

Modeling Design Versions

R Ramakrishnan[†]

D Janaki Ram[‡]

Distributed and Object Systems Group
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
India

[†]rrama@bronto.iitm.ernet.in

[‡]djram@iitm.ernet.in

Abstract

Due to the evolutionary nature of the design process, engineering design databases need to provide adequate support for versioning of designs corresponding to their different stages of evolution. Existing mechanisms of class and object versioning in Object-Oriented Databases fail to capture the design process as a concept in its entirety. For instance, a new class or object version does not always imply that it is a new design. Further, semantic information regarding each version of a component needs to be captured. The versioning mechanism must also capture the interdependencies between the designs of various components. In this paper, we propose a model to capture versioning of designs in Object-Oriented Databases based on the concepts of Design Attributes, Design Equivalences and Design Versions. The proposed model achieves Design Versioning through Semantic Classes and Version Reference Classes.

Keywords Design Versions, Engineering Design Applications, Object-Oriented Databases

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 22nd VLDB Conference
Mumbai(Bombay), India, 1996

1 Introduction

Engineering design applications involve the management of complex and evolutionary designs of the numerous components in engineering artifacts. Examples of such artifacts include aircrafts, ships and cars. Engineering Databases are hence needed to aid in the design process of such artifacts. Object oriented databases, with their rich modeling power through concepts of classification, inheritance, generalization and aggregation are well suited for such applications[Spo86][Sri89].

A complex artifact consists of many components. Each component may, in turn, consist of lower level components. This consist-of(part-of) relationship is recursive till the leaf level primitive component. Each component evolves continuously during the design process. Due to this evolutionary nature of the design process, it is required to maintain different versions of the component corresponding to its different stages of evolution. As the design process is typically a multipath one, wherein there exists several correct designs of a component, the designer needs to keep track of these versions of the component. Further, each version of the component may use different versions of lower level components. Hence, adequate support for versioning is required for design applications[Cho88][Kat90].

Existing concepts define class versioning and object versioning mechanisms[Kim89b][Kim90]. However, these are inadequate for design applications as they do not provide support to capture the design semantics. A class or object version need not necessarily mean that it is a new design. Further, the designer would like to know more information about the version, like the necessity that led to its creation and the constraints that govern the use of that version. In this paper, we propose a new versioning mechanism based on Design Attributes, Design Versions and Design Equivalences, which overcomes these defects. Semantic Classes and

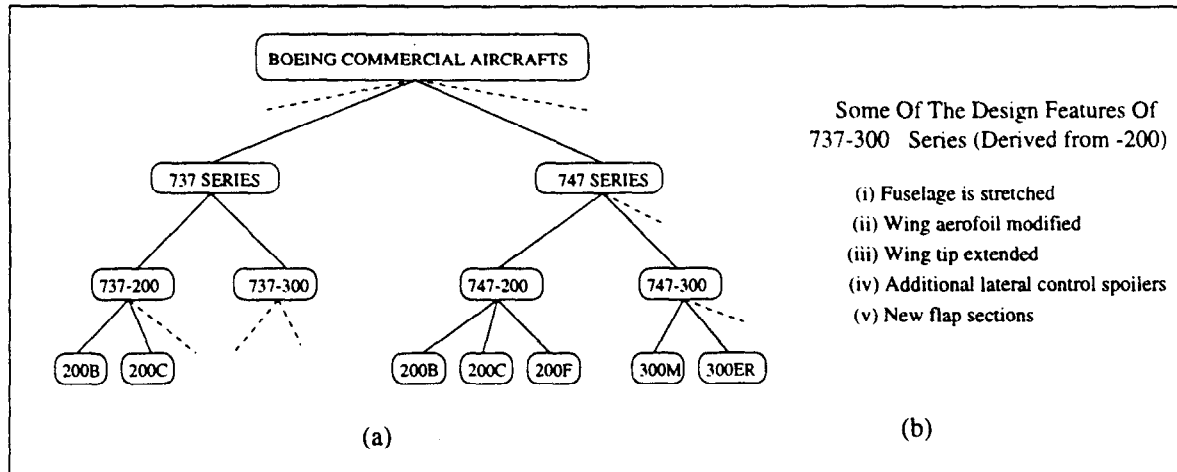


Figure 1: Product Versions

Version Reference Classes form the building blocks of this model.

The rest of the paper is organized as follows: in section 2, the versioning requirements for engineering designs are discussed. Section 3 discusses the inadequacies of existing versioning mechanisms. Section 4 illustrates the concepts of Design Attributes, Design Versions and Design Equivalences. Section 5 explains Semantic Class and Version Reference Classes. Section 6 discusses the issues of inheritance, change propagation and configurations. Section 7 discusses the Design Transaction Model and Section 8 has the Conclusions.

2 Versioning Requirements For Engineering Designs

Engineering design applications involve the management of complex and evolutionary designs of engineering artifacts such as aircrafts, ships and cars. These artifacts are characterized by the numerous components they constitute. These components are to be efficiently stored and managed by the system. The design task is characterized by a set of rules of thumb, constraints and techniques. These are used by the designer to design the various components of the artifact. The designer often uses the knowledge of earlier designs to solve the problem, referred to as design reuse. Routine designing often involves modification of existing designs to suit the additional requirements and constraints. Due to this nature of the design process, the system should provide adequate support for managing the different versions of a component corresponding to its different stages of creation and evolution.

2.1 Existence of Product Versions

Typically, in the industry, the design department is involved in creating an improved design of its product(in

this paper, we use the term "product" to mean the final product the industry manufactures. For example, in a car industry, the term Product refers to the car. For producing the car, the industry would also be producing the sub-components: chassis, engine etc. These sub-components are not referred to as Product). The improvement may be, in terms of better performance or additional facilities, or to suit different requirements. Hence, for any industry, there will be several versions of its product. For instance, in the Boeing Commercial Aircraft industry, there exists several versions of Boeing Commercial Aircrafts: the 737 series, the 747 series, the 757 series etc.(fig. 1a) In each series, there exists different versions to suit customer requirements. For instance, Boeing 747-200F is the freighter version and 747-200C is the convertible passenger-cargo version. Each series is derived from an existing series with suitable modifications incorporating the latest advances in technology. Fig. 1b illustrates some of the design features of B737-300, compared to 737-200 series[Jan95]. It is seen that the designer has taken the version of the wing used in 737-200 and modified it to suit the new requirements: he has modified the aerofoil section and extended the wing tip. Further, different versions of the aircraft may use same version of a component. The system must efficiently manage these versions. It must support queries such as "which version of the component is used in a product version".

2.2 Versioning in Design Process

A complex artifact consists of many components. For instance, an aircraft consists of fuselage, wings, powerplant, landing gear, control surface etc. These components may in turn consist of lower level components. For example, fuselage consists of nosecone, frame etc., while wings are composed of aerofoil sections, ribs,

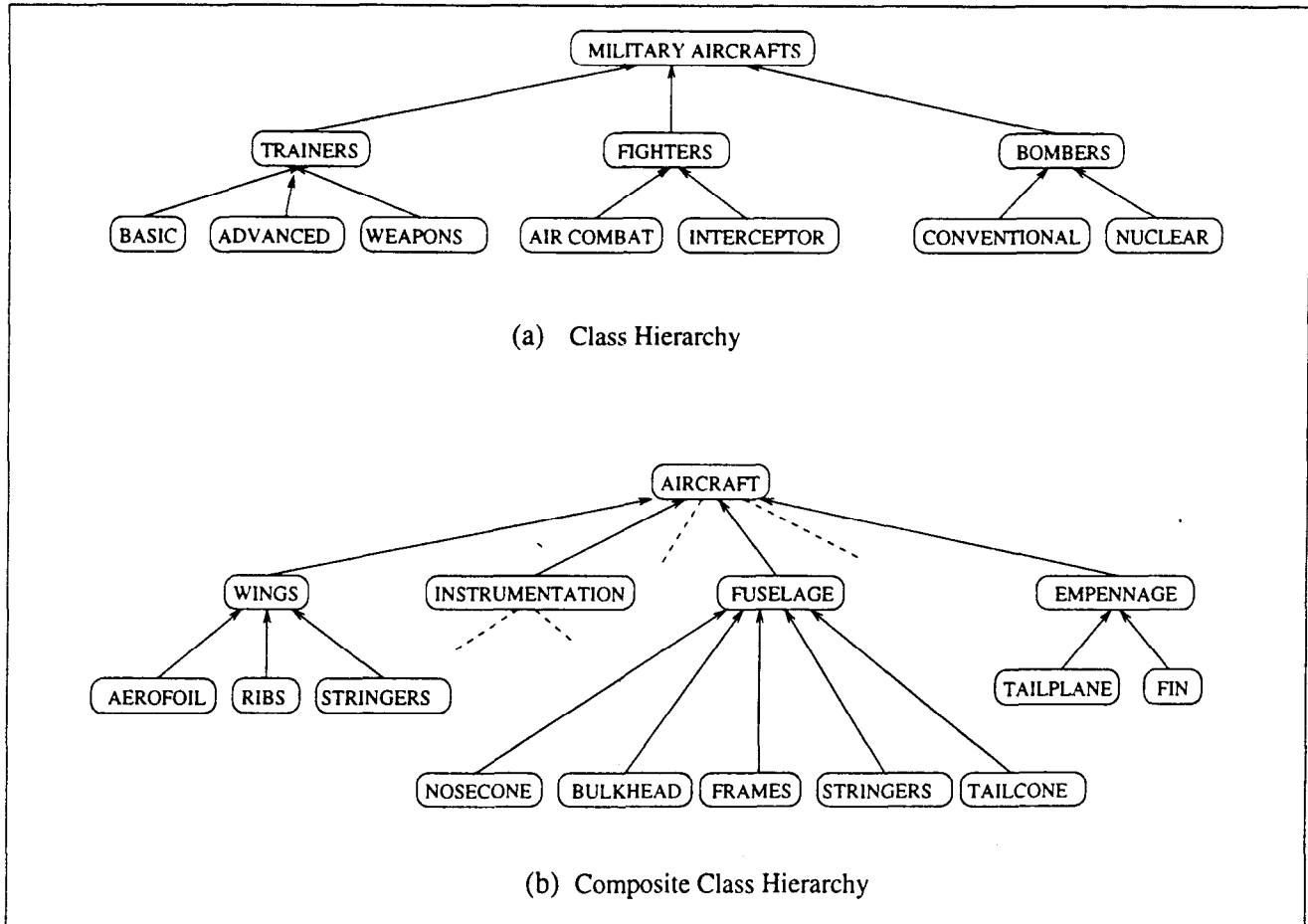


Figure 2: Organization Of Classes

stringers etc.[Kes93] Each component is normally designed by a design engineer. The designer may use an existing version of the component retrieved from the database if he finds that it satisfies his requirements. Else, he may modify an existing version to suit the requirements. There may be several versions of the component which would satisfy the requirements. Hence, the designer may need to experiment with multiple versions of the design. When the designer retrieves a version, he must be able to understand the following: the necessity that led to its creation, its characteristics, the constraints that governs its use, the differences from other versions and the lower level component versions used. This knowledge must be captured as a part of the versioning mechanism to give full support to the design process and also a meaning to the different versions of the component.

2.3 Interdependencies

Interdependencies between the versions of different components exists in a product. For example, changing the version of wing may require that the tailplane

be modified or replaced by another version. This interdependency between the designs of the wing and the tailplane needs to be captured adequately by the versioning mechanism to ensure correct design.

2.4 Support for Maintenance Department

When the designer retrieves a component version, he may also be informed of the manufacture and maintenance aspects of the component, so as to aid him in the design process. Capturing these aspects of the component would also help the maintenance department. For instance, when a component arrives for maintenance, the engineer must be able to identify the version of the lower level components used in it. This is so because different versions of the component may use different versions of lower level components. For instance, a passenger aircraft version and a freighter aircraft version may use the same version of the wing, but different versions of the fuselage. After identifying the versions, the engineer needs to be informed of the maintenance aspects of that version, thereby aiding him in the process.

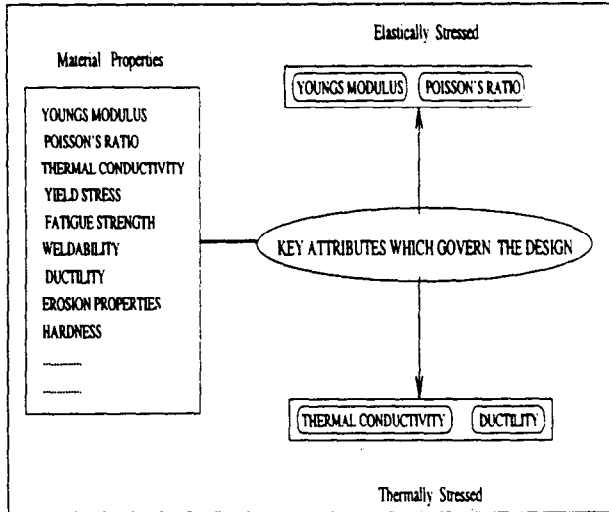


Figure 3: Key Attributes

3 Versioning In Object Oriented Databases

Object-oriented databases are well suited for engineering design applications. The core object model consists of classes, objects, messages and inheritance. Real world entities are modeled as classes. The state of the entity is described using the attributes, and its behavior using the methods in the class definition. Schema refers to the hierarchical organization of all the classes in the database and their interrelationships. The class hierarchy represents the generalization relationship among classes in terms of superclass (fig. 2a). The composite class hierarchy represents the aggregation relationship between classes (fig. 2b). The complex artifact can be represented as a set of components using the part-of relationship [Kim89a][Kim90].

3.1 Class and Object Versioning

Due to the evolutionary nature of the design process, changes to the schema are more as a rule than an exception. As the design evolves, the schema evolves by changes to the class definition, changes to the class hierarchy and changes to the composite class hierarchy [Ban87][Ngu89]. As explained in the earlier section, versioning is an important requirement for design applications. Schema versioning refers to the creation of a new version based on changes to the hierarchical organization of the classes [Kim88]. Class versioning refers to the creation of a new version of an existing class definition [Mon93]. An object is stated as versioned when it changes its state, i.e. the attribute values are changed. The version derivation hierarchy can be expressed in a version hierarchy graph.

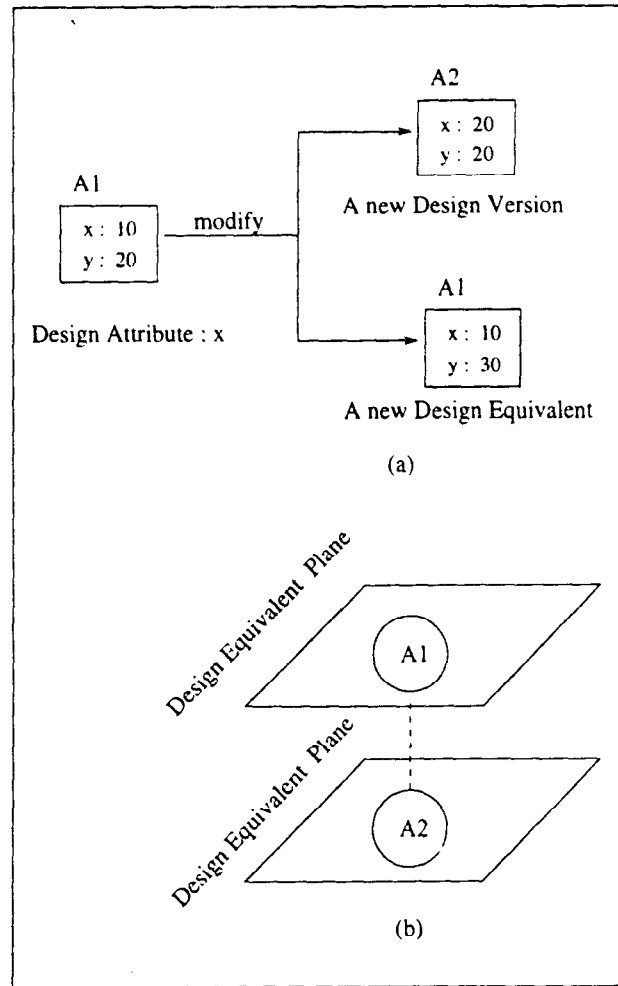


Figure 4: Design Version and Design Equivalence

3.2 Inadequacy of Existing Versioning Mechanisms

Any component has certain characteristic attributes associated with it. These key attributes play an important role in the design of the component. For example, consider the design of a stressed element. An element, for example steel, has many properties associated with it, such as Young's modulus, Poisson's ratio, yield stress, melting point and thermal conductivity. Depending on the design characteristics, only few of its properties are key to the design. For instance, if the design is that of an elastically stressed element, its Young's modulus and yield stress may be key to the design (fig. 3). Changes to the values of these key attributes will result in a new design. In other words, it is seen that a new object version does not necessarily imply that it is a new design. Depending on the design characteristics, object version may or may not lead to a new *design* version. Hence, it is seen that the existing versioning mechanisms fail to capture the

```

(a) Design Class A{
    Attributes:
        x,y,...;
        Component B *b;
        ...
    Methods:
        m1();
        ....;
};

(b) Semantic Class A{
    Design Attributes:
        x,B
    Interdependencies:
        notify_to : Q
};

(c) Semantic Class Wing{
    Design Attributes:
        aspect_ratio;
        thickness_to_chord_ratio ...
    Interdependencies:
        notify_to : tailplane
};

```

Figure 5: Semantic Class Definition design process as a concept. The versioning mechanism should, rather, seamlessly integrate with the design process. It should create new *Design Versions* based on the functionality and the behavior of the component rather than merely based on changing class definitions or attribute values.

Further, when the design of a component is changed, the design of another component may have to be altered. For instance, changing the design of the wing in an aircraft may require that the tailplane design also be changed. This is due to the interdependencies between the wing and the tailplane. These interdependencies need to be captured so as to ensure correct design of the artifact. In addition, as explained in section 2.2, the versioning process should also capture the semantics of each version, i.e. information regarding the necessity for the creation of that version, constraints that govern the use of that version, differences from other versions etc.

4 Modeling Primitives

As explained in the earlier section, the existing versioning mechanisms are inadequate for design applications. We propose a model which captures the design process as a concept. It also captures the interdependencies between the component designs and the semantic information regarding the version, thereby providing full support to the designer. The model is based on

```

Component Version Reference Class{
    Attributes:
        DerivedFrom :
        VersionedAttribute :
        DesignEquivalences:
        DefaultRetrieval :
        Annotations:
    Methods:
        add_delete_annotations();
        set_default_retrieval();
        update_design_equivalences();
        view_design_equivalences();
};

```

Figure 6: Component Version Reference Class the concept of Design Attributes, Design Versions and Design Equivalences.

4.1 Design Attributes and Design Versions

In our model, versions are created based on the functionality and behavior of the design component. This is achieved by capturing the semantic knowledge of the design through the concept of *Design Attributes*. Changes to values of the Design Attributes of a component will lead to a new *Design Version* of the component. The Design Attribute of a component can be a characteristic property of the component or a lower level component (in the composite hierarchy). For example, for military aircrafts, the Design Attributes are speed, rate of climb, combat radius, weapons system etc. These Design Attributes are important characteristics of military aircrafts. Versions of military aircrafts are distinguished based on these Design Attributes. Also, certain lower level components can be identified as characteristics of the higher level component. For example, in fig. 2b, which depicts the composite hierarchy of an aircraft, the bulkhead can be identified as the main component of the fuselage as it takes most of the structural load and governs the design of the fuselage. Hence, the bulkhead can be the Design Attribute of fuselage. Changes in the design of the bulkhead would create a new Design Version of the fuselage. Changes in non-Design Attributes such as doors and windows will not create a new Design Version. Hence, it is seen that the concept of Design Attributes and Design Versions captures the design process. It avoids unnecessary and meaningless versions.

4.2 Design Equivalence

Consider the composite object A (fig. 4). Its attributes are x and y , x being the Design Attribute. Changing the value of x will create a new Design Version of A1. However, changing the value of y will not create a new

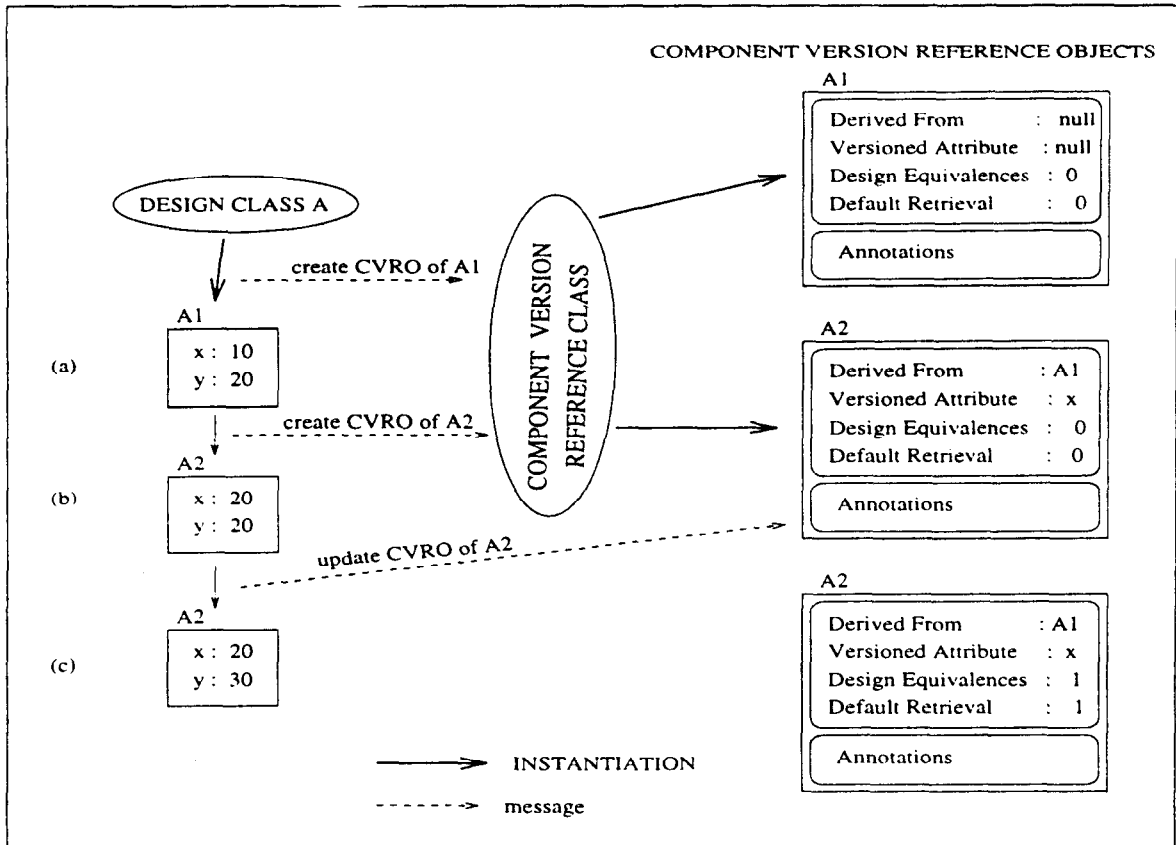


Figure 7: CVRO Management

Design Version as y is not a Design Attribute. Changing the value of a non-Design Attribute creates an *Equivalent Design Version*. In other words, Equivalent Designs differ only in the value of non-Design Attributes. In the version hierarchy, Design Equivalences are captured in the same plane. In fig. 4b, A1 and A2 are two Design Versions. The horizontal plane at A1 captures the designs equivalent to A1. Similarly, the horizontal plane at A2 captures the equivalent designs of A2.

5 Design Version Management

In this section, we focus on the issue of Design Version management. Management of Design Versions and Design Equivalences is achieved through Semantic Classes. The semantic information regarding each version is captured through the concept of Version Reference Classes.

5.1 Semantic Class

Each component is represented through its Design Class (same as conventional class definition). Fig. 5a gives the Design Class definition for a component A. The component A has certain attributes x, y... and lower level components B, C... (lower level components

are specified using the keyword component). With each Design Class, there is an associated *Semantic Class*. The Semantic Class of A captures the Design Attributes of the component A in the Design Attribute Interface and the interdependencies in the Interdependencies Interface. Fig. 5b gives the Semantic Class for the component A. It specifies x and B as the Design Attributes. Changes to the values of x, B would lead to a new Design Version. When y is changed, an Equivalent Design is created. Further, when a version Ai is modified to create a new Design Version Aj, the component Q which is interdependent with A is notified through the "notify-to" construct in the Interdependency Interface. Thus, the semantics expressed in the Semantic Class is used by the system to manage the creation of new Design Versions.

Fig. 5c gives the Semantic Class for the wing in an aircraft. The Design Attributes are aspect ratio, thickness to chord ratio etc. The interdependency between the wing and the tailplane design is captured in the "notify-to" construct.

5.2 Component Version Reference Class

The *Component Version Reference Class* captures the information regarding version derivation, ver-

COMPONENT VERSION REFERENCE OBJECT
FOR THE WING USED IN 737-300

Derived From	: wing_737-200
Versioned Attributes	:
Design Equivalences	: 2
Default Retrieval	: 1

Annotations	
Attribute:	
aspect_ratio ::	
chord_length ::	
.....	
Design:	
[1] wing aerofoil modified by 4.4%	
[2] new flap sections	
[3] additional lateral control spoilers	
[4] wing tip extended	
.....	
Manufacture:	
[1] materials	
[2] tolerance	
[3] manufacturing process and equipments...	
Maintenance:	
.....	
Miscellaneous:	
.....	

Figure 8: CVRO For Wing

sioned attributes, design equivalences and annotations(fig. 6). This class also supports methods like `set_default_retrieval`, `update_design_equivalences` etc.

For a component object A1 instantiated from the Design Class of A, the system creates the *Component Version Reference Object* (instance of Component Version Reference Class) for A1. The values of Derived From and Versioned Attribute is set to Null. The object A1 is referred, by default, as Design Equivalent 0(DE0) of A1. The values of the Design Equivalences and the Default Retrieval are set to 0(fig. 7a). When the designer retrieves A1 from the database and changes the value of a Design Attribute x, a new Design Version A2 is created. Automatically, the CVRO(Component Version Reference Object) for A2 is created(fig. 7b). The new version A2 is by default referred to as DE0, with the number of Design Equivalences being set to 0. When the designer retrieves A2 from the database, and changes the value of a non-Design Attribute y, an Equivalent Design is created, and is referred to as DE1 of A2(Design Equivalent 1 of A2). The value of Design Equivalences in CVRO of A2 is now incremented to 1(fig. 7c). The designer can explicitly retrieve the DE1 of A2 from the database. However, if he specifies just "retrieve A2", DE0 is retrieved as the value of Default Retrieval in CVRO of A2 is DE0. If the designer changes the value of Default Retrieval from DE0 to DE1, with the help of `update_design_equivalences` method, the command "re-

```
(a) Product Design Class Aircraft{
    speed, AUV, range, ...
    component Wing *w;
    component Fuselage *f: ...
};

(b) Product Version Reference Class {
    Annotations: ....
    Methods:
        add_delete_annotations();
        ....
};

(c) Product Version Reference Object A320:
    Annotations:
    Attribute:
        speed :: ...
        AUV :: ...
        ....
    Design:
        [1] centralized maintenance system
        [2] new gust load elevation system
        [3] fuselage cross section increased
        [4] wider aisle for quick turnarounds
        ....
    Manufacture:
        [1] assemblage details
        ....
    Maintenance: .....
    Miscellaneous:
        [1] first subsonic commercial aircraft
        to use Fly-by wire control throughout
        normal flight ...
```

Figure 9: Product Version Management
retrieve A2" will retrieve A2DE1. The designer can also view the design equivalents of A1 with the help of the `view_design_equivalences` method.

5.2.1 Annotations

The annotation is in the form of a text document giving information specific to the version. [Sci91] discusses the concept of annotated variables. In our model, annotations are classified into 5 sections: Attributes, Design, Manufacture, Maintenance and Miscellaneous annotations. The Attribute annotation describes each attribute of the component. The Design annotations give information regarding the attributes and lower level components corresponding to the version. The information is regarding the necessity for the creation of the version, characteristics of the version, and constraints that govern the use of the version. The Manufacture section gives information regarding the manufacture of the component version, like the machine tools and equipments to be used. The Maintenance section gives information regarding the maintenance aspects of the version. The Miscellaneous section can hold any other information the designer would like to specify.

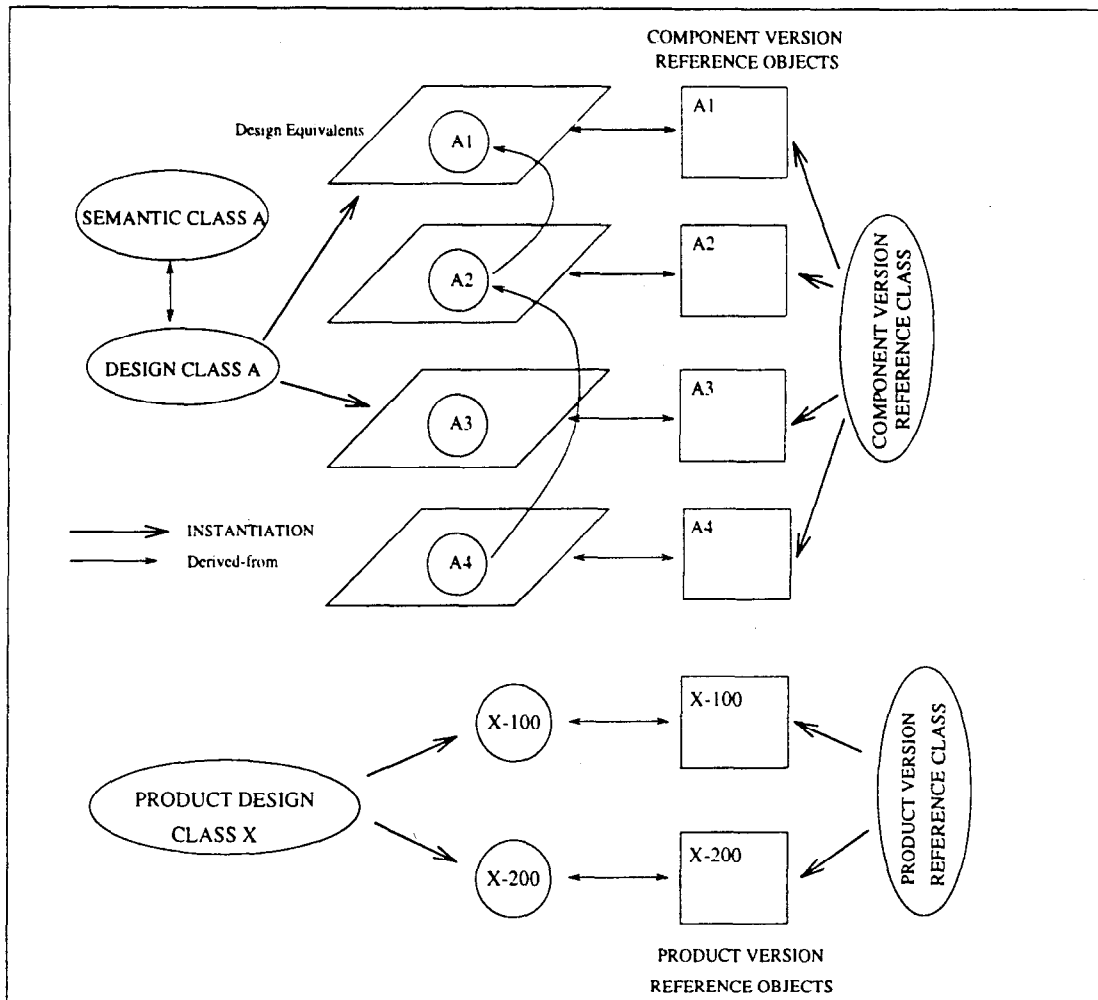


Figure 10: Design Version Organization

Fig. 8 gives the Component Version Reference Object for the wing used in B737-300 passenger version. This encompasses all the details regarding that version[Jan95]. When the designer retrieves the version from the database, he is provided with all the necessary information to proceed with the design. The incorporation of Manufacture annotations helps, in addition to the manufacturing engineer, the designer to analyze the version from the viewpoint of manufacturability. Similarly, the Maintenance annotations assist the maintenance engineer.

5.3 Product Version Reference Class

There is a Product Design Class associated with the product. Fig. 9a gives the Product Design Class for an aircraft. There may be different versions of the product. The *Product Version Reference Class* captures the information regarding the product version in the form of annotations(fig. 9b). As in the Component Version Reference Class, the annotations are sub-

divided into 5 sections, giving the attribute annotations, design, manufacture, maintenance and miscellaneous annotations specific to that product version. Fig. 9c gives the Product Version Reference Object for A320 aircrafts[Jan95].

5.4 Version Organization

The version organization is as shown in fig. 10. X denotes the product the company manufactures. A,B,C... are the lower level components. Each of these components is associated with a Semantic Class. The Component Version Reference Object is for each version of the lower level component. For example, Ai represents the version i of A, for which there is a unique Version Reference Object, which is an instance of the Component Version Reference Class. Ai can be either instantiated directly from the Design Class of A or can be derived from an existing version of A. Designs equivalent to Ai are captured in the same plane. For each version of the product, there is an associated Product Version

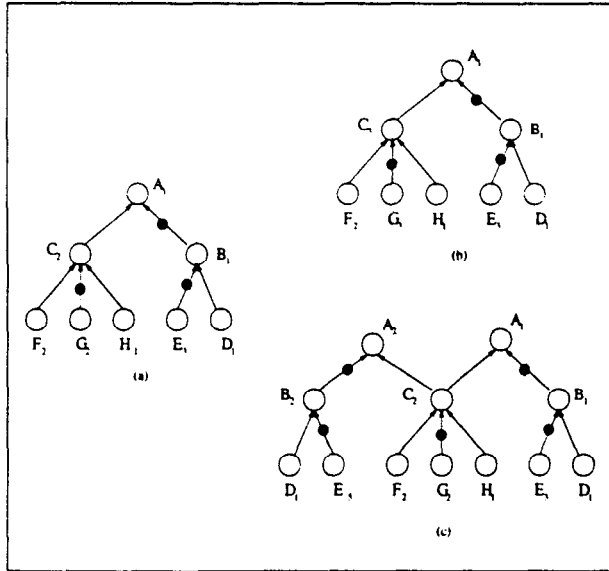


Figure 11: Change Propagation

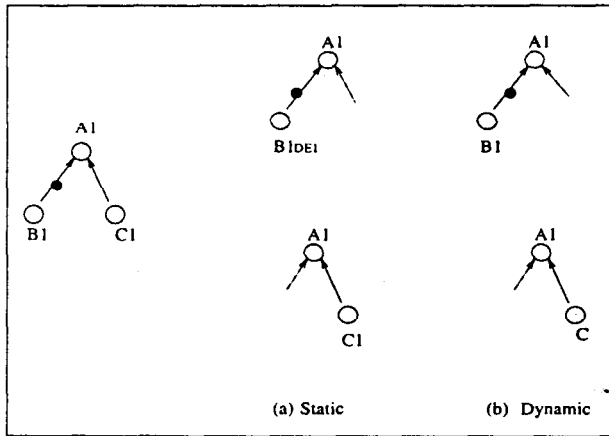


Figure 12: Configurations

Reference Object which is an instance of the Product Version Reference Class.

The designer can access each product version and its associated components and annotations. For example, he can say "retrieve the wing used in Product version 737-300". Else, he can also directly access all the wing design versions that exist in the database by specifying "retrieve versions of wing".

6 Issues In Version Management

In this section, we focus on the issues of inheritance among semantic classes, change propagation and configuration in version management.

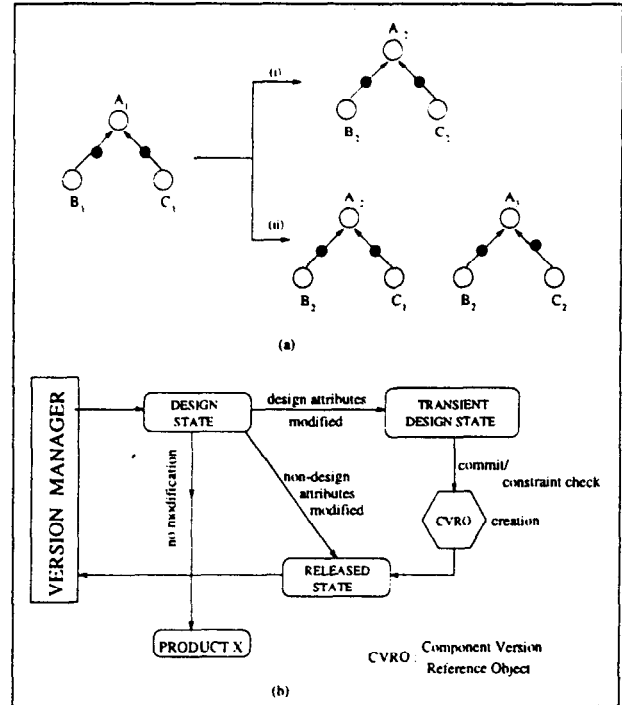


Figure 13: Design Transaction

6.1 Inheritance Among Semantic Classes

When a Design Class inherits from another Design Class, the Semantic Class remains the same for the derived class also. In case some of the attributes of the derived class is to be specified as Design Attributes, it is done in a new Semantic Class that is defined for the derived class.

6.2 Change Propagation

Change propagation is one of the key issues involved in version management [Kat87][Ska86]. The mechanism needs to have a disambiguous path to limit the scope of change propagation. Creating new versions all the way till the root of the composite hierarchy consequent to the creation of a new version of a component down the hierarchy, is certainly undesirable as it would create a large number of unnecessary meaningless versions. However, it must also be ensured that necessary versions are not missed. By necessary, it is meant that the changes caused must not be overlooked. This is possible only if the semantics of the versioning process is taken into account. In our model, the semantics of the versioning mechanism involved in the design process is captured through the concept of Design Attributes. Design Versions are created based on the changes to these Design Attributes. As explained earlier, Design Attributes for a component may be its characteristics or lower-level components.

Fig. 11 illustrates change propagation through the

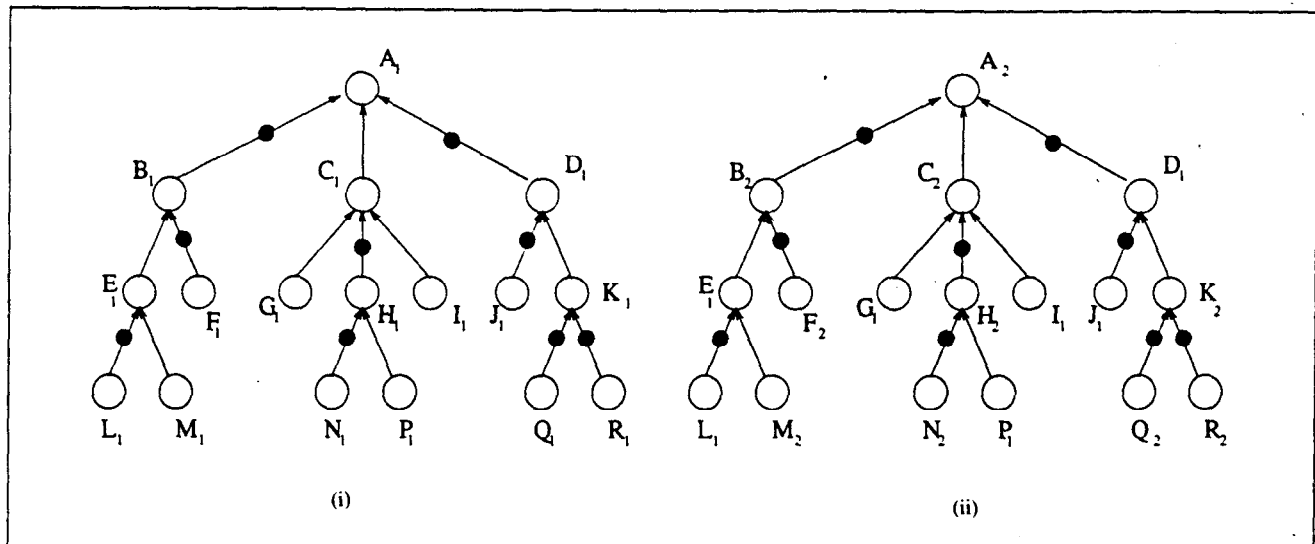


Figure 14: Transient State Propagation

concept of Design Versions and Design Attributes in our model. An arrow with a dot mark from B_j to A_i implies that B is a Design Attribute of A . The composite object A_1 is a version of A , and its Design Attribute is the lower level component B_1 . For B , the Design Attribute is E . Similarly, G is the Design Attribute for C . Fig. 11a gives the initial configuration of A_1 .

Suppose, if G_2 is replaced by another version G_3 , a new Design Version of C is created, but the propagation ends as the link between A_1 and C_3 is not a Design Attribute link. The resulting configuration is as shown in fig. 11b. However, when E_3 is replaced by E_5 , a new version of B is created. Further, as B is a Design Attribute of A , a new version of A is created. This propagation is atomic till a non-Design Attribute composite link is encountered. The resulting configuration in this case is as shown in fig. 11c.

6.3 Configuration

Consider fig. 12. A is a component, which has lower level components B and C . B is the Design Attribute of A . The binding between a version of A and its lower level component can be *static* or *dynamic*, depending on whether the link is a Design Attribute link or not.

For the link between a version of A and B , which is a Design Attribute link, the binding can be static wherein a particular Design Equivalent of B_1 is bound to A_1 . Else, the binding can be dynamic, wherein A_1 is just bound to B_1 . The designer must, at a later stage specify which Design Equivalent of B_1 he wishes to use in the product.

For the non-Design Attribute link between A and C , A_1 can be statically bound to a particular version of C , say C_2 . The designer, using his discretion can

decide which Design Equivalent of C_2 he wants to use, at a later stage. Else, A_1 can be dynamically bound to C wherein the designer decides which version of C he would like to use in the design.

7 Design Transaction Model

Consider fig. 13a. A_1 is a component with lower level components B_1 and C_1 , which are its Design Attributes. When the designer replaces B_1 and C_1 with B_2 and C_2 , the issue arises whether the resulting version will be as shown in (i) or (ii). As is evident, it is for the designer to specify which he wants. The designer usually retrieves an existing version of a component, makes all the changes required and then uses it in the product. To capture this process, the Transaction Model must provide some sort of transient state wherein the designer is permitted to make all the modifications required as an atomic operation.

When the designer retrieves a version from the database, it enters the *Design State*. The designer can use the component version directly in the product. Else, he can modify it to suit the requirements. In case he modifies non-Design Attributes, a new Design Equivalent Version is created, and it enters the *Released State*, whereby it can be used in the product, or stored back in the database. In case the designer modifies Design Attributes, the version enters the *Transient Design State*. In this state, change propagation is done such that a component is versioned only once. For example, in fig. 13a, when B_1 and C_1 are modified, A enters the Transient Design state. In this state, change propagation is done such that a new version is created only once. Hence, though B_1 and C_1 are modified, only one version of A is created. The designer then promotes the

version to the Released state, wherein all the Interdependencies are checked and Version Reference Objects for the new Design Versions are created. Fig. 13b illustrates the Design Transaction model. Fig. 14 gives an example. The initial configuration of A1 is as shown in fig. 14(i). The designer modifies M1, F1, N1, Q1 and R1. Though both Q and R are Design Attributes of K, when the designer versions Q1 and R1, K is versioned only once, as the modifications are done in the Transient State. When the designer commits, the propagation is done and the version of A enters the Released state. The resulting configuration is as shown in fig. 14(ii).

8 Conclusion

In this paper, we have proposed a new model for capturing Design Versions in Object-Oriented Databases. We are currently looking at the query language support for our model. We plan to integrate this with a collaborative design model [Ram97] so as to provide full support for engineering design applications.

References

- [Ban87] J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311-322, San Francisco, CA, May 1987
- [Cho88] H.T. Chou and W. Kim. A Unifying Framework for Version Control in a CAD Environment. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 336-344, Kyoto, Japan, August 1986
- [Jan95] *Jane's All The World's Aircrafts*, Jane's Information Group, 1995
- [Kat87] R.H. Katz and E. Chary. Managing Change in a CAD Database. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 455-462, England, September 1987
- [Kat90] R.H. Katz. Towards a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, Vol 22, No.4, pages 375-408, December 1990
- [Kes93] S.C. Keshu and K.K Ganapathy. *Aircraft Production Technology and Management*. Interline Publishings, Bangalore, India, 1993
- [Kim88] W. Kim and H. Chou. Versions of Schema for Object-Oriented Databases. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 148-159, September 1988
- [Kim89a] W. Kim, E. Bertino, and J.F. Garza. Composite Objects Revisited. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 337-347, Portland, Oregon, June 1989
- [Kim89b] Won Kim and Lochovsky, editors. *Object Oriented Concepts, Databases and Applications*. ACM Press, New York, 1989
- [Kim90] Won Kim. *Introduction to Object Oriented Databases*. MIT Press, 1990
- [Mon93] S. Monk and I. Sommerville. Schema Evolution in OODBs using Class Versioning. *SIGMOD RECORD*, Vol.22, No.3, pages 16-22, September 1993
- [Ngu89] G.T. Nguyen and D. Rieu. Schema evolution in Object Oriented Database Systems. *Data and Knowledge Engineering*, Vol.4, No.1, pages 43-67, 1989
- [Ram97] D. Janaki Ram, N. Vivekananda, Ch. Srinivasa Rao and N. Krishna Mohan. Constraint Meta-Object: A new model for Distributed Collaborative Designing. To appear in *IEEE Transactions on Systems, Man and Cybernetics*, Vol.27, Part A, Issue 3, May 1997
- [Sci91] E. Sciore. Using Annotations to Support Multiple Kinds of Versioning in a Object-Oriented Database System. *ACM Transactions on Database Systems*, Vol. 16, No. 3, pages 417-438, September 1991
- [Ska86] A.H. Skarra and S.B. Zdonik. The Management of Changing Types in an Object-Oriented Database. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 483-495, Portland, September 1986
- [Spo86] D.L Spooner, M.A. Milican and D.B. Fatz. Modeling Mechanical CAD Data with Data Abstraction and Object Oriented Techniques. In *Proceedings of the IEEE 2nd International Conference on Data Engineering*, pages 416-424, Washington, 1986
- [Sri89] D. Sriram, R. Logcher, A. Wong and S. Ahmed. An Object oriented Framework for Collaborative Engineering Design. In *Computer Aided Cooperative Product Development*, Proceedings of MIT-JSME Workshop in LNCS 492, pages 51-92, Springer-Verlag, 1991