Computable Dynamic Design Repository for Product Data Representation

Al Zeiny, Ph.D., P.E.[*]

ABSTRACT

The product representation for storing design information presented in this paper is based on a dynamic object-oriented data model. This model stores design data as they are generated during design in a computable format, as well as supports case-based reasoning and sharing of data among all design teams. This model represents each entity in the product as a generic container that encompasses its form, function, behavior, taxonomy, composition and relationships. Different features can be added dynamically to the container as needed. The model integrates multiple views of various design teams, supports design evolution and exploration, and is extensible.

INTRODUCTION

The first step necessary in solving a problem with a computer involves the development of a representation for the problem (Simon 1996). Hence, the development of any design environment requires the definition of a design data model, known as the design repository. The product development process has some inherent characteristics that impose requirements on such a model. The design of a product requires a team of designers from numerous design disciplines, each contributing a particular body of knowledge and expertise to the overall effort. A product is considered successful when the contributions of the designers and their respective views are integrated into a harmonious whole. The product design process produces design information that naturally grows in quantity and quality as the design evolves. Thus, data that are only outlined in the early stages of design are gradually modified, enhanced, and detailed as the design unfolds. The product design process is also exploratory in nature. It is an iterative process in which design alternatives are synthesized and analyzed until a satisfactory solution emerges (Rivard and Fenves 2000). The relevant features of the design problem manifest themselves as the design proceeds and depend on the decisions taken. Hence, the product representation should support dynamic changes not just in terms of values and instances, but also in terms of model structure (Eastman and Fereshetian 1994). The product design process evolves over time with the advent of new methods, technologies, requirements, and products. It is impractical to require a design-information model to be fully defined before implementation. Changes need to occur as a result of the following:

1. Selecting, changing or elaborating the material or fabrication technology used in any part of the design: These define a particular compositional structure of an object and/or the addition of particular attributes to the object.

[*] Assistant Professor, Department of Mechanical and Civil Engineering, University of Evansville, Indiana 47722. Email: az12@Evansville.edu

2. Changing any of a predefined set of analyses: Each such evaluation requires its own, possibly unique, set of materials or geometric properties. These must be added to the affected objects, by modification or specialization. Extraction and formatting capabilities are possibly required if the evaluation is run as an external program.

It has been apparent for some time that information model evolution is a requirement for design. Design involves both the definition of an information structure and also the filling of that structure with values. Dynamic capabilities cannot be simply appended onto a model initially defined statically. Problems arise in relation to actions that modify an object that is specialized into many other objects, modifying the definitions of possibly many objects. Changes in general cause model integrity issues to rise. The implications of dynamic change become apparent in the formal definition of a data model, in relation to its axiomatic properties. These must explicitly deal with extension and modification.

RELATED WORK

Several product design representations have been proposed by both industry and academia. There are two international standardization efforts that address the representation of product designs: STEP (STandard for the Exchange of Product model data) being developed by the International Organization for Standardization (Burkett and Yang 1995); and the Industry Foundation Classes, which are specifications for a set of standardized object definitions, being developed by the International Alliance for Interoperability (Kiviniemi 1999). Both standards address later stages of design, do not explicitly support design evolution, and define static product representations aimed at the transfer of information between applications. Other product or design representations have been proposed in the literature such as RATAS (Bjork 1989, 1992, 1994), EDM (Eastman 1992), multiple views and representations (Rosenman and Gero 1996), principal and joint model (Dias 1996), Generic Object-Oriented Detailed Design of Products (GOODoB) (Biedermann and Grierson 1995) and the Building Entity Model (BENT) (Rivard and Fenves 2000). In addition, another approach in the research in the area of intelligent design systems is the approach that attempts to integrate three fundamental facets of an artifact representation: the physical layout of the artifact (form), an indication of the overall effect that the artifact creates (function), and a causal account of the operation of the artifact (behavior). Different models of this type have been developed by various researchers, including (Goel et al 1996, Gorti et al 1998, Qian and Gero 1996, Szykman et al 1999, Umeda and Tomiyama 1997, Iwasaki and Chandrasekaran 1992, and de Kleer and Brown 1983) among others.

DATA MODEL

As shown in Figures (1) and (2), the design repository model consists of a set of constructs that provide the basic building blocks for representing designed artifacts and is based on a generic container, called Design Entity, that encompasses a product design entity's properties, classification, parameters, composition, relationships and geometry. The Design Entity object glues together the modular information of a product design entity and provides a single standardized interface to access information about any entity in the artifact. Because designers typically break down complex problems into small sub-problems, it is natural to arrange these entities in a hierarchical manner that parallels the divide-and-conquer strategy. The Design Entity object could represent the entire artifact or a small piece of it. The decomposition client object, when attached to the Design Entity object or any other data object, it allows the breakdown of this object into sub-objects and establishes the necessary containment hierarchy relationships.

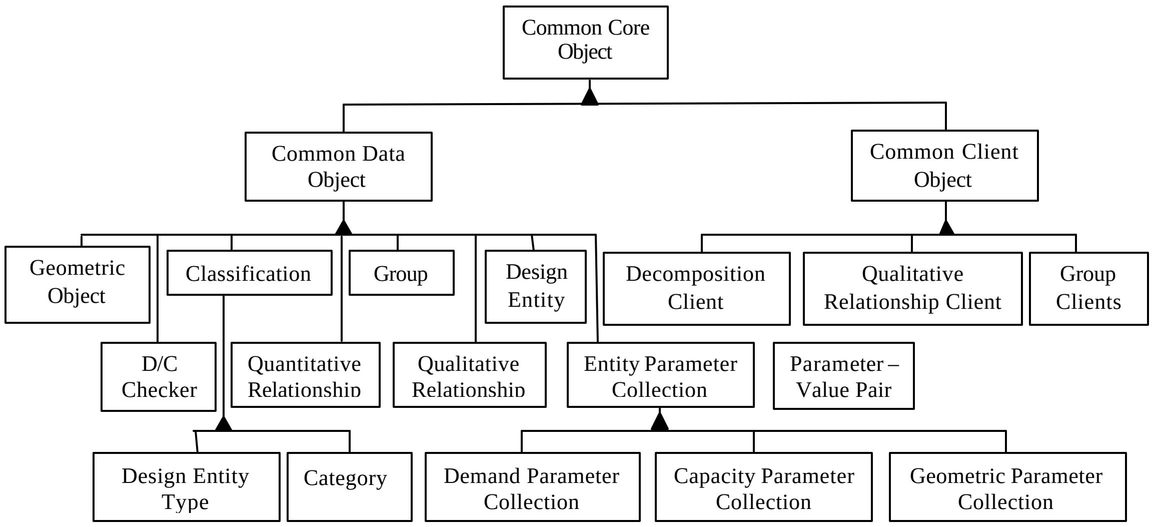

Figure (1): Inheritance Relationship between Model Object

Common Core Object

The main purpose for the Common Core Object is to provide common interface and features to all objects in the model. Among these features is to make the model dynamic and extensible. This achieved by allowing objects inherited from it to be dynamically attached together and routing the messages to the appropriate object among the group of attached objects. When a message is sent to an object inherited from Common Core Object and that object is not able to respond to the message, it simply forwards the message to the next attached object, which in turn either responds to it or forwards it to the next attached object. The message forwarding continues until one of the attached objects responds to it. To illustrate this functionality, let us examine the object arrangements in Figure (3). When a message is sent the Design Entity object to request a pointer to its Super Design Entity for example, it routes the message to the attached

Group Client object to respond to it. Since the Group Client object can not respond to this message, it forwards it to the attached Decomposition Client that responds to the message by returning the requested pointer. The routing function has a virtual interface and it simply calls the same function in the attached object. If the attached object can respond to the message, it will then execute the corresponding method, otherwise, it will forward the message to the next attached object, and so forth. This allows adding additional functionalities to any object as needed during run time by dynamically attaching a descendent of a Common Client Object to it. In the example shown in Figure (3), the Design Entity object has three client objects attached to it to add three different functionalities to it; the Group Client enables the Design Entity object to participate in various groups; the Decomposition Client object enables the Design Entity object to participate in a breakdown structure where Design Entity objects may have super- and sub- objects; The Qualitative Relationship Client object enables the Design Entity object to have one or more Qualitative Relationships attached to it. The client objects are only created and attached during run time as needed. For example, the Group Client is only created and attached to a descendent of the Common Data Object only if it is desired for the data object to participate in groups. The Common Core Object provides the common interface between various model objects as well as the routing functions.

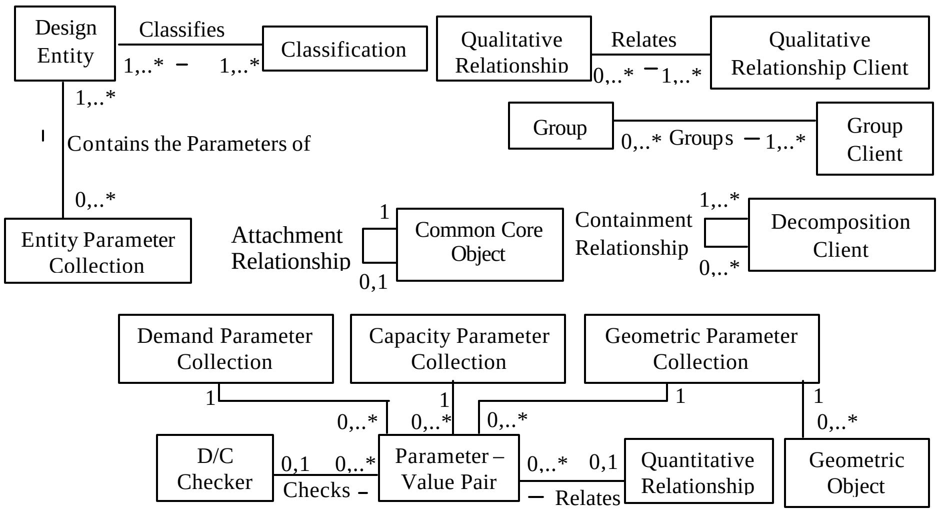

Figure (2): Relationships between Model Objects

Common Data and Client Objects

The Common Data Object contains data common to all objects such as the object name, author and id. It also provides common interface among all data objects. The decedents of this object are the ones that contain the product design data while the decedents of the Common Client Object provide added functionalities to them. Currently, three functionalities are available which are; participating in groups through the Group Client object, ability to be broken down into a decomposition tree where each object has sub-

and super-objects through the Decomposition Client and ability to have a qualitative relationship object through the Qualitative Relationship Client.

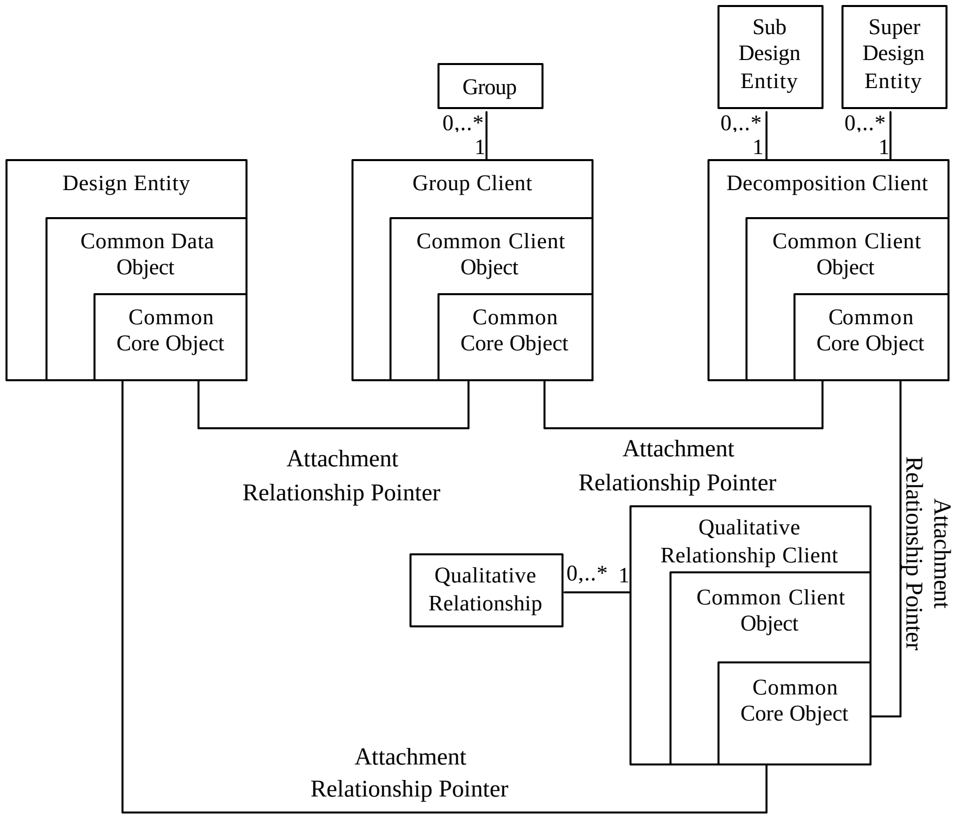

Figure (3): Example of Message Routing

Storing Product Design Information

Design information are stored in a containment hierarchy of the Design Entity objects. The Design Entity object is generic container used to store the design information. The object in the root of the design entity hierarchy may represent the entire project while the object in the bottom of the hierarchy tree may represent the smallest entity in the project. An entity is something that can be distinctly identified in a building design and about which data are accumulated. Each design entity represents a concept meaningful to design participants such as a gear, a couple, or a structural frame. An entity can be a system, a sub-system, a constituent, a part, a feature of a part, a space, or a connection. All product entities are modeled with the Design Entity generic container that glues together the modular information of a product entity and provides a single standardized interface to access information about any entity in a product.

The Design Entity object is linked to Entity Parameter Collection objects that stores the design parameter data in a set of parameter-value pairs. These pairs may be organized according to the following three approaches (Rivard and Fenves 2000):

1. The collection of all parameters defining an entity may be grouped into one flat structure.
2. The parameters may be divided into small cohesive subsets.
3. Each parameter may be represented as a distinct structure.

The first and last approaches correspond to the two extremes of a scale. The first approach leads to the creation of exceedingly complex entities that require values that may not be defined at instantiation time, do not facilitate the display or exchange of a subset of the parameters, and are difficult to maintain, extend, and understand by a single specialist. Furthermore, it cannot address unexpected situations that require the addition of new parameters in an already instantiated object, because each attribute has to be hard-coded in advance in the structure. In the third approach, at the other extreme, each attribute is stored separately. The attribute values are accessed by attribute names. This approach leads to complex and obscure naming conventions (e.g., preliminary_thickness, nominal_thickness, and actual_thickness).

The approach selected for the presented model is located between these two extremes. The parameter-value pairs characterizing a product entity are organized at two hierarchical levels, the group level and the collection level. At the group level, data are grouped into three subsets, the Capacity Parameter Collection group, the Demand Parameter Collection group, and the Geometric Parameter Collection group. The Capacity Parameter Collection group defines intended purposes, requirements, and constraints on the entity that have to be satisfied to realize the intended purpose. The Demand Parameter Collection group includes all the physical and spatial characteristics that define the actual design of the entity as well as the behavior of this entity under various conditions. In order for the design to be successful, designed demands must not exceed capacity (e.g., D/C ration is less than 1). This requirement is checked by the D/C Checker objects. The Geometric Parameter Collection group has the geometric parameters linked to the drafting software to ensure that the product drawings change automatically as geometric parameters change. While Qualitative Relationship objects link design entities together with domain specific relationships, the Quantitative Relationship objects link various parameters together to ensure the propagation of any parameter changes accordingly in a fashion similar to what happen in spreadsheets. Such a feature makes the model highly computable and various design alternatives may be explored easily, not to mention the ability of generating similar new designs from old ones by creating templates from old designs.

At the second level of data aggregation, the parameter-value pairs of an entity are combined into small cohesive subsets, each of which is called a parameter collection. A collection is defined as a group of closely related parameters that are found together in a repository (access-cohesive), instantiated at the same time (time-cohesive), and that represent the same concept (concept-cohesive). Cohesion is the only criterion used in decomposing entities. It is defined as a measure that shows how closely the parameters of an entity relate to one another. An example of a Parameter Collection is one that collects together the parameters used to describe the section properties of structural members such as depth, width, cross section area, and moment of inertia. Parameter collections allow entities to be refined in staged steps by adding sets of parameter-value pairs to the entity as they are generated in the design process. Hence, there is no need to predict all possible parameter-value pairs needed in a product entity at the outset. Parameter collections also allow the integration of multiple views by multiple design teams in one entity by including collections that are specific to each view and each design team as well as components that are shared among all views and all design teams (e.g., material properties.

Classification

A classified instance is a label assigned to a Design Entity for the purpose of classifying and indexing it in the knowledge base. Classifier instances are required to classify the generic building entities as they are being refined during the design process. Classifier instances are also used as indexes for querying the case library. The classifications are arranged together in containment hierarchy that refines the entity classification as we traverse down the hierarchy tree. Each tree node in the classification tree has a super classification, which is more generic and a sub-classification, which is more specific. In other words, classifiers are arranged in a subsumption hierarchy in which a narrower classifier is recognized to be part of a broader one. The root of every classification tree is a Category object, which defines a specific aspect of an object and consists of a hierarchy of classifiers. Each category object is linked to one or more Design Entity Type that declares the type of that specific design entity. Only classifications that are linked to a Design Entity object are possible classification objects for this Design Entity object. The Design Entity Types are also arranged together in a hierarchical decomposition fashion that describes possible types of product entities some of which may be alternatives to one another and others may complement each other.

CONCLUSION

The presented model has the following advantages:

- A generic container that that encompasses a product design entity's properties, classification, parameters, composition, relationships and geometry. The Design Entity object glues together the modular information of a product design entity and provides a single standardized interface to access information about any entity in the artifact.
- A supporting system of classification that supports querying the knowledge base for the purpose of case adaptation of the entire product or a portion of it
- The flexibility of adding and removing various client objects to the data object dynamically during run time to add or remove desired features from the data object as well as accommodate various changes

- The ability of any data object to participate in a group that reduces the amount of data entry and computations for similar design entities
- Related parameters are linked together with a quantitative relationship object to ensure the propagation of changes to the dependent parameters when the independent parameters change. Geometric objects are also linked to geometric parameters to ensure the automatic update of design drawings when design parameters change

REFERENCES

Biedermann, J. D., and Grierson, D. E. (1995). "A Generic Model for Building Design," Engrg. with Comp., London, 11(3), pp. 173–184.

Bjork, B. (1994). "RATAS Project—Developing an Infrastructure for Computer-Integrated Construction," J. Comp. in Civ. Engrg., ASCE, 8(4), pp. 401–419.

Bjork, B. (1992). "A Unified Approach for Modeling Construction Information," Build. and Environment, Oxford, U.K., 27(2), pp. 173–194.

Bjork, B. (1989). "Basic Structure of a Proposed Building Product Model," Comp.-Aided Des., Oxford, U.K., 21(2), pp. 71–78.

Burkett, W. C., and Yang, Y. (1995). "The STEP Integration Information Architecture," Engrg. with Comp., London, 11(3), pp. 136–144.

de Kleer, J. and Brown, J. (1983). "Assumptions and Ambiguities in Mechanistic Mental Models," Mental Models, D. Gentner and A. L. Stevens (Eds.), Lawrence Erlbaum Associates, New Jersey, pp. 155-190.

Dias, W. P. S. (1996). "Multidisciplinary Product Modeling of Buildings," J. Comp. in Civ. Engrg., ASCE, 10(1), pp. 78–86.

Eastman, C. M., and Fereshetian, N. (1994). "Information Models for Use in Product Design: A Comparison," Comp.-Aided Des., Oxford, U.K., 26(7), pp. 551–572.

Eastman, C. M. (1992). "A Data Model Analysis of Modularity and Extensibility in Building Databases," Build. and Envir., Oxford, U.K., 27(2), pp. 135–148.

Goel, A., Gomez, A., Grue, N., Murdock, J. W., Recker, M. and Govindaraj, T. (1996). "Explanatory Interface in Interactive Design Environments," Artificial Intelligence in Design '96, J. S. Gero (ed.), Kluwer Academic Publishers, Boston.

Gorti, S., Gupta, A., Kim, G., Sriram, R. and Wong, A. (1998). "An Object-Oriented Representation for Product and Design Processes," *Computer-Aided Design*, Vol. 30, No. 7, pp. 489-501.

Iwasaki Y. and Chandrasekaran, B. (1992). "Design Verification through Function and Behavior-Oriented Representations: Bringing the Gap between Function and Behavior," Artificial Intelligence in Design '92, J.S. Gero (Ed.), Kluwer Academic Publishers, Boston, pp. 597-616.

Kiviniemi, A. (1999). "IAI and IFC—State of the Art." Proc., 8th Int. Conf. on Durability of Build. Mat. and Components, M. A. Lacasse and D. J. Vanier, eds., Vol. 4, NRC Research Press, Ottawa, pp. 2157– 2168.

Qian, L. and Gero, J. (1996). "Function-Behavior-Structure Paths and Their Role in Analogy-Based Design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 10, No. 4, pp. 289-312.

Rivard, H. and Fenves, S. (2000). "A Representation for Conceptual Design of Buildings", Journal of Computing in Civil Engineering, ASCE, July, Vol. 14, No. 3, pp. 151-159.

Rosenman, M. A., and Gero, J. S. (1996). "Modelling Multiple Views of Design Objects in a Collaborative CAD Environment." Comp.-Aided Des., Oxford, U.K., 28(3), pp. 193–205.

Szykman, S., Racz, J., Bochenek, C. and Sriram, R. (1999). "A Web-based System for Design Artifact Modeling," Design Studies.

Simon, H. A. (1996). "The Sciences of the Artificial," 3rd Ed., MIT Press, Cambridge, Mass.

Umeda, Y. and T. Tomiyama, "Functional Reasoning in Design," IEEE Expert Intelligent Systems and Their Applications, Vol. 12, No. 2, 1997, pp. 42-48.