Inheritance Semantics for Features

H.E. Otto and F. Kimura

*Dept. of Precision Machinery Eng., The University of Tokyo
Bunkyo-ku, Hongo 7-3-1, Tokyo 113, JAPAN
tel. +81-3-3812-2111 - fax. +81-3-3812-8849
e-mail:[otto, kimura]@cim.pe.u-tokyo.ac.jp*

F. Mandorli

*Dept. of Mechanical Eng., The University of Ancona
Via Breccie Bianche, Ancona I-60131, ITALY
tel. +39-71-2204-796 - fax. +39-71-2204-801
e-mail:ferro@ied.unipr.it*

Abstract

Within recent research trends and an increased interest in targeting not only product modeling but product life cycle modeling as a whole, feature technology finds itself in a new, challenging position. Although, it is a powerful tool that at certain representation levels, can be successfully used to capture the relationship of shape and functionality in many ways, to properly utilise functionality of feature technology on a broad base, framework and entity concepts developed up to now, need to be extended. In most systems, design and implementation of a feature-based framework are based on an existing specification environment, which provides only a quite simple type-instance inheritance. However, new requirements demanding structure inheritance and value propagation between varying design configurations and different versions of feature sets used, cannot be approached with those rather traditional inheritance mechanisms. Therefore we propose an extension of feature frameworks in form of inheritance semantics that facilitates a flexible information inheritance path based on a strictly controlled, limited entity-to-entity inheritance.

Keywords

Feature-based applications, engineering design, structure inheritance, value propagation

1 INTRODUCTION

Feature technology as it evolved in the past 15 years grew from its origin to identify part regions of interest for manufacturing of products to a powerful technology with many frameworks and system architectures developed as well as implemented. Within recent research trends and an increased interest in targeting not only product modeling but product life cycle modeling as a whole, for feature technology used to support and enhance mainly computer-aided design and computer-aided manufacturing, new dimensions of requirements arise. Further demands on higher automation, more flexibility, enhanced man-machine interaction and increased built-in knowledge of application systems, require a thorough extension and perhaps even local re-design of feature frameworks developed. Feature technology is a powerful tool that at certain representation levels, can be successfully used to capture the relationship of shape and functionality in many ways. However, to properly utilize functionality of feature technology on a broad base, framework and entity concepts developed up to now, need to be extended.

Within the scope of the work described in this paper, the focus will be on inheritance semantics for form features. Feature classes other than form features might require modified or even different inheritance semantics. In most systems, design and implementation of a feature-based framework are based on an existing specification environment. These environments do provide only a quite simple type-instance inheritance. However, applications where features in a model become modified in a way that new feature classes or further versions of modified features are required, need an extended inheritance semantics that is capable to maintain present information such as equivalent representations, ancestor-descendant relationships and context-element formations.

2 FEATURE TECHNOLOGY AND INHERITANCE

2.1 Standard approaches

Initial work on features originated during the late 1970's and early 1980's within problems of identifying regions of part geometry important to manufacturing. After half a decade of untiring efforts, when feature technology left its initial embryonic state, integration of feature models and geometric models was targeted with the basic goal to integrate different processes such as manufacturing and design. Note feature-based frameworks developed earlier were not able to keep different representations of features consistent. Especially insufficient detection of spatial interference between feature representations in the geometrical model and a lack of appropriate feedback to as well as reaction in the feature model caused consistency problems that even today are plaguing several commercial CAD systems pretending to master flawless feature-based design.

Besides integration and consistency problems partially mastered in frameworks developed in the late 1980's and early 1990's, successful modeling of relationships

and control of feature properties across features as well as across different feature classes employing different techniques such as constraint propagation, associated self-evaluation rules or ATM reasoning, represent a further important milestone in the historical development of feature technology (cf. Shah, 1991). Recent use of object-oriented methods improved the quality of implementations of feature classification schemes while concepts such as class generalization and class refinement helped to reduce inconsistencies and type anomalies in feature class hierarchies, though sometimes also due to misuse the opposite was achieved (Firesmith, 1995). Despite all these achievements, feature relationships across time are not attained yet (cf. discussions in Wozny et al., 1994 and Otto et al., 1998). Strictly speaking, the concept of version and its context a configuration, which will be discussed later in more detail, were and still are unknown to almost all traditional feature-based frameworks regarding both instance-instance information forwarding and modeling of different context-element formations.

2.2 New perspectives

Extended research goals

With the recent introduction of life cycle modeling, an all embracing approach to support sustainable product development in providing a framework for modeling data, processes, activities and their relationships incurred within all stages of a product's life, new dimensions and requirements for research and practice inevitably arise. Within such an extension of research goals, traditional approaches showing limited views for application fields treated on an isolated base, are no longer feasible. Especially for activities in early product life cycle stages such as product planning and product design, approaches such as *over the wall design* (Boothroyd et al., 1992) and specification of unrelated, singular design object descriptions dedicated to a few perspectives, i.e., satisfying only a very restricted set of constraints without any alternatives, are becoming obsolete.

Restructuring of priorities and requirements

To accommodate to extended research goals as outlined above, it is inevitable to not only extend and bring into line but also restructure existing priorities and requirements. For CAD systems in general and feature-based applications in particular, this means changing from documentation of "as it turns out finally to be", i.e., nominal object specifications to documentation of design object evolution and variations of possible design solutions, capable to satisfy a larger, more complex set of product requirements (Mantyla, 1990, Hubka et al., 1996). Within the latter, aspects of representation across configuration and time pervade again.

Shifting our view a little bit from the big picture of life cycle modeling and its focus on economizing all processes while trying to save all kinds of resources and recycling as much as possible, similar goals, thought with different environments and structures in mind, can be set for (digital) object life cycles in system and software development. Here, reuse of portions of designs and their (digital) descriptions and documents is becoming a rather important issue too, though disposal is no subject to worry about now.

Although, some time ago developed mainly for reasons of enhancing functionality of programming environments, inheritance and its ramifications, model inheritance, software inheritance and variation inheritance (Meyer, 1996) to only name a few, now appear with a different priority and dimension in a much broader field of application.

Scope and objectives

These new requirements are clearly beyond a possible rapprochement employing standard type-instance inheritance only. To solve problems pointed out earlier on the way to further exploit feature technology within future product modeling systems we target, in a first approach, an extension of feature frameworks in form of inheritance semantics that can be based on different inheritance paths. In other words, an inheritance semantics that features a flexible information inheritance path based on a controlled entity-to-entity inheritance.

3 INHERITANCE ADAPTED TO NEW REQUIREMENTS

To outline some common, dedicated knowledge background, used as a context, the dimension of inheritance and variations of features within design object descriptions are briefly discussed. Based on this context, a notion of inheritance path is worked out that is later employed to develop an abstraction as well as formally described concept of path controlled inheritance semantics for features. To remain independent of any possible implementation or system platform used, abstraction and scheme are based on the neutral concept of typed attributed entity structures that can be realized with different concrete data structures and software environments. For example these could be variant records within procedural programming languages, attributed symbols within rule-based rewriting systems or entity-record structures within object-oriented database systems.

3.1 Outline of the approach

Variations of features

In traditional feature frameworks, as discussed above, inheritance is usually limited to type-instance inheritance, making it very difficult, if not impossible, to model complex hierarchical structures of different designs across time. To completely document several configurations, i.e., different sets of features used within design object descriptions, support in form of parameter inheritance, i.e., value propagation between implemented objects is required in addition to type-instance inheritance. The same counts for modeling different versions of one feature. Due to the usually non-singular representation of form features within different integrated models, parameter inheritance across different representations of a feature is required as well. With inheritance across all three basic variation classes of features possible within design object descriptions, that is configuration, version and representation as depicted in Figure 1, one should be able to better support the envisioned goal of utilizing features within future intelligent CAD systems,

integrated in a holistic framework such as life cycle modeling.

Figure 1 Variation classes of features within design object descriptions

A notion of inheritance paths

To describe a multi-dimensional inheritance space over configurations, representations and versions of features, first an adequate scheme is required, to facilitate abstraction as well as description of a course or a route data structure inheritance and value / parameter propagation can be based upon. In a further step, this scheme needs to be exactly formalized and combined with an exact mathematical representation framework to unambiguously document all relevant aspects. To approach step by step this problem, let us first determine all possible relationships between data architecture objects, i.e., defined types and actual objects, that is computed instances of those types as well as relationships between these instances themselves.

Basically, as already outlined above, there are type-instance and instance-instance relationships. The former, in general being used for creating actual instances of specified types as well as supporting reuse and probably refinement of already existing type specifications. The latter, being used for propagation of values and parameters. Although, modeling of type-instance relationships might involve complex data structures, the actual procedure is mostly, from the viewpoint of inheritance, a straightforward one-to-one mapping. However, different is the case of instance-instance inheritance. From a theoretical point of view, there is no limitation of routes for propagated values between objects. In other words, propagation of values and parameters is only limited by the number of type properties and specified relationships between these types themselves. For example, an instance implementing a cylindrical blind hole feature with a property *hole_diameter* should not be able to inherit the value of a property *flange_length* from an instance implementing an elongated flange. For feature-based applications

within product design we propose to sub divide the instance-instance relationship according to previously discussed variations of features, meaningful to engineering design, into three further relationships, that is representation-equivalence, context-element and ancestor-descendant. Propagation of values along equivalent representations of a feature supports creation of consistent configurations consisting of sets of different versions of features. Inheritance of particular properties between an ancestor and its descendant directly supports design of different versions of one feature while propagation of properties from a context to its component can be effectively used to support configuration design (cf. discussions on system requirements in Rosen et al., 1996, Shen et al., 1996).

Integrating all above discussed type-instance and instance-instance relationships and using them as a base to form an abstraction of a set of all allowed inheritance routes within a defined data model, one result can be visualized as a 5 dimensional structure inheritance / value propagation space as shown in Figure 2.

Figure 2 Multi-dimensional inheritance / propagation space

Actual description of an inheritance / propagation path, i.e., routing of forwarded information can then be abstracted by defining for each inherited unit- may this be a type property or a propagated value - a point set consisting of sets of 5 tuples, which traces a path within relevant considered feature variations as introduced. On the other hand, due to the dualism of this scheme, an abstraction of such an inheritance path can be used to derive a valid route, inherited / propagated information within an implemented system is supposed to follow.

3.2 Theoretical base

Order relations

Domains are supposed to be sets of partially ordered elements. The order relation is the degree of informational contents (definedness) an element consists of. Formally, a tuple (D, Ξ) with a set D and with a binary relation Ξ , being reflexive, anti-symmetric and transitive, is called a *partially ordered set* (poset). A subset $S \subseteq$

D is *directed*, if all (finite) sets $U \subseteq S$ have a least upper bound $x \in S$. If there is a least element $\perp_D \in D$ and all directed subsets $S \subseteq D$ have a least upper bound $\sqcup S$, the poset D is a *complete partial ordered set* (cpo). An element x with $x \sqsubseteq \sqcup S$ of a cpo D is *compact*, if there is a point $z \in S$ with $x \sqsubseteq z$ for a directed subset S of D .

Entity structures

Typed attributed entity structures can be conceptualized in different ways for example employing attribute value clusters (AVC) or labeled directed graphs (Otto et al., 1994). For the representation of entity structures and their properties an AVC is a better choice. However, for the representation and discussion of structure traversing and attribute / value propagation we prefer to use a labeled directed graph.

Definition: Given a complete partial ordered set $(T, \sqsubseteq)$ of types T and a subsumption relation $\sqsubseteq$, *typed attributed entity structures* can be defined as a labeled rooted graph G over types T and attributes A as a tuple $TAE = (N, n_r, \gamma_d, \gamma_a)$ with a finite set of nodes N starting at a root node $n_r \in N$, a total node typing function $\gamma_d: N \rightarrow T$ and a partial attribute value function $\gamma_a: N \times A \rightarrow N$.

Based on above given definition, a type $t_l \in T$ labeling a node $n_l \in N$ can be retrieved by applying the node typing function $\gamma_d(n_l) = t_l$ that is defined for each type. An attribute $a_l \in A$ labeling an arc located between two nodes n_l and n_2 with $\gamma_d(n_l) = t_l$ and $\gamma_d(n_2) = t_2$ can be used to access the successor node n_2 from n_l and retrieve an attribute value by applying both a partial attribute value function and a total node typing function as shown in equation (1).

$$\gamma_d(\gamma_a(n_l, a_l)) = t_2. \quad (1)$$

Structural extensions

To explicitly identify an entity of interest, being either an element of a data architecture specification abstracted as a typed attributed entity structure ($tae \in TAE$) or an object implemented as an instance and abstracted as a typed attributed entity ($tae' \in TAE'$), one needs to replenish the concept of TAE . Hence, we introduce a total node identification function $\gamma_n: N \rightarrow (\mathbb{N} \times I)$, mapping a node $n \in N$ to either, a natural number identifying a type element within a typed attributed entity structure or an entity identifier ($id \in I$) assigned to an instance.

3.3 Concept and abstraction

Formal definition of paths

To formally describe a route possible inherited attributes or propagated values

might take within a given structure, a path within a set of either entity structures documenting a data architecture or actual entities of an object description based on the extended abstraction *TAE* can be used as a natural, straight forward description method being compact as well as elegant in both conceptualization and notation. Considering previously introduced concept of typed attributed entity structures from the viewpoint of a labeled rooted graph, a path can be represented as a pair consisting of a node, which indicates the beginning of a path and a sequence of labeled arcs (attributes) tracing the course of a path.

Definition: A path $P = (N, A^*)$ within a typed attributed entity structure is defined as a tuple of a node N marking the origin of a path and a finite sequence of attributes $A^* = [a_1, a_2, a_3, \dots, a_n]$ with $a_i \in A$ and $[a_i] \circ [a_2] = [a_1, a_2]$ such that

$$\begin{aligned} \gamma_a(n_p, a_k) &= \gamma_a(p_k) \\ \gamma_a(\gamma_a(n_p, a_k), a_{k+1}) &= \gamma_a(p_k \circ a_{k+1}). \end{aligned} \quad (2)$$

with an application of attributes within a given path associating to the left. The empty path denoted p_ϵ is defined as shown in equation (3).

$$\gamma_a(n_p, a_\epsilon) = \gamma_a(p_\epsilon) = n_i \quad \text{with } a_\epsilon \in A \text{ and } p_\epsilon \in P. \quad (3)$$

Note the resulting sub structures generated by traversing all possible non-empty paths while collecting all encountered attributes and types should form the complete architecture of a given typed attributed entity structure. To enforce such a relationship between valid paths and entity structures as introduced, we consider only rooted typed attributed entity structures where every node is reachable from the root n_r , that is $N = \{ \gamma_a(p_i) \mid p_i \in P \wedge \downarrow_1 p_i = n_r \wedge p_i \neq p_\epsilon \}$.

Inheritance path control

After formal definition of inheritance paths we need a method, to automate as well control their computation. Considering their internal architecture and their relevance to both type properties and attribute values within inheritance, a natural way is to classify attributes in respect to the inheritance space as shown in Figure 2. Assuming an attribute domain *AT* with domain values as indicated within the set of domain equations (4), one can specify for each attribute a function, inheritance path control (*ipc*) say, that computes an attribute controlled inheritance path for each target entity being abstracted as a node within either *TAE* or *TAE'*. Strictly speaking there is an *ipc* function for each relevant domain (cf. equation (4) and D_2), implementing *ipc* as a collection of recursive functions over classified attributes as shown in equation (5).

$$\begin{aligned} D_1 &= N \\ D_2 &= ((TAE \times TAE') + TAE + TAE') \end{aligned} \quad (4)$$

$$\begin{aligned}
D_3 &= \{P\} \\
AT &= \{GENERIC, INSTANCE, CONTEXT, ANCESTOR, EQUIVALENT\} \\
ipc &\in (D_1 \times A \times D_2) \rightarrow D_3 \\
generic &\in (D_1 \times D_2) \rightarrow D_3 \\
instance &\in (D_1 \times D_2) \rightarrow D_3 \\
context &\in (D_1 \times D_2) \rightarrow D_3 \\
ancestor &\in (D_1 \times D_2) \rightarrow D_3 \\
equivalent &\in (D_1 \times D_2) \rightarrow D_3 \\
attribute\_type &\in A \rightarrow AT \\
next\_attribute &\in (D_1 \times A) \rightarrow A .
\end{aligned}$$

A generic *ipc* function consists basically of a recursion over classified attributes assigning to each attribute class an appropriate recursive function over a sub set of types and attributes. These functions in turn compute a set of inheritance paths in respect to a particular type-instance or instance-instance relationship peculiar to an attribute class where they are designated to.

$$\begin{aligned}
ipc &= \lambda n, a, tae. \\
&(a \neq a_e) \rightarrow \\
&((attribute\_type(a) = GENERIC) \rightarrow \\
&\quad generic(n, tae) \cup ipc(n, next\_attribute(a), tae), \\
&(attribute\_type(a) = INSTANCE) \rightarrow \\
&\quad instance(n, tae) \cup ipc(n, next\_attribute(a), tae), \\
&(attribute\_type(a) = CONTEXT) \rightarrow \\
&\quad context(n, tae) \cup ipc(n, next\_attribute(a), tae), \\
&(attribute\_type(a) = ANCESTOR) \rightarrow \\
&\quad ancestor(n, tae) \cup ipc(n, next\_attribute(a), tae), \\
&(attribute\_type(a) = EQUIVALENT) \rightarrow \\
&\quad equivalent(n, tae) \cup ipc(n, next\_attribute(a), tae), \dots) \dots) .
\end{aligned} \tag{5}$$

Actual structure inheritance, i.e., forwarding of type specifications and type properties to a newly created instance or value propagation between instances can then be realized by applying total node typing and total node identification functions to typed attributed entity structures as well as typed attributed entities utilizing an inheritance path computed by *ipc* functions.

4 EXAMPLE

4.1 Environment

Theoretical work and specifications to analyze and document our approach employ methods and tools of attributed entity structures, rewriting rules and denotational

semantics. Experimental work to determine and evaluate applicability and limits of our framework extension are carried out within an implemented prototype testbed employing typed attributed entity structures and a conditional attributed rewriting system. Current system and hardware platforms for practical work are, besides an integrated commercial geometrical modeling kernel (ACIS, 1996/97) that supports attributed entities, windows NT operated Gateway 2000 / G6-333 personal computer workstations and UNIX operated Sun UltraSparc graphics workstations.

4.2 Modeling example within engineering design

To illustrate the way of how path controlled inheritance as introduced works in practice, let us discuss a section of a small example within mechanical engineering design. The test part investigated and modeled is a laser holder, used to fix and carry various components ranging from opto-electronic (laser) units, tiny mirrors and lenses to an electromagnetic unit for motion control (cf. Figure 3). This part, forming the core fixture base of a laser reader head, can be found in similar shapes in many of today's CD-ROM reader devices. Subject to demonstration is structure inheritance and value propagation between features during the creation of an alternative design (cf. Figure 4) allowing for the mounting and fixture of a different motion control unit.

To accommodate the alternative motion control unit, the size of the oblong hole in both dimensions length and width needs to be changed. Further needs the number (and diameter) of previously designed fixing holes to be reduced to three fixing holes arranged as shown in Figure 4. However, these changes affect dimensions, shape and spatial location of several other adjacent features, e.g., the rectangular and semi-rectangular pockets in the center, the large through hole in the lower left corner, the elongated housing of the circular motion guide on the far right side and of course the entire base flange itself. From a feature-based point of view using path controlled structure inheritance and value propagation an alternative design object description can be economically as well fast created as following. Employing parameter propagation along the ancestor-descendant and context-element path a new design configuration containing a new feature set with new versions of features for fixing holes, through holes, slots, pockets, flanges, etc., can be created. In the case of the fixing holes, only a few feature properties such as version number, spatial location and diameter need to be changed locally. All other attribute values representing feature properties such as parent features, bound-to relationships (in the case of depressions which cannot exist outside of protrusions), constraints on circular entrance faces and a cylindrical lateral face, etc., remain (version related) identical to those of their ancestors, i.e., selected fixing holes of the previous version. In the case of the big circular through hole located in the lower left corner, only version number and spatial location need to be updated. New versions of previously mentioned pocket seven do not need to be relocated, though their dimensions and geometrical representation require local assignment of new values. The same goes for all the other features not explicitly mentioned again. In general, if form features are considered, a value change in any property related to dimensions, spatial location or spatial orientation always

requires an update in their geometrical representation.

Figure 3 Geometry of the test part

Figure 4 Geometry of the alternative design object

Due to possible side effects linked to these local changes, sometimes resulting in undesirable spatial interference between shape representations of features, additional control mechanisms such as reflection (cf. Mandorli et al, 1996) need to be implemented to avoid inconsistent models.

5 CONCLUSIONS

Path controlled structure inheritance / value propagation in respect to a multi-dimensional inheritance space based on feature variations and type-instance as well as instance-instance relationships has been introduced and discussed. Within experimental work theoretical and practical results obtained so far appear to be quite promising in respect to originally set objectives. However, for the development of compact, efficient and portable implementations of the framework developed, further aspects such as optimal storage of related information, efficient buffer management and paging, intelligent entity clustering, fast and unambiguous object access across configurations as well as versions and perhaps parallel propagation of unrelated values, critical to system performance and data integrity need to be addressed in future work.

6 REFERENCES

- ACIS 3D Toolkit (1996) Extension Reference, Spatial Technology, Inc., Boulder, Colorado.
- ACIS Geometric Modeler (1997) Programmers Reference, Spatial Technology, Inc., Boulder, Colorado.
- Boothroyd, G. and Altling, L. (1992) Design for Assembly and Disassembly, *Annals of the CIRP*, **41**, 2, 625-636.
- Firesmith, D. (1995) Inheritance guidelines, *Journal of Object-Oriented Programming*, **8**, 2, 67 - 72.
- Hubka, V. and Eder, W.E. (1996) Design Science: Introduction to Needs, Scope and Organization of Engineering Design Knowledge, Springer-Verlag, London.
- Mandorli, F., Cugini, U., Otto, H.E. and Kimura, F. (1996) Reflective control of attributed entities in feature-based CAD systems using a CARW system manager, in *Knowledge Intensive CAD* (eds. T. Tomiyama, M. Mantyla and S. Finger), Chapman and Hall, London, 151-170.
- Mantyla, M. (1990) A modeling system for top-down design of assembled products, *IBM Research Journal of Development*, **34**, 5, 636 - 659.
- Meyer, B. (1996) The many faces of inheritance: A taxonomy of taxonomy, *Computer*, **29**, 5, 105 - 107.
- Otto, H.E., Mandorli, F., Cugini, U. and Kimura, F. (1994) Domain-Oriented Semantics For Feature Modeling Based On TAE Structures Using Conditional Attributed Rewriting Systems, in *Proceedings of ASME International Computers in Engineering Conference*, Minneapolis, 13 - 27.
- Otto, H.E. and Kimura, F. (1998) Towards a framework for unmitigated integration of feature technology within life cycle modeling, in *Proceedings of JSPE 1998 Annual Conference*, JSPE, Kawasaki, Kanagawa.
- Rosen, D.W., Bras, B., Hassenzahl, S.L., Newcomb, P.J. and Yu, T. (1996) Towards computer-aided configuration design for the life cycle, *Journal of Intelligent Manufacturing*, **7**, 145-160.
- Shah J.J. (1991) Assessment of features technology, *Computer-Aided Design*, **23**, 5, 331 - 343.
- Shen, W. and Barth, J.-P. (1996) An object-oriented approach for engineering design product modelling, in *Knowledge Intensive CAD* (eds. T. Tomiyama, M. Mantyla and S. Finger), Chapman and Hall, London, 171-187.
- Wozny, M., Pratt, M. and Poli, C. (1994) Topics in feature-based design and manufacturing, in *Advances in feature based manufacturing* (eds. J. Shah, M. Mantyla and D. Nau). Elsevier Science Publishers B.V., Amsterdam, 481-510.