ACM Computing Surveys 32 (1es)

The Framework Approach for Constraint Satisfaction

Pierre Roy[1], Anne Liret[1], François Pachet[2],

[1]LIP6, University Paris 6, [2]SONY CSL-Paris

Abstract: Constraint satisfaction programming (CSP) is a powerful paradigm for solving complex combinatorial problems, which has gained a lot of attention recently. Putting the power of constraint satisfaction into the hands of programmers and designers in a simple fashion is, however, still an open issue. We propose to classify the various approaches to this problem into three categories: the *language* approach, in which only basic mechanisms are proposed to the user, the *library* approach, in which the user may pick up pre-designed algorithms off-the-shelf, and the *object-oriented framework* approach, an intermediary position between languages and libraries. We outline the design of such a framework and stress on the various advantages of this approach compared to the other ones.

Keywords: Constraint satisfaction, Object-oriented programming, Frameworks

1 Introduction

Constraint satisfaction programming (CSP) is a powerful paradigm for solving combinatorial problems, which was initially seen as an *algorithmic* issue [Mackworth 1977], [Laurière 1978]. The first proposals for integrating constraints in a language were developed within the community of logic programming. Constraint logic programming (CLP) was primarily designed to deal with specific computation domains like integer numbers. Its best known representatives are PROLOG III [Colmerauer 1990], CHIP [Van Hentenryck et al. 1989] and CLP (FD) [Codognet and Diaz 1996].

Using CSP from within a programming language is a definitive advantage, compared to the situation where the user must call an external system. Depending on what the language offers to the user, the integration of CSP may take the three forms reviewed below: library, language constructs, or framework.

Library of generic constraints

In this approach the objective is to identify generic constraints that can be used in a wide range of applications, (e.g. global constraints in CHIP [Beldiceanu and Contejean 1994]). This approach is adapted to classical problems, and in this case the only task is to formulate the problem in terms of the predefined constraints. This can be summed up by the phrase "constrain and solve". For specific problems, since constraints are complex and domain independent, this formulation may be hard to find.

The language construct approach

This approach is illustrated by CLAIRE [Caseau and Laburthe 1996], a language for building constraint solvers. CLAIRE does not propose any predefined resolution mechanisms, but integrates general and efficient low-level constructs that can be used to build specific solvers (i.e. a save/restore and a forward chaining rule mechanisms). This approach can be seen as the opposite of the library approach: the user has a lot to do, but ends up with an efficient implementation of his algorithms. This is particularly well suited to hard problems not identified as instances of well-known classes of problems.

The framework approach

The framework approach is an intermediary position. It comes from works aiming at integrating *object-oriented languages* with constraints. Rather than providing specific computation domains as for CLP, the interest of integrating constraints and objects is to provide extensible and flexible implementations of CSP (e.g. COOL [Avesani, et al. 1990], ILOGSOLVER [Puget and Leconte 1995], LAURE [Caseau 1994]). Besides, objects provide facilities for *domain adaptation*. One particularly efficient way to achieve domain adaptation is to provide *frameworks* [Fayad and Schmidt 1997] in

[1] LIP6, Boîte 169, 4 Place Jussieu, 75 252 Paris Cedex, France. E-mail: {roy | liret }@poleia.lip6.fr

[2] SONY CSL, 6, rue Amyot, Paris Cedex, France. E-mail: pachet@csl.sony.fr

which 1) general control-loop and mechanisms are coded once for all, and 2) adaptation to specific problems can be achieved easily. More than a class library, a framework is a "semi-complete" application containing integrated components collaborating to provide a reusable architecture for a family of applications. We now outline the features of such a framework.

2 Implementation of the BACKTALK Framework

The framework described here, called BACKTALK, consists of a library of constraint classes and of a general resolution algorithm, linked up by a mechanism of demons. BACKTALK provides modern constraint satisfaction techniques embedded in an open object-oriented environment [Roy and Pachet 1997a].

2.1 Solving Algorithms

Many algorithms were developed for solving CSPs. Their respective efficiency highly depends on the problem to solve, and, as claimed in [Caseau 1991], *"no constraints solver to our knowledge holds all the techniques that we have found necessary to solve [particular problems]"*.

However, most of them are based on the same abstract scheme: combining a propagation mechanism with a backtracking strategy used to save and restore successive states of the problem. We propose to unify these algorithms into a single control-loop, and to use inheritance to adapt it to specific cases. To do so, we use the "Strategy" and "Template Method" design patterns [Gamma et al. 1994].

Following the "Strategy" pattern, we represent resolution algorithms as a library of interchangeable SMALLTALK classes. To facilitate reuse, we use the "Template Method" pattern: the algorithm is decomposed into several elementary methods representing various steps of the resolution. This allows different resolution strategies to be made up by redefining the elementary methods that vary.

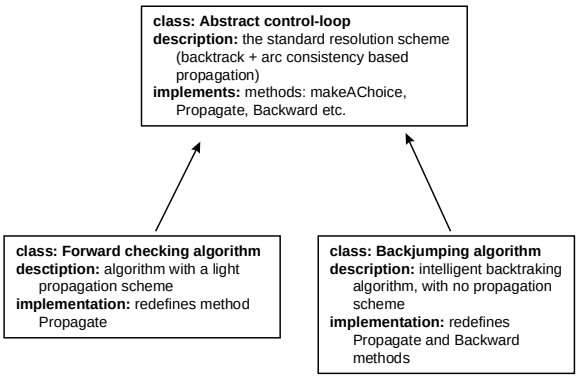

Figure 1 Following the "Strategy" pattern, resolution algorithms are represented as interchangeable classes (e.g. abstract control-loop, forward checking and backjumping algorithm). Following the "Template Method" pattern, the abstract control-loop class implements several methods that can be redefined in subclasses

2.2 Constraints Classes

The central concept is constraint filtering [Bessière and Régin 1997], which is used during the resolution to reduce the problem. Constraint filtering depends on the nature of the constraint considered but is domain independent. This leads to organizing constraints into a hierarchy of classes, each class defining its own filtering procedure. This offers the advantage of providing the user with many ready for use constraints and the ability to define new constraint from predefined ones.

The core of the framework is a mechanism of demons linking together the resolution algorithm and the constraints. The idea is to implement filtering as a set of special methods, called demons, which are *automatically triggered* during the resolution. This selective activation mechanism, close to the implementation of rules in CLAIRE [Caseau and Laburthe 1996], allows filtering methods to be specified in a modular and efficient way.

As an illustration, consider a simple constraint: $x \geq y$. To implement its filtering mechanism, we define two elementary methods. Each method is run when a specific event occurs. More precisely, the first

method is triggered when the maximal value of x is changed, while the second corresponds to a modification of the minimal value of y. (Below is the SMALLTALK code.)

```
xMaxValueChanged
    y = x maxValue
```

```
yMinValueChanged
    x = y minValue
```

3 Using the BACKTALK Framework

There are two different ways of extending BACKTALK: by composition or by inheritance; that is, following the terminology of [Johnson and Foote 1988], as a respectively black- or white-box framework.

3.1 Black-Box Framework

BACKTALK allows predefined constraints to be composed to make up more complex constraints. Different ways of combining constraints, using specific demons, are predefined in BACKTALK: conditional, disjunctive or recursively defined constraints. This is a means of extending the framework with new constraints.

For instance, *disjunctive constraints* are often needed in scheduling problems to specify that a resource must be used by at most one process at a time. Instead of implementing each disjunctive constraint as a new class, as in the "generic constraints" approach, one can simply define it by combining elementary constraints.

3.2 White-Box Framework

Besides extending the hierarchy of predefined constraints by inheritance, one can defined new resolution algorithm. As written in Section 2.1, resolution algorithms are similar. They differ in two parameters: the amount of constraint filtered at each cycle of the search and the backtracking strategy.

In BACKTALK, the amount of constraint filtered at each cycle is controlled by demons attached to the constraints of the problem. The backtracking strategy is set by one of the methods implementing the control-loop (following "Template Method"). Using the "Strategy" pattern, one can implement resolution procedures with various backtracking and filtering strategies.

We developed a new resolution algorithm (inspired by both forward checking and backjumping [Prosser 1993]) to solve a specific problem, as reported in [Roy and Pachet 1997b].

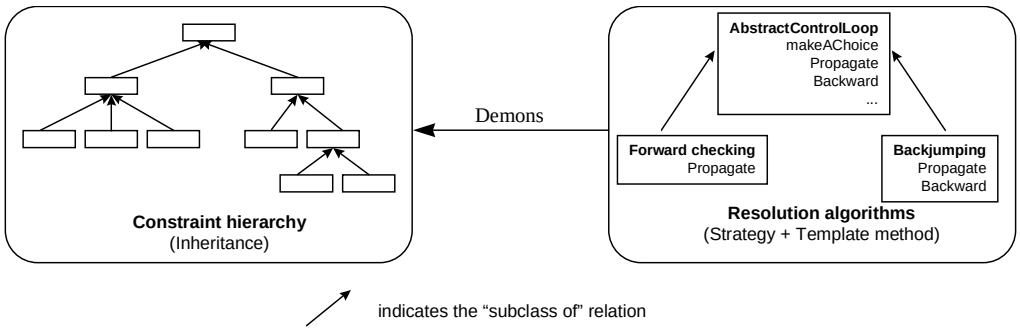

Figure 2 The overall architecture of the BACKTALK framework

4 Conclusion

The framework paradigm offers a smooth and efficient integration of CSP with objects. One way to assess the relevance of this approach, as opposed to the language-based approach, is to compare it with two extreme cases: CHIP and CLAIRE. The main difference with CHIP is that since BACKTALK provides the relevant concepts of CSP, including algorithms, as classes, it allows to redefine them by inheritance, thus gaining flexibility. The difference with CLAIRE is that BACKTALK imposes the main control-loop, whereas CLAIRE leaves it to the responsibility of the user: since CLAIRE has all the abilities of a complete hybrid language, it is suitable for highly specific applications. Table 1 illustrates the position of the framework approach.

Approach	Main characteristics	Examples
Library	Parameterized high-level constraints	CHIP
Framework	Control-loop, simple constraints	BACKTALK, ILOGSOLVER
Language	Low-level language constructs	CLAIRE

Table 1 The three approaches in proposing CSP mechanisms to a user

The framework approach is claimed more comfortable for standard applications because it provides relevant predefined abstractions. By hiding from the user the difficult mechanisms of CSP algorithms, e.g. the save-restore strategy, while allowing him to redefine parts of it, BACKTALK achieves a desirable feature of frameworks, that is a good compromise between efficiency and complexity. This echoes Steve Jobs' opinion concerning interface builder frameworks: "Simple things should be simple, complex things should be possible."

References

[Avesani, et al. 1990] P. Avesani, A. Perini, and F. Ricci, "COOL: An Object System with Constraints" proceedings of TOOLS'2, Paris, pp. 221-228, 1990.

[Beldiceanu and Contejean 1994] N. Beldiceanu and E. Contejean "Introducing global constraints in CHIP" *Journal of Mathematical and Computer Modelling*, 20 (12), pp. 97-123, 1994.

[Bessière and Régin 1997] C. Bessière and J.-Ch. Régin, "Arc-Consistency for General Constraint Networks: Preliminary Results" proceedings of IJCAI, Nagoya, Japan, pp. 398-404, Aug. 1997.

[Caseau 1991] Y. Caseau, "Abstract Interpretation of Constraints over an Order-Sorted Domain" proceedings of International Logic Programming Symposium, San Diego (Ca), pp. 435-454, 1991.

[Caseau 1994] Y. Caseau, "Constraint Satisfaction with an Object-Oriented Knowledge Representation Language" *Journal of Applied Intelligence*, vol. 4, pp. 157-184, 1994.

[Caseau and Laburthe 1996] Yves Caseau and François Laburthe, "CLAIRE: Combining Objects and Rules for Problem Solving" proceedings of the JICSLP'96 workshop on multi-paradigm logic programming, M.T. Chakravarty, Y. Guo, T. Ida eds., TU berlin, 1996.

[Codognet and Diaz 1996] Ph. Codognet and D. Diaz, "Compiling Constraints in clp(FD)", *Journal of Logic Programming*, vol. 27 (3), pp. 185-226, 1996.

[Colmerauer 1990] A. Colmerauer, "An Introduction to Prolog-III" *Communication of the ACM*, vol. 33(7), pp. 69, 1990.

[Fayad and Schmidt 1997] M. Fayad and D. Schmidt, "Object-Oriented Application Frameworks", Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, 40 (10), October 1997.

[Gamma et al. 1994] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1994.

[Johnson and Foote 1988] R. Johnson and B. Foote, "Designing Reusable Classes" *JOOP*, vol. 1(2), pp. 22, 1988.

[Laurière 1978] J.-L. Laurière, "A Language and a Program for Stating and Solving Combinatorial Problems" *Artificial intelligence*, vol. 10(1), pp. 29-127, 1978.

[Mackworth 1977] A. K. Mackworth, "Consistency in Networks of Relations" *Artificial intelligence*, vol. 8 (1), pp. 99-118, 1977.

[Prosser 1993] P. Prosser, "Hybrid Algorithms for the Constraint Satisfaction Problem" *Computational Intelligence*, vol. 9, pp. 268-299, 1993.

[Puget and Leconte 1995] J.-F. Puget and Michel Leconte, "Beyond the Glass Box: Constraints as Objects" propceedings of International Logic Programming Symposium, ILPS, pp. 513-527, Portland, Oregon, Dec. 1995.

[Roy and Pachet 1997a] P. Roy and F. Pachet, "Reifying Constraint Satisfaction in Smalltalk" *Journal of Object-Oriented Programming*, vol. 10 (4), pp. 51-63, 1997.

[Roy and Pachet 1997b] P. Roy and F. Pachet, "A Framework for Expressing Knowledge About Constraint Satisfaction Problems" proceedings of Flairs, Daytona Beach (Florida), pp. 47-51, May 1997.

[Van Hentenryck et al. 1989] P. Van Hentenryck, H. Simonis and M. Dincbas, "Constraint Satisfaction using Constraint Logic Programming", *Artificial Intelligence*, vol. 58, pp. 113-159, 1992.