

Appeared in:
J.C. Rault (ed), Actes de "Représentations Par Objets",
La Grande Motte, France, iunie 1992, pag.201-210.

The Integration of Powerful and Flexible Constraint Representation and Processing into an Object-Oriented Programming Environment

Stefan Trausan-Matu Mihai Barbuceanu Gheorghe Ghiculete

Research Institute for Informatics
8-10 Averescu
71316 Bucharest, 1
ROMANIA

Abstract

This paper presents two consecutive constraint representation and processing systems for artificial intelligence applications. The main aim of the two systems is to provide powerful knowledge representation facilities in the form of constraints. The facilities taken into account include generic constraints, parameterization, hierarchization, a specialized language for declaring the functionality of constraints, concise connections declaration, as well as flexible processing capabilities, all of these integrated into a LISP-based object-oriented programming environment. Special emphasis is placed on answering the question: "Should constraints be objects?" in connection with efficiency problems.

Keywords: Artificial intelligence, expert systems, object-oriented programming, constraint-based programming, knowledge representation.

1. Introduction

Object-oriented programming (OOP) is now widely considered as a powerful programming paradigm, able to cope with software change and reuse. In this paper we are considering OOP languages developed under a LISP environment. This kind of languages (e.g. CLOS [Kee89], KEE [FiK85], and those implemented by the authors of this paper: XRL [BTM87, BaT88], mXRL [Tra89], O3) are mainly dedicated to artificial intelligence (AI) applications. In these languages, objects are strongly related to the frame knowledge representation paradigm [Min75]. As a consequence, complex structuring of objects is used, a great emphasis is placed on multiple inheritance with method combination, and other AI knowledge representation and control mechanisms are integrated: rules, logic programming, demons, constraints. The additional processing involved when using objects is more complex than, for example, in C++ [Str87] but, of course, the representation and processing power also is significantly increased.

Constraint-based programming is a new programming paradigm, useful not only in AI but also for "traditional" applications (e.g. [Bor87, SzM88]). In AI, constraints attracted a great

interest because they are a very suitable representation for describing complex problems involving search [KK88]. Constraint processing has been used in many artificial intelligence classes of problems: electronic circuits analysis [deK84], qualitative reasoning in physics [deK86b], job shop scheduling [Fox83], resource allocation [MCKG88], planning, design [Ste81, MuS90], diagnosis [deK87], and others.

Constraints are relations between variables named the cells of the constraint. Each variable may have values in a particular set, finite or not. Constraint networks are created by sharing cells of a set of constraints. Constraint processing consists in the assignment and/or change of values in cells so that, finally, all constraints are satisfied. Usually, algorithms for constraint processing use a propagation technique. The main idea is that changes propagate locally from one constraint to another. These algorithms usually record (in an adequate data structure, for example, a queue) all the constraints to be processed. After considering one constraint, all the constraints affected by the computations are recorded at their turn. Therefore, constraint propagation may be implemented as a cycle of selecting a constraint, updating its cells in order to assure that it holds, and recording the constraints affected (which have common cells with modified cells of the current constraint). Propagation comes to an end when there are no more constraints in the data structure [Dav87].

We shall present in this paper two approaches for integrating constraint-based programming and LISP-based object-oriented programming. The first approach is discussed in the next two chapters. Its main idea is to implement constraints as objects in an object-oriented programming environment. A second constraint system is proposed in the fourth chapter. This new approach is the result of the analysis of the inefficiencies of the first approach. One important idea which differentiates it from the former system is that constraints are declared in an object-like syntax but they are not implemented as objects. Both systems were designed for providing a flexible and powerful language for constraint definition and processing.

2. The integration of constraints and objects

The integration of a constraint-oriented framework in an OOP system enhances the power of representation and processing. Relations among objects or among components of an object, transformations of the state of objects (see also [Knu91]), can be very elegantly described. If, in an AI OOP environment, besides constraints, an assumption-based truth maintenance (ATMS) [deK86a] and world mechanisms [Fil88] are integrated, complex search problems can be very naturally described and solved.

Several systems integrating OOP with constraints have been developed [EpL88, Fil88, Giu89, GJV87, MEP88, Ste81, SzM88]. Each of them uses a different manner of representing constraints. For example, in KEE [Fil88], constraints are represented as rules. In others (for example, [Giu89]) constraints are relations between object components. From our point of view, we consider that constraints must have a more complex representation. The rule representation is at a too small granularity level, and the

relation one at a too high one. We have decided to consider a constraint as an entity (something like a black box) with a number of terminals and specific behaviour (similar to [SuS80]). Behaviour can be described by a set of cases (a kind of rules) in a special language, or by a relation. This has as consequences that all the rules for a constraint are grouped together and that complex behaviour can be described.

A "first sight" idea resulting from the above considerations is to represent constraints as objects. The object encapsulation facilities support the black box view of a constraint. The inheritance might also be useful in defining constraint hierarchies. The usage of the same language for objects and constraints is benefic for uniformity and integration purposes.

It seems that the answer to the question "Should constraints be objects?" is yes. However, we shall show in the next chapters that the best answer might not be so simple for AI OOP environments in which objects have very complex implementations. The representation of constraints as objects was one of the leading ideas in the development of our first constraint processing system. Our experience with several applications written using this system has determined us to shade this idea. As a consequence, we have developed a new constraint representation and processing system. This new system, among other novel features, renounces at the object-oriented implementation for constraints. Nevertheless, the constraint declaration language is still object-like. The difference lies in implementation: Constraints are declared with a similar syntax as objects but they are implemented with simpler and more efficient structures than objects.

From the integration point of view we can say that the two systems are very similar. Both of them use a demon mechanism for coupling objects and constraints. We consider constraints as entities on their own, which communicate with objects via if-added demons attached on slots in objects and cells in constraints.

3. An object-oriented constraint representation and processing System

In an OOP environment it may seem natural, as previously discussed, to represent constraints as objects. This idea was leading the design of a constraint processing module [Tra90a] in the object-oriented, classless, COMMON-LISP-based language mXRL [Tra89].

First of all, to give a flavour of mXRL, three examples of objects are shown below. The "circle" object inherits the x and y slots from the "shape" object, has a particular slot named radius, and can respond to the draw message. The "circle32" object is a clone of the "circle" object.

```
(unit shape
  self (a unit draw draw-shape)
  x undf
  y undf)

(unit circle
  self (a unit supers (shape) draw draw-circle)
```

```

radius undf)

(unit circle32 (a circle x 2 y 3 radius 5))

```

An object's meta-description is declared in the self slot. The supers sub-slot declares the list of objects from which the current object can inherit slots and methods. The assignment of methods to messages for the current object is described as selector-method pairs in the self slot.

The drawing of circle32 is done by sending the draw message to it:

```
(msg 'draw 'circle32)
```

The representation in mXRL of an adder generic constraint and of two instances of it (S1 and S2) is described below:

```

(unit Adder
  self (a ConstraintMeta)
  cells (t1 t2 s)
  t1 undf
  t2 undf
  s undf
  computations ((t1 c1) (t2 c2) (s c3))
  c1 (- s t2)
  c2 (- s t1)
  c3 (+ t1 t2))

(unit S1 self (a Adder))

(unit S2 self (a Adder s 17))

```

The propagation of the value 2 for t1 in S2 is performed by:

```
(msg 'propagate 'S2 't1 2)
```

A compound constraint (a three terms adder) can be defined as follows:

```

(unit ThreeAdder
  self (a CompoundConstraint)
  cells (t1 t2 t3 s)
  inner-constraints ((Adder S1 S2))
  connections ((S1 s) (S2 t1))
  mapping ((t1 S1 t1)
           (t2 S1 t2)
           (t3 S2 t2)
           (s S2 s)))

```

Another example of a constraint, for a second generation expert system for diagnosis [Tra90b], is the following:

```

(unit Electrical-element
  self (a ConstraintMeta fix-bug seek-undf)
  cells (works is_good is_connected has_current))

```

```
computations ((works c1) (is_good c2) (is_connected c3) (has_current c4))
c1 (yandyn is_good is_connected has_current)
c2 (test_if_good works is_connected has_current)
c3 (enquire_if_connected works is_good is_connected)
c4 (test_if_has_current works is_good is_connected)
works undf (a slot range (yes no))
is_good (a slot range (yes no))
is_connected (a slot range (yes no))
has_current (a slot range (yes no))
```

Some important ideas gained from our experience with the implementation and usage of this first constraint representation and processing system are:

- 1) Inheritance has not been extensively used for constraints.
- 2) The structure of constraints is, in almost all cases, fixed (for example, in constraint satisfaction and modeling problems, the number of cells in constraints do not usually change). In fact, in applications involving constraints, things which usually change are not constraints but the constrained entities. We must say that this idea might not be correct for design, job-shop scheduling, and planning applications (e.g. [Fox83, Ste81]).
- 3) Generic constraints have been extensively reused, either on their own or included in compound constraints, in the same application or in other ones. In fact, the repertory of new generic constraints defined during several applications was not very large.
- 4) The number of constraint instances is, in general, much bigger than the number of the constrained objects. Taking into account that mXRL is a classless OOP language implemented in LISP, for big applications efficiency problems might appear.
- 5) The declaration of connections between constraints is a critical part for big applications.

Due to the fact that in AI OOP environments, objects are mainly used as frames, which implies that they have usually complex structures and complex additional processing, and considering the above 1), 2), and 4) remarks, we have concluded that implementing constraints as objects might not be the best idea. Nevertheless, there are several facts that supports the object-oriented representation of constraints. This is the reason for which in the COPE system, the successor of the system described above, we have decided to make a compromise between the power of constraint language and efficiency.

4. The cope constraint representation and processing System

The COPE constraint processing system has been designed according to the following goals:

- 1) The provision of facilities for declaring generic and parameterized constraints,
- 2) The provision of a specialized language for declaring the functionality of constraints,

- 3) The support of hierarchical definition of constraints,
- 4) The reduction to a minimum of the declarations of connections between constraints.
- 5) Efficiency for large applications.

Using the experience of the former constraint processing system, special attention was paid to both space and speed efficiency. One of the main decisions following from these requirements is to do not implement constraints as objects. For uniformity, we have decided to describe constraints with the same syntax as objects but to implement them with a fixed, more efficient, structure. We consider that the loss of flexibility is, for constraints, lesser than the gain in efficiency. COPE can be interfaced with any LISP-based OOP system. It is an enhancement of the old system not only in the above described aspects but also in providing newer, stronger abstraction facilities.

COPE has been interfaced with mXRL and O3. The latter is a completely new descendent of XRL with ATMS and worlds mechanisms, and which offers the possibility to implement objects with different functionality and different data structures even in the same application. The syntax of constraint declaration in COPE is very similar to the object declaration in O3.

Generic constraints are descriptions of classes of constraints which can be instantiated to particular constraints. This dichotomy is similar to the class - instance one in class-based object-oriented languages and it serves not only as an abstraction facility but also has some efficiency reasons due to the fact that the internal representation of an instance is much simpler than that of a generic constraint.

One of the main ideas on which COPE was built is the provision of a powerful language for the declaration of the constraint functionality. We have decided to describe the functionality of a constraint as a set of rules. For example, in the generic multiplier constraint with two terms below, the functionality is described by the production rules given in the slot :PropagationPrologue and those indicated as :PropagationCases. In the condition and action parts of the rules, some specific atoms and functions may be used (e.g. :newvalue, :newcell atoms, and :known, :unknown, :exists, :val, :set-cell functions).

```
(DeclareConstraint! multiplier
 :cells (p m1 m2)
 :vars (x)
 :PropagationPrologue .....
 :PropagationCases (:zero_m1_or_m2 :all_known :unknown_p
                       :unknown_m2 :unknown_m1)
 :zero_m1_or_m2 ((:exists x :in '(m1 m2)
                  :suchas (and (:known x)
                               (zerop (:val x)))
                               (:set-cell 'p 0))
 :all_known ((:known '(m1 m2 p))
             (unless (= (:val 'p) (* (:val 'm1) (:val 'm2)))
                   (Break "Unsatisfied constraint relation .....")))
 :unknown_p ((:known '(m1 m2))
            (:set-cell 'p (* (:val 'm1) (:val 'm2))))
 :unknown_m2 ((:known '(p m1))
             (:set-cell 'm2 (/ (:val 'p) (:val 'm1))))
```

```

:unknown_m1 ((:known '(p m2))
             (:set-cell 'm1 (/ (:val 'p) (:val 'm2))))

```

An instance, named `mul1`, of the multiplier is built in the first form and the value 3 for `m1` is propagated using:

```

(MakeConstraint! multiplier :instance-name mul1)

(constr-propagate 'm1 'mul1 3)

```

Parameterized constraints are a further step towards providing powerful abstraction constructs. A parameterized constraint is a generic constraint containing at least a parameter which refers at the declaration of cells in the generic constraint. For example, there might be multipliers with two, three, in general n terms. Without a parameterization facility, for each number of terms a generic constraint has to be defined. By declaring n , the number of terms as a parameter, a generic multiplier with n entries can be defined as follows:

```

(DeclareConstraint! n_multiplier
 :parameters (n)
 :cells ((n :of m :atleast 2) p)
 :vars (x)
 :PropagationPrologue .....
 :PropagationCases (:one_m_zero :all_known :unknown_p :one_m_unknown)
 :one_m_zero ((:exists x :in (:param-cells 'm)
                  :suchas (and (:known x)
                               (zerop(:val x))))
              (:set-cell 'p 0))
 :all_known ((and (:known 'p)
                  (:known (:param-cells 'm)))
             (unless (= (:val 'p) (apply #'* (:param-cells-val 'm)))
                     (Break "Unsatisfied constraint relation .....")))
 .....))

```

An instance of this constraint can be defined as:

```

(MakeConstraint! n_multiplier :instance-name n_mul1 :n 7)

```

The hierarchical definition of constraints serves two aims. The first is following the ideas of [SuS80], that means to give the possibility to define new, compound, constraints from already defined constraints. The second reason for this facility is the necessity of grouping together a set of constraints in a network of interacting constraints.

An example of a hierarchical definition of a constraint describing a resistor, using two adders and a multiplier, is:

```

(:DeclareConstraint resistor
 :cells (u1 u2 i1 i2 r)
 :GroupsOfCells ((t1 (u1 i1))
                 (t2 (u2 i2)))
 :InnerConstraints ((mul :isa multiplier)

```

```

      (add :isa adder)
      (o :isa minus))
:Connections ((:map 'r :to 'm2 :of 'mul)
              (:map 'i1 :to 't1 :of 'o)
              (:map 'i2 :to 't2 :of 'o)
              (:map 'u1 :to 's :of 'add)
              (:map 'u2 :to 't1 :of 'add)
              (:connect 't1 :of 'o :to 'm1 :of 'mul)
              (:connect 'p :of 'mul :to 't2 :of 'add)))

```

For complex problems, with a great number of constraints and with many connections, the declaration of the connections between the constraints can become cumbersome and error-prone. For this reason and for providing the possibility of using parameterized constraints in the definition of other constraints, we have designed a connections language. This language is used, of course, in a compound constraint. It reduces very much the number of the declarations of connections. One example of the usage of this language is the `:connections` component of the Resistor constraint. One further example, involving also a parameterized sum constraint is the `NSeriesResistors`:

```

(:DeclareConstraint NSeriesResistors
 :cells (u1 u2 i1 i2 r)
 :GroupsOfCells ((t1 (u1 i1)) (t2 (u2 i2)))
 :parameters (n)
 :InnerConstraints ((sum :isa (n_adder :n n))
 (res :isa (:set n :of resistor :atleast 2)))
 :connections ((:map 'r :to 's :of sum)
               (:group-map 't1 :to 't1 :of (res 1))
               (:group-map 't2 :to 't2 :of (res n))
               (do ((i 1 (1- n))) (= i n))
                 (:group-connect 't2 :of (res i) :to 't1 :of (res (1+ i)))
               (:connect 'r :of (res i) :to (term i) :of 'sum))
               (:connect 'r :of (res n) :to (term n) :of 'sum)))

```

5. Conclusions

The two systems presented in this paper has been used for developing several applications: a simple second generation expert system, a typical constraint satisfaction problem, a explanation-based learning problem, and some modeling and simulation applications in electronics and ecology [Tra90b]. All these applications have demonstrated that constraint-based representation is a powerful abstraction facility, orthogonal with object-oriented representation, which can dramatically reduce the complexity of programs from the above domains. The enhancements in the second system (the language for constraints functionality description, parameterization, concise connections declarations) are aimed at increasing the representational power of the constraint language. These enhancements are the direct consequences of the experience with the first system.

We have tried to answer to the question "Should constraints be objects?", in the context of LISP-based object-oriented programming languages. Our opinion is that the answer is not categorically yes (or, it might be yes if there could be provided different implementations for objects in the same application). This is due to the fact that in such languages objects are usually used as frames, involving a lot of processing not needed for constraints which

has a more fixed character. Another positive argument for our opinion is that constraints might have a much bigger number than that of the objects involved.

Finally, we want to emphasize that we have tested our COPE system for thousands of constraints obtaining satisfactory results with regard to both space and speed. This was not the case with the former system, in which constraints were implemented as objects

References

- [BTM87] Barbuceanu, M., Trausan-Matu, St., Molnar, B., "Integrating declarative knowledge programming styles and tools in a structured object AI environment", in: Proceedings Tenth IJCAI, Milan, Italy, (1987), pp. 563-568.
- [BaT88] Barbuceanu, M., Trausan-Matu, St., "The XRL2 Manual", ITCI Bucuresti, 1988.
- [Bor87] Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., Woolf, M., "Constraint hierarchies", OOPSLA '87 Proceedings, pp. 48-60.
- [Dav87] Davis, E., "Constraint propagation with interval labels", Artificial Intelligence, 32(1987), pp. 281-331.
- [deK84] de Kleer, J., "How circuits work", Artificial Intelligence, 24(1984), pp. 205-280.
- [deK86a] de Kleer, J., "An assumption-based TMS", Artificial Intelligence, 28(1986), pp. 127-162.
- [deK86b] de Kleer, J., Brown, J. S., "Theories of causal ordering", Artificial Intelligence, 29(1986), pp. 33-61.
- [deK87] de Kleer, J., Williams, B.C., "Diagnosing multiple faults", Artificial Intelligence, 32(1987), pp. 97-129.
- [EpL88] Epstein, D., LaLonde, W. R., "A Smalltalk window system based on constraints", Proceedings OOPSLA88, San Diego, 1988, pp. 83-94.
- [FiK85] Fikes, R., Kehler, T., The role of frame-based representation in reasoning, Communications of the ACM, Vol.28, No.9, (sept. 1985), pp. 904-920.
- [Fil88] Filman, R., Reasoning with worlds and truth maintenance in a knowledge-based programming environment, Communications of the ACM, Vol. 31, No.4, (apr.1988), pp. 382-401.
- [Fox83] Fox, M., "Constraint directed search: a case study of job-shop scheduling", Research report CMU-RI-TR-83-22, Carnegie- Mellon University, 1983.
- [Giu89] Giuse, D., "KR: constraint-based knowledge representation", Research report CMU-CS-89-142, Carnegie Mellon University, 1989.
- [GJV87] Guesgen, H. W., Junker, U., Voss, A., "Constraints in a hybrid knowledge representation system", Proceedings of IJCAI-87, pp. 30-33.
- [Kee89] Keene, S., "Object-oriented programming in COMMON LISP: A programmer's guide to CLOS", Addison-Wesley Publishing Company, Reading, MA, 1989.
- [KK88] Kanal, L., Kumar, V. (eds.), "Search in artificial intelligence", Springer-Verlag 1988, pp. 287-342.
- [Knu91] Knudsen, J.L., Object-orientation as an integrating perspective on programming, Proceedings EastEurOOPe'91, Bratislava, 1991.
- [MCKG88] Mott, D. H., Cunningham, J., Kelleher, G., Gadsden, J. A., "Constraint-based reasoning for generating naval flying programmes", Expert Systems, August 1988, Vol. 5, No. 3, pp. 226- 246.
- [MEP88] Motta, E., Eisenstadt, M., Pitman, K., West, M., "Support for Knowledge acquisition in the Knowledge Engineer's Assistant (KEATS)", Expert Systems, febr. 1988, vol. 5, nr. 1, pp. 6-27.
- [Min75] Minsky, M., "A framework for representing knowledge", in: P.Winston (ed.), "The psychology of computer vision", pp. 211- 277, McGraw Hill, New York, 1975.
- [MuS90] Murtagh, N., Shimuro, M., "Parametric engineering design using constraint-based reasoning", Proceedings of AAAI90, 1990, pp. 505-510.
- [Ste81] Stefik, M., "Planning with constraints (Molgen: Part1)", Artificial Intelligence, 16(1981), pp. 111-140.

[Str87] Stroustrup, B., The C++ programming language, Addison- Wesley, 1987.

[SuS80] Sussman, G. J., Steele, G. L., "CONSTRAINTS - A language for expressing almost-hierarchical descriptions", Artificial Intelligence, 14 (1980), pp. 1-39.

[SzM88] Szekely, P., Myers, B. A., "A user interface toolkit based on graphical objects and constraints", Proceedings OOPSLA88, San Diego, 1988, pp. 36-45.

[TBG91] Trausan-Matu, St., Barbuceanu, M., Ghiculete, Gh., "Constraint representation and processing", I.C.I., june 1991 (in romanian).

[Tra89] Trausan-Matu, St., "Micro-XRL: An object-oriented programming language for microcomputers", Research report, Technical Cybernetics Institute, Bratislava, 1989.

[Tra90a] Trausan-Matu, S., "Constraint processing in the mXRL object-oriented language", Research report, Institute for Technical Cybernetics, Slovak Academy of Sciences, Bratislava, 1989.

[Tra90b] Trausan-Matu, S., "The development of constraint processing applications in the mXRL object-oriented language", Research report, Institute for Technical Cybernetics, Slovak Academy of Sciences, Bratislava, 1989.