tools (tree maps, graphs, statistics). The
put can be displayed interactively in the CP-VIZTOOL, or can be used in mul
to produce annotated or converted output for print or WEB media.

We use XML text files to link the generation of the log files to the creat
visualization. This should allow almost any constraint programming system to
to the CP-VIZ visualization with minimal effort.

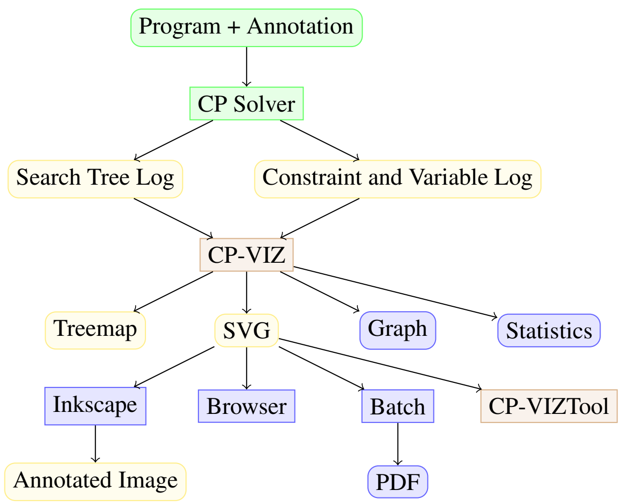

Fig. 4. CP-VIZ System Architecture

choices. In each node we have a node id, the node id of the parent node, the na
variable currently assigned, the size of its domain, and the value assigned. A
these types also allows to handle arbitrary choices, not based on variable as
These alternatives can be useful to describe more complex branching schemes
analysis is slightly more restricted. A *solution* node is used to mark choice no
complete an assignment, i.e. to mark nodes where all constraints are satisfied
mat does not assume chronological depth first search, nodes can be added for a
at any time.

Constraint and Variable Log. The second log file is used to describe snapsho
straints and variables. Its top element is *visualization*, which contains a list of
elements, describing the constraints and variables to be displayed. This is foll
sequence of *state* elements, each containing a snapshot of the execution at a g
point. Inside each state, the *visualizer_state* elements describe the current state
straint or collection of variables. The syntax used roughly follows the syntax u
global constraint catalog [3]. Constraints can be described by their named *a*
which may contain *collections* of basic types or *tuples*, which describe structu
parate types. The basic types currently allowed are integers and finite domain
integer sets and domain variables over finite sets, plus some more specialized

3.1 System Dependent XML Generators

For every constraint system that wishes to use the CP-VIZ environment, w
define an interface to generate the XML logs. Figure 5 shows such an interfac
based on two classes, *VisualSolver* and *VisualProblem*. The methods for the s
log are contained in the *VisualSolver* interface, each adds or annotates a searc
the tree.

The methods for the *VisualProblem* class are split into two groups. The a
programmer can use the method *register()* to register a constraint or a col
variables with the visualization. There is also a method *snapshot()* which tr
creation of a snapshot of all registered constraints and variables at a given
point. The snapshot is created by sending a *snapshot()* message to each regis
straint. This is then responsible for saving the current state of the constrain
log. For this it might use the remaining methods of the *VisualProblem* class,
XML elements of different types for the constraint.

3.2 CP-VIZ

The main CP-VIZ application parses the XML log files and creates SVG
the user. The search tree is parsed completely before generation, while the
and variable snapshots are handled one at a time. In order to see which cha
occurred to the variables by the current search step, the tool keeps a stack of
for all parents of the current node in memory. This not only allows to see th

```
    public void addSuccessNode(int id, int parentId,
                String variableName,int size,int value);
    public void addSuccessNode(int id, int parentId,
                String variableName, int size, String choi
    public void addFailureNode(int id, int parentId,
                String variableName,int size, int value);
    public void addFailureNode(int id, int parentId,
                String variableName, int size, String choi
    public void labelSolutionNode(int id);
}

public interface VisualProblem extends Visual {
    public void register(Constraint constraint);
    public void register(Var var);
    public void register(Var[] varArray);
    public void register(Var[][] varMatrix);
    public void snapshot();

    // for implementors only
    public void startTagArgument(String index);
    public void startTagArgument(int index);
    public void endTagArgument();

    public void startTagCollection(String index);
    public void startTagCollection(int index);
    public void endTagCollection();

    public void startTagTuple(String index);
    public void startTagTuple(int index);
    public void endTagTuple();

    void tagVariable(Var var);
    void tagVariable(String index, Var var);
    void tagVariable(int index, Var var);

    void tagInteger(String index,int value);
    void tagInteger(int index,int value);
}
```

Fig. 5. VisualSolver and VisualProblem Interface Definition

updates of the variables, but also permits to generate path based visualizat
which display the evolution of a variable or some parameter through all par
from the root to the current node.

3.3 CP-VIZ Tool

Figure 6 shows the CP-VIZTOOL, a Java application which displays the res
visualization on the screen. The application has a time-line at the top, wher
can select a state of the execution for display. The tool will then display th
the search tree in the left main pane, and the corresponding snapshot of the
and variable visualization in the right pane. The user can also step forward/b
through the execution, or display the complete solution process as a movie, pr
automatically through the different snapshots.

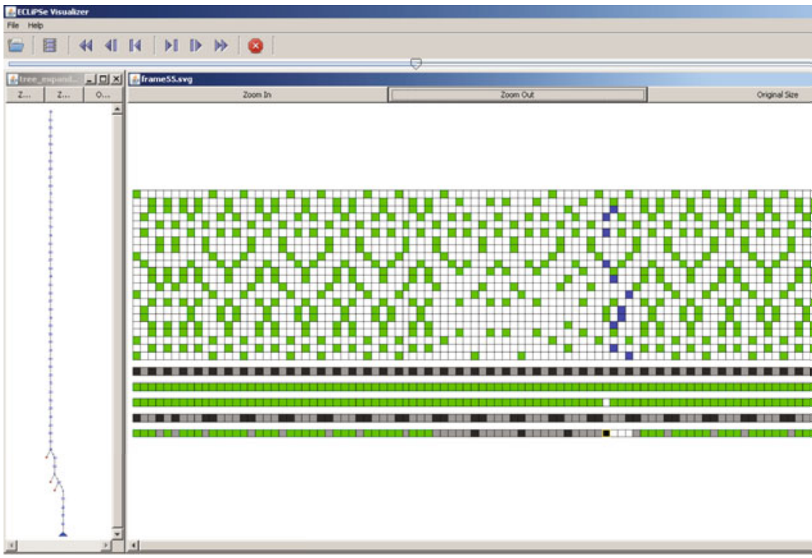

Fig. 6. Interactive CP-VIZ Tool for Car Sequencing Problem

4 Invariant Checking

By providing snapshots of the execution at fix points only, when all constr
performed their consistency checking, CP-VIZ also provides data for system
ing of execution traces. We have implemented an invariant checker, whic
ery snapshot calls an *invariant()* method for each registered constraint. Th
may return *TRUE*, also the default value, or one of the values *INTERESTIN*
ING_PROPAGATION, *INCONSISTENT* or *FALSE*. Combining all invariant c
a snapshot, the visualizer then marks the node in the search tree accordingly
lights any failed assertions in the constraint visualization. We explain the m

within the scheduling period p. A ground solution must satisfy the equations

$$\forall\, 0 \le t < p: \quad \sum_{\{i \mid s_i \le t < s_i + d_i\}} r_i \le l$$

$$\forall\, 1 \le i \le n: \quad s_i + d_i \le p$$

$$\sum_{1 \le i \le n} d_i * r_i \le l * p$$

Inequality (3) is implied by the others, but is used as it provides a good ba veloping invariants. If for a ground instance one of these equations is not sati the invariant checker will return *FALSE*.

We can rewrite constraint (3) to consider upper bounds on domain variabl This produces

$$\overline{p} \ge \left\lceil \frac{\sum d_i * r_i}{\overline{l}} \right\rceil$$

If in any snapshot this invariant does not hold, then the snapshot is incons the constraint propagator should have failed for this node. The invariant check *INCONSISTENT*. A weaker invariant checks the lower bound of p instead:

$$\underline{p} \ge \left\lceil \frac{\sum d_i * r_i}{\overline{l}} \right\rceil$$

If this invariant is violated, the lower bound of p has not been updated cor other values in the domain of p might satisfy the condition, so the invaria returns *MISSING_PROPAGATION*. In a similar way we can derive

$$\forall\, 0 \le t < p: \quad \overline{l} < \sum_{\{i \mid \overline{s_i} \le t < \underline{s_i} + d_i\}} r_i \Rightarrow \textit{INCONSISTENT}$$

$$\forall\, 0 \le t < p: \quad \underline{l} < \sum_{\{i \mid \overline{s_i} \le t < \underline{s_i} + d_i\}} r_i \Rightarrow \textit{MISSING\_PROPAGATIO}$$

We are cumulating the resource use over all obligatory parts, i.e. time periods know that a task will be active.

The weakest value is *INTERESTING*, which can be used to mark snapsh a constraint detects a special condition that the user is interested in. We wi use in section 5.3 to mark nodes where no propagation of a global method sible. One key advantage of an invariant checker inside a system independ that the invariant code can be shared for all constraint systems that impleme constraint. As it is written independently from any specific propagation m avoids problems when reused, buggy code in the validation precludes detec error. Finally, the invariant checks enhance chances to detect subtle differe declarative meaning of a global constraint between systems.

CP-VIZ tool set, and note some details of the effort required to connect a ne
to the visualizer.

5.1 ECLiPSe

We have linked the finite domain *ic* library of the Prolog based ECLiPSe syst
VIZ as part of the ECLiPSe ELearning course development. The main require
to display sufficient information of the constraint propagation to the student
overwhelming them with unwanted detail. As custom search routines are ve
write in ECLiPSe, we also needed an interface which could visualize such rou
minimal overhead. The current interface does not require hooks in the predefi
routines or constraint implementations, but rather uses logic programming f
express the visualization as annotations of the user programs. Visualizers fo
global constraints have been implemented so far, together with a series o
programs, based on the course material.

5.2 SICStus

We are currently extending the `clpfd` library module of SICStus Prolog with
predicates producing XML files for CP-VIZ. The work is being carried out a
the interface code for ECLiPSe. Like for ECLiPSe, we do not use any specia
SICStus Prolog or its `clpfd` library module and we rely on user program an
The implementation replaces the normal `labeling/2` procedure, which t
of domain variables, by another procedure taking a list of domain variables
with information for display purposes, e.g. variable name. ECLiPSe and SIC
vide different libraries of global constraints, and so the main implementation
in implementing visualizers for the global constraints not provided by ECLiP
ticular, we plan to implement 2D and 3D visualizers for the generic multi-di
geost constraint [2]. The XML files are currently being written with standard
predicates. For efficiency, this is likely to be replaced later by specific XML

5.3 Visualization of the Global Constraint SOFTPREC

As a case study for visualizing individual global constraints, consider the vis
of the SOFTPREC constraint arising in the context of the feature subscriptio
for telecommunication services. A feature subscription problem is a configura
lem defined by a set of possible features, a set of hard precedence constraint
soft precedence constraints, and a function that maps each feature and each s
dence constraint to a non-zero integer weight. The objective is to maximize
of the subscription, which is defined to be the sum of the weights of the fe
soft precedences that are included. The soft global precedence constraint S
is proposed for solving the feature subscription problem in [11] and [10].
and only if there is a strict partial order on the selected features subject to th

(http://www.emn.fr/z-info/choco-solver), which is a Java l
constraint programming. In order to visualize the search tree and the propagati
out at each node of the search tree, the implementation of SOFTPREC was ex
order to generate and save the trace in the CP-VIZ format. The implementat
extension was fairly simple. We had to extend two classes of Choco and over
of their methods. While generating the required data for visualization is q
deciding what to visualize and how to visualize it required several iterations.

A distinct advantage of the visualizer is that it is easy to get a sense of the
Visualizing solutions can give more insight than just knowing the numerica
the solution. It can help a user in deciding whether a given solution with th
value is really optimal for him/her or not. The arguments of SOFTPREC that a
might be interested in visualizing are the states of the variables associated wit
features, soft precedences, and the value of the subscription being computed
the interesting states are whether a feature (or a user precedence) is included
or undecided, or whether the current state is a result of the last choice or th
choices.

Figure 7 depicts the search tree (on the left, generated by a branch and bou
algorithm) explored until the node number 38 and the states of the variables (
straint propagation) at that node, when solving an instance of feature subscri
20 features and 10 user precedences. In Figure 7 leaf-nodes of the search t
sponding to feasible solutions are shown in green (light gray), dead-ends are
red (dark gray), and the current node (node number 38) is shown in blue (la
Figure 7 (right) visualizes the state of the variables after reaching the fix poin
ing the propagation at node number 38.

The states of the features are visualized using a vector of cells (shown at t
and to the right). When a feature is undecided the corresponding cell is unla
feature is included or excluded then the cell is labeled with either 1 or 0. The
between the features that are included/excluded in the current node from tho
decided in the earlier nodes is made through the difference in the backgroun
the cells. The states of the variables associated with the soft precedence cons
visualized through a matrix of cells. Each soft precedence constraint $i \prec j$ is
with a cell in row i and column j. If a soft precedence $i \prec j$ holds then the co
ing cell is labeled with 1 and if it is violated then the corresponding cell is la
0, and undecided soft precedence is labeled with 01. The bounds of the va
subscription being computed is displayed on the left bottom of Figure 7 (righ

SOFTPREC internally maintains transitivity on the hard precedence cons
hard precedence constraint, $i \prec j$, means that if features i and j are incluc
must precede j. From a developer's point of view it is interesting to visualize
of these variables, which is done through the background colors of the cells
trix. SOFTPREC also elicits and maintains incompatibilities between undecide
through the states of these variables. An incompatibility between undecided
and j is visualized by placing a box around the cell in row i and column j. For

are computed by associating a graph with a set of incompatibilities, and c the violation cost of each component of the graph. The components are al ized by using different colors for the incompatibilities of different compone it comes to describing the pruning rules of SOFTPREC it is much easier to exp through visualization. Initially it was agreed to implement a static variable or SOFTPREC that chooses variables associated with soft precedences before the associated with features. However, after visualizing the search tree, we disco the intended variable ordering was not implemented in the right way. Anoth tage of visualization is that it can help in understanding the impact of the s different pruning rules.

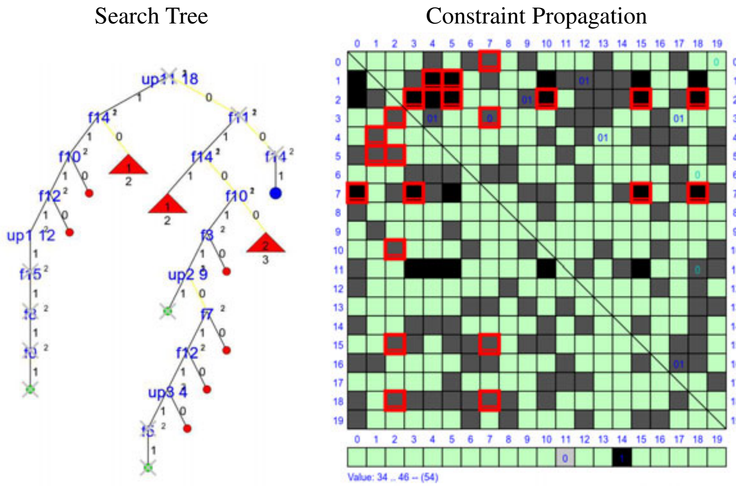

Fig. 7. Example of SOFTPREC Global Constraint Visualization

5.4 JSR331

The Java Specification Request (JSR) 331 (`http://jcp.org/en/jsr/d id=331`) is a working group in the Java Community Process trying to propos standard constraint programming API for Java. As part of a reference imple we have considered the use of CP-VIZ as an example of a visualization ext the standard API. Figure 8 shows a code example for an annotated N-queen in the proposed standard syntax. For most classes (Problem, Solver, individ constraints), variants which incorporate the visualization capabilities of CI provided. By creating for example a new constraint from ALLDIFFERENTV stead of ALLDIFFERENT , a visualization for this constraint will be provi that not all constraints and variables need to be annotated, the user can conc only parts of the model, if required. Figure 9 shows an UML sequence diagr

```
        ProblemVisual problem = new ProblemVisual("Queens
        int size = 16;
        problem.startVisualization("QueensProblem.log");
        Var[] x = problem.varArray("x",0, size-1, size);
        Var[] x1 = new Var[size];
        Var[] x2 = new Var[size];
        for (int i = 0; i < size; i++) {
            x1[i] = x[i].add(i);
            x2[i] = x[i].sub(i);
        }
        problem.register(x);
        new AllDifferentVisual(x).post();
        problem.snapshot();
        new AllDifferent(x1).post();
        new AllDifferent(x2).post();
        SolverVisual solver = new SolverVisual(problem);
        solver.startVisualization("QueensSolver.log");
        Solution solution = solver.findSolution();
        solver.stopVisualization();
        problem.stopVisualization();
    }
}
```

Fig. 8. JSR 331 Example: Visualization of N-Queens Problem

interaction of the Application, VisualProblem, VisualSolver and Constrai
which shows how the JSR331 implementation builds on the interface of Figu

6 Future Work and Conclusions

While the current CP-VIZ system already provides many useful features
standing and improving constraint programs, there are a number of features t
improve its capabilities:

- At the moment the system can tell the user which choice led to a failure, b
  provide a more detailed explanation. It would be helpful if we can integ
  explanation tools which can provide automatically derived explanations c
- Much of the development time for a constraint application is taken up wit
  ing different possible design choices. We will study how to best compare se
  and constraint and variable visualizations from multiple runs in a single
- The invariant checker provides a useful paradigm for concentrating effor
  esting parts of the search effort, but at the moment the checks are compi
  of the tool itself. It might be interesting to allow users to specify check
  tively, and display such search results inside the visualization.

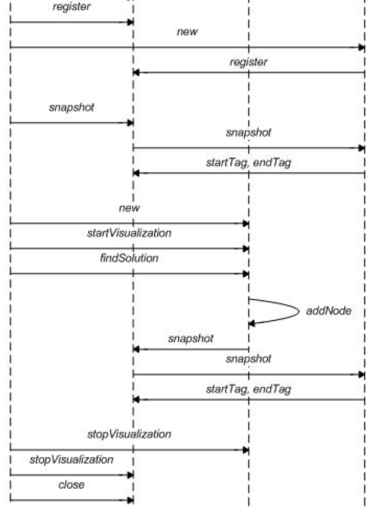

Fig. 9. UML Sequence Diagram - Message Flow between Application and Visualizati

By providing an open-source, system independent visualization platform, CI
help to reduce the amount of duplicated and redundant work required by syst
opers, while allowing specific, new features to be added without too much e
current documentation and software for CP-VIZ can be found at `http://4`
`ie/~hsimonis/CPVIZ/index.htm`.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex schedu lems. Journal of Mathematical and Computer Modelling 17(7), 57–73 (1993)
2. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A generic g constraint kernel in space and time for handling polymorphic k-dimensional Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 180–194. Springer, Heidelberg
3. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog. Techn T2005:08, SICS (May 2005)
4. Deransart, P.: Main results of the OADymPPaC project. In: Demoen, B., Lifschit ICLP 2004. LNCS, vol. 3132, pp. 456–457. Springer, Heidelberg (2004)
5. Deransart, P., Hermenegildo, M.V., Małuszyński, J.: DiSCiPl 1999. LNCS, Springer, Heidelberg (2000)
6. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., Berthier, F straint logic programming language CHIP. In: FGCS, pp. 693–702 (1988)
7. Dooms, G., Hentenryck, P.V., Michel, L.: Model-driven visualizations of const local search. Constraints 14(3), 294–324 (2009)

College Cork (2007)

10. Lesaint, D., Mehta, D., O'Sullivan, B., Quesada, L., Wilson, N.: A Soft Global
Constraint. In: IJCAI 2009, Pasadena, CA, USA (2009)
11. Lesaint, D., Mehta, D., O'Sullivan, B., Quesada, L., Wilson, N.: Consistency
for finding an optimal relaxation of a feature subscription. In: Proceeding of the
International Conference on Tools with Artificial Intelligence (ICTAI 2008), p
(2008)
12. Meier, M.: Debugging constraint programs. In: Montanari, U., Rossi, F. (eds.
LNCS, vol. 976, pp. 204–221. Springer, Heidelberg (1995)
13. Nieuwenhuis, R.: A cumulative scheduling problem. Personal Communication (2
14. Schulte, C.: Oz Explorer: A visual constraint programming tool. In: ICLP, Leuve
pp. 286–300 (1997)
15. Simonis, H.: An ECLiPSe ELearning course (2009),
`http://4c.ucc.ie/~hsimonis/ELearning/index.htm`
16. Simonis, H., Aggoun, A.: Search-tree visualisation. In: Deransart et al [5], pp. 19
17. Simonis, H., Aggoun, A., Beldiceanu, N., Bourreau, E.: Complex constraint
Global constraint visualisation. In: Deransart et al [5], 299–317