CAAD Futures 1997 (ed. R. Junge), pp.553-564, Kluwer Academic, Dordrecht, The Netherlands, 1997.

SORTS : A CONCEPT FOR REPRESENTATIONAL FLEXIBILITY

RUDI STOUFFS
Architecture and CAAD, Swiss Federal Institute of Technology Zurich
RAMESH KRISHNAMURTI
Department of Architecture, Carnegie Mellon University, Pittsburgh

This work is based on the recognition that there will always be a need for different representations of the same entity, albeit a building or building part, a shape or other complex attribute. This exigency ensues, formally, to define the relations between alternative representations, in order to support translation and identify where exact translation is possible, and to define coverage of different representations. We consider an abstraction of representations to model sorts that allows us to define algebraic operations on sorts and recognize algebraic relationships between sorts, providing us with a method for the analysis of representations, and the comparison of their coverage. We present the basis of support for a multi-representational environment.

1. Motivation

As computers become more powerful, we envision new domains, situations and environments where the computer can play an important role as a facilitator. This requires the integration and manipulation of increasing amounts of knowledge and information. In a collaboration in space and time, knowledge may be combined from multiple domains and exchanged between multiple disciplines; information may be provided and requested by all participants in this activity process; and information may change during the life-cycle of a designed artifact and because of advancements in knowledge and technology. Such collaboration creates additional impediments to effective computer support. The transfer of knowledge and information between multi-disciplinary partners and the interpretation of this data are non-trivial tasks.

Exchange of Information. A key problem is the loss of information in data exchange between representations supporting different views or abstractions of the same artifact. Within the domain of geometric modeling, many different representations have developed over the past thirty years, based on a variety of models, associated operations and underlying concepts. At the same time, much effort has been spent in order to arrive at a single representational standard for design and engineering data/tasks. The IGES (Smith et al., 1988) and STEP (Bloor, 1991) projects are prime examples. However, no single geometric representation exists to solve every problem. Over time, new design and evaluation methods develop that require new information and extended

representations. For example, the ever increasing concern for performance issues in building or engineering design necessitates the development of extended representations and manipulations of design geometries, including form and material properties, specified at various levels, in a hierarchical manner with dynamic relationships.

Multiple Representations. We strongly believe that there will always be a need for different representations of the same entity, albeit a building or building part, a shape or other complex attribute. Rather than provide specific applications for the translation between alternative representations that may serve the same or similar purpose, the need exists, formally, to define relations between alternative representations, to define coverage of different representations and to define where exact translation is possible. Our objective is to offer general translational support that recognizes the coverage of and relations between different representations, and to support different representations within a single, but, possibly locally or temporally disconnected environment (Stouffs et al., 1996). Such an environment must be able to identify when and where exact translation is possible so that the data-flow can be assessed and data-integrity can be monitored.
We conceive an algebraically based formalism that provides a handle for dealing with, operating on, and interrelating representations. This abstraction, termed *sorts*, defines formal operations on sorts and recognizes relationships between sorts, providing us with a method for the analysis of representations and the comparison of their coverage. Sorts are distinguished by their component sorts, their compositional relationships, and their assigned names. The manner in which sorts relate depends on the manner of their composition. Alternative compositions of the same component sorts give rise to alternative views of the same data, and can be derived from one another. Algebraically, sorts incorporate all embedded views, and contrary to current CAD approaches, sorts do not impose an object/geometry-centered view. For instance, by specifying a compositional hierarchy of the component sorts using a dependency relationship, the top component naturally becomes the focus of the information set. By altering the dependency relations, representations can be restructured to reflect any particular view of the information, adapting the organizational structure to the informational purpose.

Alternative Concepts and Ontologies. Probably the first and foremost problem associated with an all-encompassing representational model is the imposition of a fixed frame of reference. Instead, different disciplines adopt different ontologies; the same term may prescribe different meanings, the same concept may define alternative representations. Such semantic incompatibilities provoke data communication problems that may prevent collaboration at its fullest potential. The concept of sorts only provides for a common syntax, allowing for different vocabularies and languages to be created, and providing the means to develop translational facilities between these. For example, a point may be specified with any number of coordinates depending on its dimensionality, its coordinates may constitute integers, reals or rationals, these may be bounded in space, etc. Sorts can be defined accordingly, compared and related, and translational support provided for. There is no imposition of concepts beyond the

purely syntactical, and the alphabet of building blocks for the vocabulary definition can be readily extended at all times.

Extensibility and Adaptability. No language thus created ever needs to be static. A vocabulary may be extended from the existing alphabet or using newly developed building blocks. Augmented representations that provide support for extended information and technological advances can be achieved by combining sorts into a new composition or by specifying additional component sorts to an existing composition. Far from having to redevelop not only the data but also the applicative operations, the concept of sorts aims to provide almost continuous support to evolving representations, providing for an environment that supports exploration and trial, even with respect to the representation. Data can be readily converted to new and extended (or condensed) representations, procedural operations may remain applicative if written with flexibility in mind. This extensibility provides ample support to explore extended representations for such purposes as building performance evaluation or the inclusion of design history and design intent into an artifact's design information.

2. Definition

Conceptually, a sort specifies a set of similar models that can be described in terms of a single set of equations. We denote the system of equations for a sort its *characteristic individual*. The models of a sort constitute its *individuals*; a *form* is any group of individuals of the same sort. Given any system of equations, the corresponding set of all models described by these equations defines a sort. Then, each individual of the sort is specified by assigning a value to each of the equational parameters. For example, a point is specified by its tuple of coordinates. From a purely conceptual point of view, sorts can be related by comparing their systems of equations, in a mathematical manner. For instance, since each equation constraints the values its parameters may adopt, a sort *subsumes* another sort if its equations form a subsystem of the other sort's equations. Note that while a sort commonly specifies a continuum of models, the extent of this continuum can easily be altered with respect to any conceptualization or purpose, by adding to and/or removing from its system of equations.

In practice, we specify the characteristic individual of a *primitive sort* as an elementary data type. Primitive sorts combine to *composite sorts*, using the formal operations to specify the compositional relationships. For instance, the *attribute* operator provides for (recursively) subordinate compositions of sorts using an object-attribute relationship in both a one-to-one and a one-to-many instantiation. 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 form. The operation of *sum* allows for co-ordinating, disjunctive compositions of multiple sorts, under many-to-one and many-to-many instantiations, where each sort may -but does not have to- be represented in the data form. For example, a shape rule has both a *lhs* (left-hand-side) and *rhs* (right-hand-side) shape, either of which can be omitted. Finally, naming sorts provides for a semantic-like differentiation of sorts that

may otherwise be syntactically identical, e.g., *lhs* and *rhs* denote equivalent -not identical- sorts.
While an analysis of the representational structure may be sufficient for a one-way translation, in order to provide for continuous support for data exchange in a multi-representational environment, some domain-specific knowledge is additionally needed. Consider the conversion and subsequent re-conversion of some data between two sorts. At least a canonical representation for individuals and forms is required in order to control information-loss in a strict sense. If the data is also altered between conversion and reconversion, additionally, appropriate ways of manipulating the data within the conceptual framework of the sorts are needed to achieve the same level of control. While certain other purposes or cases may demand even further domain-specific knowledge, providing uniform ways of handling different and a priori unknown data is a necessary condition for correct translation.
As an example, consider the association of building performance data to design geometries. The behavior of these data as a result of alterations to the geometries can be expressed through a number of operations chosen to match the expected behavior. When data is presented into a collaborative environment, the applicable manipulative operations are provided as well, allowing any application to properly interpret, manipulate and re-present this information without unexpected data loss. Therefore, the definition of a sort includes a specification of the operational behavior of its individuals for common arithmetic operations. The foundation of this approach is an algebraic model for shapes (Stouffs, 1994) that offers a uniform and consistent approach for dealing with geometries of mixed-dimensionality and non-spatial attributes, using common arithmetic operations of sum, difference and product (intersection) (Stouffs and Krishnamurti, 1996, discuss the adaptability of the algebraic model to domain-specific functionality). Extending this model to sorts, these serve as an abstraction of representations that enable us to compare multiple representations in terms of their coverage and thus support translation.

Notation. In the sequel, we adopt the following notation for sorts and their definition. We reserve the letters *a* through *h* to denote sorts (i.e., their names) and *i* through *l* for characteristic individuals (as elementary data types). A primitive sort is specified by a name, a characteristic individual and, optionally, a number of arguments dependent on the particular characteristic individual (see section 4 for some examples). We write `a : [i]` or `b : [j](x)`. In some cases, a primitive sort specifies multiple aspects, each of which is assigned a name and considered a sort. The actual primitive sort only exists as a linking construct. We write `(c, d) : [k](y, z)`. The definition of a composite sort similarly consists of a name and an expression in terms of its component sorts using the symbols '^' and '+' to denote the respective attribute and sum operations. Parentheses provide for the nesting of definitions. We write `e : b + c` and `g : f ^ e`, `g : f ^ (e : b + c)`, or `g : f ^ (e : (b : [j](x)) + c)`.

3. Matching

Compositions of sorts can be compared and potential matches classified according to their similarity, in rough terms, as equivalent, similar (as composed of equivalent sorts) and convertible. A more detailed classification is provided in table 1, using a numerical ordering system, termed *levels*, to differentiate the matches computationally. Different integral levels specify different matching types, decimal levels allow for finer comparisons. Equivalent as well as similar sorts guarantee correct conversion of the data without information loss, except semantically. Incomplete or partial conversions, either through augmentations (by adding sorts under the attribute operation) or diminutions (by removing sorts under the attribute operation) always result in data loss. Whether data-loss occurs in complete conversions, i.e., through rearrangements of the component sorts (by inverting attribute relationships), depends on the behavioral categories of the constituent sorts. It also depends on conversions between primitive sorts that share the same or similar characteristic individuals, but differ in one or more constraints or arguments (i.e., equations). Finally, the operation of sum specifies a subsumption relationship on sorts, where one sort may match a part, under sum, of another sort, providing an additional (perpendicular) grading scheme.

level	**match**	**interpretation**	**example**
= 0.0	identical	semantic equality	$a \leftrightarrow a$
< 1.0	equivalent	semantic derivability	$a \leftrightarrow b : a$
< 2.0	strongly similar		$a \wedge b \leftrightarrow a\ (c : b)$ $a + (d : b + c) \leftrightarrow$ $(e : a + b) + c$
< 3.0	weakly similar	syntactic equality	$a : [i] \leftrightarrow b : [i]$
< 4.0	convertible	syntactic convertibility	$a \wedge b \leftrightarrow b \wedge a$ $a : [i](x) \leftrightarrow b : [i](y)$
< 5.0	incomplete • augmentation • diminution	partial convertibility	 • $a \rightarrow a \wedge b$ • $a \wedge b \rightarrow a$
= 5.0	incongruous	no valid conversion	

Table 1. Integral levels of sort matching.

This approach allows us to monitor data-integrity during the design process, at all times, for a large variety of data. Specifically, we know that data-integrity is maintained for

each data sort under any of the formal operations on data forms within this sort; the coverage of data sorts can be compared; data can always be moved from more-restrictive to less-restrictive sorts without data loss; and data loss can be measured when moving data in the opposite direction. Active control over which conversions should and should not be allowed or considered is presented to the user in the form of a level tuner (see table 2): three user-defined levels specify level intervals of predefined handling behavior. Additional control is envisioned using rules of exception.

level	**handling**
$< l_1$	sorts are considered equal, conversion is performed without notification
$< l_2$	user is notified of conversion
$< l_3$	user's approval is requested for conversion
Š l_3	conversion is not allowed, unless upon user's specific initiation

Table 2. Level tuner.

4. Exemplar Sorts

With the arrival of the world-wide web into the architectural domain, (virtual) architecture has conquered new territories, presenting the challenge to support, newly, any and all data types that can be envisioned. Rather than supplying slots for generic entities in a database-like fashion, sorts provide the ability to select the appropriate operational behavior, resulting in a more intelligent integration of the entities in the information environment. Currently, we consider characteristic individuals for geometric entities (i.e., points, lines, line segments, planes, plane segments and volumes), attribute entities (e.g., labels, weights, identifiers and signs), relational entities (e.g., properties) and hypermedia entities (e.g., images, text). Below, we describe some of these characteristic individuals, with their behavior, in detail.

Sorts of Labels and Points. A sort of labels provides the simplest example. A label, a string of characters, can be created, deleted or, as a combination of both, altered. When creating/adding a label to a form of labels, it becomes an additional element in the set that represents this form. When deleting/subtracting a label, the element is removed from the set. The algebraic operations behave as set operations; an empty set specifies an empty form. The resulting behavior is termed *discrete*. Discrete behavior ensures that two individuals are combined into one, only if these are identical. Another example of discrete behavior is a sort of points. Though a point can and may be considered as a composite sort, i.e., corresponding to a tuple of coordinates, specifying a characteristic

individual for points trivializes consistency checking and generally simplifies data management and retrieval. For example, a one-step creation ensures that each point has the exact number of coordinates. The conversion of such a point into any other description, e.g., as a composite sort of coordinates, can be added easily.

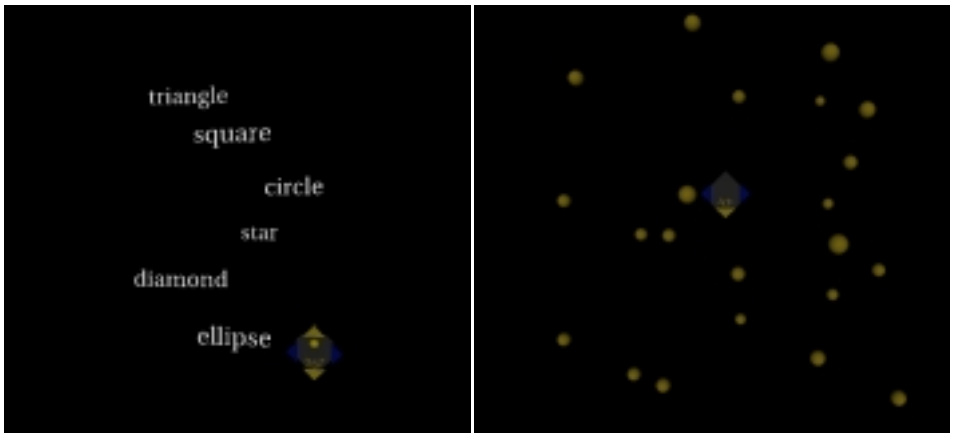

Figure 1. Visualization of forms of labels and points in VRML. Bige Tunçer.

Sorts of Images. An example of where such conversion is already provided, while it is both commonplace and necessary, is a sort of images. The characteristic individual for images, referencing an image, e.g., filename, accepts an argument specifying the image type, e.g., GIF, JPEG and TIFF. This argument to the sort's definition provides the ability to create separate sorts for gif-images, jpeg-images and tiff-images. In notational form, these become:

```
gif-images : [Image](GIF)
jpeg-images : [Image](JPEG)
tiff-images : [Image](TIFF)
```

Provided that an application(s) or procedure(s) is available for the bytecode-wise conversion of images between different types, the characteristic individual for images specifies a conversion operation that can be invoked automatically. For example, in order to convert a form of images to "web-able" images, it suffices to define sorts of

```
images : gif-images + jpeg-images + tiff-images
web-images : gif-images + jpeg-images
```

and `convert` the form of images to a form of web-images.

Sorts of Properties. More complex representations often include relationships between data entities, for instance, for the purpose of referencing library entities, or for specifying a relational system between information aspects and artifacts. For example, a boundary representation for solids specifies vertices, edges, faces and solids, with edges linking vertices, edges bounding faces, and faces bounding solids. A

characteristic individual for properties provides similar capabilities to sorts. A *property* is a two-way link between two individuals of given sorts. In order to prevent information-loss when handling properties and their associated individuals, an individual must know of any properties that link it to other individuals. All properties of the same sort thus assigned to an individual make up an attribute form with respect to the individual. Appropriate sorts must be defined accordingly under the attribute relationship. For example, a polygonal boundary representation with vertices and edges may define the following sorts:

```
points : [Point]
line-segments : [LineSegment]
(endpoints, segments) : [Property](line-segments, points)
vertices : points ^ segments
edges : line-segments ^ endpoints
```

An edge is defined as a line segment with (two) endpoints; a vertex is a point with any number of segments originating from it. The operational behavior of properties, termed *relationship*, is an extension of the discrete behavior, incorporating appropriate data-management procedures and ensuring data-consistency. For example, when deleting or otherwise modifying either individual associated to a property, this property is deleted or modified correspondingly.

5. Behavioral Categories

Most computer aided design applications, currently, adopt an object-behavioral approach. This approach specifies that once an entity is created (as an object), it remains unaltered except through explicit user intervention. Such behavior closely resembles the discrete behavior. The latter specifies that two individuals combine into one, only if these are identical. An object-oriented behavior can be achieved for any sort, by combining this with a sort of (unique) identifiers under the attribute relationship. The resulting data form is akin to a database of individuals, where each individual has a unique key assigned.

Often, a sort may profit from a more complex behavior that induces additional strengths. For instance, under the discrete behavior, a conscientious decision is required from the user on any change to the individual. This does not readily support creativity and novelty which rely on the concept of emergence, i.e., the recognition of information components and structures that are not explicitly present in the information and its representation, and on the restructuring of information (Stiny 1993; Krishnamurti and Stouffs, 1997).

Computationally recognizing emergent structures requires determining a transformation under which a specified similar structure is a *part* of the original structure (Krishnamurti and Earl, 1992). This part relationship can be freely defined, as long as it constitutes a partial order relationship. The algebraic model (Stiny, 1991; Stouffs, 1994) is based on such a part relationship. Under the algebraic model, a form is specified as an element of an algebra that is ordered by a part relation and closed under the algebraic (sum,

difference and product) operations and (affine) transformations. Fundamental to the algebraic model is that under the part relation, *any* part of a form is a form. As such, a form specifies an infinite set of (sub)forms that are each part of the original form, and users can deal with forms in indeterminate ways. The *maximal element representation* (Krishnamurti, 1992; Stouffs, 1994) captures this notion. Figure 2 illustrates this definition of a form, under the part relation, with a form of line segments. The six maximal line segments can be grouped to three by three to form two triangles. However, other interpretations are possible and up to five triangles can be recognized in the shape and manipulated as such.

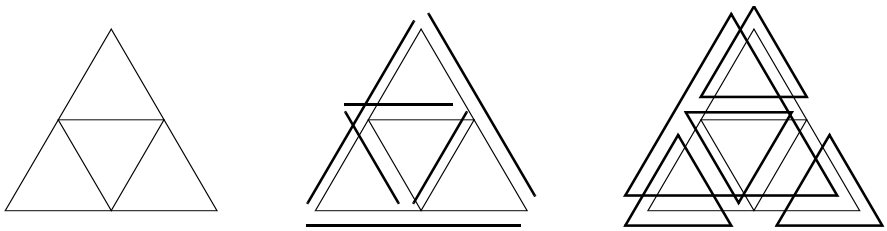

Figure 2. A shape with six maximal lines, specifying five different triangles.

Interval Behavior. The maximal element representation for line segments specifies an individual as an interval segment on an infinite line carrier, and a form as a (minimal) set of such individuals. A form is termed maximal if no two segments on the same carrier touch or overlap, i.e., these must be disjoint. Any two non-disjoint individuals are combined into one. Then, an individual is a part of another individual if its segment is embedded in the other segment on the same carrier. The algebraic operations on forms differ in their behavior from the discrete behavior in that individuals interact, not only if these are identical, but also if these overlap or touch.

Ordinal Behavior. Stiny (1992) explores the application of the maximal element representation to geometries with weights as attributes. Weights may be considered to denote thicknesses for points and lines, or tones for planes and volumes. A behavior for weights becomes apparent from drawings: a single line drawn multiple times, every time with different thickness, appears as it was drawn once with the largest thickness, even though it assumes the same line with other thicknesses. Thus, unlike behaviors described above, weight individuals from the same form always combine into a single individual. This (maximal) individual has as weight value the least upper bound of all the individuals' weight values, i.e., their maximum value. This behavior is termed *ordinal*; using numbers to represent weights, the part relation on weights corresponds to the less-than-or-equal relation on numbers.

Behavior of Compositions. A composite sort inherits its behavior from its component sorts in a manner that depends on the compositional relationship. Under the operation of sum, the behavior is that of the component sort for each component. Forms from different component sorts never interact, the resulting form, termed *metaform*,

corresponding the composite sort, is the group of forms for all component sorts. When adding an individual to a metaform, this individual gets added to the appropriate component form. When an algebraic operation applies to two metaforms of the same sort, the operation instead applies to the respective component forms. A metaform is empty only if all component forms are empty.
The attribute operation on sorts specifies a dependency relation on the sorts in a composition, where each component, except the first, defines an attribute sort to the previous component. That is, a corresponding form consists of individuals of the first component sort each of which has, as attribute, a form corresponding to the sort as a composition of all but the first component, in a recursive manner. Thus, the behavior of such a sort is defined by the behavior of its first component sort. Specifically, when an algebraic operation applies to two forms of the same composite sort (under the attribute relationship), identical individuals merge and their attribute forms combine under the same algebraic operation. Any individuals that have an empty attribute form are removed from their respective forms.
For example, a form of line segments with attributes, corresponding to a composite sort under the attribute relationship, is maximal if no two segments on the same carrier overlap and any two segments that touch have non-equal attribute forms. Then, an individual is contained in a form if a discrete classification of this individual into component individuals can be found such that each component is a part of an individual in the form, and the respective attribute forms adhere to the appropriate part relation.

Singly-associated versus multiply-associated sorts. Behaviors play an important role when assessing data-loss in information exchange between different sorts. Reorganizing component sorts under the attribute relationship into a different compositional hierarchy may alter the corresponding behavior and trigger data-loss. Consider a sort of weighted points, i.e., a sort of points with attribute weights, and a sort of points of weights, i.e., a sort of weights with attribute points. A form of the former sort is a set of non-coincident points, each of which has a single weight assigned. These weights may be different for different points. The resulting behavior of the form is discrete. A form of the latter sort is composed of a single weight with an attribute form of points, and has ordinal behavior. 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.
Sorts with ordinal behavior are also denoted *singly-associated* sorts. A form of a singly-associated sort necessarily contains only a single individual, unless empty. In contrast, forms of *multiply-associated* sorts can contain any number of individuals. Sorts with discrete, relationship or interval behavior are all multiply-associated sorts. The previous exposition always applies to compositions of both singly-associated *and* multiply-associated sorts under the attribute relationship.

6. Discussion

Our intention is not to create an all-encompassing representation as a new standard that would be "up to date" as to current research and usage. Given the sheer variety of representations available and used, any single standard, however flexible, will always typify a rather subjective evaluation that is indifferent to the purposes of the different representations. Moreover, as technology evolves and knowledge increases, any standard would become quickly outdated. Indeed, solid modelers with new capabilities are being developed, such as the parametric solid modelers found in Pro-Engineer™, SDRC's IDEAS™ and Autodesk's DESIGNER™ for which no adequate translators exist.

Instead, we propose an alternative approach that is not restricted to currently known representations and that may take away the need for standards altogether. Conceptually, we distinguish the data classes and subclasses that constitute a representation and consider a multi-way communication system using these data classes as the vocabulary elements (see also Stouffs et al., 1996). This vocabulary is not a priori limited and may be extended at all times by any application. Using sorts to achieve such a communication system, new applications that adopt sorts as their representational framework may synthesize any or all of the sorts' strengths into their functionality. How and whether these strengths are presented to the user depends on this functionality. Redeveloping an existing application to profit from sorts, without altering the original functionality, only extends the application with readily accessible communicational capabilities, but does not provide a glimpse of the flexibility that is intrinsic to sorts. How such flexibility can be presented to the user, in the best and easiest manner, makes for an interesting question on its own. Currently, we are developing database support for persistent information, data visualization tools in VRML, and a JAVA-based interpreter/interface.

Sorts were conceived in order to respond to and deal with issues of data-loss in information exchange, mainly between existing applications. These, primarily, profit from a framework for data communication constructed on sorts. When dealing with a substantial number of applications and corresponding representations, all of which may not be known at the time of development, it becomes inefficient to develop a tool for every (possible) channel linking two applications. Instead, consider a support system for data exchange consisting of a node for facilitating data transfer and translation, and a query language for communication between the node and each application. Such a system only requires from each participating application a (single) communication front-end that can pose and answer queries to and from the node. For an example, consider the domain of solid modeling. Whereas different boundary representations specify different relational systems to link the boundary entities, the entity classes of vertices, edges, faces and solids are common to the domain and, in some variations, to the representations. Using a domain-specific query language that understands these concepts, and the node's ability to explore different entity relationships, a multi-way communication system can be established that supports solid representations (Stouffs et al., 1996, present a theoretical analysis of solid representations for this purpose).

Additional non-geometric data can be treated as attributes to the geometric entities and communicated as such.
Sorts present a method for the analysis of representations, and the comparison of their coverage. A multi-way communication system based on sorts provides the ability to identify when and where exact translation is possible. While we cannot guarantee complete and correct translation, at least, data-flow can be assessed and data-integrity monitored.

Acknowledgments

The first author wishes to thank Kuk-Hwan Mieusset and Bige Tunçer for their contributions to the current developments, respectively, database support and data visualization. He also wishes to thank Gerhard Schmitt for his undaunted belief and support.

Bibliography

Bloor, M.S. (1991) STEP-standard for the exchange of product model data, Standards and Practices in Electronic Data Interchange, IEE Colloquium, 2/1-3, The Institution of Electrical Engineers (IEE), London.

Krishnamurti, R. (1992) The maximal representation of a shape, Environment and Planning B: Planning and Design 19, 267-288.

Krishnamurti, R., and Earl, C.F. (1992) Shape recognition in three dimensions, Environment and Planning B: Planning and Design 19, 585-603.

Krishnamurti, R., and Stouffs, R. (1997) Spatial change: continuity, reversibility and emergent shapes, Environment and Planning B: Planning and Design 24.

Smith, B., Rinaudot, G.R., Reed, K.A., and Wright, T. (1988) Initial Graphics Exchange Specification (IGES), Version 4.0, SAE/SP-88/767, Society of Automotive Engineers, Warrendale, Pa.

Stiny, G. (1991) The algebras of design. Research in Engineering Design 2, 171-181.

Stiny, G. (1992) Weights, Environment and Planning B: Planning and Design 19, 413-430.

Stiny, G. (1993) Emergence and continuity in shape grammars, CAAD Futures ‘93 (eds., U. Flemming and S. Van Wyk), 37-54, North-Holland, Amsterdam.

Stouffs, R. (1994) The Algebra of Shapes, Ph.D. dissertation, Departement of Architecture, Carnegie Mellon University, Pittsburgh, Pa.

Stouffs, R., and Krishnamurti, R. (1996) The extensibility and applicability of geometric representations, 3rd Design and Decision Support Systems in Architecture and Urban Planning Conference, Architecture Proceedings, 436-452. Eindhoven University of Technology, Eindhoven, The Netherlands.

Stouffs, R., Krishnamurti, R., and Eastman, C.M. (1996) A formal structure for nonequivalent solid representations, IFIP WG 5.2 Workshop on Knowledge Intensive CAD II (eds. S. Finger, M. Mäntylä and T. Tomiyama), International Federation for Information Processing, Working Group 5.2.