Computer-Aided Civil and Infrastructure Engineering **22** (2007) 98–116

Sortal Structures: Supporting Representational Flexibility for Building Domain Processes

Rudi Stouffs*

Faculty of Architecture, Delft University of Technology, PO Box 5043, 2600 GA Delft, The Netherlands

&

Ramesh Krishnamurti & Kuhn Park

School of Architecture, Carnegie Mellon University, Pittsburgh, PA

Abstract: *We present a formal approach to representational flexibility,* sorts, *to support alternative representations of an entity. The approach is constructive, based on a part relation on elements within a* sort, *which enables the recognition of emergent information. The use of data functions as a* sort *provides for the embedding of data queries within a representational structure. We discuss the application of* sorts *to supporting alternative data views, illustrating this through a case study in building construction.*

1 INTRODUCTION

In the building domain, there is always a need for multiple representations of the same entity. A building may be considered in its entirety, as a shape, a collection of parts or some grouping of properties. The building domain is multi-disciplinary, involving participants, knowledge and information from various specializations; problems require a multiplicity of views, each distinguished by particular interests and emphases. In this article, we consider the representation of such views.

Problems in the building domain are mainly data-driven. Important issues that arise in solving building domain questions stem from knowing what kind of data exists, how effectively can the data be queried and more over, as data is shared across different problem solvers, how effectively can the data be structured to suit a specific need. We have been working on a formal approach, named *sorts*, to deal with these issues, one that looks at data as opposed to knowledge as the driving force in representations

*To whom correspondence should be addressed. E-mail: *r.m.f.stouffs@tudelft.nl*.

2 MULTIPLE REPRESENTATIONS IN THE BUILDING DOMAIN

The architect is typically concerned with configurational and aesthetic aspects of a design, the structural engineer considers structural members and their relationships, and the performance engineer is interested in thermal, lighting, or acoustical performance. Each actor takes their own professionally oriented view—derived from an understanding of current problem solution techniques in their respective domain—that requires a different representation of the same (abstract) entity. Each view specifies a particular selection of information, presented in a particular way. As such, different views—or different representations—may derive from different design stages, and may also support different persons or applications within the same design stage. 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 certainly true in architecture, where the design process, by its exploratory and dynamic nature, invites a variety of approaches and representations (see, for example, Kolarevic, 2000).

There has been concerted effort in developing integrated product models that span multiple disciplines, multiple methodologies and support different views (e.g., Kiviniemi, 2006). These models 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, 2006). These efforts characterize an 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, mainly independent of any project specifics (Stouffs and Krishnamurti, 2001). We consider here how data can be effectively structured a posteriori.

2.1 Design representation and creativity

In general, integrated product models do not support the creative aspects of building design, especially, in early design phases. This is because design creativity often relies on restructuring information as yet not captured in a current information structure—that is, *emergent* information, as, for example, when the design provides new insights that lead to a new interpretation of the constituent design entities. To some extent, creativity can be supported by descriptions of design entities, which have "known" parts. When a design description is considered to have indefinite parts, new design entities become recognized as alternative collections of these parts, and the description can be reinterpreted as composed of different new "known" design entities. These descriptions can be augmented with properties that have definite descriptions, again, with definite or indefinite parts. Classic representation schemes deal with definite descriptions, generally with definite parts, and properties are certainly expressed as definite descriptions with definite parts. The classic product modeling approaches employed in CAD require specification of design entities as objects (with properties) that are maintained at all times, unless explicitly altered. Any reinterpretation of design entities requires a specification of change—for our purpose, computational change—that not only fixes the source and destination object types beforehand, but also fixes the numbers and mapping between properties. Previously, we have shown that continuity of such computational changes, from the standpoint of design rationale, requires anticipation of the particular structures that are to be changed (Krishnamurti and Stouffs, 1997). Creativity, on the other hand, goes beyond such anticipation. See Knight and Stiny (2001) for further elaborations on classical versus nonclassical computation in design.

2.2 Emergent information and rules

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). There are two prime requirements to consider.

Firstly, we need a specification of allowable transformations. Which kinds of transformations are permitted depend on the kind of information being recognized. For example, in spatial recognition, the transformations are, commonly, Euclidean: a square must be computationally recognized as a square, irrespective of scale, orientation or location. Likewise, transformations can be considered for other kinds of information—for example, search-and-replace in text editing allow for case transformations of the constituent letters.

Secondly, we need a definition for the part relationship that governs when an information structure is considered a part of another. Formally, this part relationship may be freely defined so long as it constitutes a partial order. For shapes this part relation defines an algebra—that is, a shape is specified as an element of the algebra, ordered by a part relation, closed under operations of sum, difference and product, and affine transformations (Stiny, 1991; Stouffs, 1994). Moreover, under the part relation, any part of a shape is a shape. As a shape specifies an indefinite number of (sub-) shapes, each a part of the original, it follows that we can deal with shape indeterminately, and in this way, new shapes emerge.

Geometrically, the maximal element representation (Krishnamurti, 1992; Stouffs, 1994) captures this notion precisely. Consider the combination of two squares shown in Figure 1. Each square defines an object made up of four line segments. Visually, however, the composition contains not two, but three squares. 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 pertinent. 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 indefinite number 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

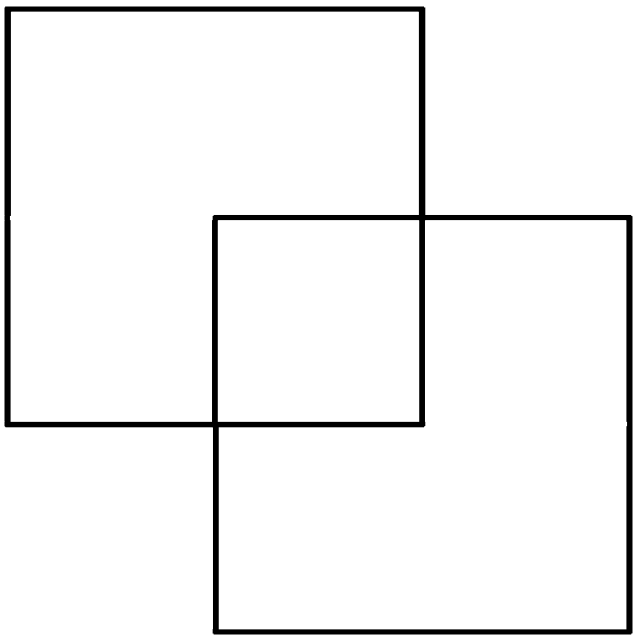


Fig. 1. A composition of two or three squares.

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 to have this interpretation supported by the system.

Recognizing emergent information is useful, especially, when the emergent information is subsequently the subject of an operation or manipulation, as in situations that rely 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. In Krishnamurti and Stouffs (1997) we discuss $s - f(a) + f(b)$ as a computational expression of spatial change, derived from 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.

Formally and technically, rules can be grouped as a device for specifying a language, namely, the set of all designs generated. Each design in the language is generated from an initial design, and employs the rules to arrive at the design. In spatial design, the specification of rules leads naturally to the generation and exploration of possible spatial designs. Both Mitchell (1993) and Stiny (1993, 1994) as well as more recently in the special issue on design spaces (Stouffs, 2006), some authors have remarked on the importance of emergent spatial elements and/or rules in 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. Individually, rules serve to facilitate common operations, for example, for the changing of one data collection into another or for the creation of new design information based on existing information in combination with a rule. Collectively, rules act as generators for the creation of certain sets of designs and for the derivation of certain information.

3 SORTS

The algebraic model for shapes and the maximal element representation have their origin in shape grammar research (Stiny, 2006). These were initially developed for shapes, made up of points and line segments, and later extended to planar and volume segments (Stouffs, 1994). Each spatial element type specifies its own algebra. We employ the notation, U_{ij}, to refer to the algebra of shapes made up of i-dimensional linear elements in a j-dimensional Euclidean space, and use U_i, as shorthand, whenever the dimensionality of the space is known (Stiny, 1991). Algebraically, a shape made up of more than one type of spatial element, belongs to the Cartesian product algebra of its constituent spatial element types, for example, a shape comprising points and line segments belongs to $U_0 \times U_1$. The algebraic model can also apply to shapes augmented with nongeometric properties, for example, labels, weights (e.g., line thickness; Stiny, 1992), and colors (Knight, 1989)—likewise, V_i refers to shapes in U_i augmented with labels, and W_i to shapes augmented with weights (Stiny, 1991).

We have extended this algebraic model to the formalism, named *sorts*, initially presented in Stouffs and Krishnamurti (2002), and one that we are still revising. Briefly, a *sort* is a basic entity, conceptually, specifying a set of similar data elements, for example, a class of objects, equivalently, a set of tuples solving a system of equations. For example, points and lines are *sorts,* as are triangles and squares. 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. *Sorts* are not limited to geometrical objects; colors are *sortal*, as are other data types. *Sorts* subsume conventional class representations.

We define a conjunctive *attribute* operation on *sorts*. For example, we can define a *sort* of labeled points, ($\approx V_0$), from a *sort* of points and a *sort* of labels. Likewise, we can define a *sort* of *weighted* line segments, ($\approx W_1$), or a *sort* of *linear* shapes ($\approx U_0 \times U_1 \times U_2 \times U_3$). Sortal conjunction has similarity to the Cartesian product of algebras. Two observations can be made in this respect.

Firstly, conjunction is not commutative: a *sort* of labeled points is not identical to the *sort* of "pointed" labels. The former *sort* comprises points with associated labels, the latter, labels with associated points. This distinction derives from the fact that each *sort* 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). Note that a table of unsorted coordinates and labels can be sorted either as labeled points or pointed labels, and the two *sorts* are distinct.

Secondly, an element belonging to a conjunctive *sort* necessarily contains elements from its constituent *sorts*. For example, a shape belonging to a Cartesian product of algebras, for example, $U_0 \times U_1 \times U_2 \times U_3$, must contain (spatial) elements from each algebra, otherwise it necessarily belongs to the algebra specified by the actual Cartesian product of the algebras of its specific element types.

To define a *sort*—where neither the ordering of component *sorts* is important, nor the presence of elements from the different component *sorts* is necessarily explicit—we define a *disjunctive* operation on *sorts*. Under this disjunction, any element of the resulting *sort* is necessarily an element of a constituent *sort*. Sortal disjunction 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 subsumes another, the entities of the former represent those of the latter, without data loss. There are a number of representational formalisms that specify a subsumption relationship, to achieve a partially ordered type structure. Most are based on first-order logic. A good example in the building design domain is Woodbury et al. (1999), who model design spaces by typed feature structures, which represent data types through record-like data structures. These structures facilitate the encapsulation of property information as (a variation of) attribute/value pairs (Aït-Kaci, 1984). Moreover, properties themselves may be typed by type structures, that is, expressed in terms of record-like data structures, containing (sub-) properties. In this way, a 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 are likewise conceived, with, at least, one exception: the association of properties to a *sort* occurs through sortal conjunction—that is, each property of a *sort* is itself a *sort*. *Primitive sorts* are the exceptions to this rule. Like primitive data types, primitive *sorts* are the smallest building blocks for building sortal representational structures. Primitive *sorts* combine to form *composite sorts* under compositional operators over *sorts*. A primitive *sort* defines the domain of possible values, for example, 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. The subsumption relationship between *sorts* then derives from the disjunction on *sorts* (similar to a union of two or more type structures) and an expression of constraints on primitive *sorts*. Specifying an intersection-like operation on *sorts* adds no further value, as the intersection of two *sorts* can always be reduced, through distributive and associative rules, to the intersection of primitive *sorts* (Stouffs and Krishnamurti, 2002). The intersection is nonempty only when the primitive *sorts* are identical, except for possible constraints.

The subsumption relation specifies a hierarchy; moreover, it can be considered in terms of information specificity. However, there is a distinction to be drawn in the way in which subsumption is treated in *sorts* and in first-order logic based representational formalisms as exemplified by type feature structures. First-order logic formalisms generally consider a relation of inclusion (hyponymy relation), commonly denoted as an *is-a* relationship. *Sorts*, on the other hand, consider a *part-of* relationship (meronymy relation).

Two simple examples illustrate this distinction. Consider a disjunction of a *sort* of points and a *sort* of line segments; this allows for the representation of both points and line segments. We can say that the *sort* of points forms part of the *sort* of points and line segments—note the part-of relationship. In first-order logic, this corresponds to the union of points and line segments. We can say that both are bounded geometrical elements of zero or one dimensions—note the is-a relationship.

This distinction becomes even more important when we consider an extension of sortal subsumption to conjunction. Consider a *sort* of labeled points as a conjunction of the *sorts* of points and labels. If we relax the requirement that an element belonging to a conjunctive *sort* necessarily contains elements from its constituent *sorts*, we can consider the *sort* of points to be part of the *sort* of labeled points, that is, a point is a labeled point without an associated label or, preferably, a point is part of a labeled point. Thus, the *sort* of labeled points subsumes the *sort* of points. In logic formalisms, a relational construct is used to represent such associations. For example, in Baader et al. (2003), roles are defined as binary relationships between concepts. Consider a concept Point and a concept Label; the concept of labeled points can then be represented as Point $\cap \exists$ has-Attribute.Label, denoting those points that have an attribute that is a label. Here, $\cap$ denotes intersection and $\exists$R.C denotes full existential quantification with respect to role R and concept C. It follows then that Point $\cap \exists$ has Attribute. Label $\subseteq$ Point; that is, the concept of points subsumes the concept of labeled points—this is quite the reverse of how it is considered in *sorts*.

Another important distinction is that first order logic-based representations generally make for an open world assumption—that is, nothing is excluded unless it is done so explicitly. For example, polygon objects may have an assigned color. When looking for a yellow square, logically, every square is considered a potential solution—unless, it has an explicitly specified color, or it is otherwise known not to have the yellow color. The fact that a color is not specified does not exclude an object from potentially being yellow. *Sorts*, on the other hand, hold to a closed world assumption. That is, we work with just the data we have. A polygon has a color only if one is explicitly assigned: when looking for a yellow square, any square will not do; it has to have the yellow color assigned.

This restriction is used to constrain emergence. More specifically, labeled points commonly serve to constrain the applicability of shape rules, which encapsulate both shape recognition (emergence) and shape transformation (computational change). Another way of looking at this distinction between the open or closed world assumptions is to consider their applicability to knowledge representation. To reiterate, logic-based representations essentially represent knowledge; *sorts*, on the other hand, are intended to represent data—any reasoning (computational change) is based purely on present (or emergent) data.

3.1 Behavioral specification

Sortal algebra implies the specification of a part relationship on the elements of a *sort*. The part relationship not only governs when an element is a part of another, but 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*.

The simplest specification of a part relationship corresponds to the subset relationship in mathematical sets. Such a part relationship applies to points and labels, for example, a point is part of another point only if they are identical, and a label is a part of a collection of labels only if it is identical to one of the labels in the collection. Here, sortal operations of addition (combining elements), subtraction, and product (intersecting elements) correspond to set union, difference, and intersection, respectively.

Another kind of part relationship corresponds to interval behavior. Consider, for example, the specification of a part relationship on line segments. A line segment may be considered as an interval on an infinite line (or *carrier*); in general, one-dimensional quantities, such as time, can be treated as intervals. An interval is a part of another interval if it is embedded in the latter; intervals on the same carrier that are adjacent or overlap combine into a single interval. Specifically, a behavior for intervals can be expressed in terms of the behavior of the boundaries of intervals (Krishnamurti and Stouffs, 2004). This behavior applies to indefinite intervals too, providing that there is an appropriate representation of both (infinite) ends of its carrier. Likewise, behaviors can be specified for area intervals (plane segments) and volume intervals (polyhedral segments). The notion extends to intervals of higher dimensions.

Behaviors apply to composite *sorts* as well; that is, the part relationship is defined on data elements belonging to a *sort* defined by conjunction (attribute) or disjunction. Specifically, a composite *sort* inherits its behavior from its component *sorts* depending upon the compositional relationship. For example, consider the following situation, common in CAD applications, of classifying data elements, say line segments, into layers. Here, we define the *sort* of labeled line segments as a composition under the attribute operator of a *sort* of line segments and a *sort* of layer labels. When layers are considered as attribute values, it may seem intuitive to allow just a single label for each line segment (most CAD applications do not normally allow the same object to exist in multiple layers at the same time). However, there is no common notion of a layer ordering (or a corresponding part relationship) that can be used to define sortal operations on layer labels. Instead, were we to allow a set of layer labels to be assigned as an attribute to a line segment, the result may be interpreted as a collection of identical copies of the same line segment existing in different layers, and could be presented as such to the user. To deal with or manipulate a single-layer copy, that is, a copy of the line segment associated with a single label (or singleton subset of labels), this copy needs to be differentiated from the other copies of the same line segment existing in the other layers. The sortal data element representing the single-layer line segment is a part of another sortal data element representing a multi-layer line segment if the first line segment is a part of the second (under the interval behavior for line segments) and if the single label of the first entity is a member of the set of labels for the second entity. Consequently, by subtracting the first entity from the second, the single-layer line segment is distinguished (i.e., selected) from the remainder. Thus, under the attribute operator, a data element is part of a data collection 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(s) of the data element(s) of the first component *sort* it is a part of. When data collections of the same composite *sort* (under conjunction) are pairwise summed (differenced or intersected), identical data elements merge, and their attribute collections combine, under this operation. Elements with empty attributes are, necessarily, removed. Under conjunction, the composite behavior, in the first instance, is that of the first component *sort*.

The disjunctive operator distinguishes, instead, all operand *sorts* such that each data element belongs explicitly to one of these *sorts*. For example, a *sort* of points and lines distinguishes each data element as either a point or a line. Consequently, 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 component *sort*. In other words, data collections from different component *sorts,* under disjunction, never interact; the resulting data collection is the set of collections from all component *sorts*. When the operations of addition, subtraction or product are applied to two data collections of the same disjunctive *sort*, each operation applies to the respective component collections.

Behaviors play an important role when assessing data loss in data exchange between different *sorts*. When reorganizing the composition of the component *sorts*, under the attribute operator, the corresponding behavior may be altered in such a way as to trigger data loss. Consider a behavior for weights (e.g., line thickness or surface tones) as is apparent from drawings on paper—a single line drawn multiple times, each time with a different thickness, appears as if it were drawn once with the largest thickness, even though it assumes the same line with other thicknesses. When using numeric values to represent weights, the part relation on weights corresponds to the less-than-or-equal relation on numeric values. Thus, weights combine into a single weight, with its value as the least upper bound of the respective individual weights, that is, their maximum value. Similarly, the common value (intersection) of a collection of weights is the greatest lower bound of the individual weights, that is, their minimum value. The result of subtracting one weight from another is either a weight that equals the numeric difference of their values or zero (i.e., no weight), and this depends on their relative values. Now consider a *sort* of weighted points, that is, a *sort* of points with attribute weights, and a *sort* of pointed weights, that is, a *sort* of weights with attribute points. A collection of weighted points defines a set of noncoincident 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 behavior of the collection is, in the first instance, the behavior for points. 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, adheres, in the first instance, to the behavior for weights. 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. An understanding of when and where exact translation of data between different *sorts* or representations is or is not possible is important for assessing data integrity and controlling data flow (Stouffs et al., 1996).

Additionally, a behavioral specification is a prerequisite for the uniform handling of different and a priori unknown data structures. Consider the association of building performance data to design geometries. The behavior 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 behavior. When an application receives the data along with its behavioral specification, the application can then correctly interpret, manipulate, and represent this information without unexpected data loss. Furthermore, the part relationship that underlies the behavioral specification for a *sort* enables the matching problem to be implemented for this *sort*. As composite *sorts* inherit their behavior and part relationship from their component *sorts*, any technical difficulties in implementing matching apply just once, for each primitive *sort*.

3.2 Data functions

Computational design relies on effective information models, not just for the creation of design artifacts, but also for querying the characteristics of such artifacts. With respect to geometry, Mäntylä (1988) remarked that these (geometric) representations must adequately answer "arbitrary geometric questions algorithmically." Even without the emphasis on geometric aspects, this remains as important today. However, current computational design applications tend to focus on representations of design artifacts, and on the tools and operations for their creation and manipulation. Techniques for querying receive less attention and are often constrained by the data representation system and methods. Nevertheless, querying a design is as much an intricate aspect of the design process as is creation and manipulation.

Querying design information, as distinguished from visual inspection, generally requires the analysis of existing information to derive new information that is not explicitly available in the information structure. A viable query language has to be based on a model for representing the different kinds of information that adhere to a consistent logic, providing access to information in a uniform and consistent manner, so that new queries can be easily constructed and posed, based on intent instead of availability. *Sorts* offer the flexibility to support constructing alternative representations, to compare representations with respect to scope and coverage, as well as support the specification of operational behavior on data in a uniform way, based on a partial order relationship.

Data recognition, or pattern matching, plays an important role in the specification of design queries, as does counting. Stouffs and Krishnamurti (1996) show how a query language for querying graphical design information can be built from basic operations and geometric relations which are defined as part of a maximal element representation for weighted geometries, augmented with operations derived from techniques of counting and pattern matching. For example, by augmenting networks of lines that are represented as volume (or plane segments) with labels as attributes, and by combining these augmented geometries under the operation of sum, as defined for the representational model, colliding lines specifically result in geometries that have more than one label as attribute. These collisions can easily be counted, whilst the labels, each associated with a geometry identifying colliding lines, and the geometry itself specifies the location of the collision.

To consider counting and other functional behavior as part of the representational approach, *sorts* consider data functions as a kind of data, offering functional behavior integrated into the data constructs. Data functions are assigned to apply to one or more selected property attributes of selected *sorts*, each of which may, itself, be a data function. The resulting value of the data function is then computed from the values of the respective property attributes of valid compositions of data entities of these *sorts*. A composition of data entities is a valid composition if the data entities are encountered along the same path of object-attribute relationships, with some restrictions. Therefore, the target *sorts* must be related to the data function's *sorts* within the representational structure under a sequence of one or more attribute relationships. The resulting value is automatically recomputed each time the data structure is traversed, for example, when visualizing the structure. As data kind, data functions specify a functional description, a result value, and one or more *sorts* and their respective property attributes.

Data functions can introduce specific behaviors and functionalities into representational structures, both for the purposes of counting and for other numerical operations. Consider, for example, a data structure corresponding to a composition of two *sorts* under the attribute operator where the attribute *sort* specifies a cost per unit length to the other *sort*. By augmenting the data structure with an inner product function, applied to the numeric value attribute property of the cost *sort* and the length attribute property of the other *sort*, the value of this function is then automatically computed as the sum of all cost values multiplied by the respective lengths of the other entities.

3.3 Implementation

For ease and brevity, we describe the implementation of *sorts* in terms of UML classes (Rumbaugh et al., 2005). Some basic definitions are, however, needed. A *sortal description* includes *sorts*, individuals and forms; a *sort* is a class structure, specifying a single data type or a composition of other class structures; an *individual* is a basic sortal element, that is, an instance of a class structure; and, a *form* is a collection of individuals of the same *sort*. These are the three main abstract classes in our Java implementation. Figure 2 shows a conceptual UML model of the abstract Sort class and its five subclasses: PrimitiveSort, AttributeSort, DisjunctiveSort, Aspect and AspectsSort. A Sort has a context in which it is defined, a description and an optional name. A PrimitiveSort specifies a single data type; its individuals have a data value of the same type. An AttributeSort is the composition of a simple *sort* (its base)—a primitive *sort* or an aspect—with another *sort* (its weight) under a conjunction. Its individuals comprise an individual of the base *sort* (the *associate individual*), which is assigned a form of the weight *sort* as its attribute (the *attribute form*). A DisjunctiveSort is a co-ordinate composition of two or more component *sorts*—which can be a primitive *sort,* an attribute *sort*, or an aspect. A disjunctive sortal form (or *metaform*) is a composition of forms from the respective component *sorts*. An AspectsSort is a *sort* of undirected relationships between two or more primitive *sorts*. Each undirected relationship is represented as two or more directed relationship Aspects. AttributeSort and DisjunctiveSort can be recursively defined. The actual implementation model for the corresponding package of classes is shown in Figure 3.

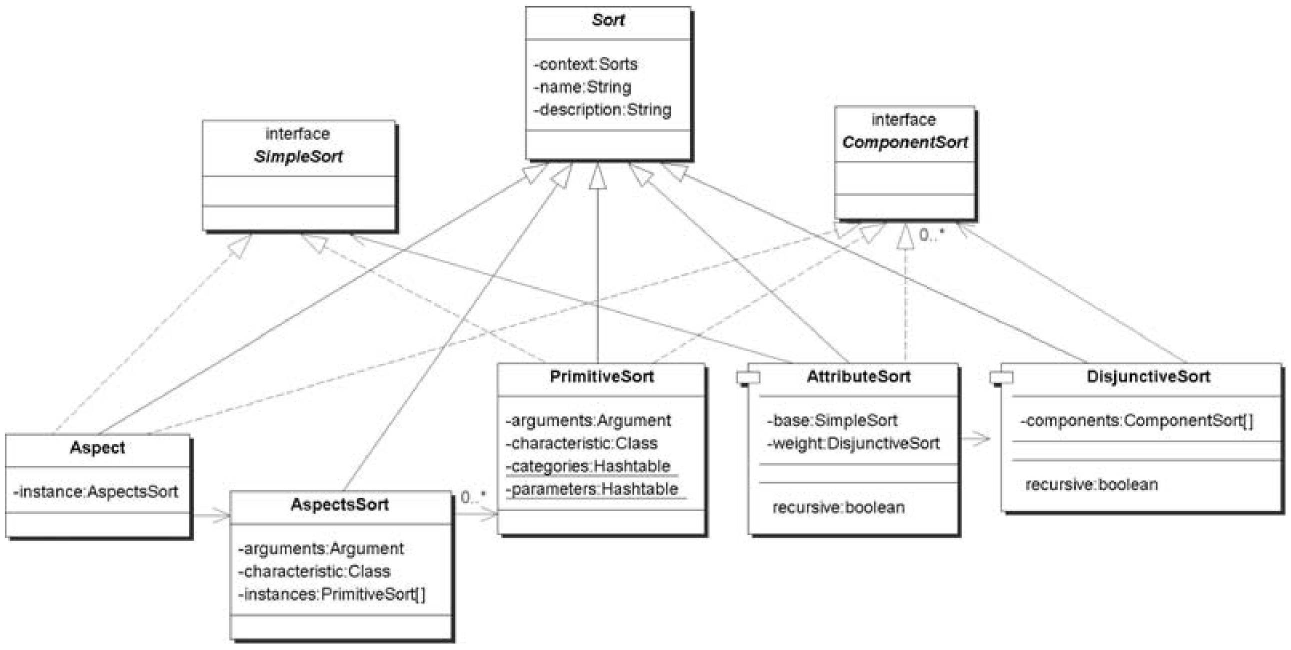


Fig. 2. UML conceptual model of the abstract Sort class and its subclasses.

Each primitive sort is specified by a name, *characteristic individual,* and *behavioral category*. A characteristic individual is a subclass of the Individual class and specifies the representation of its data values and behavioral methods. Figure 4 shows a UML implementation model for the Individual class, with some of its subclasses. The behavioral category is a subclass of the Form class and specifies the operational behavior of the sortal forms; it is assigned in a categorization of the characteristic individuals. Figure 5 shows a UML implementation model for the "form" package of classes. The following behavioral categories have been implemented: discrete, interval, ordinal, and relational.

4 ASDMCON: SORTAL STRUCTURES IN BUILDING CONSTRUCTION

4.1 Background

The ASDMCon (Advanced Sensor-based Defect Management at Construction sites) project involves collaboration among researchers from three different disciplines at Carnegie Mellon University: Architecture, Robotics, and Civil and Environmental Engineering (Yue et al., 2006). The premise underlying the project is that defects can be detected as they occur by performing frequent, complete, and accurate assessments of the actual (as-built) condition of a facility throughout the construction process. Advanced sensor technologies, including 3D imagers (i.e., laser scanners) and embedded sensors are central to this assessment process. Laser scanners provide a capability of accurately modeling the geometry of a facility, while embedded sensors monitor nongeometric aspects, such as concrete strength. This information is combined with a design model, an ontology for construction specification, and a project schedule to create an "integrated project model" (IPM), which is dynamically updated throughout the construction period (Figure 6).

Research has shown that a semantically rich integrated project database combining the views of project participants can support various project management and facility management during the construction phase (Cleveland, 1996; Froese et al., 1999; Fischer et al., 1998; Yu et al., 1998). Accordingly, an important element of our system is a capability of generating multiple views to serve the purposes of the different collaborators (and potential users). From the outset, we have looked at *sorts* towards achieving this.

The team conducted four case studies on construction sites near Pittsburgh, Pennsylvania. These case studies serve to identify the challenges in applying a specific design representation to suit the different perspectives from various objectives during the construction deviation identifying process (Gordon et al., 2003).

Overall, for each case study, the process starts with information gathering to build an IPM. The as-designed model has a level of geometric detail, useful for comparison, with features extracted from a current condition

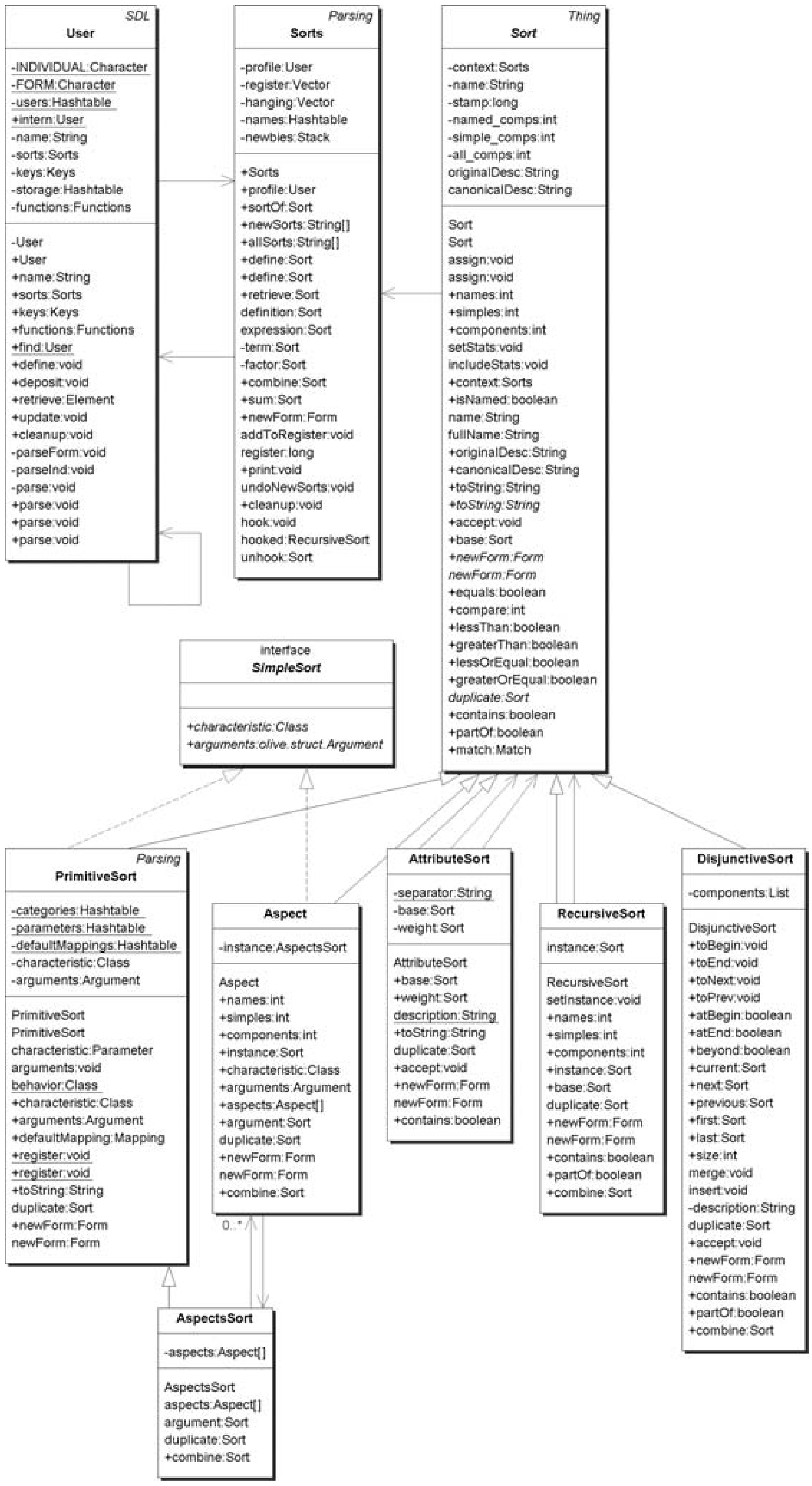


Fig. 3. UML implementation model of the classes in the "sort" package.

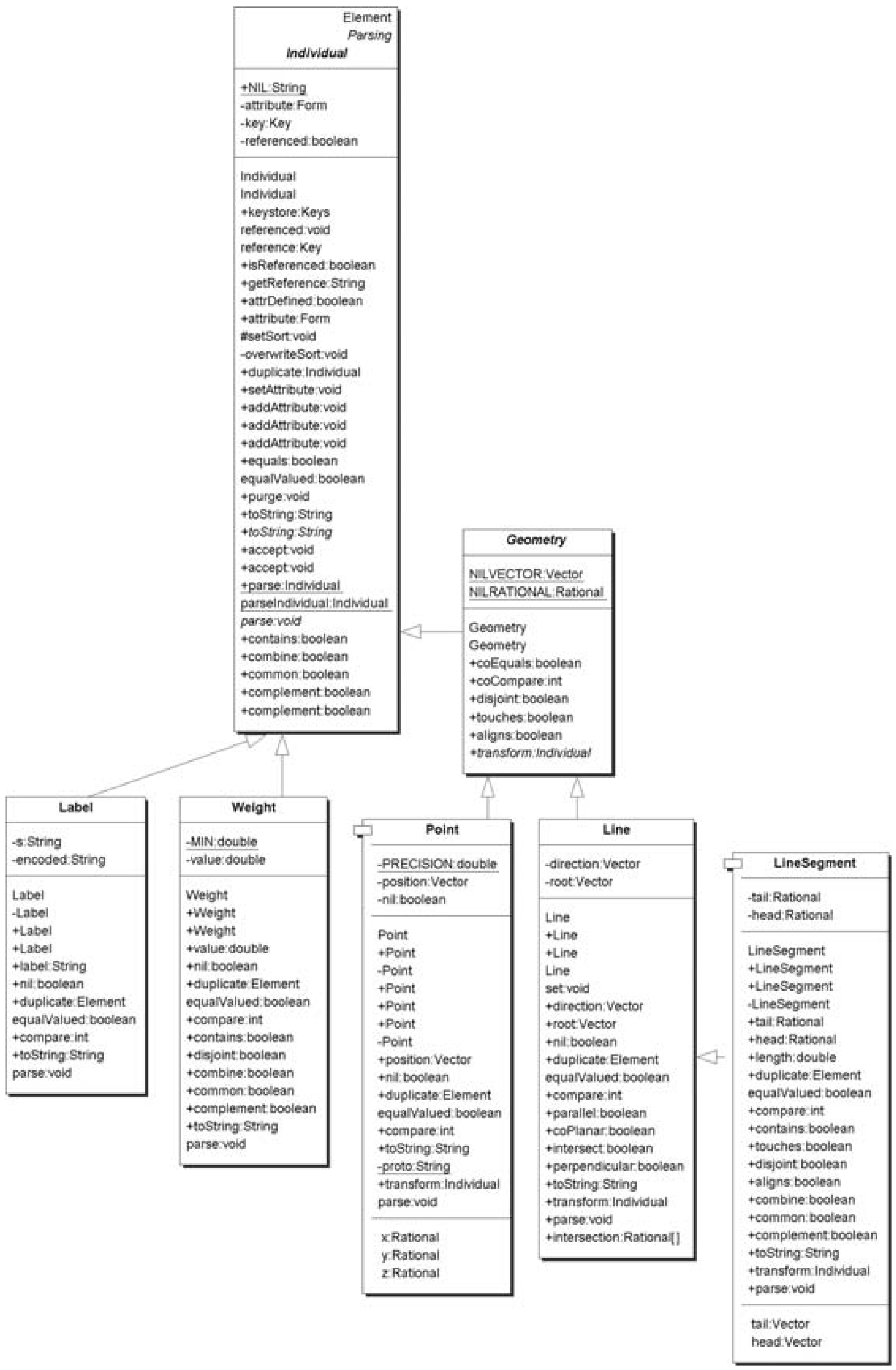


Fig. 4. UML implementation model of a subset of the classes in the "ind" package.

of construction. For nongeometric features, components are presented with expected performance attributes that correspond to gathered data. The sequence for the reality capturing process, comprises the iterative steps described below.

IPM initialization is based on architect-supplied documents, from which a building information model is developed. This model is a resource for determining measurement goals. Depending upon the property to be measured, goals are set for each specific sensing method for data collection. For example, goals with geometric information use laser scanning to be able to compare specific shapes from the as-designed and as-built models. Other properties, such as inside concrete temperature,

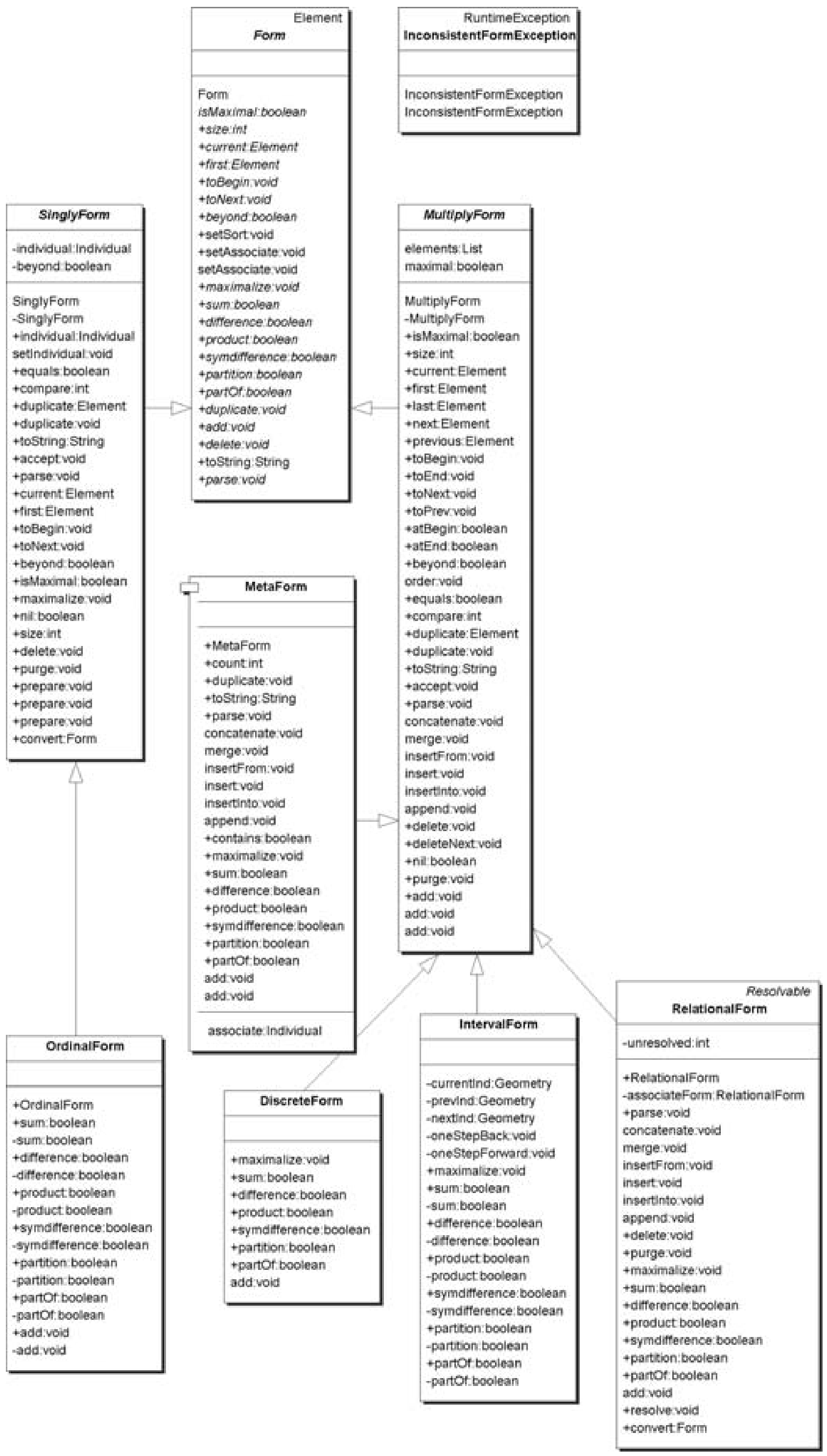


Fig. 5. UML implementation model of the classes in the "form" package.

are measured by embedded sensing. Even when a property has no geometric relevance, distributing sensors in a building element requires dealing with the as-designed geometry. Once measurement goals have been determined, planning for each method of data collection proceeds. For a given construction schedule, as-designed model, and measurement goals, an embedded sensing plan is made by multiple decisions of when, where, what

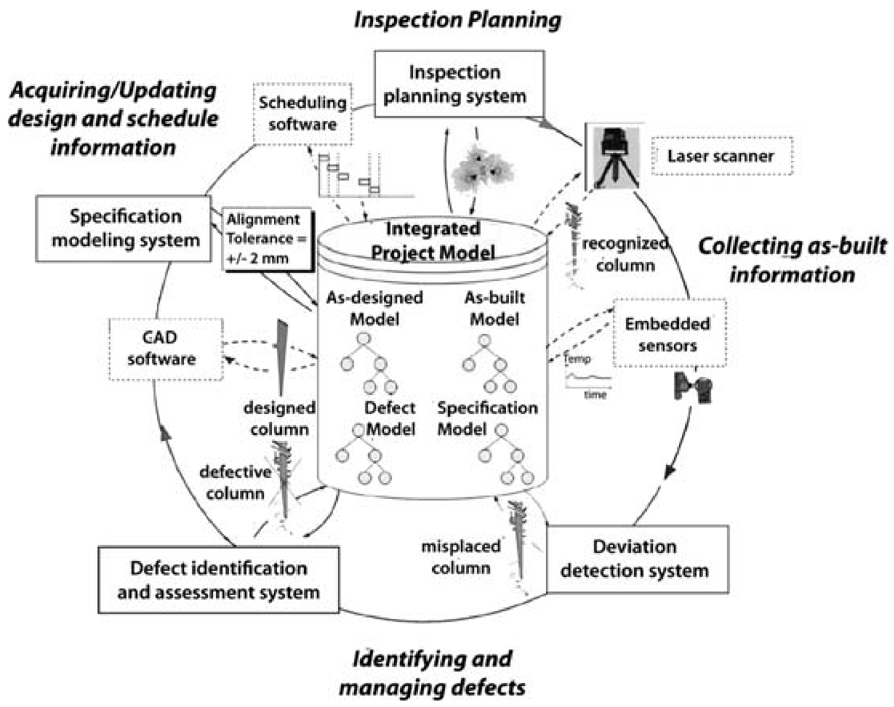

Inspection Planning
Inspection planning system
Scheduling software
Acquiring/Updating design and schedule information
Laser scanner
Specification modeling system
Alignment Tolerance = +/- 2 mm
Integrated Project Model
recognized column
Collecting as-built information
As-designed Model
As-built Model
Embedded sensors
CAD software
temp
time
designed column
Defect Model
Specification Model
defective column
misplaced column
Deviation detection system
Defect identification and assessment system
Identifying and managing defects

Fig. 6. The integrated project model and the cyclical processing pipeline used in the ASDMCon project (adapted from Akinci et al. (2006)).

properties, how long, and which sensors are needed. For laser scan planning, a further goal is to optimize scanner use to achieve a given set of measurement goals within the construction area. These iterations continue until construction is completed.

Laser scans produce low data format geometry, namely, point clouds. In the analysis stage, as-built modeling involves registering scans with one another and with the design model, segmenting the as-built data into components and associating each with corresponding components in the as-designed model. This is done using a nearest neighbor algorithm, working on the premise that the point cloud counterparts in the as-designed model are in close proximity. Suppressing point cloud noise due to certain temporary construction site conditions, for example, temporary storage of construction material, formwork, tools, etc, is through user-intervention. Matched point clouds represent surfaces, each associated with a component in the as-designed model. See Figure 7. In this way, as-built models can be created and stored in the IPM. The as-designed and as-built models can be compared, looking for discrepancies by overlaying the models within allowable tolerances described in the construction specifications. This visual inspection provides a more detailed comparison than traditional on-site inspection methods.

From our case studies we have identified changes from diverse directions—as seen from the perspective of—the architect or engineer, the construction manager, the building part fabricator, and the defect detection team. Some changes are updates to existing building elements; some others are, notably, new entities added to the building system, though not described in any current document. We can directly bring these new entities and changes into the building information model.

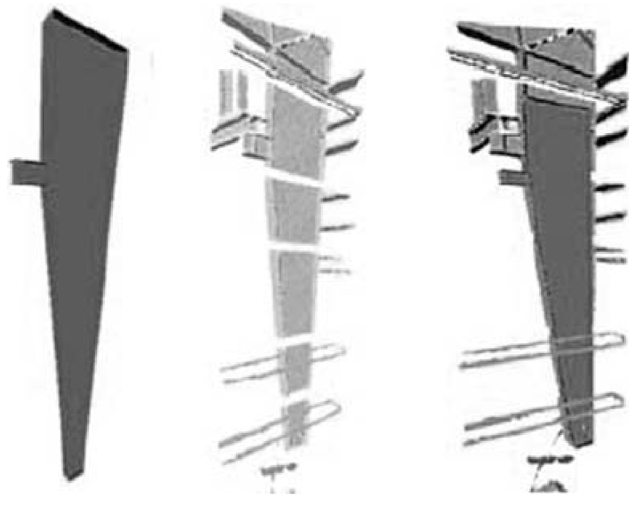


Fig. 7. The recognition progress for a steel column from a case study: (*left*) As-designed model (*center*) Point cloud, (*right*) Recognized column.

4.2 Modeling "changes" in the integrated project model

The IPM is a learning repository, containing traces of building evolution from design to completion of construction, based on incomplete attribute information, in the form of IFC data (though any data protocol will work). To capture a current site condition, two new entities were added to the IPM: embedded sensor data, and as-built geometry from the laser scans. Entities are connected through their as-designed element identity. Embedded sensor entities have as substratum, an as-designed element and a location within that element. Sensor entities contain other attributes, such as type, usage, and time-stamped values. Each as-built geometric entity maps to an as-designed element. Ideally, this mapping is one-to-one, from an as-built surface to a surface of an as-designed element. Also, a new entity named "Defect" was added into the IPM. The defect entity relates as-designed and as-built elements through their identities. It may also include either or both sensor entity and as-built geometry.

The IPM was designed to capture both changes in the as-designed model, and for a given construction site (as-built). The as-designed model contains component geometry as well as attributes data, such as material, install-start time, install-end time, etc. Defect and sensor data are attached to related components in the as-designed model. Change in the drawings implies an update of the as-designed model. All attribute and attached data are updated, or copied component by component. Data from any new updated version of the as-designed model is recorded in a different subdirectory from the previous version. Consequently, the IPM contains history data (changes). An as-built model represents whatever has been built on the construction site. It represents the latest recorded progress on the construction site, and thereby, records all changes on the construction site since the last scan session. In this way, each new scan session reflects an update of the as-built model.

4.3 Modeling "different views" in the integrated project model

For illustration, we provide three different views on the IPM, namely, that of the construction inspector, embedded sensor planner, and laser scanning technician. It is important to note that when the as-designed model is incorporated into the building construction process, information reflecting the construction specification, owner and construction contracts, construction process model, and so on, accumulate into the IPM.

The construction inspector decides upon measurement goals, to reason about deviations. For this, the construction specification, design specification, a detailed 3D as-designed geometry and construction process model are required. The information is categorized according to the required properties. For instance, geometrical elements have to be visually recognizable, connectivity to other relevant elements clearly understood, and relevant information includes nongeometric data such as material type, construction process data and construction method.

An embedded sensor planner deals with planning and sensing of embedded sensors to collect data from a construction element. For this, element geometry, its location and construction process information are required. Furthermore, a construction method is specified to relate material type to a specific time of installation. The as-designed model does not contain such information, and additional information is needed in the IPM.

To optimize scanner use to attain laser scan goals in the dynamically changing environment, the laser scanning technician employs a scanner path planner. From the path planner's perspective, construction site elements are either obstacles or goals; the net objective is to produce the shortest path for collecting data. The elements include static building entities such as walls, columns, doors and so on, and temporary site entities such as scaffolding, forms, temporary material stacks and so on. Depending on the specific range of the laser scanner, goal location and elevation are crucial. For example, the Z+F scanner is capable of scanning 360° horizontally and 70° vertically with a maximum range of 25 m. For the Z+F scanner, goals are kept to within that region to capture data of sufficient quality to produce a three-dimensional as-built model.

Table 1 summarizes the information required for the different participants (views). The as-designed geometry is sufficiently rich enough to be able to generate information appropriate to the individual needs, for example, a full model view for the construction inspector, and a regional view for laser scanner path planning. As can be seen in the table, the requirements for the three different views of the design model represent different ways of serving as domain specific viewpoints. When a construction inspector decides upon the object to be inspected, detailed information of the target is provided to the other two actors. This includes the what, when, and where data that has to be collected to achieve an inspection. The other two actors see the same object from their own perspective. The embedded sensor planner needs geometric information, material type, and construction method of the target object. On the other

Table 1
Information required by different users

View	*Requirements*
Construction inspector	• As-designed model
	• Design specification
	• Construction specification
	• Construction process
Embedded sensor planner	• Target and its inspection goals
	• Construction process information of the target
	• Construction methods of the target
	• Geometric information of the target element
	• Material type of the target
Laser scanning technician	• Target and its inspection goals
	• Construction process information of the target
	• Geometric information of the region around the target element
	• Geometric information of the target element
	• Location of the target

hand, the laser scan technician needs the target region geometry, and target object geometry and location. See Figure 8.

We define the target object geometry by its shape, and material type as a composite of the material types of the compound object. We can consider the same target object view in the two *sorts* built from the same components using the attribute operator, "∧," by taking the

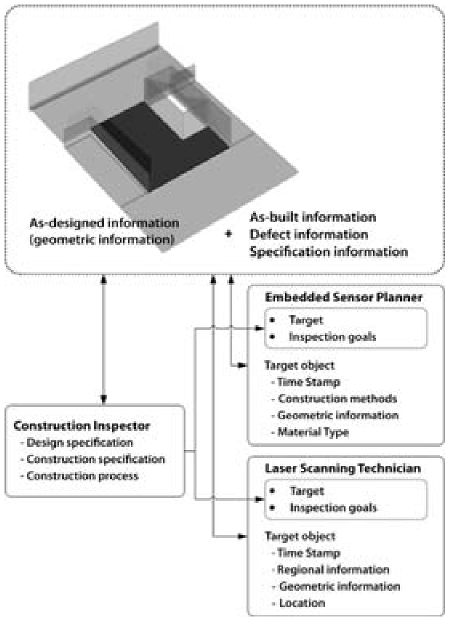


Fig. 8. Different representational needs on a slab.

components in a different order. For example, consider the components, *slabs* and *materialtypes*, both of which are *sorts* of labels, *locations*, a *sort* of points, and *shapes*, a *sort* of volumes:

In an embedded sensor view, *materialtypes* has *locations* as an attribute, and is, itself, considered, as an attribute to *shapes*, which we express as:

slab_embeddedSensor_targets: slabs ^ shapes ^ materialtypes ^ locations

In a laser scan view, *shapes* is instead considered as an attribute to *locations*, which itself is an attribute of *slabs*.

slab_laserScan_targets: slabs ^ locations ^ shapes ^ materialtypes

In this case, the laser scan technician's view provides the location of the target slab instead of multiple material locations for the embedded sensor planner.

4.4 Representing the integrated project model

Figure 9 illustrates the relationship between the various actors and the IPM.

The IPM is represented in three ways. First, we have a *general view*, which is a dynamic graph representation of the IPM and its component connectivity. Second, we have a participant-specific *sortal view* of the model, wherein *sorts* can be created, edited and/or modified. Third, for representational flexibility, *new views* can be dynamically generated, from components in the general view combined with specific functions (for example, volume calculation, face generation, etc). Figure 10 shows the current prototype, with the view panels, illustrating, in this case, the relationships of building element to both element type and material usage and the *sorts* editor. The node and edge conventions used in the different representation panels are given in Table 2.

The sortal representation schema development consists of retrieving specific relationships within the building elements, and their geometric information. One scenario that we have tested is the restructuring of building elements. Building elements have specific attributes. Some are physical, for instance, geometry, building storey containment, etc., and some conceptual, such as project, space, time, etc. Figure 11, specifically, highlights the different dependencies (and thus, the different classification) of building elements by Material Usage, Building Story, and Building Element Type, in one of the case studies.

Table 3 gives the *sorts* that were created for the case studies. Figures 12 and 13 illustrate two example composite sorts, shown as screen shots from the prototype development. As can be seen from Figure 12, the left side shows the definition of the composite sort: Building Storey Containment + Building Element + Material,

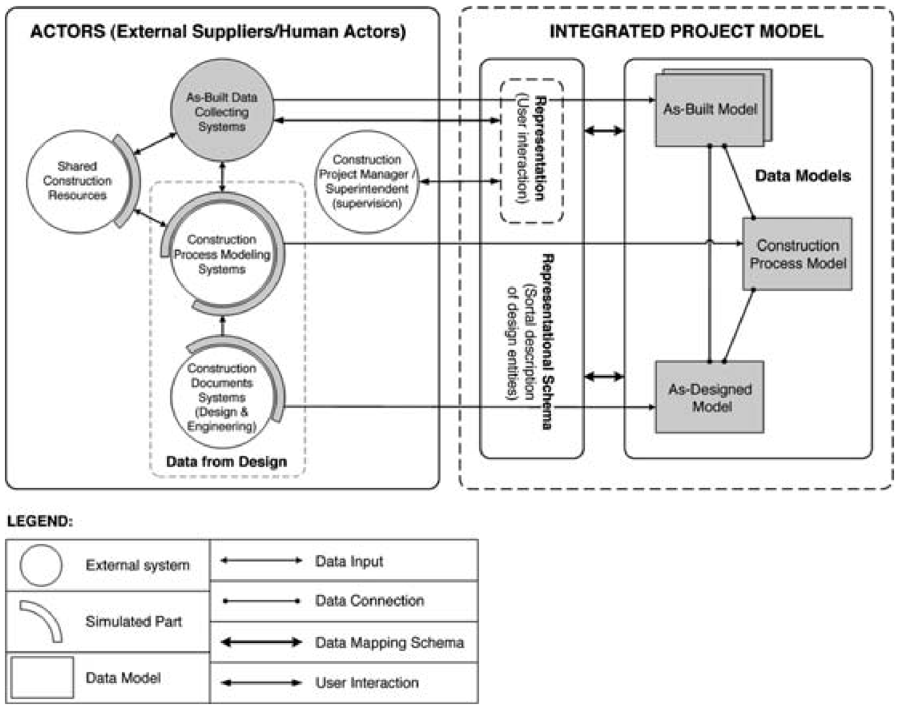
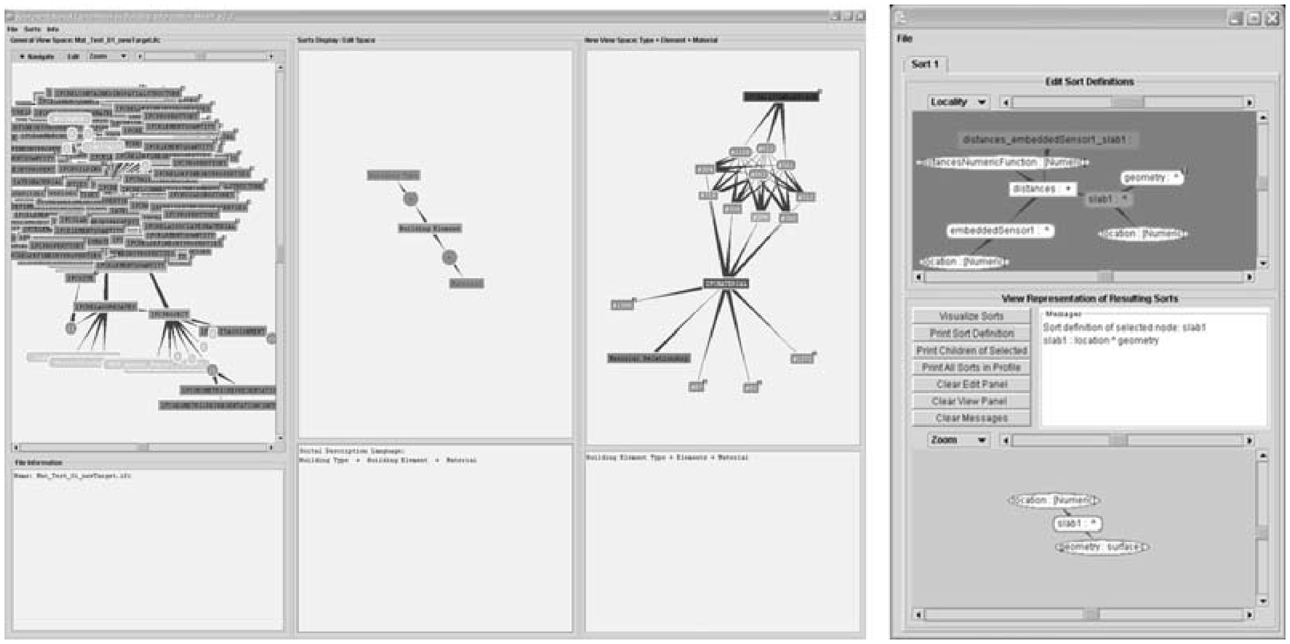

ACTORS (External Suppliers/Human Actors)
As-Built Data Collecting Systems
Shared Construction Resources
Construction Project Manager / Superintendent (supervision)
Construction Process Modeling Systems
Construction Documents Systems (Design & Engineering)
Data from Design
INTEGRATED PROJECT MODEL
Representation (User interaction)
Representational Schema (Sortal description of design entities)
As-Built Model
Data Models
Construction Process Model
As-Designed Model
LEGEND:
External system
Simulated Part
Data Model
Data Input
Data Connection
Data Mapping Schema
User Interaction

Fig. 9. Relationship between the actors and IPM.



Sort definition of selected node: slab1
slab1 : location * geometry

Fig. 10. Current prototype (*left*) view panels (*right*) sorts editor.

with the Building Element, graphically, as the focus of attention. The right hand side shows the filtering of the project database, corresponding to the Building Element, highlighting an instance and its related entities. Figure 13 is interpreted in a similar manner. Through such means we were able to easily filter the information specifically needed for laser scanning and defect detection.

Table 2
Node and edge conventions in the prototype

	Graphic representation (node)			
View panel	*Color + Shape*	*Example*	*In IFC2×2*	*In Sorts*
General	Orange Rectangle	IFCMATERIALLAYERSET	Class name	
	Light Gray Ellipse	'Concrete Structural '	Data (No outbound edges)	
	Dark Gray Circle	()	Enumeration (Outbound edges)	
Sorts Display/ Edit	Green Rectangle	Building Element Type		Primitive: Relationship
	Green Circle	+		Sum: Function
	Green Rectangle	^		Attribute: Function
New	Rectangle	IFCWALLSTANDARDCASE	Class Name	
	Ellipse	#145	Line Number (IfcBuildingElement)	
	Rectangle	#316	Line Number (Other than an IfcBuildingElement)	
	Rectangle	Building Element Type		Relationship
	Graphic representation (edge)		*In IFC2×2*	*In Sorts*
	A ——— B		A contains B	From A to B

From our experiences with the case studies, it is clear that there is a need to explicitly identify certain target building elements. To pinpoint these target elements from the rest of the building model, we embedded a string tag, "Target," as the entity—Tag: IfcIdentifier—in the IfcBuildingElement class. The text in Figure 14 shows a target tag in an IFC file, and a screen shot of the target element filtered from the building model.

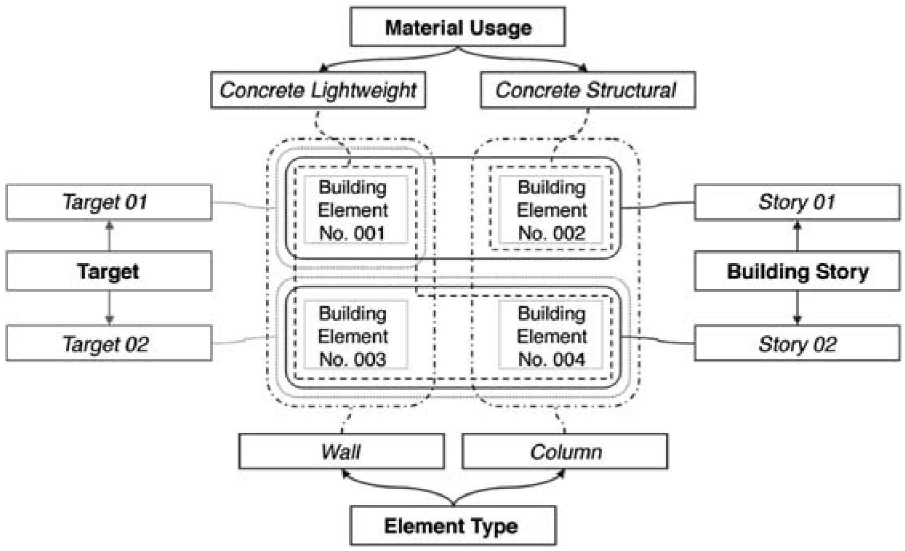


Fig. 11. Conceptual diagram of building elements dependencies.

Table 3
Example sorts defined for the construction case studies

Primitive sorts		
Spatial structure	Levels of decomposition in spatial structure for the building project (IfcSite, IfcBuilding, IfcBuildingStorey, and IfcSpace)	
Building Element	Building elements in the building project	
Building Type	Building element type information in the building project	
Building Story Containment	Storey containment information in the building project	
Material	Material usage information in the building project	
Target	Target element information in the building project	
Geometry Representation	Geometry information in a building element	
Sortal operators	*Name*	*Function*
+	Sum	Sum of primitive sorts
^	Attribute	An attribute of building element
Composite sorts	*Examples*	
	Building Storey Containment + Building Element + Material	
	#133 ∧ Geometric Representation	

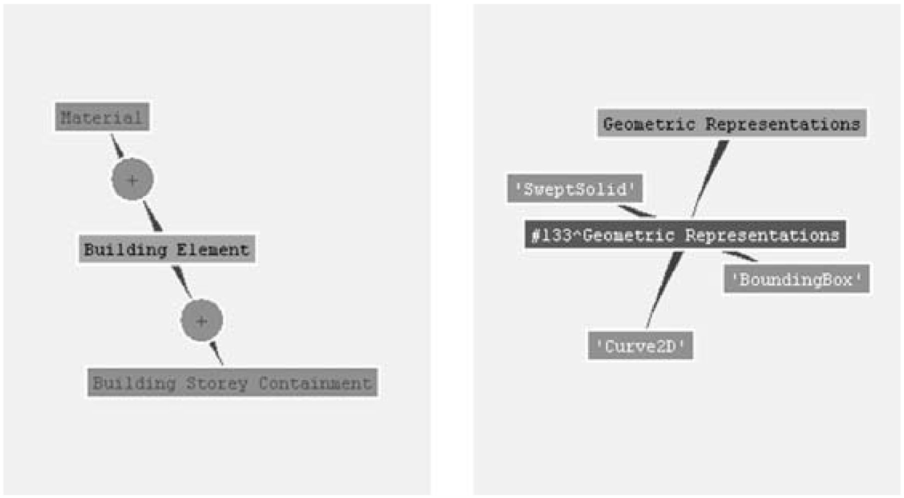

Material
+
Building Element
+
Building Storey Containment
Geometric Representations
'SweptSolid'
#133^Geometric Representations
'BoundingBox'
'Curve2D'

Fig. 12. Composite sort: (Building Storey Containment + Building Element + Material): (*left*) Sorts Display/Edit panel; and (*right*) New View panel.

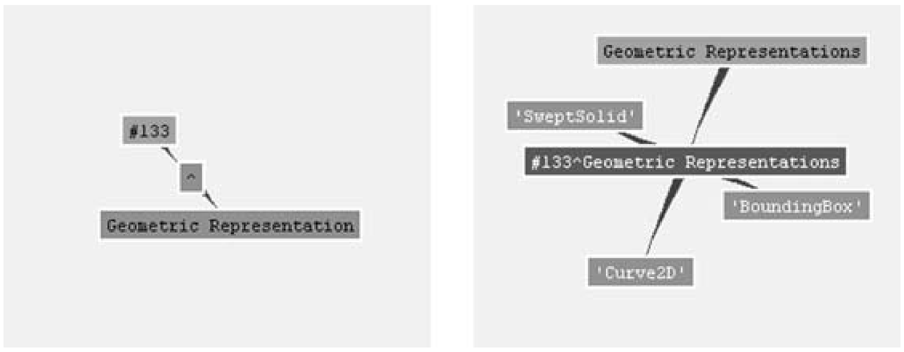
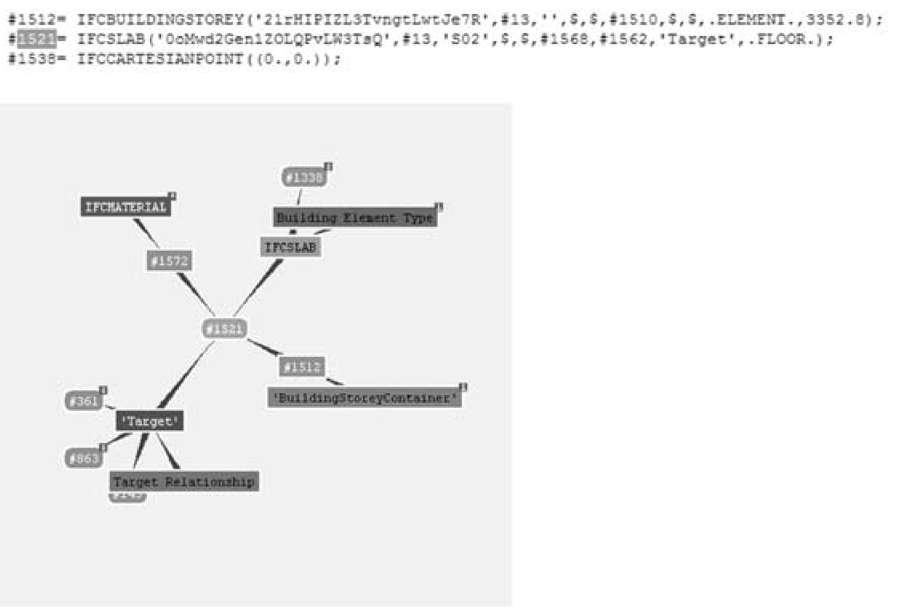


Fig. 13. Composite sort: (#133 ^ Geometric Representation): (*left*) Sorts Display/Edit panel; and (*right*) New View panel.



```
#1512= IFCBUILDINGSTOREY('21rHIPIZL3TvngtLwtJe7R',#13,'',$,$,#1510,$,$,.ELEMENT.,3352.8);
#1521= IFCSLAB('0oMwd2Gen1ZOLQPvLW3TsQ',#13,'S02',$,$,#1568,#1562,'Target',.FLOOR.);
#1538= IFCCARTESIANPOINT((0.,0.));
```

Fig. 14. Target tagged building elements: (*top*) in the IFC file; (*bottom*) filtered in the New View panel.

5 CONCLUSION

We have presented a formal approach to representational flexibility, *sorts*, for developing alternative representations of an entity, thereby supporting different viewpoints, essential to any multi-disciplinary, multi-participatory endeavor such as the building domain.

We have highlighted this in addressing certain building construction related problems.

In the case studies, we were able to show the value of *sorts* for its ability to flexibly modify/alter a representational structure, through creation, update, and deletion of sortal relationships. There are three aspects of *sorts* that we have illustrated through description and examples.

Firstly, *sorts* support the specification of operational behavior of data in a uniform way. This specification is based on a partial order, which enables: (1) the recognition of components and structures that are not explicitly present in any current information (and its representation); and (2) the restructuring of such information.

Secondly, *sorts* provide for two composition operators: (1) an attribute operator that defines a *convertibility* relation that assists in indicating compatibility and data loss, when conjunctive compositions of sorts are compared; and (2) a disjunctive composition operator that defines a subsumption relation, distinguished from the conjunctive attribute composition operator that does not adhere to the subsumption relation, quite distinct from most first order logic-based representations.

Thirdly, *sorts* formally provide support for comparing representational structures with respect to scope and coverage, to determine potential data loss whenever data is moved from less restrictive to more restrictive representations. In doing so, data loss is not necessarily an unwelcome side effect in exchange of information between different representations, rather, it can be the result of a conscious deliberate decision that takes into account any distinct advantage of the alternative views for the process under consideration.

ACKNOWLEDGMENTS

The first author is funded by the Netherlands Organization for Scientific Research (NWO), grant nr. 016.007.007; the second and third authors are funded by a grant from the National Science Foundation, CMS #0121549. Both NWO's and NSF's support are gratefully acknowledged. Any opinions, findings, conclusions or recommendations presented in this article are those of the authors and do not necessarily reflect the views of the Netherlands Organization for Scientific Research or the National Science Foundation. The authors would like to thank Michael Cumming for his work on the development of a prototype interface to build and manipulate *sorts*.

REFERENCES

Aït-Kaci, H. (1984), A lattice theoretic approach to computation based on a calculus of partially ordered type structures (property inheritance, semantic nets, graph unification), Ph.D. dissertation, University of Pennsylvania, Philadelphia, PA.

Akinci, B., Boukamp, F., Gordon, C., Huber, D., Lyons, C. & Park, K. (2006), A formalism for utilization of sensor systems and integrated project models for active construction quality control, *Automation in Construction*, **15**(2), 124–38.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P. (2003), *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, Cambridge.

Cleveland, A. B. (1996), Integrating information with 3D models for facility life-cycle support, in F. Y. Cheng (ed.), *Analysis and Computation*, American Society of Civil Engineers, Reston, VA.

Fischer, M., Aalami, F. & Akbas, R. (1998), Formalizing product model transformations: Case examples and applications, in I. Smith (ed.), *Artificial Intelligence in Structural Engineering: Information Technology for Design, Collaboration, Maintenance, and Monitoring*, *Lecture Notes in Artificial Intelligence 1454*, Springer, pp. 113–32.

Froese, T., Fischer, M., Grobler, F., Ritzenthaler, J., Yu, K., Sutherland, S., Staub, S., Akinci, B., Akbas, R., Koo, B., Barron, A. & Kunz, J. (1999), Industry Foundation Classes for project management: A trial implementation, *Electronic Journal of Information Technology in Construction*, **4**, 17–36. http://www.itcon.org/1999/2(8 Dec. 2004)

Gordon, C., Boukamp, F., Huber, D., Latimer, E., Park, K. & Akinci, B. (2003), Combining reality capture technologies for construction defect detection: A case study, in B. Tunçer, S. Özsariyildiz, and S. Sariyildiz (eds.), *E-Activities in Building Design and Construction*, Europia, Paris, pp. 99–108.

ISO (1994), *ISO 10303–1, overview and fundamental principles*, International Standardization Organization, Geneva.

Kiviniemi, A. (2006), Ten years of IFC development – why are we not yet there? Keynote lecture at the *Joint International Conference on Computing and Decision Making in Civil and Building Engineering*, Montreal, 14–16 June 2006.

Knight, T. W. (1989), Color grammars: Designing with lines and colors, *Environment and Planning B: Planning and Design*, **16**(4), 417–449.

Knight, T. W. & Stiny, G. (2001), Classical and non-classical computation, *Architectural Research Quarterly*, **5**, 355–372.

Kolarevic, B. (2000), Digital Morphogenesis and Computational Architectures, in J. Ripper Kos, A. Pessoa Borde, and D. Rodriguez Barros (eds.), *Construindo n(o) espaço digital*, PROURB, Universidade Federal do Rio de Janeiro, Rio de Janeiro, pp. 98–103.

Krishnamurti, R. (1992), The maximal representation of a shape, *Environment and Planning B: Planning and Design*, **19**(3), 267–288.

Krishnamurti, R. & Earl, C. F. (1992), Shape recognition in three dimensions, *Environment and Planning B: Planning and Design*, **19**(5), 585–603.

Krishnamurti, R. & Stouffs, R. (1997), Spatial change: Continuity, reversibility and emergent shapes, *Environment and Planning B: Planning and Design*, **24**(3), 359–384.

Krishnamurti, R. & Stouffs, R. (2004), The boundary of a shape and its classification, *Journal of Design Research*, **4**(1).

Liebich, T., Ed. (2006), *IFC 2x Edition 3*, International Alliance for Interoperability, http://www.iai-international.org/Model/R2×3_final/index.htm (23 Aug. 2006)

Mäntylä, M. (1988), *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MD.

Mitchell, W. J. (1993), A computational view of design creativity, in J. S. Gero and M. L. Maher (eds.), *Modeling Creativity and Knowledge-Based Creative Design*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Rumbaugh, J., Jacobson, I. & Booch, G. (2005), *The Unified Modeling Language reference manual*, 2nd Ed., Addison-Wesley, Boston.

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**(4), 413–430.

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

Stiny, G. (1994), Shape rules: Closure, continuity, and emergence, *Environment and Planning B: Planning and Design*, **21**, s49–s78.

Stiny, G. (2006), *Shape: Talking about Seeing and Doing*, MIT Press, Cambridge, Mass.

Stouffs, R. (1994), *The Algebra of Shapes*, Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA.

Stouffs, R. (2006), Guest Editor, *Artificial Intelligence in Design, Engineering and Manufacturing*, Special Issue on Design Spaces: The Explicit Representation of Spaces of Alternatives, 20 (2), May.

Stouffs, R. & Krishnamurti, R. (1996), On a query language for weighted geometries, in O. Moselhi, C. Bedard, and S. Alkass (eds.), *Third Canadian Conference on Computing in Civil and Building Engineering*, Canadian Society for Civil Engineering, Montreal, pp. 783–93.

Stouffs, R. & Krishnamurti, R. (2001), On the road to standardization, in B. de Vries, J. van Leeuwen, and H. Achten (eds.), *Computer Aided Architectural Design Futures 2001*, Kluwer Academic, Dordrecht, The Netherlands, pp. 75–88.

Stouffs, R. & Krishnamurti, R. (2002), Representational flexibility for design, in J. Gero (ed.), *Artificial Intelligence in Design '02*, Kluwer Academic, Dordrecht, The Netherlands, 105–28.

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

Woodbury, R., Burrow, A., Datta, S. & Chang, T. (1999), Typed feature structures and design space exploration, *Artificial Intelligence in Design, Engineering and Manufacturing*, **13**(4), 287–302.

Yu, K., Froese, T. & Grobler, F. (1998), International alliance for interoperability: Industry foundation classes, in K. Wang (ed.), *Computing in Civil Engineering*, American Society of Civil Engineers, Reston, VA, pp. 395–406.

Yue, K., Huber, D. Akinci, B., & Krishnamurti, R. (2006), The ASDMCon project: The challenge of detecting defects on construction sites, *Third International Symposium on 3D Processing, Visualization and Transmission* (*3DPVT* 2006), June.