Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2009, Valencia
Evolution and Trends in Design, Analysis and Construction of Shell and Spatial Structures
28 September – 2 October 2009, Universidad Politecnica de Valencia, Spain
Alberto DOMINGO and Carlos LAZARO (eds.)

Exceptions as a rule in computational design

Jeroen L. COENDERS*

*Arup, Delft University of Technology
Naritaweg 118, 1043 CA Amsterdam, Stevinweg 1, 2628 CN Delft
jeroen.coenders@arup.com, j.l.coenders@tudelft.nl

Abstract

Structural design deals with solving the complex problem of the realisation of a building from the initiation to final execution. During this process many complicated sub-processes take place which handle with large amounts of design information. Following the Structural Design Tools approach (Coenders [2]) computational design attempts to support the process of design by providing concepts, guidelines and computational strategies to appropriate existing and new technology for (structural) design. Computational technology is strong in storing and processing large amounts of data in a fraction of the speed of human labour. However, one of the problematic areas of computational design is the fact that design often consists of many exceptions. Current state-of-the-art computational design systems, such as Bentley's GenerativeComponents (GC; by Robert Aish [1]) or Grasshopper (by McNeel [4]) are very strong in modelling repetition, but not exception. Generating design by rules is appropriate for repetition, but by definition exceptions don't follow the rule. Traditional CAD systems are stronger in this field as the entire model is basically an exception. However, these systems are less appropriate for design process augmented by computation.

The author would like to discuss a novel computational strategy for parametric and associative design systems to address the modelling and handling of exceptions in design for design systems of the future.

Keywords: Computational Design, Parametric and Associative Design, repetition and exceptions in design.

1. Introduction

Structural design deals with solving the complex problem of the realisation of a building from the initiation to final execution: First, this solution is virtual, existing in paper, sketches, computers, and later physical when the building is being constructed on site. The engineer conceives a structural concept which describes how the structure will bear its loads and determines how the structure will be build. During this process many complicated sub-processes take place which handle with large amounts of design information. These processes are complicated, because they deal with obtaining evidence and confidence in the feasibility of the structural concept, concerning the structural limit and serviceability states, but also other criteria, such as economic feasibility, buildability, etc. In these processes the amount of computational techniques used has increased since the first computers. However, computation is often focused on precision and automation, which helps the engineer in gaining his or her evidence and confidence, but does not directly help to create this confidence. The author has proposed that to obtain a higher degree of automation in the design process, possibly even a process where all or at least the majority of information exists in computation systems, it is not enough to just develop computational systems from the technological side (Coenders [2]). Technology in its own without the essential connections to the essence and key characteristics of design, such as the recognition of gaining confidence in the design, will fail in the long run. Therefore, the Structural Design Tools approach proposes a computational design approach to support the process of design by providing concepts, guidelines and computational strategies to appropriate existing and new technology for (structural) design and its key characteristics, by studying these characteristics and linking these to qualities of software technology. Furthermore, it might be appropriate to make minor changes to these technologies to make them more suitable for design.

A simple example would be parametric and associative design (further discussed below) which in theory provides mechanisms to model and store design logic and knowledge rather than just design information, potentially increasing insight in the workings of the system. By providing structural objects in these systems these systems would become more appropriate to use in structural design. This paper discusses a more complex example of appropriating parametric and associative technology to structural design: by proposing some modifications on a more fundamental level: the ability to model and deal with exceptions in the model and in the system.

2. Parametric and associative design

The systems will be used in this paper can be classified as parametric and associative design systems. The author has defined concepts in an earlier paper [3] to identify these systems. Examples for these computational design systems are Bentley's GenerativeComponents (GC; by Robert Aish [1]) or Grasshopper (by McNeel [4]).

As stated before these systems are able to express design knowledge through processable design logic which the system is able to replay on every change in the input parameters of the system. These systems help in rationalising often complicated designs by explicitly defining the underlying logic, rather than just containing the result of a set of geometric operations.

The special feature of these systems is that they do not act as a black-box of pre-programmed logic, but the user is able to develop his or her own logic in these systems as long as the operators or combinations of operations are available, or the system can be extended towards new operators by programming add-ins.

The two mentioned systems contain another powerful concept: replication. This concept is able to apply logic defined for a single object over a series of objects and carry these replicated objects through the design. In GenerativeComponents the Series() statement is being used to initiate the replication. Replication can also be a result of for instance an intersection operation. This mechanism allows for single-definition and multiple-application which is very powerful to model complicated logic with ease.

Another mechanism proposed by the author where replicated objects can occur is collection. Collection is a mechanism where certain objects are placed in a container or collection object and obtain a single point-of-reference.

3. The practice of structural design modeling

Computational technology is strong in storing and processing large amounts of data in a fraction of the speed of human labour. Furthermore, parametric and associative design enables designers to express repetitive or replicated logic with relative ease.

However, as stated before, to assess the applicability of computational design systems the suitability for design needs to be assessed. One of the problematic areas which immediately arises is the fact that only in a small amount of cases such extensive repetition is present in design that replication becomes useful. It can almost be said that design doesn't follow the rule, but is a collection of exceptions. Designs are rarely completely rationalized to a repetitive form without exceptions.

Currently 2D CAD systems are often seen as old-fashioned and outdated, because they do not contain object logic, but only geometric representations of the objects which need to be interpreted by the user. However, one important benefit of these systems is often overseen, which is that they are very suitable for modelling exceptions, exactly because they do not prefer objects, but just model the geometric representation. On the other hand, there are also many disadvantages to these systems which have been extensively discussed in the Building Information Modelling (BIM) world. Also for the future, these systems are less appropriate for design process augmented by computation.

An additional problem is the fact that as buildings become larger and information becomes more detailed, complexity arises in the information and interrelationship of the information.

4. Dealing with exceptions

Following the issues described above and the fact that the increased complexity in building design makes more application of computational design desirable in the future, computational design would benefit from a method to deal with exceptions which while maintaining all advantages of parametric and associative modelling.

It needs to be mentioned that the concepts and techniques below will not cover every type of exception, but will aid in dealing with exception objects which contain some kind of common feature, either their type, how they are created or how the system will deal with them. Complete random objects of random object types which each have their own special behavior, naturally will not benefit from design logic on a broader level. For this class of objects the logic defined can only be applied on the object itself and therefore there is no benefit in any of the methods below.

It also needs to be mentioned that below mechanisms and operators will be described as a language to express exception logic. It needs to be noted that this language might not be comprehensive for all classes of problems, all required or available capabilities in other languages which deal with similar problems in computation, or all available classes of exceptions in design, etc. etc. Also, the language is not optimised for ease-of-use, understandability, etc. The language purely is intended to demonstrate the power of providing exception logic in design systems of the future and its conceptual ease of definition of logic.

4.1. Types

In design systems various types of objects are present. This paper will focus on two base types: (1) primitives, such as numbers, letters, characters, etc. which have a common sequences (2 follows after 1, b follows after a, etc.) and (2) object sequences or collections. The second case, object sequences or collections, contain objects and since everything can be defined as an object, this could be basically anything: objects, names of days, dates, results from an intersection operation, an imported Excel range, a database query, etc. etc. The sequence logic for these collections are defined by two factors: (1) the position in the array (Table 1 operator 9 and 10) and (2) a given index number, which is usually equal to the position in the array, but can differ on results from (partly) failed operations.

Objects are assumed to have a system base type (Object, Point, Line, etc) and a Global Unique Identifier (GUID).

4.2. Conceptual directions

Exceptions occur in various situations in modelling design in parametric and associative design systems. Two scenarios will be address in this paper: (1) during object creation and (2) after object collection.

4.3. During object creation

In the case of the “during object creation” scenario exceptions are created while the user defines a repetitive logic, but would like to make certain exceptions to the repetition rule. In most parametric and associative design systems this scenario could be created by defining a sequence of objects (for instance by using the Series() function in GenerativeComponents) and giving each of this objects a rule which determines what the behaviour should be. This is useful, but there are a number of caveats: (1) This would create objects where in some situations the user might not want an object, leaving a “turned-off” dummy object which takes memory and processing power from the system. (2) This would not work for behaviour where global knowledge is required rather than local information known to the object itself. (3) This would not work if depending on the rules, various types of objects need to be created rather than just one type.

4.4. After object collection

In the case of the “after object collection” scenario exceptions need to be dealt with which exist in the collection. These collections can either be a result of the “during object creation” scenario where a collection has been created which contains exceptions, a collection operation by the user in the design system or the result of an operation in the design system. Usually these collections contain broken sequences, only a part of the information has to be dealt with based on a criterion, certain objects need to be inserted at certain positions in the collection or the collection needs to be sorted or renumbered.

4.5. Operators

Operators are statements in the language which define a logic which refers to an object or object type or works on these objects or object types. Operators can work on various object types:

- Primitives, such as numbers (integers) or letters (characters);
- Objects or collections of objects;

Table 1 lists a number of operators or expressions in the language which can be used to express exception logic.

Nr	Operator	Description
1	*primitiveA*-*primitiveB*	Range from *primitiveA* to *primitiveB*
2	*primitiveA*-*primitiveB*;*stepsize*	Range from *primitiveA* to *primitiveB* with step *stepsize*
3	*objectA*,*objectB*	*objectA* and *objectB*
4	*collectionA*,*collectionB*	*collectionA* and *collectionB*
5	*collectionA*|*collectionB*	*collectionA* except *collectionB*
6	(..)	Compound, priority
7	*collection*[..]	Collection statement
8	*collectionA*[..]<*collectionB*	Inserts c*ollectionB* in *collectionA*
9	[..]	Array indices statement
10	a*number*	Reference to array index or position number
11	i*number*	Reference to given index number
12	n(*objectName)*	Reference to object name
13	g(*GUID*)	Reference to object's Global Unique Identifier (GUID)
14	t(*type*)	Reference to object's system type
15	Invert(*collection*)	Inverts the *collection*
16	Renumber(*collection*)	Renumbers the given indices of the *collection*
17	RegexpN(*collection*,*regexp*)	Returns the part of the *collection* that matches the regular expression *regexp* in its name.
18	RegexpT(*collection,regexp*)	Returns the part of the *collection* that matches the regular expression *regexp* in its type name.
19	Sort(*collection*,*sort*	Sorts the *collection* based on a *sort criterium* either

	criterium,ascending)	ascending or descending.
20	Select(*collection*,*ruletree*)	Selects the objects from a *collection* based on the given *rule tree* [3].

Table 1: Operators

Italic print in the table indicates a name of a primitive or an object to be replaced by the name of the primitive or object itself. Furtermore, Wikipedia [5] contains a comprehensive explanation of regular expressions used in operator 19.

In Table 2 various examples of operators have been shown applied to integer sequences.

Nr	**Example**	**Result**
1	1-10	1,2,3,4,5,6,7,8,9,10
2	1-3,6-8	1,2,3,6,7,8
3	1-10|5	1,2,3,4,6,7,8,9,10
4	1-10;2	1,3,5,7,9
5	(1-10)|(2-3)	1,4,5,6,7,8,9,10

Table 2: Examples of integer sequences

In Table 3 various examples of operators have been shown applied to letter sequences.

Nr	**Example**	**Result**
1	b-d	b,c,d
2	a,c-e,g	a,c,d,e,g

Table 3: Example of letter or character sequences

Table 4 demonstrates various examples for application of operators on object sequences. The original collections used in the examples are:

A={Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday}

B={1:PointA,2:PointB,4:PointC,5:PointD,6:PointE}

C={1:PointA,2:LineA,3:LineB,4:PointB}

With notation:
collection = {*index*:*objectA*,...,*index_n*:*object_n*}

Collection A would be a typical example of a predefined sequence of objects defined to be used throughout the model. Collection B is a typical example of what an intersection of multiple lines (6 original lines) with a surface could return. Note that one line missed the surface (number 3) and therefore no point was created at this index number.

Nr	**Example**	**Result**
1	A[1-3]	Monday,Tuesday,Wednesday
2	B[1-3]	PointA,PointB,PointC
3	B[1-5;2]	PointA,PointC,PointE
4	B[i1-i3]	PointA,PointB
5	B[i1-i5;2]	PointA,PointD
6	B[a1-a3]	PointA,PointB,PointC
7	Invert(A)	{Sunday,Saturday,Friday,Thursday,Wednesday, Tuesday,Monday}
8	Renumber(A)	{1:PointA,2:PointB,4:PointC,4:PointD,5:PointE}
9	B[i3]<PointX	{1:PointA,2:PointB,3:PointX,4:PointC,5:PointD, 6:PointE}
10	B[n(PointA)]	PointA
11	C[t(Point)]	{1:PointA,4:PointB}
12	Renumber(C[t(Point)])	{1:PointA,2:PointB}

Table 4: Examples of object sequences and collections

Example 9 in Table 4 is a powerful example of what exception logic could provide to design systems. Collection B is a point collection resulting from an intersection operation and somehow in this example it is logical that one point is missing. At this point a user-defined exception point can be inserted in the collection which can be used in concurrency with the repetitively defined points in subsequent logic by one reference: collection B.

4.6. Dealing with more complicated rules and rule structures

It can be imagined that the user would like to express even more complex statements or interrelations between statements. For this purpose the concept of rule-processing in parametric and associative design proposed in an earlier paper by the author [3] can be employed. This system is able to build complex rule objects which can interact and respond to any programmed behaviour that follows a rule-based logic. An example of this would be selection procedures from the collection where an algorithm needs be employed to determine if the object satisfies the criterion or not, for instance a point-in-polygon rule or search, selecting or filtering functions. Typically, the rule logic will be initiated by operator 20 in Table 1.

4. Discussion

This paper has demonstrated various types of exceptions and mechanisms to deal with these exception types.

However, as stated earlier, other exception types might occur in practical design problems. This would require further research in this specific topic which lies outside the scope of this proposal paper.

Furthermore, more advanced functionality might need to be addressed in the object types (for instance Booleans, Strings and Doubles as primitives), the operators or the types of collections the logic deals with, for instance nested and recursive collections or multi-dimensional collections.

It needs to be noted that the various demonstrations might be possible to model in current design systems without a language for expressing exception logic and would certainly be able to develop making use of programmed feature types. However, that would create less efficient design models (for instance with a lot of memory use on work-around dummy objects) which is a problem in the case of large and complex models or would require significantly more effort and knowledge of the end-user to apply. A language for expressing exception logic would make the logic more accessible and easy to use, creating more powerful and efficient models.

5. Conclusions

In this paper the author has discussed an important obstacle which exist in supporting the structural design process by computational design, the paradox between the ease of repetition by computation and the widespread existence of exceptions to this repetition in real building design. Furthermore, the author has proposed various strategies to deal with and overcome various types of exception in computational design systems of the future. These systems and concepts will be further researched, prototyped and developed at Delft University of Technology, the Structural Design Lab, and Arup.

References

[1] Aish R., *Introduction to GenerativeComponents, a parametric and associative design system for architecture, building engineering and digital fabrication*, white paper, http://www.bentley.com, 2005.

[2] Coenders J.L. and Wagemans L.A.G., The next step in modeling for structural design: structural design tools. *Proceedings of the international symposium on shell and spatial structures; theory, technique, valuation, maintenance*, Madrid, Spain, IASS, 2005: 85-92.

[3] Coenders J.L., Rule-processing as a cross-cutting concern in parametric associative design systems, in IASS-SLTE 2008 simposium Acapulco Mexico, New materials and technologies, new design and innovations – a sustainable approach to architectural and structural design, Oliva Salinas J.G., Acapulco, Mexico, 2008, 87-88.

[4] McNeel, *Grasshopper*, website, http://grasshopper.rhino3d.com, 2008.

[5] Wikipedia contributors, *Regular expression,* Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=289365504, 2009.