An Object Ontology Using Form-Function Reasoning to Support Robot Context Understanding

Eric Wang, Yong Se Kim and Sung Ah Kim
Sungkyunkwan University
wang@me.skku.ac.kr, yskim@skku.edu, sakim@skku.edu

ABSTRACT

A robot that acts within an everyday environment needs a machine-understandable representation of objects and their features, shapes, and usages. We report on the development of a generic ontology of objects, and the use of this ontology to instantiate a knowledge base of everyday physical objects. Generic shape representation of objects and features is obtained through form-function reasoning to deduce geometric shape requirements from an object's mechanical and other functions, which supports object recognition. Associational knowledge between objects captures typical associations among groups of objects that are commonly used together, and associations between sets of objects and typical human activities, which supports context understanding.

Keywords: Object Ontology, Feature, Generic Shape, Form-Function Reasoning

1. INTRODUCTION

Consider an autonomous service robot that interacts with a typical human environment. On a small scale, the robot must be able to recognize everyday objects from low-level sensor information. This requires reasoning about objects' intrinsic properties and intended usages. On a larger scale, the robot must interpret scenes containing multiple objects. This involves knowledge of human activities that correspond to configurations of multiple objects. Both of these tasks fall under the more general task of *context understanding*.

We have developed an ontology for everyday physical objects to support context understanding. Using this ontology, we have instantiated a knowledge base of typical objects, including furniture, appliances, office supplies, *etc.* We use the Protégé ontology editor with OWL plug-in [9] for ontology development.

This research considers typical manufactured objects in indoor environments. A manufactured object has been deliberately designed by humans to fulfill a primary usage or functionality. We adopt the approach that an object is characterized by the functions it provides. The function of an object often derives from its geometric shape, and hence dictates that shape. We decompose each object class into its key functions, using a taxonomy of mechanical and other functions. By applying form-function reasoning, we deduce geometric shape requirements for a given functionality. This provides a generic shape representation of objects within our object ontology, which can be used to support object recognition.

Similarly, a configuration of multiple objects in a room is typically "designed" by humans to support a certain kind of human activity. While such configurations of objects could be remodeled at any time, in practice they often remain static for extended durations (months or years). At an informal level, humans tend to blur the distinction between a room and its intended activity, and to label a room according to its currently intended activity, disregarding the volatile nature of the room's configuration. Hence, it is highly relevant for the robot to be able to identify such configurations, and to deduce the intended human activities. Our knowledge base includes associational knowledge between configurations of objects and human activities, which supports scene interpretation.

All objects are affected by the force of gravity. Over typical distance scales, we assume that gravity exerts a vertical downward force everywhere. We ignore specialized or transient physical effects such as acceleration, friction, surface tension, vibrations, etc.

2. RELATED WORKS

The Generic Recognition Using Form and Function (GRUFF) system [10] performs generic object recognition and context-based scene understanding. Objects are decomposed into functions based on the object's intended usage, where each function describes a minimum set of structural elements, and their geometric properties and relations. GRUFF's function-based knowledge is organized in an *is-a* hierarchy of concepts, similar to an ontology. Scenes are decomposed into a hierarchy of functional requirements, which can be satisfied by a set of one or more objects. GRUFF performs context-based scene understanding by comparing a candidate scene concept to the visible evidence, and evaluating the support for all of the scene concept's functional requirements. GRUFF has instantiated function-based knowledge for everyday objects like furniture, kitchenware, and hand tools, and everyday scenes like *office*. However, it does not define any formal representation of generic shapes.

Neumann *et al.* performs context-based scene interpretation [7] within the Cognitive Vision Systems (CogVis) consortium. Scenes are modeled as *aggregates*, where an aggregate is a set of entities and their relations, including spatial and temporal relations. An aggregate is implemented as a partonomy (*is-a* hierarchy of concepts with *part-of* relations). Neumann's current approach is to represent partonomies in description logic, which is equivalent to an ontology. Scene interpretation is then achieved by the DL reasoning service of *model construction*, using the RACER DL reasoner [3]. Visual information is converted into assertional knowledge representing the current observed situation. For each candidate scene class, RACER attempts to construct a model (instance of the scene class) that is consistent with all assertional knowledge. Scene knowledge has been implemented for a limited domain of *table settings*. This work presumes that the recognition of individual objects has already been achieved by other system modules, and thus its ontology-like knowledge representation focuses on scene descriptions, and does not model detailed intrinsic properties or shape information of each object class.

3. FEATURE-BASED OBJECT ONTOLOGY

Real objects are often assembled from many separate components, each of which may have its own primary function. To capture this compositional nature of objects in a generic manner, we adopt a feature-based perspective, shown in Fig. 1. In this work, a *feature* is a functionally significant subset of an object or another feature. We characterize each feature according to its intended functions and usage information. By applying form-function reasoning, we deduce the functional elements, called *organs* [4], which are the active elements that carry the functions, and their geometric shape requirements, as well as any geometric relations and constraints that exist between features.

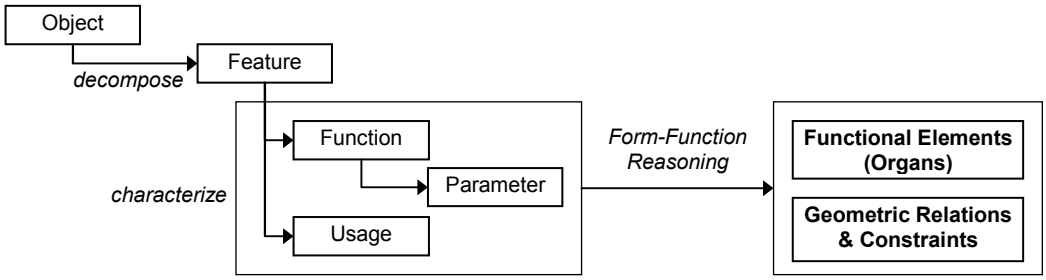

Fig. 1. Overview of Object Ontology Framework

3.1 Part-Whole Representation of Objects and Features

An object may have any number of features. Each feature may itself be composed from many sub-features. Furthermore, a feature could be a 3D solid component, which could be viewed as an object in its own right if it were disassembled from its parent object. This implies that features and objects should share the same data properties. This leads to a part-whole model with a common set of data properties. We implement this using (a variation of) the COMPOSITE design pattern [1] for part-whole models, as shown in Fig. 2. First, an abstract base Descriptor is defined to carry all of the common data properties. Each Descriptor has any number of Feature instances. The topmost Object and Feature base classes then derive from Descriptor, and inherit all of its data properties. In addition, each Feature has a hasParent relation that references its containing Descriptor individual.

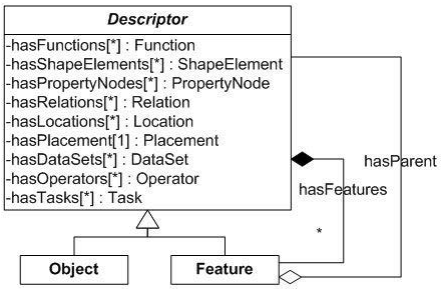

Fig. 2. Part-whole representation of Object and Feature classes

3.2 Geometric Shape Elements

A geometric shape element such as *horizontal planar surface* describes the minimum necessary condition for an object or feature to achieve a desired function. Each shape element composes a geometric datum element with geometric conditions and restrictions, as shown in Fig. 3. A ShapeElement has one GeometryDatum, which specifies the relevant subset of the object or feature's geometry, and zero or more ShapeConstraints. Each ShapeConstraint has one GeometricConstraint, which modifies the datum by specifying a geometric condition such as *horizontal* or *planar*, and zero or more QualitativeModifiers, which allow variations to be specified. The ExtentModifier subclass provides a set of keywords that discretize a range of allowed variations from the nominal condition specified by the geometric constraint: *exactly* means that 0% variation is permitted, *nearly* permits up to 5% variation, *mostly* up to 15%, and so on.

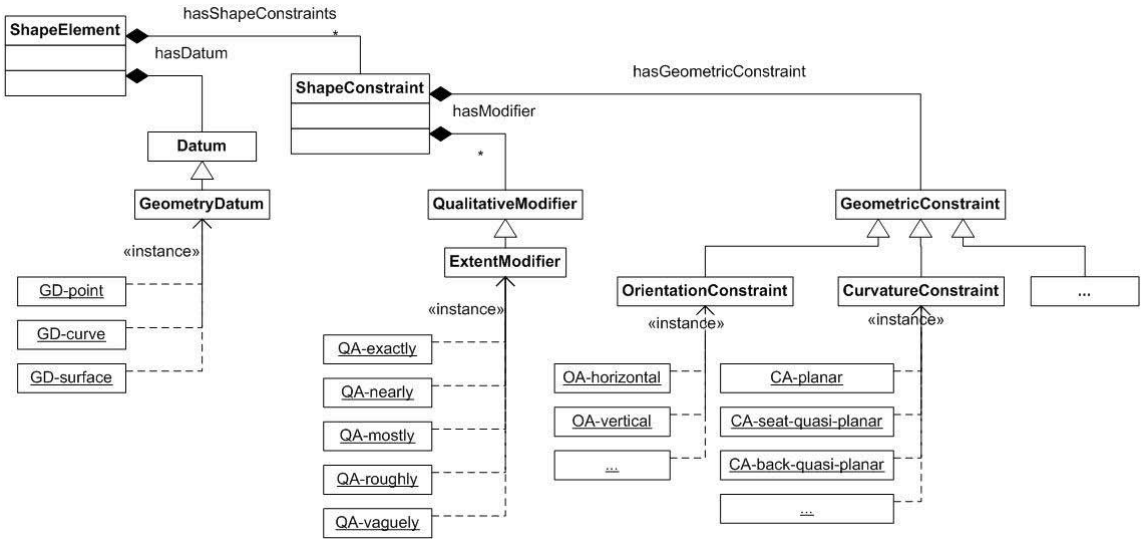

Fig. 3. Representation of shape elements

3.3 Property Nodes and Trees

Every Descriptor has 0 or more run-time data properties associated with it. These can specify simple quantitative attributes such as *height*, or acceptable ranges of values. They can also specify general *n*-ary relations between *n* individuals.

3.3.1 Design Criteria

It is convenient to be able (a) to manage a set of properties as if they were a single individual, (b) to specify properties separately from their values, and (c) to compose sets of properties from other sets of properties. This allows the knowledge of object properties to be distributed in a top-down manner in the object ontology, following a principle of *least commitment*. It also supports incremental definitions, which enables a *scaffolding* approach to ontology construction through the reuse of existing definitions.

To support this, we reify (make concrete) properties as an explicit class hierarchy within the ontology, shown in Fig. 4, rather than encoding them using the intrinsic OWL properties. Furthermore, we define a part-whole structure for properties, using the COMPOSITE design pattern, which consists of an abstract PropertyNode base class with concrete PropertyLeaf and PropertyList subclasses, where a list node may contain both leaf nodes and other list nodes.

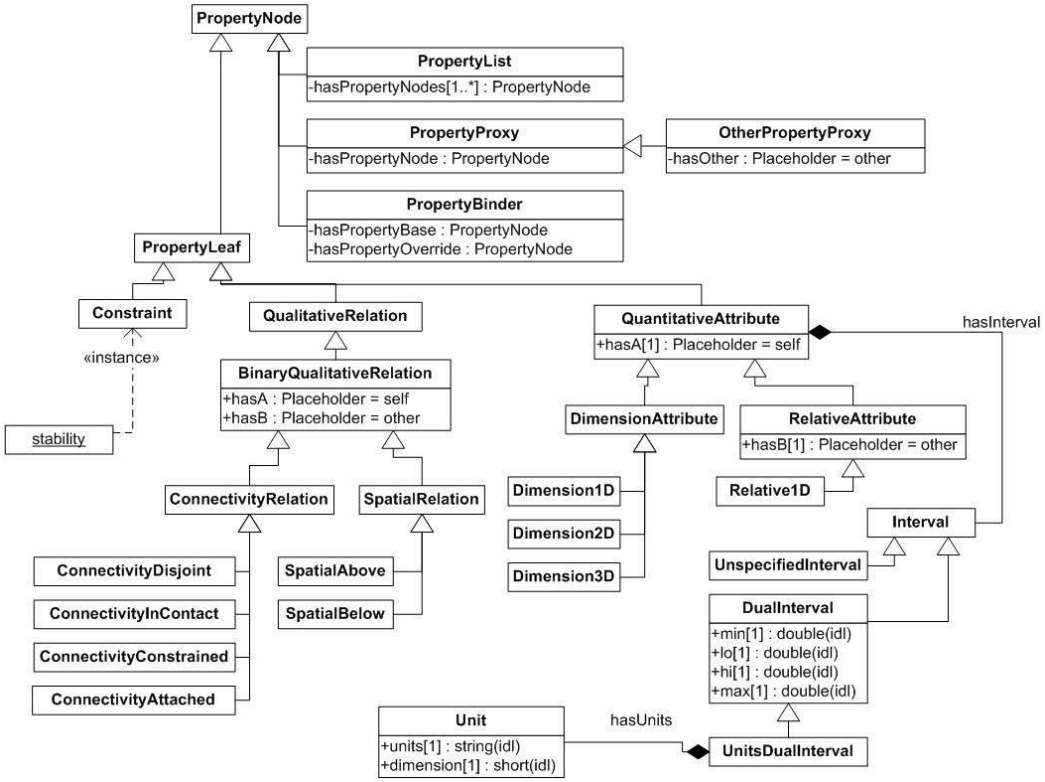

Fig. 4. PropertyNode hierarchy

To avoid combinatorial explosion when instantiating sets of properties, it is necessary to enable properties to be instantiated *without referring to specific individuals*. For this, we define a notion of a Placeholder, which is a syntactic keyword that is automatically replaced at run-time by a specific individual found at a stereotypical location relative to the current individual. The Placeholder value **self** refers to the current Descriptor, **parent** refers to its parent Descriptor, and **other** is used in conjunction with the *other* relation in an OtherPropertyProxy individual, which uses the PROXY design pattern [1].

It is also desirable for an object class to specify only the *existence* of a set of properties, without committing to any set of values, or ranges of values, for those properties. This entails the following mechanisms:

- **Unspecified values.** To allow a property to be declared to exist without specifying its value, we reify the notion of an unspecified value of a class C as an explicit subclass of C.

- **Overriding values.** The PropertyBinder class references a base property node and an overriding property node. Both of these are usually PropertyLists that define some properties. For every property in the overriding list that matches a property in the base list, the value from the overriding list is taken as the instance's value. This allows any class to override sets of property values defined in other classes.

3.3.2 PropertyLeaf

A PropertyLeaf represents a single atomic property. We define the following subclasses of PropertyLeaf.

3.3.3 Constraint

The Constraint subclass represents a geometric condition that applies to the current individual (usually an Obect). We define one individual of this subclass, *GC_stability*, which represents the stability criterion according to the standard definition: *An object is stable if the projection of its center of gravity to the ground plane is contained in the convex hull of the set of its ground contact points.* For simplicity in evaluating this criterion, it may be assumed that the object is rigid and has uniform density.

3.3.4 QuantitativeAttribute

The QuantitativeAttribute subclass represents a property with a quantitative (numeric) value. It defines four data fields:

(1) A name string.
(2) A Placeholder relation *hasA* that indicates the owner of this attribute, which is always **self**.
(3) An Interval, which represents a generic interval.
(4) A units string, which specifies the units of measure for this attribute's Interval.

Two subclasses of QuantitativeAttribute are defined.

- The DimensionAttribute subclass represents a scalar value. It defines further subclasses Dimension1D for 1D quantities such as *height*, Dimension2D for 2D quantities such as *area*, and Dimension3D for 3D quantities such as *volume*.
- The RelativeAttribute subclass represents a scalar value between two things. It defines an additional data field *hasB* to a Placeholder value that indicates the other thing, which is usually **parent** or **other**. Its subclass Relative1D represents a 1D quantity between two things, such as *distance* or *angle*.

3.3.5 QualitativeRelation

The QualitativeRelation subclass represents a general *n*-ary relationship. Its subclass BinaryQualitativeRelation is a relationship between exactly two things. Hence, it defines two Placeholder relations named *hasA* and *hasB*, which are usually assigned the values **self** and **other**, respectively. The following subclasses of BinaryQualitativeRelation are defined:

- ConnectivityRelation specifies how two things are connected. We implement it using the ontology design pattern of a *value partition* (a set of mutually disjoint subclasses) with the following subclasses:
  - ConnectivityDisjoint indicates that objects A and B are separate, and are not in contact.
  - ConnectivityInContact indicates that objects A and B are separate, and are currently in contact.
  - ConnectivityConstrained indicates that objects A and B are connected, but still have some relative degrees of freedom wholly or partially unrestricted
  - ConnectivityAttached indicates that objects A and B are rigidly connected, with all relative degrees of freedom restricted.

- SpatialRelation specifies the relative position between two things. It comprises a value partition with the following subclasses:
  - SpatialAbove indicates that A is above B (with respect to the gravity direction).
  - SpatialBelow indicates that A is below B.

For the bottommost class ConnectivityDisjoint, we instantiate one individual *CD_self_other*, and assign to it the values *hasA* = **self** and *hasB* = **other**. This individual is then a constant value that represents a generic "disjointness" property between any two things. Since it is constant, we may compose it into larger sets of properties. If every property in a set is constant, then the set itself is constant. This provides the basis of our scaffolding approach to ontology instantiation. For each of the other bottommost classes, we instantiate one individual in a similar manner.

3.4 Location and Placement Properties

Every Descriptor (object or feature) has the following properties, shown in . 5.

- **Placement** is the quantitative position and orientation, given in a global or local coordinate system.
- **Location** is a qualitative description of an object's possible locations. An object can have different locations associated with different usages. For example, a book may be stored in a bookcase, but is used on a table.

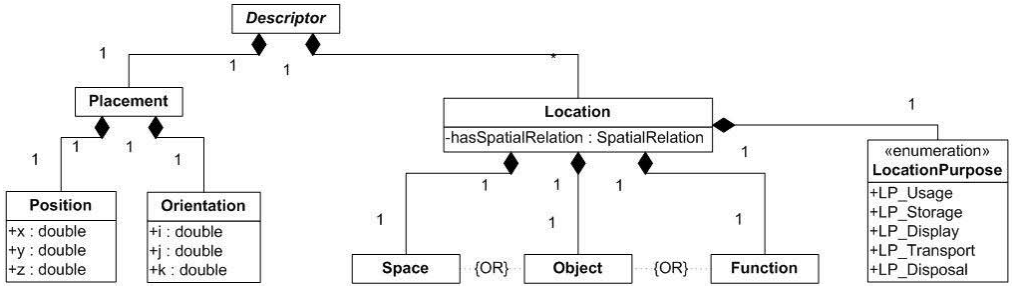

Fig. 5. Placement and Location properties

3.5 Extrinsic Data Sets

To assist in defining visual recognition routines, we provide a generic mechanism to annotate any Descriptor with arbitrary external data sets, as shown in Fig. 6. A common example of an external data set is a disk file containing a canonical image of the Descriptor. We characterize each DataSet according to the following properties.

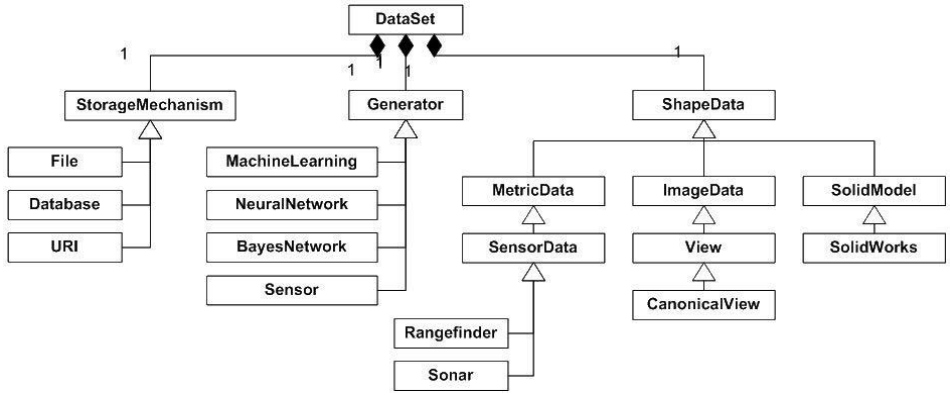

Fig. 6. Representation of external data sets of shape information

- **Storage mechanism**. This generalizes from file-based storage to other storage locations, such as database or web-based.
- **Generator**. External data sets for context understanding are typically generated or processed by other system modules, each of which may attach its own annotations to its output data.
- **Shape data.** Shape data representations can vary widely, including low-level sensor data, "ideal" image data for human understanding, or "perfect" 3D solid model data.

4. FORM-FUNCTION REASONING

The usage of an object is achieved through functions conducted by the object's detailed shape. While objects may have vast differences at the detailed shape level, their functions could be described using a common set of generic functions. Form-function reasoning has been previously considered in design methodologies [8][12], and function-based taxonomies for design have been proposed in [6][11]. From these taxonomies, we adopt the functions of (a) *limit motion* – to restrict or eliminate degrees of freedom; and (b) *support* – to maintain in a fixed configuration or orientation.

The *limit motion* function may be considered as a restriction on the relative degrees of freedom between two things. We parameterize the *limit motion* function with the following arguments:

(a) **Direction:** The set of directions in which motion is limited. Equivalently, this is the set of translational degrees of freedom that are restricted.
(b) **Type:** The type of other objects that are usually involved, according to the object's primary usage. At a high level of abstraction, we consider only Object, Liquid, and Human types.

(c) **Quantity:** The cardinality of other objects affected. This can be an explicit quantitative value, or a qualitative range such as *many*. For Liquid types, this argument is ignored.

We instantiate the following combinations of arguments.

- *Limit motion***<downward, Object, many>**. A table's top surface has a primary function of limiting vertical downward motion for many arbitrary objects. The *polar set* [2] of a vertical downward vector is a horizontal planar halfspace that faces upward with respect to gravity. Hence, we deduce a shape requirement of a horizontal planar surface whose normal vector is upward.

- *Limit motion***<downward, Human, 1>**. A chair's seat has a primary function of limiting vertical downward motion for one human. By reasoning as for a tabletop, we deduce an upward horizontal surface. However, because a seat is primarily intended as a human contact surface, ergonomic considerations provide a significant secondary contribution to its shape. This can result in some contouring to make prolonged contact more comfortable. Yet the seat must remain *approximately* planar to fulfill its primary function. We abstract away the variations in a seat's shape by defining a qualitative condition of *seat_quasi_planar*, which spans a range of surface deformations from perfectly planar to moderately contoured "so as to support a sitting human". At this level of abstraction, we do not commit to any analytic characterization of such contouring.

- *Limit motion***<all-lower-halfspace, Liquid>**. A container of liquids must provide the function of limiting all motions in the lower halfspace, which comprises all motions that have any downward component with respect to gravity, and all horizontal motions that are orthogonal to gravity. By considering this set of limited motions to be an *original normal cone* [5], we conclude that it corresponds to a *pocket* feature volume. Thus, we deduce a shape requirement of an upward pocket with respect to gravity, as shown in Fig. 7.

- *Limit motion***<all-lower-halfspace, Object, many>**. Note that a container of liquids can also contain many arbitrary solid objects. By the same reasoning as above, we still get a geometric requirement of an upward pocket. Currently, we consider only the typical case of relatively small objects that are "completely contained" within the container, i.e. the intersection of each object with the upward pocket's volume is the entire object itself. In other words, no portion of any object protrudes above the (fictitious) top surface of the container's upward pocket.

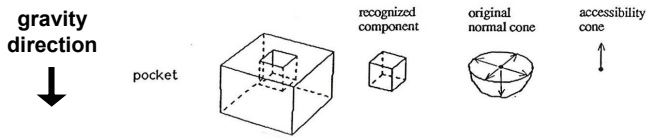

Fig. 7. Upward pocket feature.

5. ASSOCIATIONS AND ACTIVITIES

In addition to knowledge associated with individual objects, humans maintain much useful knowledge that pertains to *groups* of objects. We define a generic representation for associational knowledge among groups of objects, shown in Fig. 8.

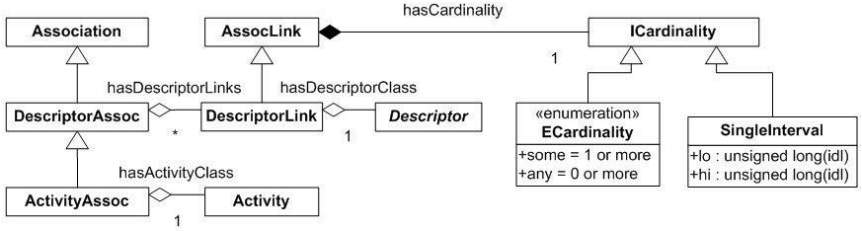

Fig. 8. Associations and Activities

We consider two kinds of associational knowledge.

- **Between objects.** Some objects are commonly used together, or always found in close proximity to each other, or with other recognizable relations. For example, a computer monitor is usually close to a keyboard and mouse.

- **From objects to activities.** Certain human activities are closely associated to some configurations of objects (given that humans will actively reconfigure the objects to achieve their intent until this becomes true). We characterize human activities in terms of the objects they involve. To annotate each object-to-activity link with additional parameters, we reify the link as an explicit class ObjectLink that stores these parameters. We define parameters such as the cardinality of objects, the strength with which the objects are correlated to the activity, and a notion of necessity. This allows the instantiation of detailed associations such as "a strong correlation for several chairs", which could be used to reject a scene that contains only one chair.

6. EXAMPLE: INSTANTIATION OF TABLE AND STANDARDTABLE CLASSES

We illustrate the use of the object ontology for generic shape representation by instantiating the Table class, shown in Fig. 9. A Table is an abstract base class for *all* possible tables. We decompose Table into the following features:

- The key functional element of a table is its top surface. We represent this as one or more Counter features, each having a ShapeElement of *horizontal planar surface*. A Counter also defines properties of *height* and *area*, but does not commit to any specific values.

- A key characteristic of a table is that the top surface remains at a fixed height somehow. This implies that it is supported against the downward force of gravity. We represent this as one or more Supporter features. At this level of abstraction, we do not commit to any geometric characteristics of each Supporter.

- A Table must be stable. Hence, it has a GC-stability property.

We define UnderSupporter as a subclass of Supporter that is always *attached below* the supported Descriptor. This is imstantiated as an OWL restriction that UnderSupporter shall always have an OtherPropertyProxy that composes an *attached below* relation between itself and another Descriptor, using constant individuals CONN-attached and SPA-below. For a (subclass of) Table, the other Descriptor will be that table individual's Counter feature.

Next, we instantiate StandardTable as a subclass of Table, shown in Fig. 10. The StandardTable class represents all typical office tables, which we characterize as follows:

- *Generally vertical legs.* A StandardTable requires that all of its Supporter features be VerticalUnderSupporter (as an OWL restriction). VerticalUnderSupporter inherits the *attached below* relation from UnderSupporter, and additionally defines a *mostly orthogonal* relation between itself and the StandardTable's Counter feature.

- *Roughly thigh-height and typical area range*. StandardTable specifies an OWL restriction that it shall have a PropertyBinder that overrides its Counter feature's *height* and *area* properties with specific ranges of values, which were derived from empirical observations.

7. CONCLUSIONS

We have presented an ontology for generic shape information, in which objects are decomposed into features, and features are associated with shape elements. Form-function reasoning is used to deduce shape elements and their geometric shape requirements from object functions. This work can support a generic object recognition capability by combining with other modules that implement the detailed recognition algorithms for each shape element, which supports context understanding.

From the example of instantiating two classes, we see that some amount of verbosity is unavoidable to fully describe even a simple object class. However, the object ontology provides mechanisms to define and reuse constant sets of properties, which eliminates unnecessary verbosity, and supports a *scaffolding* approach to ontology instantiation. It also provides a mechanism to override property values, which supports a *least commitment* approach to knowledge organization.

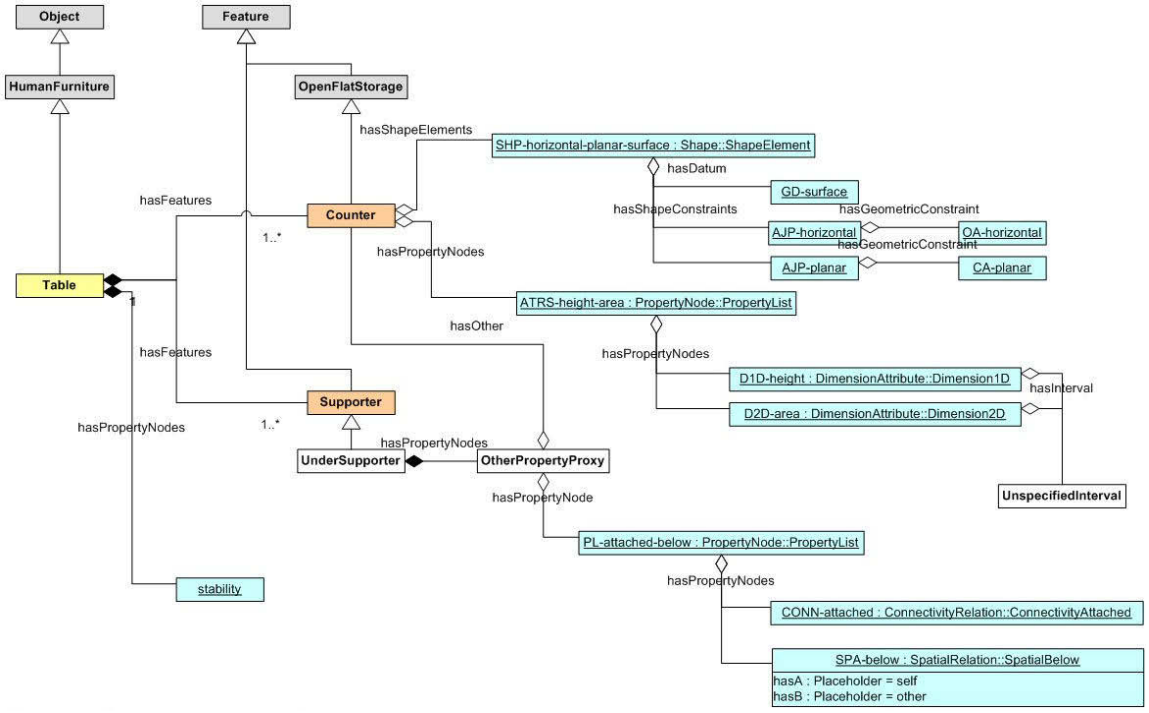

Fig. 9. Instantiation of Table class

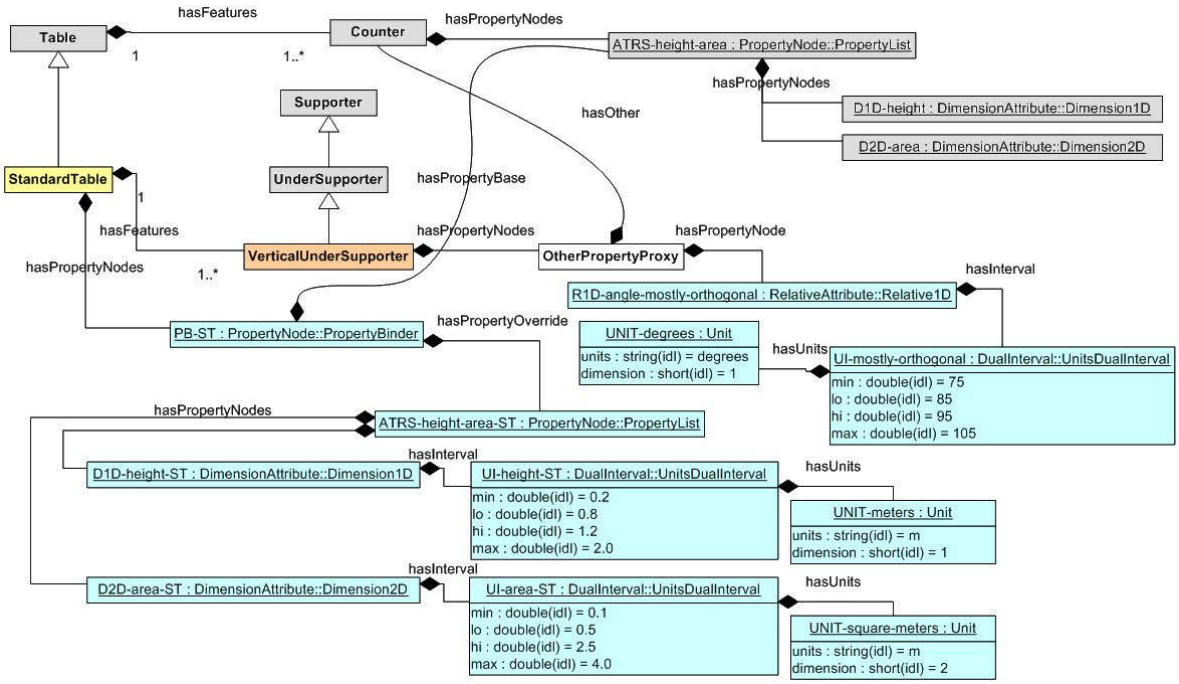

Fig. 10. Instantiation of StandardTable class

8. REFERENCES

[1] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*, Addison Wesley, 1995.

[2] Grünbaum, B., *Convex Polytopes*, John Wiley & Sons, Ltd., 1967.

[3] Haarslev, V., and Möller, R., RACER System Description, *Proc. Int'l Joint Conf. on Automated Reasoning* (IJCAR 2001), LNAI Vol. 2083, Springer, 2001, pp. 701–705.

[4] Haudrum, J., *Creating the Basis for Process Selection in the Design Stage*, Ph.D. Thesis, Institute of Manufacturing Engineering, Technical University of Denmark, 1994.

[5] Kim, Y. S., Recognition of Form Features Using Convex Decomposition, *Computer-Aided Design*, Vol. 24, No. 9, pp. 461–476, Sep. 1992.

[6] Kirschman, C. F. and Fadel, G. M., Classifying Functions for Mechanical Design, *Journal of Mechanical Design*, Vol. 120, pp. 475–482, 1998.

[7] Neumann, B., and Möller, R., On Scene Interpretation with Description Logics, *FBI-B-257/04 (Technical Report), Fachbereich Informatik*, Universität Hamburg, 2004.

[8] Pahl, G., and Beitz, W., *Engineering Design*, Design Council, London, 1988.

[9] Protégé, http://protege.stanford.edu.

[10] Stark, L., and Bowyer, K., Function-Based Generic Recognition for Multiple Object Categories, *Computer Vision, Graphics and Image Processing*, Vol. 59, No. 1, pp. 1–21, Jan. 1994.

[11] Stone, R. B., and Wood, K. L., Development of a Functional Basis for Design, *Proc. ASME Conf. on Design Theory and Methodology*, Las Vegas, 1999.

[12] Welch, R., and Dixon, J., Representing functions, behavior and structure during conceptual design, *Proc. ASME Conf. on Design Theory and Methodology*, Scottsdale, Sep. 1992.