1

Event-based.constraints: coordinate.satisfaction –> object.state [1]

Remco C. Veltkamp and Edwin H. Blake

This paper is about systems support for interactive computer graphics. The aim is to integrate the two major approaches to dealing with complexity in the design and implementation of such systems, namely, constraints and object-oriented programming.
The use of constraints in managing the complexity of designing interactive graphics systems and the use of object-oriented methods for describing simulations and systems of concrete objects have been two natural methods for building large complex graphics systems. This widely acknowledged way of dealing with the complexities of modelling and interface design has had disappointingly little practical impact.
We have identified a major cause for the lack of progress in combining constraints and object-oriented methods. We believe that a proper solution to the problem requires a radical separation of the constraint system and the normal object-oriented framework. In this paper we propose a way of dealing with these problems by means of two orthogonal communication strategies for objects: events and messages.

1 Introduction

The use of constraints in managing the complexity of designing interactive graphics systems and graphical user interfaces dates back to the earliest days of interactive graphics — consider Sutherland's Sketchpad from the early sixties [15]. Object-oriented methods with their usefulness for describing simulations and systems of concrete objects have been a natural method for building large complex graphics systems. The great benefits of class inheritance in user interface design is well recognized and is finding increasing commercial application.

The desirability of combining object-oriented methods and constraints has a similar venerable and distinguished lineage — a major system from the late seventies was Borning's ThingLab [5] which was written in Smalltalk. On the whole, and rather surprisingly, this widely acknowledged way of dealing with the complexities of modelling and interface design has had disappointingly little practical impact.

If one plans to use object-oriented methods to manage complexity in building interactive computer graphics systems, and if one also wants to provide constraints as a tool to manage the complexity of analysis, design, and interaction, then constraints and objects must be combined in a harmonious and coordinated whole. However, the integration of

[1] The syntax used in the title is explained in section 6.

constraints and objects leads to conflicts in programming methodologies [10], and we believe that this is one of the major causes of the lack of application of constraints and their low profile within the mainstream object-oriented approach (another major problem is the difficulty in providing powerful and general constraint solving methods).

We distinguish two incompatibilities between constraints and object-oriented concepts:

- a constraint solver looks at, and sets, the constrained objects' internal data, which conflicts with the data encapsulation concept in the object-oriented paradigm;
- object-oriented programming is imperative, while constraint programming is declarative.

2 Constraints and data encapsulation

To illustrate the problem, let us look at an example, say from a geometric figure editor. Suppose we have a circle `C` with data fields `x`, `y` and `r` representing the centre and radius, an axis parallel rectangle R with data fields `l`, `r`, `b`, and `t` representing the left, right, bottom, and top sides (see figure 1). Suppose further that we have the constraints that the objects touch each other and have equal area.

We could express our constraints as follows:

```
touching:     C.x+C.r=R.l
areas equal: π×C.r×C.r = (R.t-R.b)×(R.r-R.l)
```

A constraint solver may come with the following solution (see figure 2):

```
C.x=5, C.r=1
R.l=6, R.r=7
R.b=0, R.t=π
```

Encapsulation is first violated by the constraint expressions, and then by expressing the solution. To avoid this problem, approaches based on message passing have been proposed. In [12], the methods of an object that may violate constraints are guarded by so-called propagators. The propagators send messages to other objects to maintain the constraints. This technique is similar to the pre- and postcondition facilities in Go [8] [6]. This approach is limited to constraint maintenance (i.e. truth maintenance, as opposed to starting with an inconsistent situation that is then resolved).

A more powerful technique is presented in [17]. The constraint solver produces a set of programs that solve constraints which are stated in the form of equations in terms of

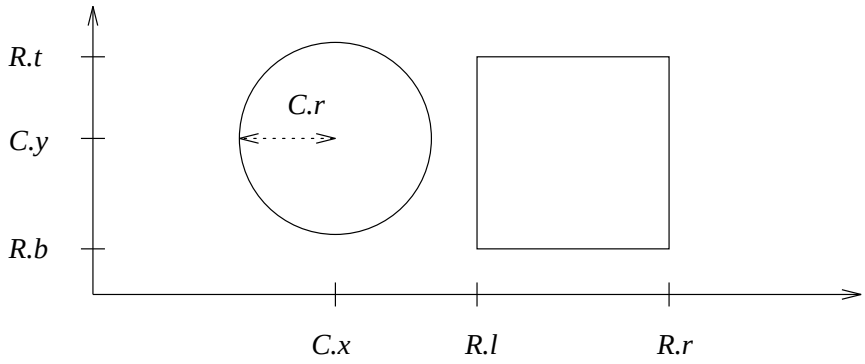

Figure 1: The circle and the rectangle must touch and must have equal area.

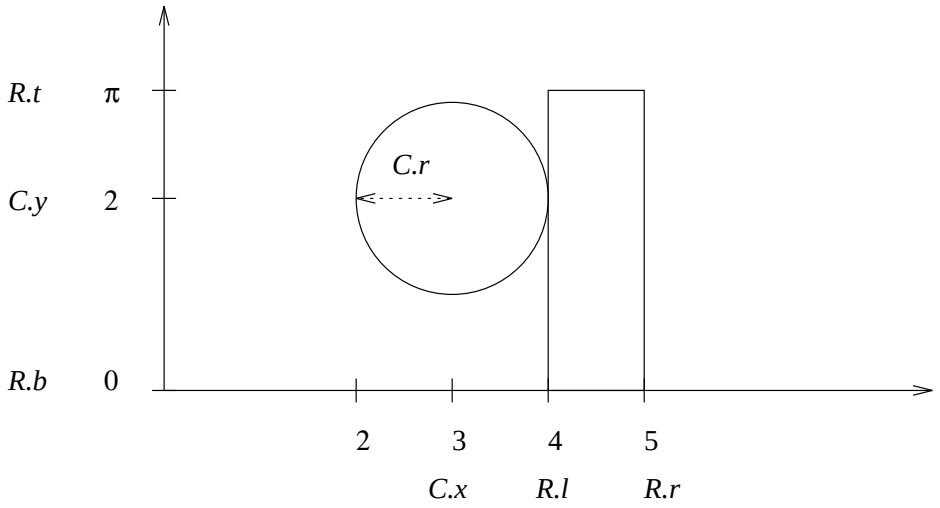

Figure 2: The circle and the rectangle touch and have equal area.

messages to the objects. It translates a declarative constraint into procedural solutions in terms of messages back to objects. This amounts to the constraint system maintaining a view on the states of the objects. The constraint solver is then able to reason about the current state of the objects and propose procedures to fulfill the constraints.

For the above example, typical constraint equations would be:

```
left(R)=right(C)
area(C)=area(R)
```

And a possible solution is:

```
scale(C,distance(R)/radius)
scale(R,area(C)/area)
```

The problem here is that the second method destroys the first constraint, which must be repaired. Doing so destroys the second constraint, etc. The real problem is the local character of the solution. More powerful solution are necessarily global in nature. The danger is that all objects need methods to get and set their internal data. This however, allows every other object to get and set these values, which is clearly against the object-oriented philosophy.

One way to restrict this, is to have an object allow value setting only when its internal constraints remain satisfied (see [14]). A constraint could be made internal by constructing a 'container object', which contains the constraint and the operant objects, but this does not solve the basic problem. In particular, the state of active objects cannot be changed without their explicit cooperation. (Active objects, or actors, conceptually have their own processor and behave autonomously, which is typical in animation and simulation.) Another approach is to limit access to private data to constraint-objects or the constraint solver-objects only. For example C++ provides the 'friend' declaration to grant functions access to the private part of objects. This is also comparable to the approach taken by [7], where special variables (slots) are accessible by constraints only. One can argue that encapsulation is still violated (and specifically that the C++ friend construct is not intended to allow changing the state of an object). Alternatively one can see constraints more as a means to manipulate information in an orderly and restricted way, than that they violate the data encapsulation principle [16], i.e. they provide controlled violation [4].

3 Relations in the Object-Oriented Paradigm

It should be pointed out that the problem of integrating constraints in the object-oriented paradigm is a sub-class of the problem of expressing relations in general in object-oriented programming. Constraints are functional relations that restrict the values which variables in an object can assume. One simple way of avoiding the encapsulation problems associated with constraints would be to include the constrained objects as part of some larger container object. It should be obvious by now that this is no real solution [3].

However, we would expect that a good approach to combining constraints and objects would provide interesting and useful pointers to dealing with problems of aggregation, parts and wholes, and inter-object relationships in general. This in turn has clear connections with object-oriented database research.

4 Imperative vs. Declarative

Object-oriented languages are imperative, and thus use a notion of state, particularly represented by objects. On the other hand, pure constraint languages are declarative, and thus specify one single timeless state: the solution to the specified problem. Both paradigms can be combined as in [9], where an imperative assignment to a variable sets a value at one moment in time, and a declarative constraint dictates a value from that moment on.

However, active objects, or actors, behave totally independently and do not by themselves need to have a notion of some sort of global time. This holds in particular in simulation and animation applications if objects are modelled as concurrent autonomous entities. One aspect of time, however, is the synchronization of objects, such as the constraint that actions of objects take place in intervals that must overlap, or have an explicit ordering. Another type of constraints on time is for modelling object behaviour during the life time of the object.

One important issue involved with constraints and time is that if the solution depends on the order in which constraints are solved, then some of the declarative semantics is destroyed.

5 Combining Objects and Constraints

The justification for combining objects and constraints derives from the fact that it addresses the problems of complexity in large interactive graphical systems which arises on two fronts. The first is the complexity inherent in specifying the behaviour of animations and interactions with many components or objects. Constraints allow the declarative modelling of the behaviour of such systems. The second front is the complexity due to the fact that we are dealing with large software systems. Sound software engineering principles, such as data encapsulation, are needed to cope with large complex software systems.

It appears that all constraint systems in an object-oriented environment infringe the data encapsulation principle to some extent. The debugging of the constraint *satisfaction* routines, which have global effects, is the responsibility of the system programmer who provides the whole interactive graphical programming environment. At least the responsibility for integrity is shifted from the constraint user to the constraint system implementor. A problem that remains is the difficulty of debugging a constraint *specification*, due to the global effects of constraints. However, these global effects should be contained within a declarative constraint programming environment where the well known

techniques of declarative software engineering are applicable (e.g., provability, executable specifications).

The time complexity of constraint satisfaction depends on both the domain and the kind of constraints. For example, linear constraints over real numbers can be solved in polynomial time, discrete constraint satisfaction problems are NP-complete, a single polynomial constraint of degree higher than four does not even have an analytical solution, and the complexity for integer polynomials of degree greater than two is still unknown. An interesting conjecture is that in the absence of global information of some kind, "interesting" constraint resolution will require exponential time [personal communication, Wilk]. It might be interesting to prove the NP completeness of an identified class of constraint resolution methods under the assumption of strict data encapsulation.

Concluding, powerful constraint solvers are global in nature and are hard to integrate with objects. Wilk's solution [17] is too complex to be really useful, and the global view on the object states does not reduce resolution complexity. Rankin's approach [14] does not allow powerful constraint solvers. The integration of Freeman-Benson [9, 10] has been taken about as far as it can in terms of efficiency. By contrast, we believe that it is worthwhile to explore a solution that keeps the paradigms distinct and does not compromise the benefits which they severally confer.

6 Event-based constraint handling

We believe that a proper solution to the problem requires a radical separation of the constraint system and the normal object-oriented framework. In this paper we propose a way of dealing with these problems by means of orthogonal communication strategies for objects. These are events and data streams on one hand, and messages on the other hand.

Events are globally broadcast communications which can be received selectively. When they are received, events cause a pre-emptive invocation of routines (interrupts). Events can be generated by state changes in objects. A *stream* is a connection between an output and an input port of processes, for example objects. Coordinators determine how these objects are interconnected by streams and how their interaction pattern changes during the execution life of the system. *Messages* are the normal communications between objects in the object-oriented sense.

For the modelling of the interaction pattern we use the Manifold model of coordination [2]. The focus of this model is on the coordination of processes and on their communication, not on the computations performed by some of the processes. These processes are considered as black boxes whose behaviour is abstracted to their input and output. The communication is supported by two mechanisms: data-flow streams and event broadcasting. The data-flow streams form a network of streams, linking input and output ports of the processes and carrying the units exchanged between them. The event broadcasting mechanism provides control on the dynamical modification of the data-flow network.

Atomic processes are external for Manifold, and atomic in the sense that they are considered as black boxes of which no internal feature or behaviour is known. At the level of Manifold, they cannot be decomposed further than their input and output channels. An atomic process can:

- raise an event,
- take a unit from a stream connected to an input port,
- put a unit out to the streams connected to an output port.

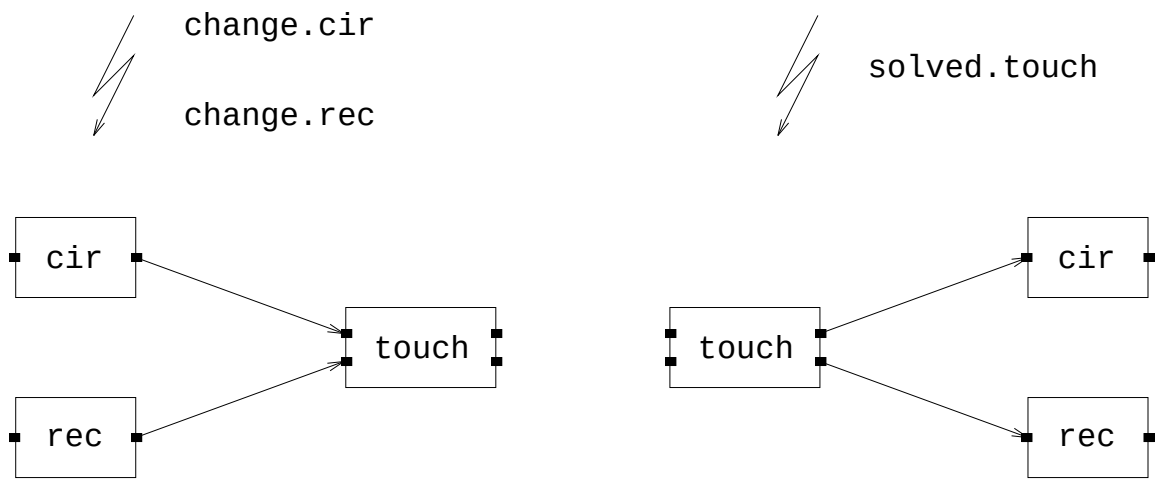

Figure 3: Data-flow networks controlled by coordinator `touch_coord`, triggered by the events.

Streams carry units from the output port to the input port. There is no assumption about the contents of units, this is left to computations in atomic processes. A 'coordinator' is a process that sets up and breaks down streams between processes, i.e. a data-flow network. When an event is raised the previous network is dismantled and the new network is set up.

In the syntax of Manifold, `ev.obj` denotes the event `ev` raised by object `obj`, and `obj1.out -> obj2.in` denotes the linking of output port `out` of `obj1` to the input port `in` of `obj2` by a stream. `(a,b)` is the parallel composition of `a` and `b`, where `a` and `b` are processes or streams. The full syntax is described in [1]. (This syntax is also used in the title.)

A possible coordinator for a global solution of the above example may partially look as follows:

```
touch_coord(cir,rec,touch)
process cir, rec, touch.
{ event wait.

  start:            do wait.

  change.cir:       (cir->touch.in1, rec->touch.in2).
  change.rec:       (cir->touch.in1, rec->touch.in2).
  satisfied.touch:  do wait.
  solved.touch:     (touch.out1->cir,touch.out2->rec).

  wait:             (cir,rec).
}
```

Event `change` from either `cir` or `rec` causes the creation of a communication network from the constraint operands to the constraint (see figure 3). In this example the constraint touch itself does the satisfaction. If it finds a solution, it raises the event `solved.touch`. Then the coordinator creates streams from the constraint to the operand objects. The

coordinator only creates the communication network, all the atomic processes are responsible for actually doing something.

7 Object and constraint models

We want a change of a variable to lead to the checking of the validity of constraints on the variable. A possible approach is to have a central data base with values of the object member variables, and a data manager. Satellites processes could then subscribe to events such as changing variables. When an event occurs, the data manager notifies all satellites that subscribed to that event. The concept of such a central data manager is hard to combine with object-oriented concepts such as data encapsulation.

In Manifold, all the objects conceptually are active objects. This means that every object has its own virtual processor with its own thread of control (as mentioned in section 2). When the value of an object's variable is changed, we let it raise the event `change`.

We are currently exploring two alternative approaches to modelling constraints. In the first approach, the constraints are solved and maintained in Manifold. In this way the application objects and the constraints are completely orthogonal. The communication between the objects and the constraint side is via data streams set up under Manifold control.

In the second approach, constraints are modeled as objects, just like the application objects. In this scheme, each constraint object `cstr` has an associated shadow coordinator `cstr_coord` (like `touch_coord` in the example above). The coordinator can listen to an event `change` for each of the constraint operands. The constraint coordinator can then decide to perform global or local satisfaction.

If the constraints are ordinary objects, the application programmer could create new constraint classes and new operand classes. The system should automatically generate the event raising behaviour of the objects, input and output ports for communication with Manifold, and the shadow coordinators for constraint objects. The programmer has to provide methods to write data into the output port and to read from the input port that are consistent with those at the other side of the stream, i.e. the stream between a constraint and an operand. This defines an interface between the two. In this way, the state of an object can be completely read and set, but the exact implementation of the object remains hidden. Note however that this state can only be read and set from the Manifold side, not by the other application objects.

8 Implications

We are currently exploring the implications of these two alternatives in terms of functionality, style, and ease of use. One of the implications of the separation of objects and constraints management is that several satisfaction techniques can easily be used in one system. Indeed it may be profitable to use a class of algorithms that can be used to eliminate local (node, arc, and path) inconsistencies [13] before any attempt is made to construct a complete solution. Another possibility is the combination of propagation of degrees of freedom and propagation of known states (also just called local propagation). Propagating degrees of freedom amounts to discarding all parts of the constraint network that can be satisfied easily and solving the rest by some other method. Propagation of degrees of freedom identifies a part in the network with enough degrees of freedom so that it can be changed to satisfy all its constraints. That part and all the constraints that

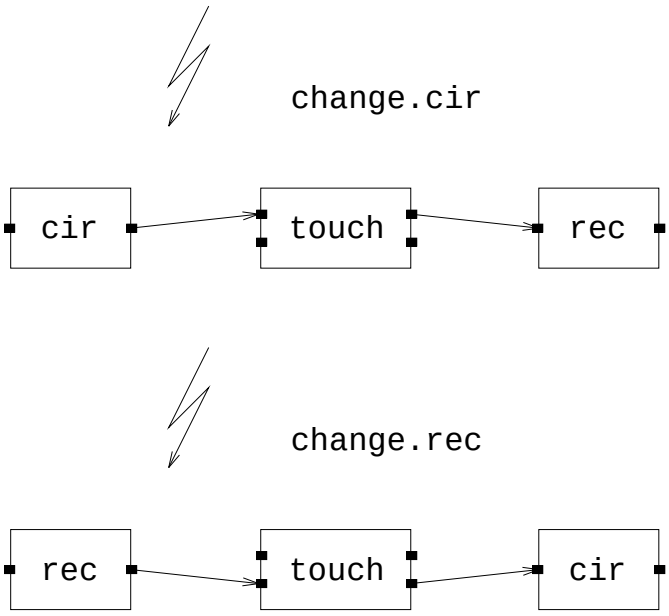

Figure 4: Local propagation networks controlled by `local_touch_coord`.

apply to it are then removed from the network. Deletion of these constraints may give another part enough degrees of freedom so as to satisfy all its constraints. This continues until no more degrees of freedom can be propagated. The part of the network that is left is then satisfied by some global method. The result can now be propagated towards the discarded parts, which are successively satisfied (propagation of known states).

Local propagation is easily coordinated. In our example this could be done in the following way (see figure 4):

```
local_touch_coord(cir,rec,touch)
process cir, rec, touch.
{ event wait.

  start:            do wait.

  change.cir:       (cir->touch.in1, touch.out2->rec).
  change.rec:       (rec->touch.in2, touch.out1->cir).
  satisfied.touch:  do wait.

  wait:             (cir,rec).
}
```

A change of one of the constraint operands results in the raising of an event. This causes `local_touch_coord` to create streams from the altered object to the constraint, and from the constraint to the other object. The constraint is responsible for finding a new solution for the other object.

Some situations allow an even simpler coordination. For example after a local distortion of the constraint, e.g. by a method 'translate' of `cir`. For such methods that make constraints fail, corresponding events (e.g. `translate.cir`) may trigger local propagation similar to the approach in [12]:

```
translate.cir: (cir.out -> rec.in).
```

The above examples are by no means complete, but give a flavour of the type of solution we propose. In our approach we retain strict encapsulation for all modelling of concrete objects. Relationships between objects which cannot logically be ascribed to the internal actions of a container object are expressed in terms of constraints. These constraints may be global but the referential transparency of functional relationships allows one to reason about them and prove their correctness. The proofs of correctness will of course only apply provided the objects, which are regarded as atomic objects from the point of view of constraints, act according to specifications. All modelling of objects with states and behaviour is done in the normal object-oriented framework. In this framework correctness depends (as it always did) on correct program design, using concepts such as modularity and hierarchical decomposition.

One of our next research goals is to model the satisfaction of meta-constraints and higher-order constraints. The strict separation between coordination and functionality of constraint satisfaction provides a way to handle constraints on the satisfaction mechanism (meta-constraints), and constraints on constraints (higher-order constraints).

9 Conclusions

This paper proposes a relevant and important contribution to systems support for interactive computer graphics. This contribution, the combination of constraints and object-oriented methods, has been much heralded but has yet to arrive. We believe that we have identified a major cause for this lack of progress.

The problem is to combine two important approaches to software engineering: object-oriented and declarative programming, in casu constraint programming. The two naturally come together in computer graphics when the behaviour of active objects is partly modelled through constraints. Several approaches to integrate constraints and objects have been taken, see section 5. We have proposed a solution that keeps the object-oriented and constraint programming paradigms distinct and does not compromise the benefits which they severally confer.

The results of our research will lead to better design, analysis, and implementation of interactive graphics systems. The abstraction developed will have immediate application in graphical simulation and visualization as well as graphical user interface management systems. More generally a way of expressing relations between objects, within a robust software engineering based approach, is urgently needed in multimedia applications and other complex interactive graphical applications.

This paper describes a research project in progress. We are currently elaborating and implementing the alternative object and constraint models with the event-based mechanism. Our next research goal is to model the satisfaction of meta-constraints and second-order constraints. This enhances the power of constraint resolution, which alleviates a second reason for the lack of impact of constraints in the object-oriented approach: the lack of powerful and general satisfaction techniques.

The example of the touch-constraint on the rectangle and the circle has been implemented in C++ and Manifold. A complete demo program is available for ftp in the directory ftp.cwi.nl:/pub/remco/EventBasedConstraints.

10 Acknowledgement

We like to thank Richard Kelleners for programming the Manifold demo. This research has been supported in part by NWO (Dutch Organization for Scientific Research) under Grants NF-51/62-514, SION-612-322-212, and SION-612-31-001.

References

[1] F. Arbab. Specification of Manifold. Technical Report CS–9220, CWI, Amsterdam, The Netherlands, 1992.

[2] F. Arbab, I. Herman, and P. Spilling. An Overview of Manifold and its Implementation. *Concurrency: Practice and Experience*, 5(1):23 – 70, February 1993.

[3] E. H. Blake and S. Cook. On Including Part Hierarchies in Object-Oriented Languages, with an Implementation in Smalltalk. In *Proc. ECOOP'87*, volume 276 of *Lect. Notes Comp. Sci.*, pages 41–50. Springer-Verlag, Berlin, 1987.

[4] Edwin H. Blake and Quinton Hoole. Expressing Relationships Between Objects: Problems and Solutions. In *Proceedings of the Third Eurographics Workshop on Object-Oriented Graphics,* Champery, Switzerland, 1992.

[5] Alan Borning. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353 – 387, October 1981.

[6] Bull – Imaging and Office Solutions. *GoPATH 1.2.0 — A Path To Object Oriented Graphics, a public domain environment for graphical and interactive application development*, 1993.

[7] Eric Cournarie and Michel Beaudouin-Lafon. Alien: a Prototype-Based Constraint System. In Laffra et al. [11].

[8] Jacques Davy. Go, A Graphical and Interactive C++ Toolkit for Application Data Presentation and Editing. In *Proceedings 5th Annual Technical Conference on the X Window System*, 1991.

[9] Bjorn N. Freeman-Benson. Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming. *(ECOOP/OOPSLA '90 Proceedings) SIGPLAN Notices*, 25(10):77 – 88, October 1990.

[10] Bjorn N. Freeman-Benson and Alan Borning. Integrating Constraints with an Object-Oriented Language. In O. Lehrmann Madsen, editor, *Proceedings ECOOP'92 – European Conference on Object-Oriented Programming, Utrecht, 1992*, Lecture Notes in Computer Science 615, pages 268 – 286. Springer-Verlag, 1992.

[11] Chris Laffra, Edwin Blake, Vicky de Mey, and Xavier Pintado, editors. *Advances in Object-Oriented Graphics II & III.* Springer-Verlag, 1995.

[12] Chris Laffra and Jan van den Bos. Propagators and Concurrent Constraints. *OOPS Messenger*, 2(2):68 – 72, April 1991.

[13] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99 – 118, 1977.

[14] John R. Rankin. A Graphics Object Oriented Constraint Solver. In Laffra et al. [11].

[15] Ivan E. Sutherland. Sketchpad: A Man-Machine Graphical Communication System. In *Proceedings of the Spring Joint Computer Conference, Detroit, Michigan, May 21-23 1963*, pages 329 – 345. AFIPS Press, 1963.

[16] Remco C. Veltkamp. A Quantum Approach to Geometric Constraint Satisfaction. In Laffra et al. [11].

[17] Michael Wilk. Equate: an Object-Oriented Constraint Solver. In *Proceedings OOPSLA'91*, pages 286 – 298, 1991.