

# **Designing with constraints**

*Towards mass customization in the housing industry*

R.A. NIEMEIJER, B. DE VRIES, J. BEETZ

*Eindhoven University of Technology*

*Den Dolech 2*

*5612AZ Eindhoven*

*The Netherlands*

*r.a.niemeijer@tue.nl*

**Key words:** user-oriented architectural design, CAD, mass customization, constraints, natural language parsing

**Abstract:** Mass customization, while common in other industries, has yet to find widespread adoption in the housing industry. Current methods of mass customization are either labour-intensive or allow only a limited degree of freedom. In this paper, we look at a method of mass customization that allows buyers to make modifications to the design of their house, after which the new design is automatically checked against building regulations and the architect's requirements.

## 1. INTRODUCTION

For the last century, the predominant type of housing development in the Netherlands has been social housing – large-scale, mass-produced housing developments. Although this method of building results in reduced costs, it does not allow for meeting the needs of individual buyers, as mass production calls for large amounts of identical houses, which means that they cannot be tailored to the needs of individuals. Other industries, such as the car industry, that previously operated based on the principle of mass production have since adopted alternate methods to provide a far greater variety of choice to customers. The housing industry, however, has been slow to respond to this trend. There are multiple reasons for this, ranging from the somewhat limited adoption of computers to the fact that houses cannot benefit from the same economies of scale that a lot of consumer products can.

Houses are subject to considerably more regulations than other consumer products such as clothing or computers, and also offer a far greater range of possible alterations. Trying to cover the whole of the valid solution space by means of predesigned alternatives is infeasible. The only practical way to offer buyers designs that are adapted to their wishes is to involve them in the design process. Since a one-on-one meeting with the architect is impractical due to the size of housing developments and limited time, buyers must be able to modify the design without the support of the architect. In this scenario, however, the amount of regulations raises a technological barrier. Since requiring buyers to familiarize themselves with all of the building regulations is not a viable option, an automated way of verifying building code compliance of designs is needed.

The automated checking of building regulations is a relatively new concept in the housing industry, with the first steps only having been taken in the past few years with projects such as SMARTcodes (Wix et al. 2008). Most applications of this approach, however, have so far focused on communication between experts, and not on the possibilities it offers for simplifying communication between buyers (non-experts) and the architect. In this paper, we will explore the basic concepts of such a system, as well as some of the technological challenges, focusing on the building code compliance.

## **2. MASS CUSTOMIZATION**

Before the invention of computer-aided design, most manufacturing processes were limited to one of two options: mass production or customization. Mass production is the production of large amounts of identical parts. The idea has existed for hundreds of years (the Venetian Arsenal, a shipyard in Venice, employed mass production to produce one ship per day as far back as the 14th or 15th century), but it did not achieve widespread popularity until its adoption by Henry Ford's Ford Motor Company. Due to economies of scale, mass production reduces costs significantly, but it prohibits individual choice. A well-known quote from Henry Ford about the Model T Ford is "Any customer can have a car painted any colour that he wants, so long as it is black."

On the other end of the spectrum is customization, where each product is unique and built according to a different specification. A textbook example would be the art of portrait painting, where each portrait is necessarily unique. Since the lack of repetition reduces the possibilities for standardization, costs are typically higher than for mass production. Important to note is that both alternatives can exist within the same industry and even within the same project. In the housing industry, for instance, housing projects are usually executed as mass production while one-of-a-kind buildings such as museums are customized, although they are often constructed out of mass-produced components.

Mass customization is a mix between the two systems that combines the advantages of both. It is made possible by the use of computer aided design, which allows for more flexible output with little or no additional variable costs (though at the cost of a higher up-front investment). The car industry is a good example of this approach. Instead of merely producing one car model as in the days of the Model T Ford, customers can now choose from a wide range of models in different colours and with different extras, which are all assembled on the same assembly line by use of robots.

### **2.1 Mass customization types**

In the book "Mass Customization: The New Frontier in Business Competition" (Pine II 1993), four different types of mass customization are identified: collaborative, adaptive, transparent and cosmetic customization.

### **2.1.1 Collaborative customization**

In collaborative customization, there is a dialog between the producer and the consumer to determine the consumer's exact needs. This information is then used to manufacture a product specifically for that customer. The automobile industry operates on this basis; the customer chooses a model, a colour, etc. and this information is sent to the factory, where the chosen car is produced.

Using collaborative customization in the housing industry would involve asking buyers for their needs and then designing the house accordingly. This approach is used already, but typically only in one-off projects where the buyer contacts an architect directly to design a custom house. Due to the costs involved, only upscale houses tend to be built in this way. In larger developments, this method is used only rarely.

### **2.1.2 Adaptive customization**

In the case of adaptive customization, the manufacturer's product is standardized, but can be modified by the customers. An example would be a typical office chair, which can be adjusted to accommodate people of varying sizes by adjusting a few levers.

In the housing industry, one can think of movable walls, which allow spaces to be modified according to the user's needs. On a larger timescale, there is IFD (Industrieel, Flexibel, Demontabel, i.e. Industrial, Flexible, Demountable) building (van Gassel 2003). In this philosophy, buildings are constructed in such a way that they can be disassembled into their respective components (floors, walls, etc.) This reduces building waste, but also means that more involved modifications, such as additions to the house, are easier to realize.

### **2.1.3 Transparent customization**

In transparent customization, customers receive a customized product without being aware of it. Fanta Orange, for example, has a different flavour and colour in different countries, but has the same name everywhere.

Adoption of this in the housing industry is more difficult to envision. If the target audience is known up front a certain amount of customization could be done, but without inquiring about their specific needs, which would be categorized as collaborative rather than transparent customization, only a limited amount of customization can take place.

### **2.1.4 Cosmetic customization**

Finally, cosmetic customization means marketing the same product to different target audiences in a different way. For example, SUVs (Sport Utility Vehicles) are marketed to people in more rural areas for their off-road capabilities and to people in the suburbs for their safety.

Given the fairly limited number of common dwelling types (apartment, detached house, row house, etc.) a certain amount of cosmetic customization is unavoidable in the housing industry. Doing so deliberately would necessitate a large-scale market research to discover if there are target groups that share a large proportion of their requirements.

## **2.2 Participatory design**

Up until the 19th century, houses were typically built individually, and were customized to the owner. This was of course not cheap, and hence proper houses were reserved for richer citizens. Less affluent people often lived in shacks they built themselves. This started to change at the start of the 20th century when increasing urbanization, industrialization and the Woningwet (Housing law) of 1901 started a trend towards social housing. Social housing, like the industrialization that led to it, is based on mass production. Large amount of identical – and thus cheap – houses were constructed for the lower classes. After the Second World War, the required rebuilding effort and the baby boom resulted in an even greater push towards social housing.

The late 20th and early 21st century, however, saw the beginning of a movement away from mass-produced identical houses towards individually customized homes, partly driven by increased freedom of choice in other industries.

### **2.2.1 Habraken's alternative to mass housing**

In 1964, the BNA (Bond van Nederlandse Architecten, i.e. the Union of Dutch Architects) presented a young architect by the name of John Habraken with the opportunity to start a research foundation. Three years prior, Habraken had written a book called "Supports, an alternative to mass housing" (Habraken 1972). In this book, he claimed that the elimination of the buyer that occurred in mass-produced housing was incorrect. Instead, he argued for dividing a house in two different spheres, which he called "support" and "infill". The infill sphere refers to parts of the house that are likely to change between different owners, such as kitchens, the floor plan and floor material. The support sphere contains the more permanent features

of the house (load-bearing walls, ducts, etc.). A project would have only a small number of different support spheres, but the infill sphere was unique to each dwelling. The support sphere therefore has to be flexible enough in its layout to accommodate many different infills. Buyers could view the infill in showrooms, and make a selection with guidance from a specialist.

The research foundation that John Habraken founded was named SAR (Stichting Architecten Research, i.e. Foundation for Architect's Research). This foundation further developed and promoted Habraken's "Open bouwen" (open building") method, and this marked the first step towards mass customization in the Netherlands.

Although Habraken's method is a significant increase in freedom of choice for the buyer, there are a few limitations. The potential for more involved changes, such as adding an extra floor, is limited, since the construction method for the supports is based on mass production. Additionally, the need for buyers to confer with a specialist when choosing the infill means that this method becomes labor-intensive on larger projects.

### **2.2.2 Multiple choice housing**

Another method of participatory design that has been growing in popularity in the past decade or two is to present buyers with a limited number of variants for different parts of the house, resulting in a sort of "multiple choice" style of participatory design. This increased flexibility usually takes the form of brochures in which people can choose between different alternatives, e.g. kitchen layout A, B or C and an optional extra storey. The brochure can take the form of a traditional paper brochure, or of a computer application that allows you to interactively modify the design on a computer. Examples of the latter approach include the "Wenswonen" projects by Heijmans (Heijmans 2010), "Woonwijzer" by architecture firm BBVH (BBVH 2010) and TNO's iBuild (TNO 2010).

This "multiple choice" approach to architecture requires that the architect designs all the alternatives up front. Due to the combinatorial growth of these alternatives – five choices of three alternatives each already result in  $3 \times 3 \times 3 \times 3 = 243$  total variations – the amount of options is usually kept fairly limited. This means that while a number of different alternatives are offered, there is still a good chance that a customer's desired design variation is not offered. Additionally, there is the risk that some of the designed alternatives will not be used at all, since they are created by the architect instead of by the users.

### **2.3 Mass customization in the housing industry**

Surveys have shown that buyers are willing to pay up to 10% of the total house price extra if that means the house design is adjusted to their needs (Halman et al. 2008). This represents a lucrative opportunity for housing project developers. Compared to other countries, however, customization in the Netherlands is relatively rare. In the United States and Japan, for example, this practice is more prevalent (van den Thillart 2004).

The previous two paragraphs discussed two methods of increasing user participation in the design. Neither was free of disadvantages though. Habraken's method is labour-intensive and multiple choice housing only offers a limited amount of choice. Ideally, it would be possible to combine the advantages of both without incurring their disadvantages. One way of achieving this is to allow people to modify the design on their own computer. This way, architects don't have to create variants themselves, saving a lot of time, and buyers are able to design their house exactly the way they want. Allowing non-experts to design a house, however, introduces a new problem; being unaware of building regulations, they are likely to create illegal designs. Therefore, all the buyers' designs will have to be checked against building regulations. Since verifying large amounts of blueprints isn't a viable option (aside from the fact non-experts will have difficulty creating correct blueprints), a building information model will have to be used.

### **2.4 Building information models**

If customers are to customize their own homes, a separate set of drawings will have to be created for each house. Continuing to make these drawings in the traditional, two-dimensional, way is costly and error-prone. Representing a 3D design as a series of 2D drawings means that some information is specified multiple times; failure to update all relevant drawings will introduce errors into the design. These errors are difficult to spot since the information is spread out overall several drawings instead of one design. Additionally, representing the design as lines on paper (either real or virtual) makes it very difficult for a computer to reason about the design. The importance of this will be discussed in a later paragraph. To combat these problems and to achieve other benefits compared to the traditional design method, such as improved communication, the past two decades have seen the emergence of Building Information Models (BIMs). First coined by architect and Autodesk employee Jerry Laiserin, the term refers to the data models found in architectural software such as Revit and ArchiCAD, the latter of which was the first CAD package to introduce the concept of a BIM.

In a BIM, architectural designs are stored as elements such as walls and windows, along with non-geometrical information such as building schedules. The desire to store a design not just in terms of pure geometry, but as a parametric model, can also be found in the mechanical engineering industry, where it is referred to as feature-based modelling (van Leeuwen 1999, Bidarra and Bronsvort 1999). This way, the computer knows much more about the design than in the case of the traditional, non-semantic, representation of blue prints consisting of lines. This allows features such as 3D visualization and automatic construction planning and parts list generation. It also makes it easier to make changes to the design, as changes only have to be made once, as opposed to in all the relevant 2D drawings.

Most CAD packages nowadays have an internal BIM. This works well, but results in problems when information must be exchanged between two different packages. Since each vendor uses a proprietary BIM with different amounts of available information, exchange is problematic and frequently results in loss of information. In order to solve this problem, the International Alliance for Interoperability (IAI 2010) developed a vendor-independent specification for a building model called IFC (Industry Foundation Classes). The intent is to provide an open standard for BIMs that all CAD vendors can read and write to facilitate interoperability between the various CAD programs.

### **3. CONSTRAINTS**

One of the greatest challenges in adopting mass customization in the housing industry is making sure that all the resulting designs comply with building regulations, as well as the architect's intentions. These checks could be done manually, but due to large amount of both designs and rules this would be exceedingly labour-intensive. Finding a method of automating this checking process would greatly benefit this phase. Naturally, this requires that building regulations can be expressed in a way that can objectively be checked by a computer. Although this is true for a large subclass of all building regulations, there are exceptions. Regulations such as "the architectural quality of the addition must match that of the surrounding buildings" have no objective interpretation, as "architectural quality" is an ill-defined term. As such, a system based on this approach will not be able to handle every regulation that is currently found in practice. It will, however, be able to check a sizeable amount, if not the majority, of the regulations, removing the need for people to worry about the trivially checked rules and giving them more time to focus on questions of aesthetics and such.

### 3.1 Constraints in different industries

The idea of using constraints is not new. Constraints have found the most widespread adoption in the mechanical engineering industry. Applications include automatically inferring constraints from sketches (Anderl and Mendgen 1996), creating designs for piping (Gross 1996) and parametric solid modelling (Bettig and Shah 2003).

Conversely, the building industry has so far not seen much application of constraints. The only major CAD program to offer support for constraints is Revit (Strömberg 2006), and even then the support is limited to basic geometrical constraints. Projects that use more complex constraints include “De Digitale Dakkapel” (the digital dormer), which allows people check online whether a dormer needs a permit (van Leeuwen et al. 2004), and the SMARTcodes project, in which building models are checked for building code violations. The constraints in the SMARTcodes project are specified by means of the constraint framework present in the IFC building model mentioned earlier (Borrmann et al. 2009).

Although not quite the same in terms of application, constraints can be compared to universal quantifications from the field of mathematics. Quantifications indicate how many elements of a given set satisfy the given predicate. For instance,

$$\forall x \in \mathbb{N} . x \geq 0$$

means “For all elements  $x$  in the set of natural numbers,  $x$  is more than or equal to zero”. The similarity with constraints, such as “dormers must be higher than 1.5 m”, can easily be seen.

### 3.2 Methods of using constraints

There are two ways of dealing with constraints, depending on who creates the design. Although both approaches have the same end result, the way in which they arrive there is different.

#### 3.2.1 Constraint solving

The first way to use constraints is constraint solving, i.e. taking the constraints as input and trying to find a design that satisfies them. This method is typically used when the design is generated by the computer (Belbidia and Alby 2003, Donath and Böhme 2007, Eggink et al. 2001, Kelleners 1999). The precise manner in which this is done depends on the method that is used to generate the design and the complexity of the constraint. For simple constraints such as “the height of the wall must be

between 2 and 2.6 m”, it is easy to generate all possible valid solutions. Computing all combinations of all the possibilities for each element results in a list of all valid design alternatives, from which an alternative can then be picked using any criterion, such as minimal cost. The major downside of this approach in practice is that the solution space is very large, resulting in prohibitively long calculation times. For that reason it is more common to use a different approach, namely genetic algorithms.

The idea of genetic algorithms strongly resembles evolution; first, a random design is created. This design receives a score based on how well it meets the given constraints (in all likelihood this score will be very low, as a completely random design is unlikely to meet many of the criteria). From this initial design, a new generation of designs is created by making small adjustments to the original (mirroring the way mutation produces variation in species). Each design in this new generation is also given a score based on how well they satisfy the constraints. The mutations with the highest scores are selected as the new basis for future generations. The process continues until a solution with a sufficiently high score is found. To reduce to risk of getting stuck on a local maximum (a solution that is not yet optimal, but all its mutations have a lower score), it is customary to also include some lower-scoring alternatives in each generation to increase genetic diversity.

### **3.2.2 Constraint checking**

The second way of dealing with constraints is to make a design, check to see if it meets all the constraints, and adjust it if necessary. This approach is called constraint checking. This approach is appropriate when the designer is a human. Because the designer has the advantage of common sense, the initial design will be significantly better than the first designs in the constraint solving method. In this scenario, the constraints serve more as a reminder than as the defining factor in the design. An example of this method of constraint handling can be found in the CAD package Navisworks and the Solibri Model checker (AECbytes 2010), which offer clash detection, a feature that detects intersections between different parts of the 3D model, which would be impossible in real life.

This method better matches the original goal of having the computer support the buyer, since constraint solving allows no user input beyond setting the constraints. Combined with the problem of computational complexity this means that constraint checking is a better approach for our purposes.

## 4. CONSTRAINT LANGUAGE

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:

The first option is to use a programming language. This is a natural choice because the amount of expressive power required of the constraint system is similar to that of a (simple) programming language. The advantages of this option are easy implementation and the guarantee that most, if not all, constraints can be entered this way. The main disadvantage is that programming syntax will most likely be unfamiliar to architects, requiring training. The specific programming language can be chosen from the large number of languages that exist today. For instance, in Java the translation of the constraint “The height of windows in brick walls must be between 1 and 2 m” might result in the following code:

```
if (wall.material == materials.Brick) {
    for(window : wall.windows)
        assert(window.height >= 1 &&
               window.height <= 2));
}
```

As the example above demonstrates, a lot of popular programming languages have syntax that is not immediately recognizable to non-programmers (the use of && for and, curly braces, etc.). A philosophically similar, if technically different, approach is to express the constraints in XML, as is done in the SMARTcodes project (Wix et al. 2008). It has the same problem of having a non-intuitive syntax that programming languages have.

Some of these issues could be remedied by using an Application Programming Interface (API) or a Domain-Specific Language (DSL) targeted specifically at defining architectural constraints (Spinellis 1999). This reduces the amount of unfamiliar syntax the architect has to deal with. The same example constraint might then be expressed as something along the lines of:

```
window height between 1 and 2 for window in windows
of wall if wall made of brick.
```

If this approach is taken even further, we arrive at natural language parsing. This means that the architect can enter the constraints in plain

English. This removes the requirement for training on the part of the architect, since he can use the language he is familiar with. On the other hand, it greatly increases the difficulty of the implementation, as natural languages are far harder to interpret than computer languages, due to the freedom of expression they offer. For example, using the same constraint again, the following expressions are all functionally identical:

- The height of windows in brick walls must be between 1 and 2 m
- Windows in walls made of brick must be between 1 and 2 m high
- The height of any window in a brick wall must be higher than or equal to 1 m and lower than or equal to 2 m

However, the superior usability of this approach means that is still the preferred option, provided a suitable implementation can be found.

## **4.1 Parsing**

Parsing is the process of using a grammar to convert a text to a data structure – typically a hierarchical data structure such as a tree. Parsing is used in several domains, the most common ones being interpreting programming languages, reading data files and analyzing natural text. Parsing typically consists of three steps; lexical, syntactic and semantic analysis. These steps will be explained in the following paragraphs.

### **4.1.1 Lexical analysis**

In the first step, the goal is to split up the single input string into individual units known as tokens. The algorithm that is typically used to do this is to loop over the input string one character at a time and either adding it to the current token or starting a new one, based on the possible tokens that have been defined. The end result is a list of tokens. As an example, a simple calculator might define three kinds of tokens: integers, which consist of 1 or more digits, an addition operator consisting of a single plus sign, and a subtraction operator (a single minus sign). Lexical analysis would then split the input string “12+3-7” into the following tokens: “12”, “+”, “3”, “-”, “7”.

### **4.1.2 Syntactic analysis**

The second step is to convert the list of tokens into a (usually hierarchical) data structure. This is done through rules that define a series of input tokens and the resulting output. The exact way these rules are expressed varies, but most resemble the Backus-Naur Form (BNF), which is a syntax used to describe grammars. For instance, a grammar for parsing a very limited subset of the English language might look as follows:

```

<sentence> ::= <subject> <predicate>
<subject> ::= <article> <noun>
<predicate> ::= <verb> <direct-object>
<direct-object> ::= <article> <noun>
<article> ::= THE | A
<noun> ::= MAN | DOG
<verb> ::= BITES | PETS

```

These grammars are often specified in a BNF-like syntax and subsequently converted to the host programming language by use of a parser generator such as ANTLR, Bison or Lexx/Yacc. Alternatively, the grammar can be written in the host language directly through the use of parser combinators. Continuing with the simple calculator parser from the previous paragraph, a basic binary tree data structure might be defined as follows (using Haskell for the syntax):

```

data Operator = Plus | Minus
data CalcTree = Node CalcTree Operator CalcTree
               | Leaf Int

```

I.e. a tree is either a node containing an operator (plus or minus) and two subtrees, or a single integer. The parse rule could then become

```

calc = try (Node <$> int <*> operator <*> calc) <|>
        int

```

which first checks if the remainder of our input string starts with an integer followed by an operator and a calculation. If the match succeeds, a node is created. If not, the parser backtracks over the input and tries to match just an integer. If the match succeeds, the syntactic analysis ends, otherwise it fails, since no matching rule can be found. Backtracking is required since tokens are consumed as they are matched. In the case where there is one remaining integer in the input, trying to match the first alternative in `calc` would consume it and then fail since there is no operator. Matching the second alternative requires undoing this. Note that not all parsers support backtracking. Without backtracking the rules must be formulated in a different way so that it is not required. In this case, a common solution is the following (using ANTLR syntax for a C# program):

```

calc returns[CalcTree tree] :
    a=INT {$tree = a;}
    (op=operator b=INT
     {$tree = new Node($tree, op, b)};)*;

```

which means that a calculation is an integer followed by zero or more occurrences of an operator followed by an integer.

### 4.1.3 Semantic analysis

The final step is to evaluate the resulting parse tree. Naturally, the algorithm used in this step depends entirely on the specific problem being solved. In the case of the simple calculator example, the desired value is the result of the calculation. Since the calculation is now stored in a binary tree, calculating the result can be done with a simple recursive algorithm:

```
eval (Node l Plus r) = eval l + eval r
eval (Node l Minus r) = eval l - eval r
eval (Leaf i) = i
```

## 4.2 Natural language parsing

The previous paragraphs describe a deterministic parser, i.e. at no point in the process is there any doubt as to what type of token is encountered. This is usually the case in domains such as programming languages or data files. This, however, is not true in all domains, with natural language processing being the most prominent counter-example. A well-known example is the sentence:

“Time flies like an arrow, but fruit flies like a banana.”

The first occurrence of “flies” is a verb, but the second occurrence is a noun. The deterministic approach stops working here, since the interpretation involving flying fruit would be grammatically correct, but logically incorrect. The common solution for this problem is to use a statistical approach, often based on a large database of tagged text, which is known as a corpus (Brown et al. 1990, Collins 2003, King 1983, Manning 1999). The parser can then check the neighbouring words to determine the more likely scenario. Since “fruit flies” will occur more often as “Noun, Noun” than “Noun, Verb”, this will be chosen as the correct interpretation. Naturally, this requires that such a database is available.

The requirement for a corpus presents two complications. Since the application is intended to be used in the Netherlands, Dutch is the preferred language for entering the constraints. Unfortunately, the availability of Dutch corpora is extremely limited. As an added complication, constraints will contain a lot of building industry-specific words that are not found in general-purpose corpora, rendering this approach somewhat problematic. Since no building industry-specific Dutch corpora exist, interpreting natural language will require either creating such a corpus or finding a parsing algorithm that does not require a corpus.

## 5. CONCLUSIONS

Giving buyers more freedom of choice when it comes to the design of their new house will require their participation in the design process. The two existing methods of doing this – direct consultation and multiple choice housing – both have downsides related to efficiency and the amount of choice, respectively. Choosing the approach where people modify the design themselves, after which their designs are checked automatically for design constraints and building code compliance could solve both of these problems. Going down this path requires a method of entering the constraints into the computer.

The goal of this project is to develop a method for architects to enter design constraints. The entry method should consist of natural language or as close an approximation as is practically possible. This significantly reduces the required amount of training since no new techniques will have to be mastered and since it does not significantly differ from the current way of dealing with constraints, since they are also text-based.

Constraint will be represented internally as functions that take a building model as their input and return a Boolean value (true or false) that indicates whether or not the constraint was violated. These functions will be generated by parsing the natural language input.

The feasibility of this language will be tested by creating prototypes, followed by performance tests, which will test both the usability of the system as well as the success rate of interpreting the natural language input. This will be done by presenting participants with a series of designs with mistakes. Their task will be to formulate constraints to prevent such mistakes in the future. The resulting constraints will be entered into the prototype to determine the percentage that is correctly interpreted.

## 6. REFERENCES

- AECbytes, Solibri Model Checker: AECbytes Product Review,  
<http://www.aecbytes.com/review/2009/SolibriModelChecker.html>
- Anderl, R. and R. Mendgen, 1996, "Modelling with constraints: theoretical foundation and application", *Computer-Aided Design*, 28(3), p. 155-168
- BBVH, Woonwijzer - hoogvliet, <http://www.bbv.nl/woonwijzer/>
- Belblidia, S. and E. Alby, 2003, "Implicit handling of geometric relations in an existing modeler", in: *CAADRIA 2003 Conference, Bangkok, Thailand*, p. 613-622
- Bettig, B. and J. Shah, 2003, "Solution selectors: a user-oriented answer to the multiple solution problem in constraint solving.", *Journal of Mechanical Design*, 125(3), p. 443-451
- Bidarra, R. and W.F. Bronsvort, 1999, "Semantic feature modelling", *Computer-Aided Design*, 32, p. 201-225

- Borrmann, A., J. Hyvärinen and E. Rank, 2009, "Spatial constraints in collaborative design processes", in: *Procedures of the International Conference on Intelligent Computing in Engineering (ICE'09)*
- Brown, P.F., J. Cocke, S.A.D. Pietra, V.J.D. Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer and P.S. Roossin, 1990, "A statistical approach to machine translation", *Computational Linguistics*, 16(2), p. 79-85
- Collins, M., 2003, "Head-Driven Statistical Models for Natural Language Parsing", *Computational Linguistics*, 29(4), p. 589-637
- Donath, D. and L.F.G. Böhme, 2007, "Constraint-Based Design in Participatory Housing Planning", in: *eCAADe 2007 Conference, Frankfurt am Main, Germany*, p. 687-694
- Eggink, D., M.D. Gross and E. Do, 2001, "Smart Objects: Constraints and Behaviors in a 3D Design Environment", in: *eCAADe 2001 Conference, Helsinki, Finland*, p. 460-465
- van Gassel, F., 2003, "Experiences with the Design and Production of an Industrial, Flexible, and Demountable (IFD) Building System", *NIST special publication SP*, ISSU 989, p. 167-172
- Gross, M.D., 1996, "Elements That Follow Your Rules: Constraint Based CAD Layout", in: *ACADIA 1996 Conference, Tuscon, USA*, p. 115-122
- Habraken, N.J., 1972, *Supports, an alterSupports, an alternative to mass housing native to mass housing*, Architectural Press, London
- Halman, J.I.M., J.T. Voordijk and I.M.M.J Reymen, 2008, "Modular Approaches in Dutch House Building: An Exploratory Survey", *Housing Studies*, 23(5), p. 781-799
- Heijmans, Wenswonen - Ieder mens een woning naar Wens, [www.wenswonen.nl](http://www.wenswonen.nl)
- IAI, Welcome - IAI Tech Interantional, [www.iai-tech.org](http://www.iai-tech.org)
- Kelleners, R.H.M.C., 1999, "Constraints in object-oriented graphics", *PhD Thesis*, Eindhoven University of Technology
- King, M., 1983, *Parsing Natural Language*, Academic Press Inc. Ltd., London
- van Leeuwen, J.P., 1999, "Modelling Architectural Design Information by Features", *PhD Thesis*, Eindhoven University of Technology
- van Leeuwen, J.P., A.J. Jessurun and E. de Wit, 2004, "The Digital Dormer - Applying for Building Permits Online", in: *European Conference on Product and Process Modelling 2004, Istanbul, Turkey*, p. 355-361
- Manning, H., 1999, *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge, MA
- Pine II, B.J., 1993, *Mass Customization: The New Frontier in Business Competition*, Harvard Business Press, Watertown
- Spinellis, D., 1999, "Reliable software implementation using domain-specific languages", in: *ESREL, 10th european software conference on safety and reliability*
- Strömberg, J., 2006, "Integrating Constraints with a Drawing CAD Application", *Masters Thesis*, Stockholm University
- van den Thillart, C.C.A.M., 2004, *Customised Industrialisation in the Residential Sector: Mass customisation modelling as a tool for benchmarking, variation and selection*, Sun, Amsterdam
- TNO, iBUILD, [www.ibuild.nl](http://www.ibuild.nl)
- Wix, J., N. Nisbet and T. Liebich, 2008, "Using Constraints to Validate and Check Building Information Models", in: *ECPPM 2008 Conference, Sophia Antipolis, France*, p. 467-476