Interface Design Issues in a Solid Model Derivation System

William A. Lange Jr., Manager, R&D InfoTech Enterprises, 4900 Seminary Road, Suite 850, Alexandria, VA., 22311. Email: blange@infotech.org

Dr. Oscar N. Garcia, NCR Distinguished Prof. and Chair, Department of Computer Science and Engineering, 303 Russ Engineering Center, Wright State University, Dayton, OH 45435. Email: ogarcia@cs.wright.edu

Abstract

Difficulties with part oriented solid modeling tools have motivated the development of a derivation-based solid modeling system to aid in preliminary submarine design tasks. The projected system is a graphically editable geometric rule system based on Extended Pattern Grammars with Variables (EPGVs). This paper reviews relevant characteristics of Boundary Representation Modelers, Feature-based Modelers and Extended Pattern Grammars, then establishes a fundamental relationship between the Inference Dependency Graph that is constructed during an EPGV derivation and a parametric system Feature tree. The primary technique involved with editing an EPGV derived model is identified with Dependency Driven Backtracking, and a description of the process is presented.

This work is funded under NAVSEA contracts N00024-96-C-4180, and N00024-97-C-4186, in association with the Small Business Innovative Research (SBIR) program.

Introduction

Early stage submarine design entails a multi-step process wherein:

- Ship geometry is established and compartments are arranged to determine the minimum size ship that satisfies the mission requirements;
- Displacements and weights are calculated;
- If the ship's displacement exceeds its weight, ballast is added to achieve neutral buoyancy and no further compartment arrangement work is required. However, if the ship weight exceeds its displacement, ship size is increased (usually by adding length) until weight and displacement are equal. Since this larger ship provides more volume and deck area than was necessary to accommodate the required systems, compartment rearrangement is not critical.

Past attempts to accomplish the first step quickly using conventional Computer Aided Design (CAD) systems or Solid Modelers have presented frustrating complications due to time consuming model construction, mass property calculation inaccuracies, complicated editing operations and sometimes model-corrupting software upgrades. The problems encountered are similar to those that limit the application of Solid Modelers to fast generation of building Architecture designs. Potential solutions to the problem were examined through a Small Business Innovative Research (SBIR) Phase I evaluation program. A promising direction was found which takes advantage of compilation possibilities offered by viewing geometric construction as a grammatical generation process, and which utilizes an inferencing system within which productions of the grammar are derived. The resulting system holds the promise of defining a derivation-based Feature Solid Modeling environment, supporting more open editing at higher levels of abstraction. The following sections present a brief review of the involved technology, identify some problematic user interface issues and present the solutions which are currently being explored in Phase II of the SBIR work.

Boundary Representation Modelers

Most commercial architecture and engineering oriented solid modeling systems use some form of Boundary Representation (Brep) Modeler kernel. The currently predominant commercial kernels are Spatial Technologies' ACIS system and EDS's Parasolid system, both of which are original descendents from work on boundary-defined solids initiated in Cambridge England. (ACIS 96a,b,c) A notable exception to the use of commercial kernels is that of the Pro / Engineer solid modeling system which uses a proprietary modeler. Detailed descriptions of the Brep approach to solid modeling can be found in a number of texts. (HOFFMAN 89) (FOLEY 90) (MORTENSON 85) There a a few points which are important in the context of defining grammars in such environments:

- Modern Brep kernels can represent both manifold and non-manifold constructions.
- Topological set definition is characterized by a complex graph of inter-linked entities.
- The inter-linked entities represent a selection of geometric primitive types, such as points, curves, surfaces and bodies, each of which is used to define boundaries for entities of higher dimension.

The modelers also provide both data representation editing procedures and evaluation procedures that calculate functions of the data representation such as mass properties or surface areas.

Feature-based Modelers

The provision of only Boolean editing operations in Brep kernels is generally insufficient to support an adequate editing interface for engineering and architectural design work. Limitation to only these operations results in many unintuitive construction operations. For example, the inside of a compartment or room must be created as a void within a larger solid. From a designer's point of view, a more natural approach is to construct an interior volume by the sequential introduction of bounding surfaces. Feature-based modelers have evolved this constructive approach toward the editing of solid models. Pro / Engineer was one of the first of this class of modelers, however today most part-oriented design modelers - for example SolidWorks - offer feature-based construction and editing mechanisms. (MCLELLAN 96) (SOLIDWORKS 97)

Generally, a feature-based modeler supports user definition of 2D sketches using predefined primitives, then generates solids by the control of Brep operations such as sweeps or extrusions performed using the sketches. Complex 3D solids are the results of composition of the first order solids under the control of subsequent Boolean or other Brep operations. The key points here are that:

- Each feature can be regarded as mapping to an expression that is evaluated to construct entities composing the solid model. In some systems these expressions can be regarded as constituting a single function. Most commercial modelers use some variant of a relational constraint manager; such as D-Cubed's 2D dimension constraint manager (DCM) that is used in the evaluation of user defined 2D sketches in SolidWorks. The D-Cubed product is a two-pass constraint manager. The first pass establishes an order of calculation specific to the constraints and entities involved, and the second pass calculates the final parameter values. The first pass is on the order of five times slower than the second pass. (D-CUBED 97a,b)

- The set of all features in a model defines a feature tree, which represents a partial order for the evaluation of the involved expressions. It is also important to note that dependencies are established between the input parameters of the expressions in the tree and the preexisting model entities to which they refer. We say that the preexisting entities *support* the expression, or that the expression is *dependent-on* the preexisting entities. Since preexisting entities must also be a function of the evaluation of prior expressions, we say that these entities are dependent on the expressions that generated them. It is therefore possible to define transitive dependency relationships over the system of expressions and entities.
- Generation or regeneration of a model is accomplished by traversal of the feature tree. Successful evaluation of the tree results in the controlled generation of a Brep solid model. Evaluation can, and often does fail, when parameter values are introduced which are outside the domains of the functions from which the expressions are composed. For example, a model which assumes a four sided rectangle as one of its sketch features, and which constructs geometry on each of the four sides, can fail if the length of one of the rectangle sides is reduced to zero. Very often such failures coincide with a change in the underlying topological set representation of the Brep model.

As suggested in the first point above, the means by which dependencies of the feature tree expressions and resulting geometry are defined or detected is important, and is one of the ways by which different systems can be classified. Parametric systems are function composition systems. The dependencies established by constructive operations define a static order for the calculation of feature tree expressions. Such systems tend to be fast but often unfriendly with respect to user editing. Variational systems, such as those using the D-Cubed product, attempt to detect a dependency order first, which is then evaluated. (PIERRA 96) Fudos' and Hoffman's work on representation and evaluation of 2D constraints also falls into this category. (FUDOS 93) (HOFFMAN 95) These systems are more forgiving from the user's point of view, however they require a two-pass implementation and the dependency detection pass is generally computationally expensive.

All current feature-based modelers have problems generating a set of solid models that vary significantly in topological set definition. They can handle some degree of variation of the input parameters for the basic topological set that the feature tree constructs, but consideration of alternate topological set definitions requires radical surgery to the feature tree which results in a great deal of user interaction. The most help that current feature-based modelers can offer in this regard is the provision of arrays of features, which are simply repeats of an already defined feature, or procedurally programmed choices between alternate, predefined partial models. This problem is closely associated with the problems that arise when a feature-based modeler is to be extended to support use of a library of features and parts. The system that our project is developing attempts to deal with some of these model construction problems.

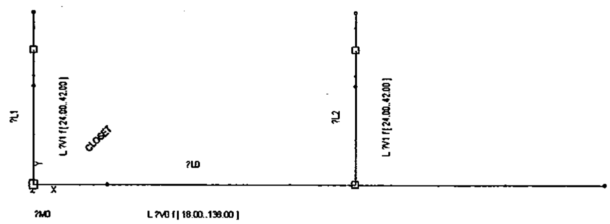
Extended Pattern Grammars

In the 1970's K.S. Fu defined Pattern Grammars by mapping a coded set of geometric primitives onto conventional string grammars. The primitives, termed chain codes, defined a very limited set of two or three-dimensional vectors defined within a standard size square or box. These systems have found use in image recognition tasks and more limited application in image generation tasks. (FU 74)

Recently, Extended Pattern Grammars have been defined by generalizing Pattern Grammars using the following means. We generalize the rewrite location by introducing a string labeled three-dimensional spatial marker which records a rule to model space transform. These constructs correspond to the non-terminal symbols of a conventional string grammar. The vector primitives of Pattern Grammars are generalized by substituting parametric definitions for lines. Further extension of the resulting class of grammar can be accomplished by introducing variables that associate a bounded real number variable with each parameter. In the case of point and direction parameters, we also introduce simple functions that characterize the degree of closeness or match between a possible value and a default value for each parameter. For example the match function for variable points can be the Euclidean distance between a default point and a possible matching point. This class of grammar is called an Extended Pattern Grammar using Variables (EPGV). (LANGE 95,96,97) This type of grammar has much in common with Stiny's Shape Grammars and Heisserman's Boundary Solid Grammars. (STINY 80,81a,81b,90) (GIPS 80) (CHASE 89) (HEISSERMAN 91) EPGVs need not maintain a normal form and as a result are generally faster and easier to implement than shape grammars. The EPGV marker supports EPGV use in assembly contexts, unlike boundary solid grammars. Finally, EPGV rules are embeddable in forward chaining inference systems, thus providing access to an assortment of implementation techniques. They are composed of much higher level rewrite rules than the rule based geometry systems described in (JUAN-ARINYO 95a,b).

It is notable that the speed and memory capacity of computers in the 1970's generally would have prevented effective implementation of EPGVs. EPGV systems, like any inferencing system, depend heavily on pattern matching. The implementation techniques used to provide matching support generally have been the primary stumbling block for effective use of such systems since they require CPU speeds and memory sizes that only now are becoming widely available. An example of a graphically represented EPGV rule is shown in illustration 1.

IF: Condition



THEN: Action

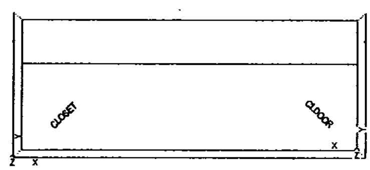


Illustration 1: Graphic Representation of an EPGV rule for house closets.

A properly bounded EPGV rule encapsulates a single topological set that is defined over an infinite set of possible real valued parameter bindings. Such rules can be represented by a characteristic pattern that utilizes some valid choice of parameter value bindings. Any valid instantiation of the variable bindings can be chosen to define a pattern that is regarded as the rule's characteristic pattern. Characteristic patterns can be used to provide a graphically editable form for an EPGV rule. In addition, since each instantiation of a rule in a derived model also can be regarded as a characteristic pattern, many of the same basic editing operations can be used to edit the rule "in place". However there are significant limitations on this form of editing, because the dependencies of later constructions on the supported entities of the currently edited rule must be taken into account.

Forward Chaining Control Strategy

An OPS-like forward chaining language might be used as the framework within which to define an EPGV inferencing environment. (BROWNSTON 86) However we have found the classical conflict resolution strategies- LEX and MEA – are problematic. There is no underlying theoretical justification for the use of these strategies, and they produce a substantial complication for debugging, since they introduce a measure of non-deterministic control over rule selection. Our experiences with EPGV rule sets has strongly suggested that these rule sets should be regarded as non-monotonic – that is, the order of firing of the rules does affect the ultimate solution derived. EPGV forward chaining engines therefore store the conflict set and systematically explore every rule firing sequence. This strategy is, however, controlled through the use of a simple, finite state control mechanism based on the identification of named global states. Our use of the conflict set is closely related to the design of Prolog, in which Choice Points are recorded, however this concept is implemented in the context of a forward, rather than a backward inferencing system. (BÖRGER 95) (AIT-KACI 91) (BOIZUMAULT 93)

A second design issue involves handling of failure during the evaluation of the consequent of a rule. This can occur if user input is allowed in the consequent, or if lower level system exceptions occur during consequent evaluation. We also need a means of recovering from a derivation in which an evaluative rule has failed. A backtrack stack must be introduced, which again is used in the same way that this type of stack is used in Prolog. Using this design, an EPGV forward chaining (FC) control strategy, in combination with a properly constructed rule set, is capable of systematically generating all solution variants. Unlike Prolog, no claims are made with respect to logic programming. There is also no limitation with respect to the number of terms that can exist in the consequent set of the rule. Such limitations would produce very significant user interface problems, since the construction of shapes that a designer would regard as a single event would have to be distributed among many rules. This characteristic is one of the reasons that Prolog is not, in our opinion, a practical implementation environment.

Inference Dependency Graph

In recent work, the backtrack stack has been extended to record all dependencies among rules and constructed entities. These fall into the dependent-on and support relations defined above, in the context of feature trees. This augmented backtrack stack is termed an *Inference Dependency Graph* (IDG).

The IDG can be seen to map to the underlying relationships of a Feature Tree that has resulted from a parametric system by considering the following procedure.

Given any feature tree, a corresponding IDG and linked EPGV rules can be generated. For each feature, from earliest to latest do the following: Create an empty rule associated with the current feature. Identify the set of other features that support the current feature. We call this the support set. For each entry in the support set, generate an expression that matches the entry's geometric type, introducing variable names as needed, and add this expression to the antecedent set associated with the current feature. Establish an orientation for the new rule by recording the current spatial origin in the global spatial frame of reference in a marker object, and give the marker a unique label. Add the marker to the antecedent set of the current feature's rule and the consequent set of the preceding feature's rule. Establish a topological location in the Brep data structure from which to start matching by determining the lowest level in the Brep structure from which all support set entities can be reached, and record this pointer in the marker that was entered in the preceding feature consequent set. Identify each entity constructed due to the effect of the evaluation of the current feature, this is the depends-on set. For each entity in the depends-on set, generate an expression (as was done for the support set) and add it to the consequent set of the associated rule. Link the antecedent terms of the current rule to the support set entries from which the term expression was generated, link the consequent terms of the rules to the dependent Brep structures that they generate. When no features remain, the set of rules and the IDG has been formed.

The above procedure generates a set of rules that map one to one to defined features. However, an EPGV rule need not be limited to one feature, but could introduce multiple features. This situation is similar to that of production systems in which multiple rules may be bundled into a single more complex rule.

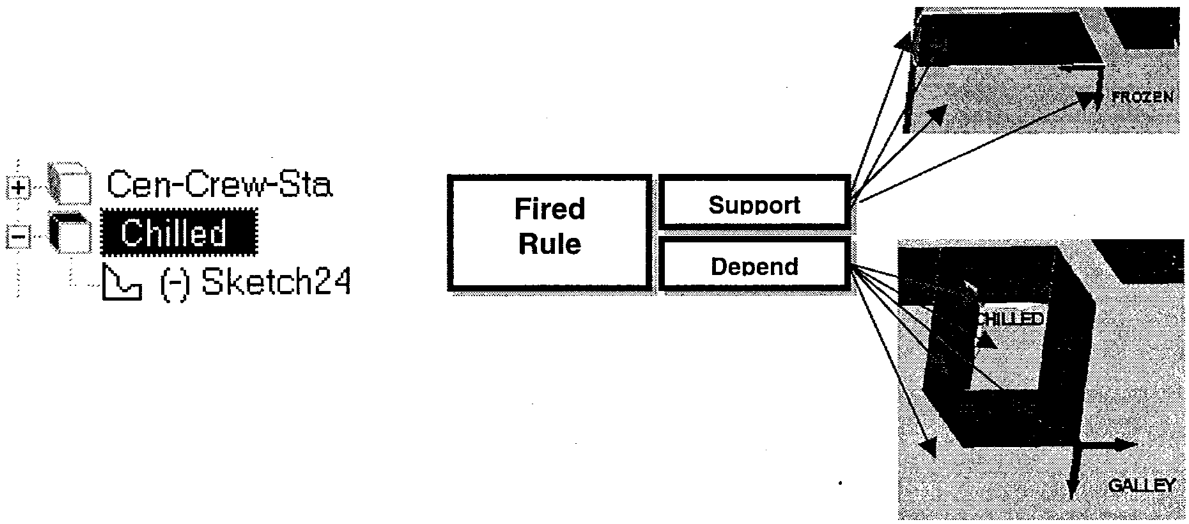


Illustration 2: A parametric feature corresponds to a fired rule in an Inference Dependency Graph.

Derived Design Editing

The Feature to IDG generation procedure just described establishes a relationship between the product of an EPGV derived model and a parametric system Feature-based model. The implications of this relationship are:

- A parametric system Feature Model can be regarded as the product of an EPGV derivation.
- EPGV rules can be regarded as one or more context sensitive parametric system Features.

• An entry in a parametric system Feature Library that checks for conditions of application can be regarded as an EPGV rule.

The IDG can be used as the basis for implementation of in-place editing of instantiated rules, which is the same as editing parametric system Features. Editing requires dependency driven backtracking utilizing the IDG. There are two cases that have been identified, and the following editing strategy has been outlined.

First, editing may cause changes to parameters in such a way that new parameter values may remain within allowable bounds for the entity's support rule. In this case, newly calculated values should be propagated down the supports paths from the entity. If encountered rules – which, by definition must have followed the entity's support rule in firing order - are now given invalid parameter values, those rules must be marked for retraction. If no boundary conditions are invalidated, the topology of the current model is still valid, and the current IDG structure is still valid.

Second, a parameter set may now invalidate the bounds of the support rule, and it must be marked for retraction. Deletion of an entity falls into this category.

For any rule marked by the above cases, mark all entries of its depends-on set for retraction, and for any marked entity, mark its depends-on rules for retraction.

After all marking is done, form a dependency ordering of the entities to be retracted. Retract the entities that have no depends-on entities, and follow their support links back through the order until all entities are retracted. As entity retraction is proceeding, if an entity's support rule is determined to have no depends-on entities, retract the rule. After all retractions are complete, restart forward chaining at the state which was matched by the earliest fired rule.

Rule Editor

Editing issues associated with the development of EPGV rules are complicated and numerous. Currently, editing of an EPGV rule antecedent and consequent are regarded as separate Brep editing events in which variables and entities that have been defined earlier in the session are available and from which parameter values can

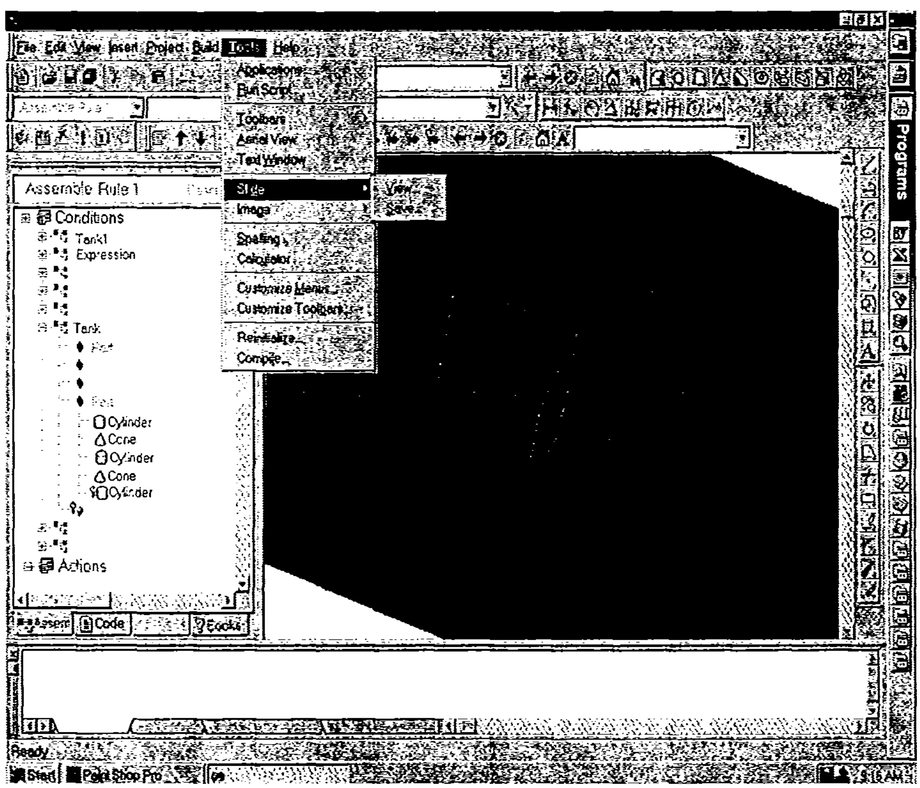


Illustration 3: Rule Editor Concept

be obtained. However there are advantages to editing both rule components in a single context, since this contributes to the user's orientation within the rule. Work on the rule-editing interface is at a very early stage, illustration 3 indicates some of the features which are planned.

Conclusion

The EPGV approach provides an inferencing framework within which graphically represented solid geometry construction rules take on a status equal to that of alphanumeric rules in conventional expert systems. Incorporation of this framework into feature-based modeling systems may provide more open modeling interfaces that allow engineers to design at a more natural level of abstraction.

References

(ACIS 96a) ACIS 2.1 Basic Rendering Husk Development Manual, Spatial Technology, Inc., 1996.

(ACIS 96b) ACIS 2.1 Save File Format Manual, Spatial Technology, Inc., 1996.

(ACIS 96c) ACIS 2.1 Geometric Modeler Application Guide, Spatial Technology, Inc., 1996.

(AIT-KACI 91) Ait-Kaci, H., Warren's Abstract Machine: A Tutorial Reconstruction, Logic Programming. MIT Press, Cambridge, Mass; London, UK, 1991.

(BOIZUMAULT 93) Boizumault, P., The Implementation of Prolog, Princeton university Press, 1993.

(BÖRGER 95) Börger E. and D. Rosenzweig, "The WAM - Definition and Compiler Correctness". in: Logic Programming: Formal Methods and Practical Applications (C.Beierle, L.Plümer, Eds.), North-Holland, Series in Computer Science and Artificial Intelligence, pp. 21-90, 1995.

(BROWNSTON 86) Brownston, L., Farrel, R., Cant, E., Programming Expert Systems in OPS5, An Introduction to Rule-Based Programming. Addison-Wesley, Reading, Mass., 1986.

(CHASE 89) Chase, S.C., "Shapes and shape grammars: from mathematical model to computer implementation," Environment and Planning B , Vol.16, No.2, 1989, pp.215-242.

(D-CUBED 97a) The 2D DCM Technical Overview. D-Cubed Ltd. 1997.

(D-CUBED 97b) The 3D DCM-P3 Technical Overview. D-Cubed Ltd. 1997.

(FOLEY 90) Foley, J., van Dam, A., Feiner, S, Hughes, J. Computer Graphics Principles and Practice, 2nd edition. Addison-Wesley, Reading, Mass., 1990.

(FU 74) Fu, K.S., Syntactic Methods in Pattern Recognition , Academic Press, N.Y., 1974.

(FUDOS 93) Fudos, I., Editable Representations for 2D Geometric Design, Unpublished Master of Science Thesis, Purdue University, Dec. 1993.

(GIPS 80) Gips, J., Stiny, G., "Production systems and grammars: a uniform characterization," in Environment and Planning B , 1980, pp.399-408.

(HEISSERMAN 91) Heisserman, J., Generative Geometric Design and Boundary Solid Grammars. Unpublished Ph.D.Arch. Thesis, Carnegie Mellon University, May 1991.

(HOFFMAN 89) Hoffman, C., Geometric and Solid Modeling. Morgan Kaufmann, San Mateo, 1989.

(HOFFMAN 95) Hoffman, C., "Geometric Constraint Solving: Algebraic Issues" in ACA Proceedings, 1995.

(JUAN-ARINYO 95a) Juan-Arinyo, R., Soto, A., "A Rule-Constructive Geometric Constraint Solver" in Report LSI-95-25-R, May 1995.

(JUAN-ARINYO 95b) Juan-Arinyo, R., "Triangles, Ruler, and Compass" in Report LSI-95-6-R, 1995.
(LANGE 95) Lange, W.A., A Production System Development Environment for Building Design based on Extended Pattern Grammars, Unpublished Master's Thesis, George Washington University, 1995. Thesis directed by Dr. Oscar Garcia.

(LANGE 96) Lange, W.A., Choung, H.S., Submarine Compartment Arrangement Using Extended Pattern Grammars with Variables, Phase I Final Report. Task SBIR N96-100, Database Driven 3D Compartment Arrangements. Phase I funding under Contract Number N00024-96-C-4180.

(LANGE 97) Lange, W.A., Submarine Compartment Arrangement Using Extended Pattern Grammars with Variables, Phase I Option Final Report. Task SBIR N96-100, Database Driven 3D Compartment Arrangements. Funding under Contract Number N00024-96-C-4180.

(MCLELLAN 96) McLellan, T., Karam, F., Pro/ENGINEER ® Tips and Techniques, Onward Press, Santa Fe, NM, 1996.

(MORTENSON 85) Mortenson, M., Geometric Modeling, John Wiley and Sons, 1985.

(PIERRA 96) Pierra, G., Ait-Ameur,Y., Besnard, F., Girard, Pl, Potier, J., "A general framework for parametric product models within STEP and Parts Library" in Proc. European Conf. On Product Data Technology, London, 18-19 April 1996, 1996.

(SOLIDWORKS 97) SolidWorks 97 Plus Evaluation software and Documentation. 1997.

(STINY 80) Stiny, G., "Introduction to shape and shape grammars," Environment and Planning B , Vol.7, 1980, pp.343-351.

(STINY 81a) Stiny, G., "A note on the description of designs," Environment and Planning B, Vol.8, 1981, pp.257-267.

(STINY 81b) Stiny, G., March, L., "Design machines," Environment and Planning B, Vol.8, 1981, pp.245-255.

(STINY 90) Stiny, G., "What is a design?," Environment and Planning B, Vol.17, No.1, 1990, pp.97-103.