One Size Fits None – A User Interface for Constraint-Based Design

Remco Niemeijer
Eindhoven University of Technology

Bauke de Vries
Eindhoven University of Technology

Jakob Beetz
Eindhoven University of Technology

ABSTRACT

Flexible mass customization of buildings is still in its infancy. Current systems for the automated support of owner-driven configuration management are limited with regard to the degree of freedom they offer to end-users, due to the lack of an easy way to specify those freedoms. In this paper we present the prototype of an interface that allows architects to define constraints to which user-customized dwellings must conform.

1 INTRODUCTION AND RELATED RESEARCH

People who buy a house typically have little, if any, input into the design. Because changing the technical drawings is a laborious and error-prone process, no more than a few alternatives are typically offered. By using constraints, however, changes can be made while guaranteeing the new design is still valid. This enables the client to make changes to the design without incurring significant additional costs.

The idea of automatically checking building, while common in the mechanical engineering industry (Gross 1996; Bettig and Shah 2003), has yet to be widely adopted in the building industry. The only major commercial CAD package to feature enforceable rules so far is Revit (Strömberg 2006). Projects that use such rules in practice, such as SMARTcodes (Wix, Nisbet and Liebich 2008) exist, but are few and far between. When such rules are used, they are frequently only applied to the geometry and not to non-physical aspects such as materials, costs or thermal isolation (Eggink, Gross and Do 2001; Nassara, Belblidia and Alby 2003; Thabeth and Beliveau 2003).

2 SYSTEM DESCRIPTION

We propose a system where the buyer of a house can change the architect's design before the house is built, to achieve a better match with their needs. The traditional way of doing this, where each client would have to separately confer with the architect face-to-face, works only for small projects and does not scale up well to the more typical large housing projects. In our approach, the architect, after creating the initial design, defines a set of constraints (Kelleners 1999; Donath and Böhme 2007). After the initial design and the rules are completed, the buyer can modify the design within the rules of the architect.

3 CONSTRAINTS

We define a constraint to be an assertion about building elements. It is effectively a function that takes as its argument a list of building elements and returns a Boolean, indicating whether all the elements satisfy the constraint. Unfortunately, constraints like this only work for requirements that can be objectively judged. Rules about aesthetical quality, for instance, will still have to be checked by humans. In our system we opt for constraint checking instead of constraint solving, since the solution space is far too large to effectively solve the constraints automatically and because this process does not give the user enough input.

At some point, the constraints will have to be entered into the system. This action will most commonly be performed by architects, as most of the building codes and law constraints will only have to be entered once. The goal, therefore, is to find a method of constraint entry that is easy for architects to work with. There are several alternatives, shown in Table 1:

Table 1

Input type	Advantages	Disadvantages
Programming language e.g. Java, Python, Haskell	Powerful Easy implementation	Difficult syntax Requires training
Domain-Specific Language	Easier syntax than programming language	Still requires training
Natural language	No training required Very easy to use	Very difficult to implement
Tree-based input, e.g. Yahoo Pipes	Graphical User Interface	Difficult to read
Block input, e.g. Lego Mindstorms NXT	Fairly easy to read	Not as compact to display as a language-based solution

We have chosen the block input, or "puzzle piece," option since we believe it offers the most advantages and the least disadvantages. One change we make, however, is that instead of representing every constraint as a single sequence of pieces, they are split up into four different sections. This is because initial paper-only tests revealed that all but the shortest sentences would quickly become very hard to understand. We have opted to split constraints into elements, definitions, conditions, and rules. The elements section defines which elements the constraint applies to. The definitions section allows creating definitions similar to those found in legal documents. While most useful if a concept is referred to more than once, they can also help to split up long sentences. Conditions are a further filtering of the elements the constraint applies to. Finally, rules determine the rules the filtered elements must obey. Table 2 shows two example translations, taken from the Dutch building codes for dormers.

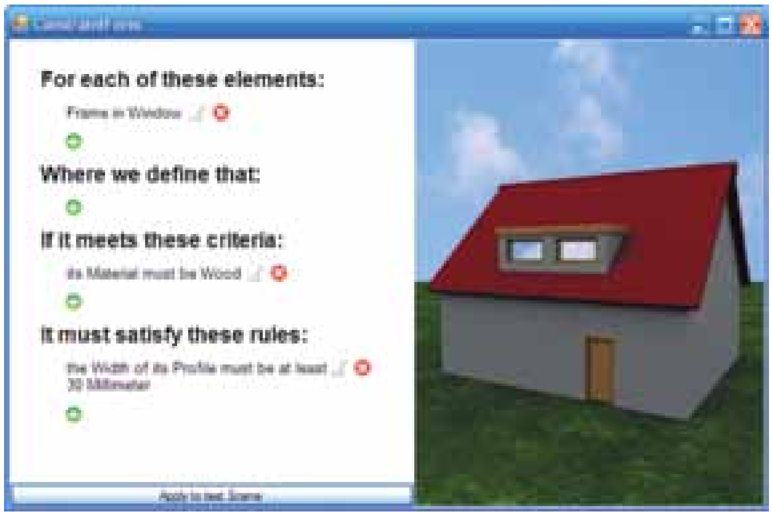
Table 2
Example translations

Original constraint	Translated constraint
The material of walls of dormers must be wood sheet, wood or zinc	Element: Walls in Dormers Definition: m is its material Condition: - Rules: m must be wood or m must be wood sheet or m must be zinc
Half-span roofs with an angle of less than 30 degrees cannot contain dormers	Element: Dormers Definitions: - Conditions: The type of its roof must be half-span roof Rules: The angle of its roof must be more than 30 degrees

4 PROTOTYPE DESCRIPTIONS

We developed a working prototype based on the approach described above, to test whether or not it works in practice. Figure 1 shows the main screen of the prototype, where the four sections of a constraint can be edited. In these four sections, rules can be added, edited and deleted. Creating or editing a rule brings up the puzzle sequence editor, shown in figure 2. In the puzzle sequence editor, the left side of the screen contains the "library" of available pieces. These can be dragged over to the work area on the right, where they are assembled into constraints. When a puzzle piece is placed, the input is checked by a grammar to see which pieces can legally follow it. The puzzle piece library on the left is updated to show only those pieces. As a result, the user sees only useful pieces, which reduces information overload and makes it easy to see if a mistake has been made: if the desired piece is not listed there must be an error in the sentence. When the constraints are checked, the sequences of puzzle pieces are converted to functions that are applied to every building element of the house (since the filtering takes place in the functions) and any violations are then reported to the user.

Figure 1
Prototype main screen



For each of these elements:

Frame in Window

Where we define that:

If it meets these criteria:

its Material must be Wood

It must satisfy these rules:

the Width of its Profile must be at least 30 Millimeter

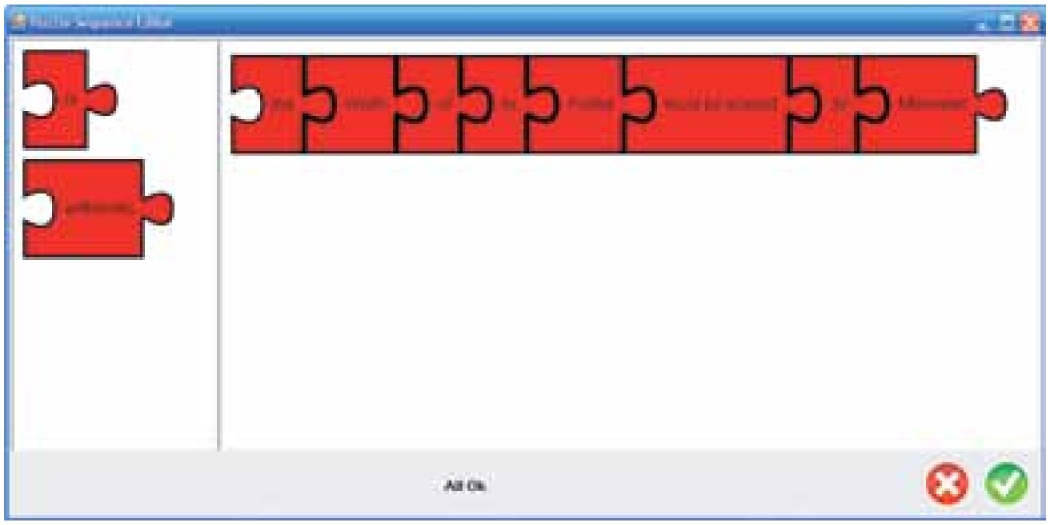


Figure 2
Puzzle sequence edit window

5 DISCUSSION

Although the chosen puzzle piece approach seems to be reasonably successful in allowing people to define constraints, it is not a very fast method. Having to drag all the pieces to the correct location is a rather laborious process. Having said that, the task load should decrease in time, because many constraints will recur in different projects. The architect can build up a library of constraints for many different building elements that can be re-used after minor adaptation in other projects. As for the system as a whole, it will most likely not replace any existing methods. It will instead become an additional option for the architect. Some architects make one design, some offer several, and in the future some architects will allow clients to make decisions of their own.

REFERENCES

BELBLIDIA, S., E. ALBY. 2003. IMPLICIT HANDLING OF GEOMETRIC RELATIONS IN AN EXISTING MODELER. PAPER PRESENTED AT THE MEETING OF THE THE EIGHTH CONFERENCE ON COMPUTER-AIDED ARCHITECTURAL DESIGN RESEARCH IN ASIA.

BETTIG, B., J. SHAH. 2003. SOLUTION SELECTORS: A USER-ORIENTED ANSWER TO THE MULTIPLE SOLUTION PROBLEM IN CONSTRAINT SOLVING.. JOURNAL OF MECHANICAL DESIGN 125 (3): 443–451.

DONATH, D., L. BÖHME. 2007. CONSTRAINT-BASED DESIGN IN PARTICIPATORY HOUSING PLANNING. PAPER PRESENTED AT THE MEETING OF THE PREDICTING THE FUTURE.

EGGINK, D., M. GROSS, E. DO. 2001. SMART OBJECTS: CONSTRAINTS AND BEHAVIORS IN A 3D DESIGN ENVIRONMENT. PAPER PRESENTED AT THE MEETING OF THE ARCHITECTURAL INFORMATION MANAGEMENT.

GROSS, M. 1996. ELEMENTS THAT FOLLOW YOUR RULES: CONSTRAINT BASED CAD LAYOUT. PAPER PRESENTED AT THE MEETING OF THE DESIGN COMPUTATION: COLLABORATION, REASONING, PEDAGOGY.

KELLENERS, R. 1999. CONSTRAINTS IN OBJECT-ORIENTED GRAPHICS. PHD DISS., EINDHOVEN UNIVERSITY OF TECHNOLOGY.

NASSARA, K., W. THABETB, Y. BELIVEAU 2003. BUILDING ASSEMBLY DETAILING USING CONSTRAINT-BASED MODELING. AUTOMATION IN CONSTRUCTION 12: 365–379.

WIX, J., N. NISBET, T. LIEBICH. 2008. USING CONSTRAINTS TO VALIDATE AND CHECK BUILDING INFORMATION MODELS. PAPER PRESENTED AT THE MEETING OF THE PROCEEDINGS OF THE 7TH EUROPEAN CONFERENCE ON PRODUCT AND PROCESS MODELING.