Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

A Model for Strategy in Constraint Solving

J.J. van Wijk

Software Engineering (SEN)

SEN-R9714 August 31, 1997

Report SEN-R9714
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

A Model for Strategy in Constraint Solving

Jarke J. van Wijk[1]
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

The use of constraints for the definition of graphical user interfaces has been recognized as a great concept. However, often many valuations of the variables will satisfy the constraints, and which particular valuation matches best with the expectation of the user cannot be decided without further information. Three typical examples of user interfaces are presented where this occurs, and from these, requirements on a more cooperative constraint solver are derived.

A new method for the definition and implementation of the strategy to decide which variables to adapt is presented. The model is based on two notions: hierarchy and grouping. Variables are divided into groups, and for each group three parameters are set. These are used to determine the level of variables, dependent on which group they belong, and which variables are modified. These levels are used in turn to select the variables to be adapted.

An implementation of this method is described, as part of the Computational Steering Environment (CSE) developed at CWI. The resulting constraint solver can handle simultaneous sets of non-linear, multi-way constraints; and can handle a high-level definition of the strategy to be followed. Finally, the results are discussed, and suggestions for further work are done.

1991 Computing Reviews Classification System: D.2.2, I.3.4
Keywords and Phrases: user interface, constraints, strategy.
Note: Work carried out under CWI project SEN-1.3, Interactive Visualization Environments.

1. INTRODUCTION

The use of constraints in graphical user interfaces has a long history, going back to Ivan Sutherland's SKETCHPAD system [1]. Imperative programming languages, such as Fortran, Pascal, and C, require the developer to specify exactly what must be done at which time. Constraints allow for a declarative style: the developer only specifies constraints between variables, and a constraint solver takes care that these constraints are satisfied by deriving a particular valuation for the variables. For an extensive introduction, see [2]. Hence, the developer can specify a system at a higher level, and concentrate on functional aspects instead of the implementation. This leads to shorter development times and systems that are easier to maintain and modify.

Although many systems for constraint solving have been developed, constraint based methods have not succeeded to replace more conventional methods. A general method that is fast, reliable, and applicable to all kinds of data types and constraints does not exist. Therefore, compromises must be made and the system has to be tuned to the particular application domain.

We consider here the use of constraints for graphical user interface design. A data flow diagram of our view on this is shown in figure 1. Central is a store of variables, which contains the names, attributes, and values of variables. The user can modify the values of variables via a graphical user interface. Before this input is sent to the application, a constraint solver reads these modified values, and adapts values of variables so that a set of constraints, specified by a developer, is satisfied. In this article we use the word *modify* for changes of the values of variables by the user, and *adapt* for changes of values of variables by the constraint solver. Note further that the terms *user* and *developer* denote roles, not persons.

[1] Netherlands Energy Research Foundation ECN, P.O. Box 1, 1755 ZG Petten, The Netherlands.

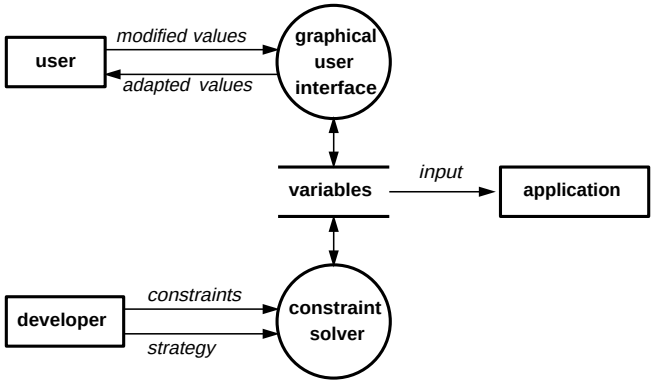

Figure 1: Constraint solver to augment a user interface

The number of constraints will typically be much smaller than the number of variables. This is a typical situation in user interfaces: if all values of variables could be derived from a fixed set of constraints, then we would not need a user interface at all. Hence, the constraint solver must make a decision which particular valuation must be chosen.

The central problem in this article is the selection of a particular valuation for the variables by the constraint solver. Formally, each valuation that satisfies the constraints is acceptable. However, users will typically prefer some valuations above others: they have an intuitive notion about the effect that a change in the value of a variable should have on the values of other variables. As a conceptual solution we propose that the developer must be enabled to specify a strategy to be followed by the constraint solver to make this decision.

We will first discuss some simple examples of applications, derive requirements from these, and review related work. Next, a model for strategy will be presented. The two guiding principles are the notions of grouping and hierarchy. With this model the developer is enabled to specify the strategy to be followed by the constraint solver on a high level. The numerical solution of constraints is discussed separately. The constraint solver is implemented as part of the Computational Steering Environment (CSE), developed at CWI. The description of the implementation and results are finally followed by a discussion and conclusions.

2. PROBLEM

In this section we consider three simple, though real-life examples of situations where we want to use constraints to augment a user interface to an application. Next these examples are analyzed, and requirements for a constraint solving system are derived. In all cases we need facilities to define and implement a strategy for solving the constraints. Common sense heuristics and rules can easily be defined, but how to formalize them ? Finally, related work is reviewed.

2.1 Chain

Suppose we must position a sequence of N points $\mathbf{p}_i$ with coordinates $p_{i,x}$ and $p_{i,y}$. The distances r_i between the succeeding points are given by:

$$r_i \quad = \quad (p_{i+1,x} - p_{i,x})^2 + (p_{i+1,y} - p_{i,y})^2,$$

with $i = 1, \ldots, N-1$. We have $3N-1$ unknowns ($N-1$ distances and $2N$ coordinates) and $N-1$ constraints.

Which particular valuation fits best depends on the meaning of the edges that connect the points. If they are rigid links, the values of r_i must be fixed when positions are modified. If they are used for measuring purposes, the user is free to modify the points, and the distances must be adapted. Another question is if all points have the same status. This could be, but it is also very well possible that the status of outer points differs from the inner

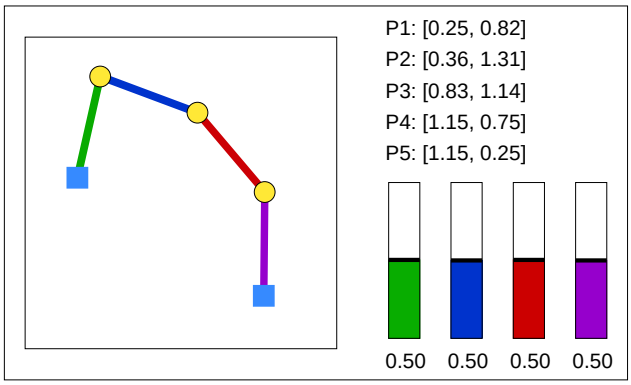

Figure 2: Chain

points. For instance, they could be prescribed externally, and the inner points must be positioned to bridge the gap between them. Figure 2 shows a user interface. We want to enable the user to change values numerically, but also by dragging sliders, points and links. If a link is dragged, four variables are modified simultaneously.

2.2 Budget

Suppose, we must distribute a budget B over two departments P and Q. Each department has three different activities where the budget is spent, namely a, b and c. Figure 3 shows an example of a user interface. A spreadsheet style representation and a hierarchical view are shown. Variables can be changed numerically or by dragging sliders.

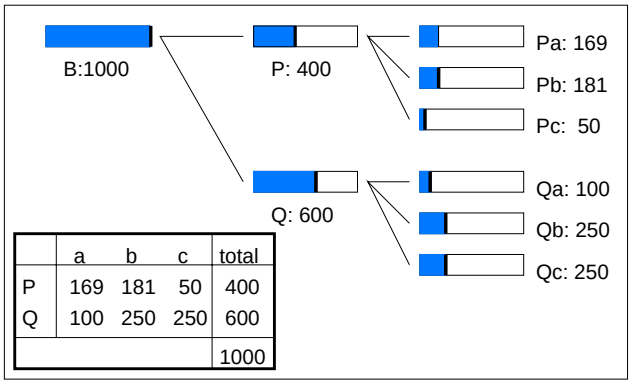

Figure 3: Budget

The constraints that apply here are:

$$\begin{aligned} B &= P + Q\,, \\ P &= P_a + P_b + P_c\,, \\ Q &= Q_a + Q_b + Qc\,. \end{aligned}$$

Here we have nine variables and three constraints: each time a variable is modified, many different valuations will satisfy the constraints. Several different strategies are possible. Suppose we are managing activity Qc and we want more budget. This extra budget could come from:

- other activities within Q: Qa and Qb must be reduced;
- the other department: P, Pa, Pb, Pc must be reduced and Q must be increased;
- the outside world: B and Q must be increased.

The decision which strategy to use cannot be made by just looking at the constraints, additional information is required.

2.3 Hot and Cold

Suppose, we have a simulation of a physical process where somewhere a flow of warm water F with a temperature T must be specified as input. The simulation as such does not care where this water comes from, but we want to show in the user interface that this warm water is a mix of hot and cold water, with flows F_h and F_c and temperatures T_h and T_c.

Figure 4 shows an tentative example of a user interface. The user can specify all variables numerically, change T_h and T_c via the thermometers, change F_h and F_c with the taps, and change T and F simultaneously with the lab tap.

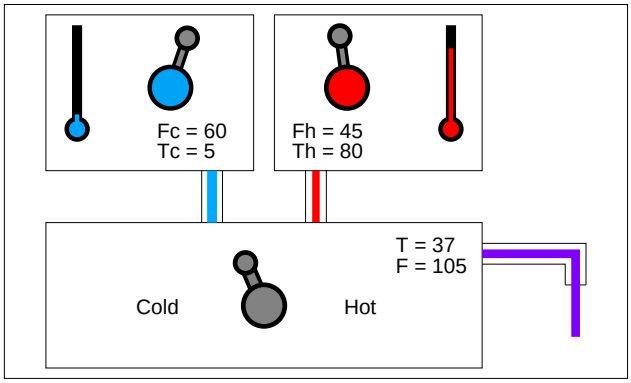

Figure 4: Hot and cold water

The relations (constraints) between the variables follow simply from the laws of mass and energy conservation:

$$\begin{aligned} F &= F_h + F_c \\ TF &= T_h F_h + T_c F_c \end{aligned}$$

We want to enable the user to modify all variables. We have six variables and two constraints, hence a severely underconstrained system. Without further information it is impossible for the constraint solver to select a reasonable valuation. The rules to be followed are:

- If F or T are modified (the lab-tap is used), then F_h and F_c must be adapted.
- If F_h or F_c are modified (a traditional tap is used), F and T must be adapted.
- T_h and T_c are presets, which cannot be changed easily. If they are changed, then the constraint solver must keep the values of F and T fixed, and adapt F_h and F_c to satisfy the constraints.

2.4 Requirements

If we want to provide a developer with a constraint solver that enables him to implement the previous examples easily, what are the requirements on this constraint solver ? First, consider the constraints themselves. The developer must be enabled to specify them directly to the solver. This implies:

- Multiple constraints must be maintained simultaneously, where the system takes care of valuation;
- Constraints are multi-way (e.g. in the constraint $B = P + Q$ both B, P, and Q may change);
- Constraints are multi-dependent (i.e. in one constraint a set of variables depends on another set of variables: $TF = T_h F_h + T_c F_c$);
- Constraints can contain cyclic dependencies of variables (see the equations for the Hot and Cold example);
- Constraints can be non-linear (see the Hot and Cold, and the Chain example).

To keep our task within reasonable limits however, we make three simplifying assumptions:

- All constraints are equalities;
- All variables are real numbers;
- The number of variables and constraints is small (say, less than 50), so that performance issues can be neglected for the time being.

The examples given fit within these last assumptions, although it is easy to extend them or to define other examples that do violate them.

We have seen in the examples that solving the constraints is not enough: to find good, reasonable, expected, intuitively right solutions the developer must be enabled to specify a strategy, and the solver must be able to find a solution according to this strategy. We can discern two steps in this process. Consider the constraint $z = x + y$, and suppose that z is modified. The first step is to select the variables to adapt. If both x and y are selected, in the second step the change in z must be distributed over x and y. The first step we call the *selection* step, the second step the *distribution* step. For both steps the developer must be enabled to specify a strategy.

One solution could be to specify the strategy imperatively: via a standard programming language or via some dedicated mini-language. However, this is in conflict with the functional spirit of constraints: we want to specify what the strategy is, not how it is implemented. From the examples more specific requirements and directions for solutions can be derived.

First, the number of variables is larger than the number of constraints, so in most cases all constraints can be satisfied. Also, if the constraints are defined for a user interface in such a way that no valuation exists that satisfies them all, then this is presumably an error in the definition. Further, all constraints have a law-of-nature character. If a constraint (such as the mass-balance, the calculation of distances) is violated, then the values of variables make no sense. Therefore, the strategic decision to be made is not which constraints must be satisfied and which not, but which variables must be adapted and which not. Hence, the strategy should be specified based on variables.

Second, two concepts must be supported: *hierarchy* and *grouping*. Concerning hierarchy: some variables are more important and should be adapted less quickly than other variables. The heat of the incoming hot water (T_h) should only be adapted if no other solutions can be found. Also, we typically can distinguish groups of variables. If we modify F and T directly, then F_h and F_c must be adapted; if we modify F_h and F_c, F and T must be adapted. In other words, we distinguish two groups of variables here, and which variables are to be adapted depends on which group is modified. Note that this cannot be specified via hierarchy alone.

Third, if the result of the selection step is that more variables are selected then needed, we require that the changes in the variables are evenly distributed, i.e. minimal in a least-squares sense.

2.5 Related Work

The problem of deciding which variables to change has been recognized earlier, but only few solutions are given. According to Leler [2], the most popular method in systems based on local propagation is to pick some variables to change, even at random, and to allow the user to complain if a "wrong" choice was made. Other strategies are to let the developer specify in advance which variables have to be changed, to pick the variables that cause the least change, or to pick the variables that are closest to the change. None of these methods enables the developer to specify a strategy in advance.

In the nineties the theory of constraint hierarchies was developed by the group of Borning at the University of Washington [3]. This concept was used for the SkyBlue constraint solver and the Multi-Garnet user interface development system [4]. A constraint hierarchy is a set of constraints, each labeled with a *strength*, indicating how important it is to satisfy each constraint. Given an overconstrained set of constraints, weaker constraints can be left unsatisfied. If the hierarchy is underconstrained, the solver can choose any of the possible solutions. The user can control which solution is chosen by adding weak *stay* constraints to specify variables whose value should not be changed.

The use of constraint hierarchies is much more powerful than the previous (non-)solutions. Furthermore, highly efficient algorithms for the handling of such constraint hierarchies have been developed. On the other hand, constraint hierarchies are not powerful enough. For instance, the simple Hot and Cold problem just described cannot be handled with a fixed set of strengths on constraints and/or variables.

Standard computer graphics constraint methods typically use optimization methods to deal with underconstrained problems. Often L_2 norms are used, in other words, the sum of squared differences is minimized. The use of an L_∞ norm implies that the number of variables adapted is minimized. For the problems presented here those methods applied to all variables simultaneously fall short. Both the notion of grouping (if x has to be adapted, then y may be adapted as well) and hierarchy (z should not be adapted if we can satisfy the constraints by adapting x) are not supported.

3. STRATEGY

In this section a new method for handling strategy in constraint solving is presented. First a model is presented, next an algorithm to find a solution. Here only the selection of variables to be adapted is discussed, in the next section the numerical solution of constraints is discussed. The use of the new method is illustrated with the previous examples.

3.1 Model

Without loss of generality, let $\mathbf{x}$ be a vector of n variables $x_i \in \Re$, with $i = 1, \ldots, n$, and suppose that m constraints are defined on these variables:

$$C_j(\mathbf{x}) = 0,\ j = 1, \ldots, m.$$

We define the set $\mathcal{M}$ of modified variables as:

$$\mathcal{M} = \{i \mid x_i \text{ is modified}\}.$$

The developer divides the variables into groups $\mathcal{G}_k, k = 1, \ldots, p$. These groups $\mathcal{G}_k$ must be a partition of $\{1, \ldots, n\}$, i.e.

$$\begin{aligned} \bigcup \mathcal{G}_k &= \{1, \ldots, n\}, \\ \mathcal{G}_k &\neq \emptyset, \\ i \neq j &\Rightarrow \mathcal{G}_i \cap \mathcal{G}_j = \emptyset. \end{aligned}$$

Central in this model is the concept of a level. A high level of a variable means that the value of the variable should be kept. The developer does not assign levels to variables, but to groups. Each group has three associated level values:

- l_k: the standard level of a group;

- d_k: the difference in level for modified variables;
- g_k: the difference in level for unmodified variables in modified groups.

The levels of variables are derived from these values. Suppose that $i \in \mathcal{G}_k$. The level v_i of a variable is given by:

$$v_i = \begin{cases} l_k + d_k & \text{if } i \in \mathcal{M}; \\ l_k & \text{if } \mathcal{G}_k \cap \mathcal{M} = \emptyset; \\ l_k + g_k & \text{otherwise.} \end{cases}$$

We distinguish three types of variables here. If a variable is modified, its level is the group level plus an offset d_k. Typically this offset will be positive: the values of modified variables must be kept. If a variable is not modified, nor is one of the variables in the same group, its level is the group level. Finally, if the variable is not modified, but one or more of the variables in the same group is, the level is the group level plus an offset g_k. If this offset is positive, then in principle the constraints must be solved using variables in other groups, otherwise the variables in the same group are more likely to be adapted.

The developer defines the strategy via the groups and the levels per group. The task of the constraint solver now is to find the minimal value for l_{min} such that:

$$\begin{aligned} C_j(\mathbf{x}') = 0, &\quad j = 1, \ldots m, \\ x'_i = x_i, &\quad \text{if } v_i > l_{min} \end{aligned}$$

where primes denote adapted values. In other words, all variables x_i with $v_i <= l_{min}$ are adapted such that the constraints are satisfied, and this set of adapted variables must be as small as possible.

3.2 Algorithm

The minimum level l_{min} can be determined with a straightforward algorithm. Let $\mathcal{C}$ be the set of active constraints, i.e. constraints that are considered in the solution process. Let $\mathcal{X}$ be the set of active variables, i.e. variables whose values can be adapted. The set of variables $\mathcal{V}_j$ that is used in a constraint C_j is defined as:

$$\begin{aligned} \mathcal{V}_j \quad = \quad & \{i \mid \exists x_i : C_j(\mathbf{x}) \neq C_j(\mathbf{y}) \ \wedge \\ & \quad k \neq i \Rightarrow x_k = y_k\}. \end{aligned}$$

The algorithm proceeds as follows.

1. Initialize the set of active constraints to all constraints that are not satisfied:

$$\mathcal{C} \leftarrow \{j \mid C_j(\mathbf{x}) \neq 0\}.$$

2. Initialize the minimum level:

$$l_{min} \leftarrow \max_{j \in \mathcal{C}}(\min_{i \in \mathcal{V}_j}(l_i)).$$

Each constraint must have at least one active variable, hence for each active constraint we search for the variable with the lowest level. The initial value of the minimum level is the highest of these.

3. Determine the active variables:

$$\mathcal{X} \leftarrow \{i \mid v_i \leq l_{min}\}.$$

4. Determine the active constraints. These are all constraints that use active variables:

$$\mathcal{C} \leftarrow \{j \mid \mathcal{V}_j \cap \mathcal{X} \neq \emptyset\}.$$

5. Try to find a valuation for the variables x_i, $i \in \mathcal{X}$, such that the constraints $C_j(\mathbf{x}) = 0$, $j \in \mathcal{C}$, are satisfied. If a solution was found, **stop**. The problem has been reduced to a numerical one, which will be elaborated further in the next section.

6. Increase l_{min} such that at least one new variable becomes active:

$$l_{min} \leftarrow \min(v_i \mid i \notin \mathcal{X} \wedge i \in \mathcal{V}_j \wedge j \in \mathcal{C}),$$

i.e. search for the non-active variable with the lowest level that is used in an active constraint.

7. Go to step 3.

This algorithm can be accelerated in several ways. Incremental techniques to update the sets can be used, in combination with additional data-structures to store references and indices. Further, depending on the type of constraints additional heuristics can be used. For instance, if all constraints are equalities, linear and independent, then the number of active variables must be greater than or equal to the number of active constraints. In step 3 all variables with levels below l_{min} are made active. This is often unnecessary. For instance, if we have two constraints

$$\begin{aligned} a &= x + y; \\ b &= u + v; \end{aligned}$$

and the first is satisfied and the second is not, then the variables a, x and y do not have to become active, regardless of their levels. An improved version of step 3 is:

3.a Initialize $\mathcal{X}$ to all variables with $v_i \leq l_{min}$ that are used in invalid constraints:

$$\begin{aligned} \mathcal{X} \quad \leftarrow \quad & \{i \mid v_i \leq l_{min} \wedge \\ & \exists k : C_k(\mathbf{x}) \neq 0 \wedge i \in \mathcal{V}_k\} \end{aligned}$$

3.b Make additional variables active, such that for each constraint with active variables all variables with $v_i \leq l_{min}$ are made active:

$$\begin{aligned} \mathcal{X}' \quad \leftarrow \quad & \{i \mid v_i \leq l_{min} \wedge \\ & \exists k : \mathcal{V}_k \cap \mathcal{X} \neq \emptyset \wedge i \in \mathcal{V}_k\} \end{aligned}$$

3.c If new variables were added (i.e. $\mathcal{X}' \neq \mathcal{X}$):

$$\mathcal{X} \leftarrow \mathcal{X}',$$

and go back to step 3.b.

However, if the number of constraints is small, a straightforward implementation suffices. Note further that in the model and in the algorithm no assumption is made yet about the type of constraints and variables: the model can also be applied for instance for inequality constraints and integer variables.

3.3 Examples

The strategy to be followed is fully defined by the grouping of variables, and the assignment of values to the levels for each group. We will illustrate this via the examples presented before. The notation used is:

$$\{x, y, \ldots\} : l, d, g;$$

where x and y are variables in a group, followed by the level parameters.

We can now define the strategy for the chain example as:

$$\begin{aligned} \{r_1, \ldots, r_N\} : &\quad 30, 1, 0; \\ \{p_{1,x}, p_{1,y}, p_{N,x}, p_{N,y}\} : &\quad 20, 1, 0; \\ \{p_{2,x}, p_{2,y}, \ldots, p_{N-1,x}, p_{N-1,y}\} : &\quad 10, 1, 0. \end{aligned}$$

We express here that the distances have the highest level, followed by the end-points, and with the interior points on the last place. Hence, if the value of r_1 is modified, then the positions of interior points will be adapted. If we modify the position of interior point $\mathbf{p}_2$, then the constraint on r_1 cannot be satisfied by repositioning the other interior points (which have the lowest level: 10). Therefore, the level l_{min} has to be increased to 11, the level of $\mathbf{p}_2$, and $\mathbf{p}_2$ itself is adapted. The net effect is that the user can drag $\mathbf{p}_2$, but only over a circle of radius r_1 around $\mathbf{p}_1$.

For the budget case we define the strategy as follows:

$$\begin{aligned} \{B\} : &\quad 30, 1, 0; \\ \{P, Q\} : &\quad 20, 1, 0; \\ \{P_a, P_b, P_c\} : &\quad \alpha, 1, \beta; \\ \{Q_a, Q_b, Q_c\} : &\quad \alpha, 1, \beta. \end{aligned}$$

Again, we consider what must be done if Q_c is modified. We have discussed that different strategies are possible. The model for strategy enables them to be parametrized, so that at run-time the strategy can be changed. First, if we use $\alpha = 10$ and $\beta = -1$, then Q_a and Q_b have the lowest level (9) and will be adapted. Second, if we use $\alpha = 25$ and $\beta = 1$, then P and Q (level 20), P_a, P_b, and P_c (all level 25) will be adapted. Finally, if we set $\alpha = 40$ and $\beta = 1$, then B and Q will be adapted, since they have the lowest levels in the invalidated constraints. These different strategies are depicted in figure 5.

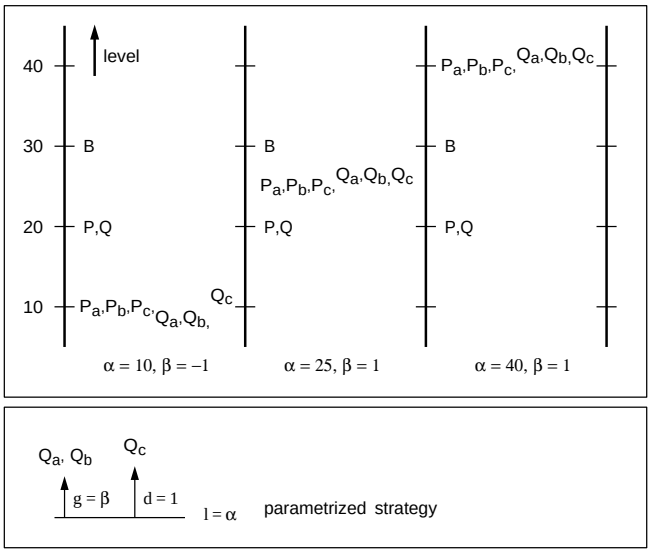

Figure 5: Levels of variables when Q_c is modified

A definition of the strategy to be applied for the Hot and Cold water example is:

$$\{T_h, T_c\} : \quad 100, 1, 0;$$

$$\begin{aligned} \{F, T\}: &\quad 51, 10, 5; \\ \{F_h, F_c\}: &\quad 50, 10, 5. \end{aligned}$$

If for instance F_h is modified, then the level of F_c becomes 55, while the levels of F and T stay 51. Hence, F and T are selected to be adapted. If F is modified, the variables in the other group will be adapted. This situation is a good example of a case that cannot be handled with a straightforward scheme, using only static levels or strengths per constraint or variable. Finally, if T_c is modified, then F_h and F_c will be adapted. Their group level (50) is slightly lower than that of F and T.

It might seem that much input has to be specified to define the strategy. However, if we suppose that 1 is the default value for d_k and 0 is the default for g_k, then in many cases it suffices to group the variables and to specify l_k. The meaning of grouping and l_k are easy to understand. In more demanding situations, more complex behavior can be specified via the d_k's and g_k's.

4. NUMERICAL SOLUTION

In the preceding section we left the actual solution of the constraints as a separate problem. We now discuss how to find valuations for the active variables x_i, $i \in \mathcal{X}$, such that the constraints $C_j(\mathbf{x}) = 0$, $j \in \mathcal{C}$, are satisfied, and that the change in the variables is minimal. The latter we define as:

$$\text{minimize} f(\mathbf{x}'_i) = \sum_{i \in \mathcal{X}} \left(\frac{x'_i - x_i}{s_i} \right)^2,$$

where s_i is a scale-factor per variable, to be defined by the developer. The scale-factors s_i denote the unit amount of change of each variable. If all variables have the same meaning (say, position) then the scale-factors can all be set to 1, but if for instance temperatures must be compared to pressures, additional information is required to get meaningful results.

For the solution of the problem we use some standard methods. The overall scheme is similar to the Method of Approximation Programming of Griffith and Stewart [5]: we reduce the non-linear problem to a sequence of quadratic programming steps. To this end, first the non-linear constraints are reduced to linear ones: the functions C_j are expanded about the initial point $\mathbf{x}$ in a Taylor series which is truncated after the linear term:

$$C_j(\mathbf{x}') \approx C_j(\mathbf{x}) + \nabla C_j(\mathbf{x})(\mathbf{x}' - \mathbf{x}).$$

The function to be minimized can be formulated as:

$$\begin{aligned} g(\mathbf{x}') &= \mathbf{x}'^T A \mathbf{x}' - 2\mathbf{b}^T \mathbf{x}', \text{ with} \\ A &= \text{diag}(1/s_1^2, \ldots, 1/s_n^2), \text{ and} \\ \mathbf{b} &= A\mathbf{x}. \end{aligned}$$

We use a method of Fletcher [6] to solve this standard quadratic programming problem. This gives a new value for $\mathbf{x}'$. Next, the procedure is repeated with this new value as the initial trial point, until the constraints are satisfied within a certain tolerance.

5. IMPLEMENTATION

The described methods are implemented as part of the Computational Steering Environment (CSE), developed at CWI. The aim in the development of the CSE [7] is to provide researchers with an environment that enables them to develop and use graphical user interfaces to running simulations, so that they can get more insight in their models via computational steering.

5.1 CSE

Central in the architecture of the CSE is the Data Manager: an active data-base that maintains a list of variables. Their attributes (name, type, size) and values are stored. Other processes, called satellites, can connect to the Data Manager, and define new variables, and read and write values. Further, satellites can request to receive event messages from the Data Manager when changes in the attributes or values of variables occur.

A simple application programmers interface is available for the connection of simulations to the Data Manager. Often an initial sequence of declarative calls and an update call in the main loop suffices. Standard satellites are available to log, transform, calculate and to perform other standard operations on the values of variables.

For graphical input and output also a general tool has been developed: the PGO-editor, where PGO stands for Parametrized Graphics Object. With this tool the user can design a user interface in a MacDraw-like way. Graphics objects, such as circles, text, rectangles, lines, etc., are defined by entering a sequence of points. The position of these points, as well as other attributes like color and linewidth, can be parametrized to variables in the Data Manager. This is done by assigning Degrees of Freedom (DOFs) to points. Each DOF has an associated variable, and a minimum and maximum position. For each variable a linear mapping can be specified to map these geometric minima and maxima to bounds of the variable. In run-mode the PGO-editor takes care of maintaining a one-to-one mapping of the data in the Data Manager to the DOFs: If a value changes, the associated positions and other graphic attributes are updated; If the user drags an object, the associated values are modified. Both a 2D and a 3D version [8] of this tool have been developed.

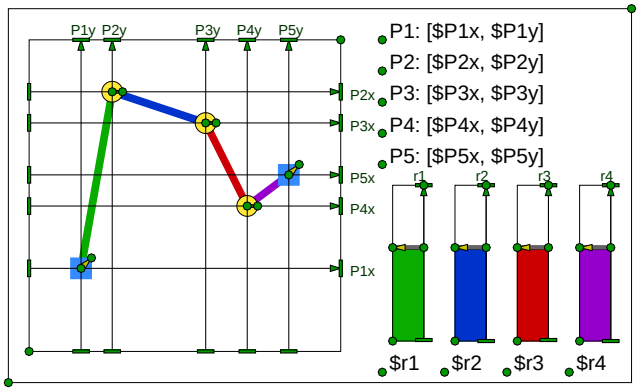

Figure 6: Chain, edit-mode

All examples shown are running applications, developed with the CSE. Figure 6 shows the chain example in edit-mode. The centers of the circles and squares are parametrized, as well as the rectangles used in the sliders. Further, in labels references are made to variables by prefixing them with a dollar-sign. This drawing can be considered as a functional specification of a graphical user interface. In run-mode, interaction with the graphics is translated into modifications of values. Dragging the first blue square induces modification of $P_{1,x}$ and $P_{1,y}$, dragging the green link induces modification of $P_{1,x}$, $P_{1,y}$, $P_{2,x}$, and $P_{2,y}$. Text items can be clicked on, upon which new numerical values can be entered. Before the screen is repainted after graphical interaction, the values of variables are updated by the constraint solver, which is yet another satellite.

With this set-up, running on an SGI Indy workstation, the user can drag the objects smoothly in real-time, while simultaneously the constraints are maintained. Furthermore, each satellite can be switched independently from edit-mode to run-mode. This enables developers to build user interfaces and to define constraints interactively. Each example shown in took less than an hour to implement. And most important, the end-user (a researcher) is enabled to define and adapt the user interface himself: the user and the developer are the same person.

5.2 Constraint Satellite

The constraint solver is implemented as a general purpose satellite within the CSE. The developer enters constraints as equations, using the notation of standard programming languages. Mathematical functions (sin, log, sqrt, etc.) are supported. The strategy is entered via the same notation as described in the section on strategy. If the values for d and/or g are omitted, 1 and 0 are used respectively. Instead of values for l, d, and g, also variable names can be specified. If these variables are bound to PGOs, the user is enabled to change the strategy at

run-time.

When the constraint solver is switched from edit-mode to run-mode, first the constraints and the strategy are interpreted. The constraints are converted into parse trees. Also, parse trees for the computation of partial derivatives are derived by symbolic differentiation. Synchronization of the constraint solver and the PGO-editor is done via trigger variables. Often several variables are modified simultaneously (by dragging a link, for instance). After a graphical interaction first the variables are modified, and next a trigger variable is changed. This change is caught by the constraint solver, which reads the new values, checks the constraints and finds a new valuation, if necessary. Next the constraint solver changes another trigger variable, which signals to the PGO-editor that the image has to be repainted.

6. DISCUSSION

The described system meets the requirements. Multiple, multi-way, multi-dependent, cyclic, non-linear constraints are maintained simultaneously, where the constraint solver takes care of valuation. The user can specify the strategy to be followed (i.e. which variables to adapt) in a compact and declarative style. The strategy is defined in natural concepts for the user, i.e.:

- which groups of variables can be distinguished;
- how important is each group;
- if a variable is changed, how should its importance be changed;
- if a variable is changed, what does this mean for the importance of other variables in the same group ?

We have shown how this information can be used by a constraint solver to adapt values in an expected, natural, and intuitive way.

The model for strategy presented here is more powerful than the concept of constraint hierarchies. In constraint hierarchies the notions of grouping and context-dependency (i.e. is a variable modified or not) for levels are not embedded, and hence for many cases it will not be possible to define the strategy according to the expectations of the user.

6.1 Further Work

Nevertheless, further work still has to be done. We made some simplifying assumptions, which are often valid within our type of applications, but in general too limiting. First, a more general system must be able to handle inequalities. This is not a serious problem. They can be handled with the method described by Fletcher [6], but this has not been implemented yet.

The handling of constraints on integer, boolean, and string variables is much more complex. The model for strategy however, is independent of the types of constraints and variables, so the handling of mixed type constraints is a separate problem.

The methods presented have a brute force character, which is only justified if the number of variables and constraints is small. If constraint solving is used to augment a user interface, this assumption is often valid; if a complete system is defined in terms of constraints, it is not. With our system we found in practice that an application with 25 constraints still allows for real-time interaction. We have already given suggestions how the selection of variables and constraints can be speeded up further. Many techniques for optimization in general, and quadratic programming in particular have been developed. Probably more efficient methods can be used for this. Another approach could be to augment an existing efficient constraint solver with the method described here to define strategies. Specifically, the strengths used in the SkyBlue solver could be derived dynamically from a specification of the strategy as presented here.

Numerical solvers for nonlinear simultaneous equations are notorious for being unstable: if no reasonable initial guess is provided, no solution can be found. The solver described here is no exception. Fortunately, when the user changes values by dragging, the steps are often small, so the old values are close to the new solution sought.

We assumed here that the problems are underconstrained: many valuations exist that satisfy the constraints. If this assumption is invalid, the user must be enabled to indicate which constraints are more important than others. This is not yet included in our model, but could be added. An open question is if a hierarchy alone is enough, or that grouping etc. is required here also.

The model for strategy presented here does not allow us to specify each kind of strategy. An extreme, exhaustive model that does allow this, is to require from the user to specify for each possible subset of $\mathcal{X}$ the levels of all variables. It is clear that such a model is not usable in practice. A less extreme solution would be to relax the requirement that the groups $\mathcal{G}_k$ are a partition of $\{1, \ldots, n\}$. These groups could be defined hierarchically, or even arbitrary. However, this would add complexity: when a variable is an element of several groups, some modified, some not, additional rules are required to determine the level of the variable. We regard the model for strategy presented here as an optimal compromise between flexibility on one hand and ease of use on the other hand.

ACKNOWLEDGEMENTS

This work is supported by the Netherlands Computer Science Research Foundation (SION), with financial support of the Netherlands Organization for Scientific Research (NWO).

REFERENCES

1. Sutherland, I., SKETCHPAD: A Man-Machine Graphical Communication System. IFIPS *Proceedings of the Spring Joint Computer Conference*, January 1963.
2. Leler, Wm, Constraint Programming Languages Their Specification and Generation. Addison-Wesley, Reading MA, 1988.
3. Borning, A., B. Freeman-Benson, and M. Wilson, Constraint Hierarchies. *Lisp and Symbolic Computation*, 5(3), pp. 223–270, September 1992.
4. Sanella, M., Constraint Satisfaction and Debugging for Interactive User Interfaces. PhD-Thesis, Technical Report 94-09-10, Dept. of Computer Science and Engineering, University of Washington, September 1994.
5. Griffith, R.E. and R.A. Stewart, A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems. *Management Science*, 7, pp. 379–392, 1961.
6. Fletcher, R., A General Quadratic Programming Algorithm. *J. Inst. Maths. Applics.*, 7, pp. 76–91, 1971.
7. Wijk, J.J. van, and R. van Liere, An Environment for Computational Steering. Presented at the Dagstuhl Seminar on Scientific Visualization, 23-27 May 1994, Germany, Proceedings to be published.
8. Mulder, J. and J.J. van Wijk, 3D Computational Steering with Parametrized Geometric Objects. In: Nielson, G.M. and D. Silver (eds.), *Proceedings IEEE Visualization'95*, CS Press, pp. 304–311, October 1995.