

Representing Computational Change in Design

Rudi Stouffs
Chair for Technical Design & Informatics
Faculty of Architecture
Delft University of Technology
r.stouffs@bk.tudelft.nl

1. Introduction

There will always be a need for different representations of the same entity; this is particularly true in the building domain, whether it be a building in its entirety that is under consideration, or a part of a building, albeit a shape, or some other complex collection of properties. The building domain, at all stages, is multi-disciplinary, involving participants, knowledge and information from various specializations. Problems in building design require a multiplicity of viewpoints, each distinguished by particular interests and emphases. In the main, the architect is concerned with aesthetic and configurational aspects of a design, the structural engineer is engaged by structural members and their relationships, and the performance engineer is interested in the thermal, lighting, or acoustical performance(s) of an eventual design. Each has views – derived from an understanding of current problem solution techniques in their respective domain – that requires a different representation of the same (abstract) entity. Even within the same task, or by the same person, various representations may serve different purposes defined within the problem context and selected approach. This is especially the case in architectural design, where the design process, by its exploratory and dynamic nature, invites a variety of approaches and representations (see, for example, Kolarevic 2000).

Integrated data models span multiple disciplines, and support different views. Such models allow for a variety of representations in support of different disciplines or methodologies, and enable information exchange between representations and collaboration across disciplines; examples include the ISO STEP standard for the definition of product models (ISO 1994), and the Industry Foundation Classes (IFCs) of the International Alliance for Interoperability (IAI), an object-oriented data model for product information sharing (Liebich 2004). These efforts characterize a, primarily, a priori top-down approach: an attempt is made at establishing an agreement on concepts and relationships, which offer a complete and uniform description of the project data, independent of any project specifics (Stouffs and Krishnamurti 2001a).

As such, these models do not support the creative aspects of the building design process, especially, in the early design phases. Creativity, in design, relies on a restructuring of information that is not yet captured in a current information structure – that is, emergent information – for example, when the design provides new insights that lead to a new interpretation of constituent design elements. The standard object-oriented approach (in CAAD) requires a specification of design elements as objects (with properties) that is maintained at all times, unless explicitly altered. Any reinterpretation of design elements, then, requires the specification of a change – in this case, computational – that not only fixes beforehand the source and destination object types, but also their numbers as well as the mapping between properties. It has been shown that continuity of computational change requires anticipation of the particular structures that are to be

changed (Krishnamurti and Stouffs 1997). Creativity, on the other hand, is without such anticipation.

Computationally, recognizing emergent information structures requires determining a transformation under which a specified structure is similar to a part of the original (Krishnamurti and Earl 1992). It requires the definition of a part relationship that governs when one information structure is considered a part of another. This part relationship may be freely defined, as long as it constitutes a partial order. The algebraic model for shapes (Stiny 1991; Stouffs 1994) is based on such a part relationship. In this model, a shape is specified as an element of an algebra, ordered by a part relation, and closed under the operations of algebraic sum, difference and product, and the affine transformations. Moreover, under the part relation, any part of a shape is a shape. Whence, each shape specifies an infinite set of (sub-)shapes, each a part of the original shape; in this way, users can deal with shape in indeterminate ways. As such, shapes emerge.

The maximal element representation (Krishnamurti 1992; Stouffs 1994) captures this notion precisely. Consider Figure 1, which shows a combination of two squares. In the object-oriented approach, each square may define an object as made up of four line segments. Visually, however, the composition in Figure 1 contains not two, but three squares. In order to recognize this third square, each square object needs to be reinterpreted as a collection of six line segments, such that two can be taken from each to define the middle square object. However, the transformation of a square object into such a collection of six line segments is specific to this particular context, and not generally relevant. Under the maximal element representation, each object is, in a minimal way, made up of maximal line segments, and each such maximal line segment specifies an infinite set of (sub-)segments that are each part of the original segment. Thus, the four line segments defining the middle square object can always be found in the representation of the original two squares, each a collection of four maximal line segments. Furthermore, although an indefinite number of other collections of line segments can be determined and represented, none is of any higher importance, except by designer choice. This provides the designer with the freedom to reinterpret a design in any way, and have this interpretation supported by the system.

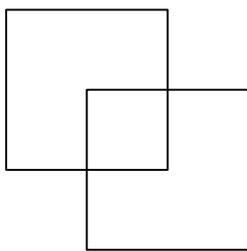


Figure 1: A composition of two or three squares.

Both the algebraic model for shapes and the maximal element representation have their origin in shape grammar research (Stiny 1980; 1991). They were initially developed for shapes, made up of points and line segments, in one and two dimensions, later extended to three dimensions and to planar and volume segments (Stouffs 1994). Each type of spatial element specifies its own algebra, e.g., U_{12} for line segments in two dimensions, U_{23} for plane segments in three dimensions. The notation U_j refers to linear shapes made up of i -dimensional elements in a j -dimensional Euclidean space, with U_i as shorthand, when the dimensionality of the space is

known (Stiny 1991). When a shape is made up of more than one type of spatial element, it belongs to the algebra given by the Cartesian product of the algebras of its spatial element types, e.g., a shape of points and line segments belongs to $U_0 \times U_1$. The algebraic model has also been applied to shapes augmented with non-geometric properties, such as labels, weights (e.g., line thickness; Stiny 1992), and colours (Knight 1989). The notation V_i refers to linear shapes made up of i -dimensional elements augmented with labels and W_i to linear shapes made up of i -dimensional elements augmented with weights (Stiny 1991). Both V_i and W_i can be defined as the Cartesian product of the algebras of the spatial element type and, respectively, an algebra for labels and weights.

By extending the algebraic model for shapes, a framework for representational flexibility, named *sorts*, can be defined that supports, computationally, the recognition of emergent information. In this framework, each *sort* defines its own algebra: a *sort* of points corresponds to U_0 , a *sort* of line segments to U_1 , a *sort* of plane segments to U_2 and a *sort* of volume segments to U_3 . Corresponding to the Cartesian product of algebras, we can define a conjunctive operation on *sorts* such that a *sort* of labelled points can be defined corresponding to V_0 , given a *sort* of points and a *sort* of labels. Likewise, we can define a *sort* of weighted line segments corresponding to W_1 , or a *sort* of linear shapes corresponding to $U_0 \times U_1 \times U_2 \times U_3$.

Two remarks are in order with respect to the conjunctive operation on *sorts*, and its similarity to the Cartesian product. Firstly, the operation is not commutative, thus a *sort* of labelled points is not identical to a *sort* of ‘pointed’ labels. The former *sort* consists of points with associated labels; the latter consists of labels with associated points. This distinction derives from the fact that each *sort*, or algebra, defines its own algebraic operations, based on its own part relationship, and the resulting algebraic operations for the two conjunctive *sorts* may behave differently (see below). Secondly, an element belonging to a conjunctive *sort* necessarily contains elements from each constituent *sort*. For example, a shape belonging to a Cartesian product of algebras, e.g., $U_0 \times U_1 \times U_2 \times U_3$, must contain (spatial) elements from each algebra, otherwise it necessarily belongs to the algebra given by the actual Cartesian product of the algebras of its specific element types.

In order to define a disjunctive *sort* — where neither the ordering of the component *sorts* is important, nor is the explicit presence of elements from the different component *sorts* necessary — we can define a disjunctive operation on *sorts*. Under this disjunctive operation, any element of the resulting *sort* is necessarily an element of a constituent *sort*. The disjunctive operation on *sorts* consequently defines a subsumption relationship on *sorts*: a disjunctive *sort* subsumes each constituent *sort*, because each element of a constituent *sort* is also an element of the disjunctive *sort*.

Subsumption is a powerful mechanism for comparing alternative representations of the same entity. When a representation is subsumed by another, the entities represented by the former can also be represented by the latter representation, without any data loss. There are many representational formalisms that consider the subsumption relationship in order to achieve partially ordered type structures; most are based on first-order logic. Applied to building design, a good example is Woodbury et al. (1999), who adopt typed feature structures as the model for design space exploration. Like many other formalisms, typed feature structures consider a record-like data structure for representing data types. Record-like data structures facilitate the encapsulation of property information in (a variation of) attribute/value pairs (Ait-Kaci 1984). Furthermore, the properties may themselves be typed by type structures, i.e., expressed in terms of record-like data structures, containing (sub-)properties. Then, the subsumption relationship

defines a partial ordering on type structures. Furthermore, the algebraic operations of intersection and union (or others similar) may be defined on type structures so that the intersection of two type structures is subsumed by either type structure, and the union of two type structures subsumes either type structure.

Sorts can be similarly conceived, with at least one exception: the association of properties to a *sort* occurs through the conjunctive operation on *sorts* — that is, each property of a *sort* is itself a *sort*. Primitive *sorts* are the exception to this rule. Like primitive data types, primitive *sorts* are the smallest building blocks for building *sortal* representational structures. A primitive *sort* defines the domain of possible values that elements from this *sort* may hold, e.g., a primitive *sort* of weights specifies the domain of positive real numbers, and a primitive *sort* of line segments specifies the domain of intervals on linear carriers. Primitive *sorts* may be constrained over the extent of their domain, for example, limiting weights to values between 0 and 1. Then, the subsumption relationship between *sorts* derives from the disjunctive operation on *sorts* (similar to a union of two or more type structures) and the expression of constraints on primitive *sorts*. The specification of an intersection-like operation on *sorts* adds no further value because the intersection of two *sorts* can always be reduced, through distributive and associative rules, to the intersection of primitive *sorts*; the intersection is non-empty only when the primitive *sorts* are identical, except for possible constraints (Stouffs and Krishnamurti 2002).

Another important distinction is the fact that first order logic-based representations generally consider an open world assumption — that is, nothing can be excluded unless it is explicitly excluded. For example, consider polygon objects that may have a colour assigned. When looking for a yellow square, a square without any colour specified is considered a potential solution — unless, it has another colour explicitly specified, or it is otherwise known not to have the yellow colour. The fact that a colour is not specified does not exclude an object from potentially being yellow. *Sorts*, on the other hand, hold to a closed world assumption. A polygon only has a colour if one is explicitly assigned: when looking for a yellow square, any square will not do, unless it has the yellow colour assigned. This restriction is commonly used in shape grammars to constrain emergence. More specifically, labelled points commonly serve to constrain the applicability of shape rules, which encapsulate both shape recognition (emergence) and shape transformation (computational change). *Sorts* provide a component-based approach to developing grammar systems, utilizing a uniform characterization of grammars (Stouffs and Krishnamurti 2001b). Another way of looking at this distinction between adopting the open or closed world assumption is to consider their applicability for knowledge representation. Logic-based representations are developed essentially for representing knowledge. *Sorts* are intended only to represent data and information — any reasoning (computational change) is based purely on present (or emergent) information.

2. A constructive approach to *sorts*

A *sort* constitutes the basic entity for our formalism. Conceptually, a *sort* defines a set of similar data elements, e.g., a class of objects, or a set of tuples solving a system of equations. For example, points and lines are each a *sort*, and so are plane and volume segments. *Sorts* are not limited to geometrical objects, colours are a *sort*; other data types too define *sorts*. Elementary data types define the basic building blocks for the construction of *sorts*; these building blocks are denoted as *primitive sorts*. Primitive *sorts* combine to form *composite sorts* under compositional operators over *sorts*.

The *attribute* operator (corresponding to the operation of Cartesian product), specifies a conjunctively subordinate composition of *sorts*. The resulting representational structure is a combination of both operand structures under the familiar object-attribute relationship. For example, a *sort* of labelled points is specified as a *sort* of points, with one or more labels assigned to each point in the data collection. The attribute operator is non-commutative.

The *disjunctive* operator allows for disjunctively co-ordinate compositions of *sorts*: the resulting data collection combines the different types of data elements from its operand *sorts*, without imposing any hierarchical relationships. The resulting representational structure distinguishes all operand structures such that each data element belongs explicitly to one of the operand *sorts*. For example, a *sort* of points and lines distinguishes each data element as either a point or a line. The disjunctive operator is commutative.

The algebraic model implies the specification of a part relationship on the elements within a *sort*. This part relationship not only governs when one element is a part of another but, as a result, also how elements combine and intersect, and what the result is of subtracting one element from another or from a collection of elements from the same *sort*. Since this part relationship may be freely defined, as long as it constitutes a partial order, different part relationships may be applied to different *sorts*, reflecting on a desired behaviour for the *sorts*' elements. For example, points and labels adhere to a discrete (or set-like) behaviour, with the part relationship corresponding to the subset relationship on mathematical sets; weights (e.g., line thickness or surface tones) adhere to an ordinal behaviour, with the part relation on weights corresponding to the less-than-or-equal relation on numeric values; and line segments and other one-dimensional quantities, such as time, adhere to an interval behaviour (Stouffs and Krishnamurti 2002).

Behaviours also apply to composite *sorts*, that is, a part relationship can be defined for data elements belonging to a *sort* defined under the attribute or disjunctive operator. Specifically, a composite *sort* inherits its behaviour from its component *sorts* in a manner that depends on the compositional relationship.

Under the attribute operator, the composite behaviour is that of the first component *sort*. The attribute operator specifies a dependency relation on component *sorts*, where each component, except for the first, defines a *sort* of attribute data to the previous component. The corresponding data collection consists of data elements of the first component *sort*; each element then has, as attribute, a data collection corresponding to the *sort* as a composition of all but the first component, in a recursive manner. Thus, a data element is part of a data collection, composed under the attribute operator, if it is a part of the data elements of the first component *sort*, and if it has an attribute collection that is a part of the respective attribute collection of the data element(s) of the first component *sort* it is a part of. When data collections of the same composite *sort* (under the attribute operator) are pairwise summed (differenced or intersected), identical data elements merge, and their attribute collections combine, under this operation. All elements with empty attributes are, necessarily, removed.

Under the disjunctive operator, the behaviour is that of the component *sort* for each component — a data element is part of a disjunctive data collection if it is a part of the partial data collection of elements from the same primitive *sort*. In other words, data collections from different component *sorts*, under the disjunctive operator, never interact; the resulting data collection is the set of collections from all component *sorts*. When the operation of addition, subtraction or product is applied to two data collections of the same disjunctive *sort*, the operation instead applies to the respective component collections.

Behaviours play an important role when assessing data loss in data exchange between different *sorts*. When reorganizing the composition of components *sorts* under the attribute operator, the corresponding behaviour may be altered in such a way as to trigger data loss. Consider a *sort* of weighted points, i.e., a *sort* of points with attribute weights, and a *sort* of pointed weights, i.e., a *sort* of weights with attribute points. A collection of weighted points defines a set of non-coincident points, each having a single weight assigned (possibly the maximum value of various weights assigned to the same point). These weights may be different for different points. The behaviour of the collection is discrete. On the other hand, a collection of pointed weights, which is defined as a single weight (which is the maximum of all weights considered) with an attribute collection of points, has an ordinal behaviour. In both cases, points are associated with weights. However, in the first case, different points may be associated with different weights, whereas, in the second case, all points are associated with the same weight. In a conversion from the first to the second *sort*, data loss is inevitable.

A behavioural specification is also a prerequisite for a uniform handling of different and a priori unknown data structures. Consider the association of building performance data to design geometries. The behaviour of these data, as a result of an alteration to the geometry, can be expressed through a number of operations chosen to match the expected behaviour. When an application receives the data together with its behavioural specification, the application can correctly interpret, manipulate, and represent this information without unexpected data loss.

3. Computationally recognizing emergent information

Recognizing emergent information requires matching a specified information pattern onto a similar part of the original information. This match is determined under an allowable transformation. The kinds of transformations that are allowed depend on the kind of information that is being recognized. For example, for spatial recognition, this transformation is geometric, commonly, a Euclidean transformation: a square must be computationally recognized as a square irrespective of scale, orientation or location. Similar transformations can be considered for other kinds of information. For example, search-and-replace functionalities in text editors allow case transformations of the constituent letters. The part relationship underlying the behavioural specifications for *sorts* enables the matching problem to be implemented for each primitive *sort*. Composite *sorts* inherit their behaviour and part relationship from their component *sorts* — thus, any technical difficulties in implementing matching apply just once, for each primitive *sort*.

Recognizing emergent information is useful if the emergent information is subsequently the subject of an operation or manipulation. This is particularly true of design, where creativity relies on a restructuring of emergent information. Data recognition and subsequent manipulation can be considered part of a single computation $s - f(a) + f(b)$, where s is a data collection, a is a representation of the data pattern, f is a transformation under which a is a part of s , and $f(b)$ is the data replacing $f(a)$ in s . See, for example, Krishnamurti and Stouffs (1997) for a discussion on spatial change. $s - f(a) + f(b)$ is an expression of computational change and can also be written as a design rule: $a \rightarrow b$. Rule application, then, consists of replacing the emergent data corresponding to a , under some allowable transformation, by b , under the same transformation.

Rules can be grouped (into grammars). A grammar is a formal device for the specification of a certain language, which is the set of all designs generated by the grammar. Each generation of a design in the language starts with an initial design, and employs the rules to achieve a design that contains only elements from a given terminal vocabulary. In spatial design, the specification of (spatial) rules and grammars leads naturally to the generation and exploration of possible

spatial designs. According to some (Mitchell 1993; Stiny 1993; 1994; and more recently, Woodbury and Burrow 2004), spatial elements or shapes emerging under a part relation is highly enticing to design search.

The concept of search is much more fundamental than any generational form alone might imply. Any mutation of a data collection into another, or parts of others, constitutes an action of search. As such, a rule may be considered to specify a particular composition of operations and/or transformations that is recognized as a new, single, operation and applied as such. Rules can serve to facilitate common operations, e.g., for changing one data collection into another or for creating new design information based on existing information in combination with a rule. Similarly, a grammar is more than a framework for generation; it is a tool that permits a structuring of a collection of rules or operations that have proven their applicability to the creation of a certain set (or language) of designs or the derivation of certain information. Stouffs and Krishnamurti (2001b) present a few examples of design grammar formalisms that can be expressed using *sorts*.

4. Future work

The research described here is part of the Informatics research programme at the Faculty of Architecture, Delft University of Technology. This research programme focuses on the use of various aspects of Information, Communication and Knowledge Technology (ICKT) to support collaboration and communication in the building sector.

The research described here is also partly the result of a collaboration with researchers at Carnegie Mellon University. Park and Krishnamurti (2004) investigate the application of *sorts* to a case study from the construction domain considering the comparison of as-designed and as-built building models. In this case study, the as-designed model is a 3D design model obtained from a commercial parametric design software in the form of an IFC file, and the as-built model is a 3D geometric model derived from laser scanning results. The particular focus in this project is on the integration of different models into an integrated project model, that allows for both models to be compared in order to look for discrepancies outside of allowable tolerances described in the construction specifications, and that is designed to capture both changes in the drawings (as-designed) and changes on a given construction site (as-built).

Future work at Delft concerns the specification of a query language through the embedding of numeric functions into the representational structure that supports the derivation of new information. It is our intention to apply this query language to Industrial, Flexible and Demountable (IFD) building models for the extraction of such information as quantities and distances for various purposes.

5. Acknowledgments

I'm indebted to Ramesh Krishnamurti, with whom I collaborate in this research. The research has been partly funded by the Netherlands Organization for Scientific Research (NWO), grant nr. 016.007.007, whose support is gratefully acknowledged. Any opinions, findings, conclusions or recommendations presented in this paper are those of the author and do not necessarily reflect the views of the Netherlands Organization for Scientific Research.

