データベース・システム 63－2
（１９８８．１．１８）

設計データベースのバージョン管理とその実現

[1] 宇田川 佳久 ・ [2] 溝口 徹夫

[1] 三菱電機株式会社・情報電子研究所
[2] 三菱電機株式会社・ＳＥセンター

本文では，設計データベースのためのバージョン管理について論じ，データ・ディレクトリを利用したアプローチが有効であることを示す。ここで提案しているデータ・ディレクトリは，
（１）バージョンに関する情報（例えば，識別名，時刻印，導出履歴など）を管理するリレーションと，
（２）コンフィギュレーション（すなわち，設計オブジェクト間の階層構造）を表現するグラフ構造，
とから構成されている。
このバージョン管理機能を実現し，本文で述べた方法が設計データを管理する上で有効なものであることを確認した。

Design and Implementation of Version Control Mechanisms for a Design Database

[1] Yosihisa Udagawa and [2] Tetsuo Mizoguchi

[1] Information Systems & Electronics Dev. Lab.
[2] SE Center

In this paper, we discuss version control issues for a design database and show that data directory approach is useful in controlling version histories. Our data directory consists of two parts :
(1) a relation for maintaining information about a design object (e.g. an identifier, timestamps and derivation history),
(2) graph structures to represent configurations (i.e. hierarchies among the design objects).
The proposed version mechanisms have been implemented by one of the authors. Experiments have shown that our approach is quite effective in keeping track of design data.

1．はじめに

ＣＡＤ（Computer Aided Design）システムの構築法に変化の兆しがある。設計データを集中管理する“設計データベース”に基づいて設計ツールを統合し，より生産性の高いＣＡＤシステムを構築しようとする動きである〔14,21,22,23〕。

従来のＣＡＤシステムの主な目的は，効率的に“設計データを生成し検証”することであった。一方，ＣＡＤシステムが普及するに伴い，優良な過去の設計事例を有効に使いたいという要求が高まってきた。即ち，近い将来のＣＡＤシステムでは，“設計データを検索し再利用”する機能が重要視されると予想される。また，ＣＡＤシステムの守備範囲を設計の上流または下流にまで広げようとする動きもある。この場合，設計データを複数の設計ツールが共用するようになり，データの整合性管理に関心が寄せられるようになる。当然，設計データの並列アクセスやセキュリティ，リカバリに対する要求も強くなってくる〔2,4,7〕。

このような問題は，４半世紀あまり前，事務処理の分野でも起こっている。事務処理分野におけるこの問題に対する解決策は，データベース管理システムを使うものであった。すなわち，必要なデータをデータベース管理システムによって一元管理し，応用プログラマが必要とするデータは，すべてこのデータベース管理システムを経由してアクセスされるようにする。これにより，ファイル構成の変換に煩わされることも，データ検索のために多大な応用プログラムを作成する必要もなくなった。さらに，データの変更に際してデータの整合性を維持することも，データベース管理システムを使わない場合に比べて容易になるという利点があった〔11,19〕。

設計データベースに基づいてＣＡＤシステムを構築する場合，同様の利点を期待することができる。事務用データベースも，設計データベースも解決すべき問題は類似しており，問題解決のためのアプローチも良く似ているからである。これまでの研究から，事務用データベースと設計データベースには，以下に示すような機能が共通していることが知られている〔2,4〕.

（１）２次記憶上のデータへの効率的なアクセス
（２）同時アクセス機能
（３）応用プログラムのための効率的な言語インターフェースの提供
（４）表，または図形によるレポート生成機能
（５）ユーザ・フレンドリ言語の提供
（６）バック・アップとリカバリ機能
（７）セキュリティ機能

一方，設計データベース特有の機能としては，以下のものが指摘されている。

（８）設計オブジェクトを色々な形態で表現する機能
（９）設計オブジェクトの階層設計を支援する機能
（10）設計オブジェクトのバージョン管理機能
（11）設計オブジェクトの生成，変更，検証，利用状況の管理機能

設計オブジェクトは，設計のフェーズごとに異なる表現を持ち得る。例えば，ＶＬＳＩ設計の回路は，論理設計のフェーズでは回路図または状態遷移図，論理シミュレーションのフェーズではネット・リストといった表現形態をとる。（８）の機能を実現するためには，文字数値の他に図形や画像などの，いわゆるマネチメディア・データを扱うことが本質であり，この方面からの研究が行われている〔13,14,15〕。

（９）,（10）,（11）の機能を実現するためには，オブジェクト指向アプローチが有効であると目されている。オブジェクト指向の意味するところは，研究者によって若干の違いがある。本文では，オブジェクト指向という言葉を“現実世界の実体をデータ構造とそれを操作する演算によって表現すること”という意味に用いるものとする〔5,9,10,16,17,18,20〕。オブジェクト指向アプローチの利点は，内部データ表現と操作を隠蔽することができることである。この性質は，設計オブジェクト単位の操作／管理を容易にする。例えば，回路図はより単純な回路図の組合せとして表現される。この場合，回路図が操作／管理の単位であり，回路図がどのようなデータ構造と操作演算によって表現されているかは別の次元の問題である。設計オブジェクトの階層，バージョン，それに変更履歴の管理は，設計オブジェクトを単位として行うのが自然なやり方であると考えられる。

本文では，ＡＤＡＭ（Advanced Database with Abstraction Mechanism）と称するデータベースのバージョン管理機能と，それを実現するためのデータ・ディレクトリについて論じたものである。ＡＤＡＭは，リレーショナル・データモデルに図形・画像を操作する機能，及びデータ構造と操作演算を一体化する機能を付加したもので，ディジタル回路の管理に適用されている〔12〕。

本文第２章では，ＡＤＡＭデータモデルの概要とＡＤＡＭの設計オブジェクト管理機能について述べる。第３章では，設計オブジェクトの変更通知をするために３種類の時刻印を使う方法が有効であることを論じている。第４章では，設計オブジェクトを管理するためのデータ・ディレクトリの構造と操作の概要について述べている。第５章では，データ・ディレクトリのインプリメンテーションと利用法について述べる。

2．　ＡＤＡＭデータモデル

2．1　オブジェクト指向

設計オブジェクトは，一般に，階層構造を成す。この階層構造をＣｏｄｄが提案したリレーショナル・データモデルによって表現することも可能である〔1,3〕。しかし，設計アプリケーション・プログラムは，階層中のオブジェクトを意味のある単位として処理するこ

とが多い。すなわち，
・データのアクセスは，階層中のオブジェクトを単位とすることが多い，
・操作演算は，オブジェクトごとに異なることが多い。
このような理由から，階層中のそれぞれのオブジェクトに対して，データとそれに対する操作演算をひとまとめにして記憶／管理する，いわゆるオブジェクト指向アプローチが妥当であると考えられている。

本文で述べるＡＤＡＭデータモデルは，
（１）オブジェクトの名前
（２）オブジェクトに対するパラメータ
（３）オブジェクトの概略図形表現
（４）オブジェクトの概略構造表現
（５）オブジェクトの詳細図形表現
（６）オブジェクトの詳細構造表現
を一つの単位としてアクセスできるようにしている（図１）。構造表現は，リレーショナル・データベースを基本としており，図形表現はリレーション中のデータをパラメータとして記述されている。この意味で，ＡＤＡＭはリレーショナル・データベースのひとつの拡張と位置付けることができる。

個々のオブジェクトの定義情報と集約・汎化階層に関する情報は，オブジェクト指向データベースのメタ情報と考えることができる。ＡＤＡＭでは，これらのメタ情報をデータ・ディレクトリによって管理している。ＡＤＡＭのデータ・ディレクトリの構造については，本文第４章で論じられている。

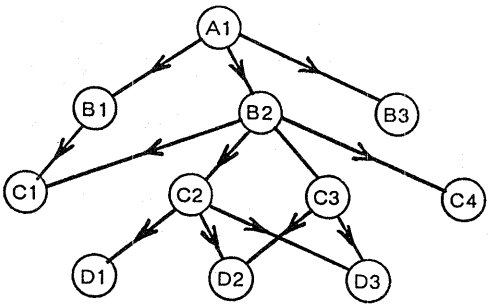

図１．ＡＤＡＭデータモデルの概要

２．２　ＡＤＡＭにおける設計オブジェクト管理機能

設計データベースでは，設計オブジェクトをインターフェースの記述（仕様）と実現法の記述に分けて管理することが有効であると言われている〔1,4〕。ＡＤＡＭデータモデルもこの手法を支援する機能をもっている。設計オブジェクトをインターフェースと実現法の記述に分けて管理する場合，バージョンとは“同じインターフェースを有するが実現法が異なる設計オブジェクト”と定義される。一つの設計オブジェクトから複数の設計バージョンが作られ得るので，設計バージョンの導出履歴は，一般に木構造となる。

ＡＤＡＭにおける設計オブジェクト管理機能は，次のとおり。
（Ａ）設計バージョンの導出履歴管理
（Ｂ）コンフィギュレーションの管理
（Ｃ）未具体化オブジェクトの管理

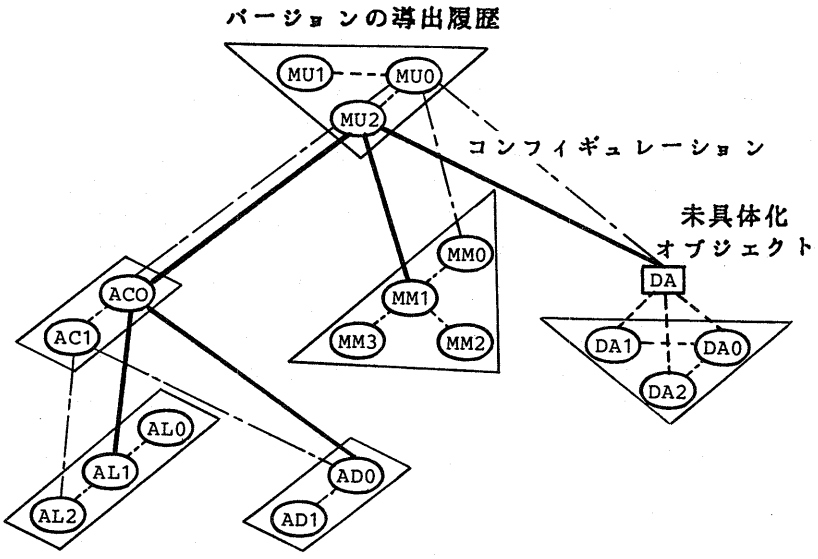

図２．ＡＤＡＭのオブジェクト管理の概要

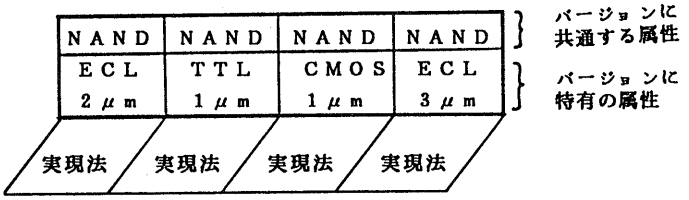

図3．インターフェースの属性の汎化とバージョン

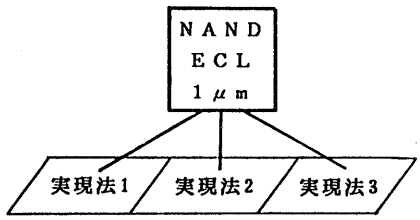

図4．未具体化オブジェクトと
実現法の記述

図2は，ＡＤＡＭのオブジェクト管理の概略を示している。

設計データベースでバージョンの管理が必要とされるのは，設計オブジェクトが次の性質をもっているためである。

（1）設計オブジェクトは，一度に完成されることは稀で，設計とテスト（デバッグ）を繰り返しながら作られることが多い。

（2）設計ミスや出来栄えの良い設計オブジェクトを捜すために，設計オブジェクトの導出履歴を追いかける必要がある。

（3）インターフェースを満たす設計オブジェクトは，一般に，論理的にも物理的にも複数の実現法がある。

（4）同じインターフェースを有する設計オブジェクトを複数のエンジニアが競作することがあり，これを支援する必要がある。

設計バージョンは，より単純な構造をした設計オブジェクト（あるいは部品）から組立てられている。部品は，さらに単純な構造をした部品から構成されている。このようにして，設計オブジェクトは階層構造を構成する。上記のように，階層構造中の設計オブジェクトには複数のバーシーョンが付随している。コンフィギュレーションは，階層構造中のオブジェクトから特定のバージョンを指定し，目的とする設計オブジェクトの代替案を作るものである。いわば，バージョン管理が部品レベルの代替案を管理するものであるのに対し，コンフィギュレーションは（サブ）システムの代替案を管理するものと位置付けることができる。設計データベースでコンフィギュレーションの管理が必要とされるのは，設計オブジェクトが次の性質をもっているためである。

（1）設計オブジェクトには，それが使用できる条件や，他の設計オブジェクトとの依存／排他関係がある。これらを考慮した，ある時点における全体の設計オブジェクトを管理する必要がある。

（2）ある設計オブジェクトが全体の設計オブジェクトから見ても整合性のとれたオブジェクトであることをテストするために，一時的な全体の設計オブジェクトを定義する必要がある。

（3）全体の設計オブジェクトからみた，それぞれの要素オブジェクトの機能や性能を考慮し，要素オブジェクトの分割や合併を支援する必要がある。

分散処理技術の進歩により，一つのオブジェクトが複数のエンジニアによって同時に設計されることが可能になってきた。同時設計を支援することは，設計データベース全体の性能向上に結び付く。未具体化オブジェクト（Uninstantiated Object）は，設計の同時作業を支援するために用意されたもので，設計オブジェクトのインターフェースと実現法の記述を任意の時点で結合することを可能とするものである。未具体化オブジェクトは，Katzら〔5〕が提唱している動的コンフィギュレーション・バインディングの機能を提供することができる。

バージョンと未具体化オブジェクトの相違を次に述べる。ＡＤＡＭデータモデルにおけるオブジェクトの概要構造表現には，オブジェクトに関する属性が含まれている。これらの属性のうちのいくつかはオブジェクトに共通なものである。ＡＤＡＭにおいて，バージョンは共通な属性に基づいて汎化されたオブジェクトとして扱われる。例えば，ゲートは機能を分類する属性（NAND.NORなど）と製造技術に関する属性（プロセス，配線幅など）によって特徴付けられる。このとき，ゲートを機能によって分類すれば，“ＮＡＮＤゲート”というバージョンが作られることになる（図3）。バージョンにおいては，インターフェースと実現法の記述は1対1に関連付けられている。

これに対して，未具体化オブジェクトは，図4に示したように，実現法の記述が特定されていないオブジェクトである。未具体化オブジェクトをバージョンと同じ機能をもつ設計オブジェクトとするためには，実現法の記述を特定し，インターフェースの記述と特定された実現法の記述に関連付けなければならない。この関連付けを行うためには，一般に，オブジェクトに依存した手続きを起動しなければならない。というのは，関連付けをするには，オブジェクトを記述しているデータ構造を操作しなければならないからである。

3．設計オブジェクトの変更通知

設計バージョンは，バージョン間の参照関係に於ても，時間的な変更過程に於ても複雑な構造をしている。第2章で述べたように，設計オブジェクトは階層的に構成され，しかも各種の代替案が関連し合っている。

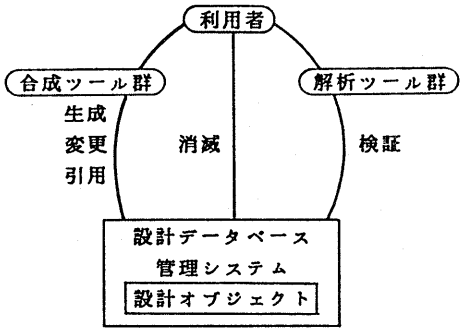

図５．設計オブジェクトと操作の概要

従って，ある設計バージョンの変更に伴って，関連するオブジェクトが影響を受けることが，しばしば起こり得る。ある設計バージョンが変更されたときに，影響が及ぶオブジェクトにその変更を通知することを設計オブジェクトの変更通知（Change Notification）言う。

設計バージョンは，生成，変更，他のバージョンの引用，他のバージョンからの引用，検証，消滅といったライフ・サイクルを辿る。図５は，設計オブジェクトとそれに対する操作の概要を示したものである。設計オブジェクトの生成，変更，引用は，ジェネレータやエディタなどの合成ツール群によって行われる〔4〕。他方，設計オブジェクトの検証は，各種のシミュレータや設計ルール・チェッカなどの解析ツール群によって行われる。いわば，上記のような設計ツールを介して設計オブジェクトは，生成され，成長してゆくわけである。

設計オブジェクトに操作が加わるたびに，設計オブジェクトのコピーを取っておく方法は，あらゆる時点での設計オブジェクトを再現できるわけであるから，原理的には万能なものである。しかし，実用的な実現が難しいことも確かである。設計オブジェクトを操作が加わるたびに記憶するには，膨大な記憶容量を必要とすること，さらに，一般的には，ひとつの操作は設計オブジェクトに対して意味のある操作になり得ないからである。

これに対して，合成ツールまたは解析ツールの処理が終了した時刻を基準に，設計オブジェクトの変更履歴を管理しようとするアプローチがある。これが先に述べた，設計オブジェクトの変更通知で，設計データベースの重要な機能のひとつとなっている。設計オブジェクトの変更通知に関する研究は，設計オブジェクトに対して意味のある操作が行われた時刻に基づいて設計オブジェクトの状態を定め，好ましくない状態が発生したときに，その旨を，利用者に通知するものと位置付けることができよう。

Batoryら〔１〕は，変更通知が必要となるタイミングを２種類の時刻印（変更した時刻ＣＮＴと変更が受諾された時刻ＣＡＴ）を用いて判定する手法を提案している。設計データベース中のあらゆるバージョンは，２種類の時刻印をもっている。すなわち，Ｖを設計バージョンとするときＶ．ＣＮＴはＶが変更された時刻，Ｖ．ＣＡＴはＶの変更が受諾された時刻を表している。

Ｖ．ＣＡＴ ＞ Ｖ．ＣＮＴであるとき，バージョンＶは設計変更された後に，その変更が受理（検証）されていることから，実現無矛盾であると呼ばれる。バージョンＶの実現のために使われている設計バージョンの集合をＪとする。条件

〔∀Ｘ∈Ｊ〕（Ｘ．ＣＮＴ＜Ｖ．ＣＡＴ）

が成り立つとき，バージョンＶは参照無矛盾（Reference Consistent）であると定義されている。これは，Ｖが受諾された後に，Ｖを構成するあらゆるオブジェクトが変更されていないことを保証するものと解釈することができる。また，

（Ｖ．ＣＮＴ ＜ Ｖ．ＣＡＴ）＆
〔∀Ｘ∈Ｊ〕（(Ｘ．ＣＮＴ ＜ Ｖ．ＣＡＴ)
＆(Ｘ．ＣＮＴ ＜ Ｘ．ＣＡＴ)

であるとき総合無矛盾（Totally Consistent）と定義されている。総合無矛盾の定義には，若干の問題点がある。例えば，設計バージョンＶが，バージョンX_1，X_2，X_3から構成され，時刻印ＣＮＴ，ＣＡＴに図６のような関係があるものとする。このとき，バージョンＶは，総合無矛盾である。なぜなら，

Ｖ．ＣＮＴ ＜ Ｖ．ＣＡＴ

であり，$X \in \{X_1, X_2, X_3\}$なるすべてのＸに対して，

(Ｘ．ＣＮＴ ＜ Ｖ．ＣＡＴ)
＆(Ｘ．ＣＮＴ ＜ Ｘ．ＣＡＴ)

が成り立つからである。

バージョンX_1，X_2，X_3がすべて受諾されたのなら問題は起こらない。しかし，図６のＸ　のように時刻Ｖ．ＣＡの後に受諾されないバージョンがあると，バージョンＶは妥当でない設計バージョンを含むものとなり，Ｖも妥当なものとは言い難くなる。このような問題が生じたのは，総合無矛盾の定義にはＸ．ＣＡＴとＶ．ＣＡＴに関して何の制約もないことが原因である。このために，バージョンＶが受諾された後に，

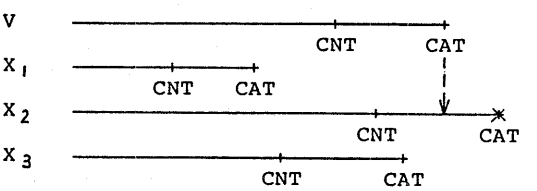

図６．２時刻印方式の例

その構成要素であるバージョンXが受諾されないことが起きると，Vが妥当でないバージョンXを含んだものとなってしまう。

このような状態を排除するために，我々は次の論理式で定義される局所無矛盾（Locally Consistent）なる概念を提案する。

〔∀X∈J〕((X.CAT < V.CAT)
& (X.CAT < X.LRT))

ここで，第3番目の時刻印LRTが導入された。LRTはバージョンが引用された最新の時刻を保持している。局所無矛盾の意味するところは，バージョンVが受諾される以前に，Vを構成するすべてのバージョンXが受諾されており，しかもXが引用されたのはXが受諾された以後であるということである。この定義によれば，局所無矛盾でない（局所矛盾である）のは，次の3つの場合が考えられる。

（1）要素バージョンXのうち，受諾される前に引用されたものがある。

（2）バージョンVがすべての要素バージョンが受諾される以前に受諾された。

（3）（1）と（2）の両方が起きた。

図7に，局所無矛盾であるバージョンの例を示した。図中の破線で示した部分が局所無矛盾の定義に反する部分である。

設計バージョンが，実現無矛盾，総合無矛盾，局所無矛盾であることを調べるためには，3種類の時刻印（CNT，CAT，LRT）が必要である。次の章で述べるように，ADAMのデータ・ディレクトリにはデータベース内のすべての設計バージョンに対して，この3種類の時刻印を保持している。

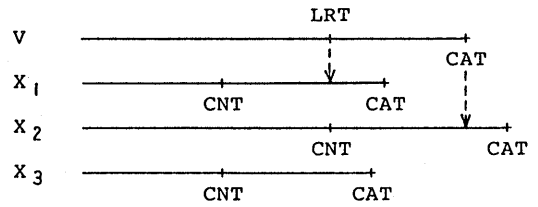

図7．局所矛盾である例

4．データ・ディレクトリによる設計オブジェクトの管理

4．1　データ・ディレクトリの概要

本文第3章，第4章で論じた設計オブジェクトの管理機能は，図8に示したデータ・ディレクトリによって実現される。データ・ディレクトリは，設計オブジェクトの属性情報を管理するリレーションDDD，バージョン導出木，コンフィギュレーション木の3種類のデータ構造で構成されている。以下，設計オブジェクト管理機能とデータ構造との関係を述べる。

4．2　設計バージョンの管理

設計バージョンを管理するためには，設計バージョンの導出履歴とバージョンに付随する属性情報を扱わなければならない。設計バージョンは設計オブジェクトの名前（O－ID）とバージョン識別子（VSN）の組合せで表現される。それぞれの設計バージョンは，本文第3章で述べたように，バージョンが変更された時刻（CNT），変更が受諾された時刻（CAT）それに他のバージョンから引用された時刻（LRT）が付随している。ADAMのデータ・ディレクトリでは，これらの情報に加えて，設計バージョンのテスト・レベルを記憶する項目（TESL）と，バージョン導出木を表現するための項目（PVE）を備えている。

リレーションDDDのオブジェクト名の第8文字目は，設計バージョンの実現法が定義されているか否かを区別するために用いられている。この文字が“D”であるとき実現法が定義されていることを示している。従って，この設計バージョンの設計変更時刻（CNT）の項目には，バージョンが定義または変更された時刻が記憶されている。他方，オブジェクト名の第8文字目が空白であるとき，実現法が未定義であることを示している。従って，CNT項目の時刻は未定義となっている。ただし，実現法が未定義であっても，他のオブジェクトから引用されていれば，LRT項目に引用時刻が記憶されている。

設計バージョンの導出木は，バージョンの生成とともに構成されてゆく。ひとつの設計バージョンからは

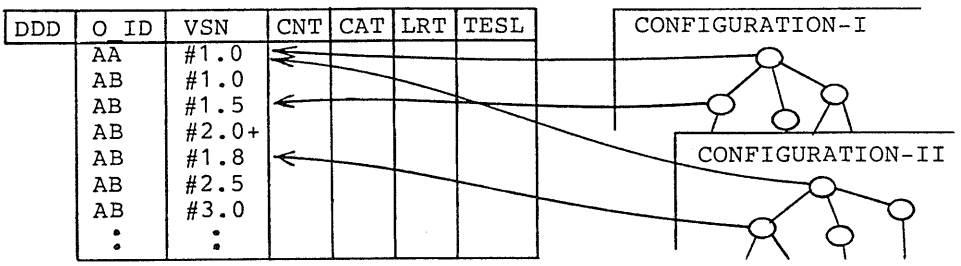

図8．データ・ディレクトリの概要

$DPRI(1,90)

? NO	O-ID	VSN	PVE	CNT	CAT	LRT	TES
1	PSYS	D #1.0 +	0	91987/135129	0/ 0	0/ 0	1
2	PSUPP	D #1.0 +	0	91987/134311	0/ 0	90987/194017	1
3	MSYSA	#1.0 +	0	0/ 0	0/ 0	90987/194017	0
4	MSYSB	#1.0 +	0	0/ 0	0/ 0	90987/194017	0
5	P	#1.0 +	0	0/ 0	0/ 0	90987/194017	0
6	PSYS	D #2.0	1	90987/194449	0/ 0	0/ 0	1
7	PSYS	D #3.0	1	90987/194122	0/ 0	0/ 0	1
8	PSYS	D #2.4	6	90987/194249	0/ 0	0/ 0	1
9	E2	#1.0 +	0	0/ 0	0/ 0	91987/134311	0
10	DY	#1.0 +	0	0/ 0	0/ 0	91987/134311	0
11	B	#1.0 +	0	0/ 0	0/ 0	91987/134311	0
12	S	#1.0 +	0	0/ 0	0/ 0	91987/134311	0
13	E4	#1.0 +	0	0/ 0	0/ 0	91987/134311	0
14	PSYS	D #4.0	0	91987/134345	0/ 0	0/ 0	1
15	PSYS	D #4.2	14	91987/134421	0/ 0	0/ 0	1

図９．リレーションＤＤＤの例

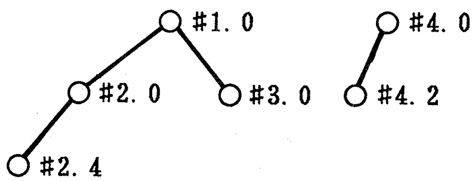

図10. オブジェクトPSYSのバージョン導出木

任意の数の設計バージョンが導出されるが，ある設計バージョンの親は唯一つである。この性質を利用してバージョンの導出木を管理することができる。すなわち，それぞれのバージョンに対して，親バージョンのデータが記憶されている場所を記憶するＰＶＥ（Position of Version Entry ）フィールドを用意する。図9に示したリレーションＤＤＤは，図１０に示したバージョン導出木を含んでいる。Katzらは〔５〕，バージョン導出木を一つのルート・ノードを持つ木構造として論じている。しかし，一般には，与えられた設計仕様を満たす設計オブジェクトには複数の実現法がある。この場合，図１０に示したように複数のルート・ノードを持つ木構造となる。ＡＤＡＭのバージョン管理の特徴の一つは，複数のルート・ノードを持つ木構造を扱っていることにある。

一般に，バージョン導出木には数多くの設計バージョンが含まれており，設計オブジェクトを引用するたびに，どのバージョンを引用するかいちいち指定するのは煩わしい。バージョン導出木で管理されているバージョンのうち代表となるものを決めておけば，このような煩わしさはなくなる。バージョン導出木中の代表となるバージョンは，現在有効バージョン（Current Version）と呼ばれている〔５〕。現在有効バージョンは，任意の時点で，それぞれの設計バージョンに対してただひとつだけ存在する。リレーションＤＤＤのバージョン識別名の第８文字目は，設計バージョンが現在有効バージョンであるか否かを識別するために使われている。バージョン識別名の第８文字目が“＋”であるとき現在有効バージョンであることを示し，空白であるとき現在有効バージョンではないことを示している。

４．３　コンフィギュレーションの管理

コンフィギュレーションは，設計バージョンが，他のどんな設計バージョンから構成されているのかを管理するためのものである。設計バージョンをノードとし，バージョンの引用関係を親バージョンから子バージョンに向かうアークに対応付けると，コンフィギュレーションはサイクルを含まないグラフとして表現される〔４〕。このグラフ構造は，あらかじめ定められたものではなく，設計の進展とともに変化してゆく。バージョン導出木の管理と同様に，コンフィギュレーションもリレーションＤＤＤとは独立したデータ構造によって管理するのが適当である。ＡＤＡＭのコンフィギュレーション管理では，バージョン間の引用関係の他に，親バージョンが子バージョンを引用している個数も管理している。すなわち，コンフィギュレーションは，＜親オブジェクト名，親バージョン識別子，子オブジェクト名，子バージョン識別子，引用個数＞という構造をもったデータの集合によって表現されている。

DDD						
O_ID	VSN	PVE	CNT	CAT	LRT	TESL
DA	%					
DA	#1.0					
DA	#1.8 +					
DA	#2.2					

具体化 ↓ ↑ 未具体化

DDD						
O_ID	VSN	PVE	CNT	CAT	LRT	TESL
DA	%1.8					
DA	#1.0					
DA	#1.8 +					
DA	#2.2					

図１１．具体化／未具体化操作の例

４．４　未具体化オブジェクトの管理

未具体化オブジェクトを管理するためには，データ・ディレクトリの操作とともに，設計オブジェクトのインターフェースの記述と実現法の記述を関連付けたり，関連付けを解いたりする手続を起動する必要がある。実際，ＡＤＡＭのデータ・ディレクトリによって管理されるのは，未具体化オブジェクトが存在するか否か，存在するときインターフェースの記述と実現法の記述が関連付けられているか否か，という情報に限られている。

図11は，未具体化オブジェクトの管理法の概略を示している。未具体化オブジェクトであることは，リレーションＤＤＤのバージョン識別子の第１文字目が“%”であることで判別している。ちなみに，未具体化オブジェクトでないとき，バージョン識別子の第１文字目は“#”である。バージョン識別子の第２文字目から第７文字目までが空白であるとき，インターフェースの記述と実現法の記述が関連付けられていないことを示している。これらの文字列にバージョン識別子が指定されているとき，指定されているバージョンの実現法と未具体化オブジェクトが関連付けられていることを示す。

インターフェースの記述と実現法の記述が関連付けられたとき，時刻印ＣＮＴとＣＡＴとには，関連付けが完了した時刻に書き換えられる。また，実現法を提供したバージョンの時刻印ＬＲＴも，同上の時刻に書き換えられる。

５．　データ・ディレクトリのインプリメンテーション

５．１　インプリメンテーションの概要

ここでは，本文第４章で述べたデータ・ディレクトリのインプリメンテーションの状況について述べる。まず，データ・ディレクトリに関する機能とそれらの機能を実現するためのコマンドを示す。次に，標準IC回路図の管理に適用した例を示し，このデータ・ディレクトリが回路図の概略情報を手掛かりに，類似した回路図を検索する上で役に立つことを述べる。

５．２　データ・ディレクトリの機能と操作コマンド

データ・ディレクトリの機能を以下に示す。

（１）バージョンの定義・検索・削除
（２）現在有効バージョンの変更
（３）コンフィギュレーションの定義・検索・削除
（４）未具体化オブジェクトの管理
（５）タイム・スタンプの妥当生のチェック

現在までに（１）－（３）までの機能がインプリメントされており，ＦＯＲＴＲＡＮ言語で約３０００行を要している。表１は，（１）－（３）までの機能をインプリメントするために必要となった基本操作コマンドの一覧表である。（３）を除く機能は，リレーションＤＤＤに対する操作によってインプリメントされるものであり，論理的にはリレーショナル演算の範囲で記述できる。（３）のインプリメンテーションのためには，オブジェクトの階層を頂上まで，または麓まで辿る機能（$REFALLP, $REFALLC）と，オブジェクトの階層中にサーキットが含まれているか否かを判定する機能（$REFCR）が必要となる。

```
$REF(?,?,NAT2,?,?)
 $REFPRI
  LN   PARENT      P.VSN       CHILD       C.VSN       NUM
-----------------------------------------------------------
   1   SWD      D  #1.0    +   NAT2     D  #1.0    +     1
   2   SWDD     D  #1.0    +   NAT2     D  #1.0    +     1
   3   M26      D  #1.0    +   NAT2     D  #1.0    +     4
   4   M37      D  #1.0    +   NAT2     D  #1.0    +     4
   5   PPP      D  #1.0    +   NAT2     D  #1.0    +     1
   6   PP       D  #1.0    +   NAT2     D  #1.0    +     1
   7   M00      D  #1.0    +   NAT2     D  #1.0    +     4
```

図12．回路図と構成要素回路図の一覧

5．3　標準IC回路図管理への適用

本文で論じたデータ・ディレクトリは，いわばオブジェクトの概要を管理しているものである。この情報を使えば，ユーザが探しているオブジェクトに近いものを検索することができる。現在のＡＤＡＭデータベースには標準IC回路図など約２５０枚が管理されている。この程度の数でも，データベース中にどんな回路図が入っているのかを図面の識別名だけをたよりに管理することは難しい。ここで，データ・ディレクトリが大きな威力を発揮する。例えば，２入力・トーテンポール出力のＮＡＮＤゲート（ＮＡＴ２）を構成要素としている回路図を検索した結果を図12に示した。検索結果には，指定された回路図の引用個数も表示されているので，所望の回路図であるか否かのおおよその見当を付けることができる。こうして見当を付けた回路図をもう少し詳しく見るためには，親の回路図名を指定し，その回路図を構成する要素回路図の一覧表をつくれば良い。図13は，回路図名が“Ｍ１８３”，バージョンが“＃１．０”である回路図を構成する部品回路図と引用個数を示している。回路図の生成時刻などを知りたければリレーションＤＤＤを検索すれば良い。図14は，部品回路図の引用個数が８個を超える回路図の一覧である。

表１．データ・ディレクトリの基本操作コマンド

機　能	コマンド名／引数の数
バージョンの定義	$DDEF/（1..6）
バージョンの検索	$DRET/（1..6）
バージョンのプリント	$DPRI　/2
バージョンの削除	$DDEL　/1
現在有効バージョンの変更	$CVDES　/2
バージョン情報の入力	$DREAD　/0
バージョン情報の出力	$DWRTE　/0
引用関係の定義	$REFDEF　/3
引用関係のプリント	$REFPRI　/0
引用関係の削除	$REFDEL　/5
引用関係の変更	$REFUPD　/5
引用関係の一段の検索	$REF　/5
引用関係の上方検索	$REFALLP/2
引用関係の下方検索	$REFALLC/2
サーキットの検出	$REFCR　/2
Configuration の定義	$CFDEF　/1
Config. 情報の入力	$CFREAD　/1
Config. 情報の出力	$CFWRTE　/1

```
$REF(M183,#1.0,?,?,?)
 $REFPRI
  LN   PARENT     P.VSN      CHILD      C.VSN      NUM
-----------------------------------------------------
   1   M183  D  #1.0  +   NOT3  D  #1.0  +    1
   2   M183  D  #1.0  +   NOT4  D  #1.0  +    1
   3   M183  D  #1.0  +   ANT2  D  #1.0  +    3
   4   M183  D  #1.0  +   ANT3  D  #1.0  +    4
   5   M183  D  #1.0  +   INT   D  #1.0  +    3
```

図13．回路図と構成要素回路図の一覧

```
$REF(?,?,?,?,?>8)
 $REFPRI
  LN   PARENT     P.VSN      CHILD      C.VSN      NUM
-----------------------------------------------------
   1   M139  D  #1.0  +   INTS  D  #1.0  +   10
   2   M82E  D  #1.0  +   ANT2S D  #1.0  +   12
   3   MLS194 D  #1.0  +   ANT3S D  #1.0  +   16
   4   MLS194 D  #1.0  +   INTS  D  #1.0  +    9
   5   MS194 D  #1.0  +   ANT3S D  #1.0  +   16
   6   MS194 D  #1.0  +   INTS  D  #1.0  +    9
   7   M179  D  #1.0  +   INTS  D  #1.0  +   12
   8   M95   D  #1.0  +   ANT2S D  #1.0  +   10
   9   ML83  D  #1.0  +   ANT2S D  #1.0  +   24
  10   M445E D  #1.0  +   NAT4S D  #1.0  +   10
  11   M49   D  #1.0  +   ANT3S D  #1.0  +    9
  12   M45E  D  #1.0  +   NAT4S D  #1.0  +   10
```

図14．回路図と構成要素回路図の一覧

6．おわりに

本文では，リレーショナル・データベースを拡張したＡＤＡＭと称するデータベースのバージョン管理機能とその実現結果について述べている。設計オブジェクトは，一般に，設計やテストを繰り返しながら作られてゆく。この過程で多くの設計バージョンが作られる。設計バージョンの導出履歴は木構造を成すために，従来の事務用データベースには見られなかったバージョン管理機能が必要となる。

これまでに，設計やテストが完了した時刻に基づいて，設計対象物の妥当性を管理するアプローチが研究されてきた〔１〕。本文では，従来提案されていた２時刻印方式に問題があることを述べるとともに，問題点を解決するための３時刻印方式を提案した。これらの議論に基づいて，ＡＤＡＭのデータ・ディレクトリを設計し，その実現結果を示した。このデータ・ディレクトリは，設計オブジェクトの構成要素，生成・検証時刻などを管理しており，類似したオブジェクトの検索にも役立つことを確認した。

謝辞

本研究を遂行するに当たり，御指導いただきました当社情報電子研究所・首藤　勝 副所長，市川　照久 知識処理開発部長に感謝します。また，日頃より有益なコメントをいただきます関係者の方々，とりわけ，当社コンピュータ製作所・西川　正文主事，情報電子研究所・石川　洋氏に感謝します。

参考文献

[1] Batory,D.S. and Kim.W : Modeling Concepts for VLSI CAD Objects, ACM Trans. Database Syst. 10,3 (Sept 1985), 322-346.

[2] Eastman,C.M. : System facilities for CAD databases, Proc. 17th ACM/IEEE Design Automation Conference, 1980, pp.50-56.

[3] Haskin,R. and Lorie,R. : On Extending the Functions of a Relational Database System, Proc. ACM SIGMOD 1982, ACM, New York, 207-212.

[4] Katz,R.H. : Information Management for Engineering Design, (Surveys in Computer Science) Springer-Verlag, 1985.

[5] Katz,R.H., Chang,E. and Bhateya,R. : Version modeling concepts for computer-aided design databases, Proc. ACM SIGMOD Intel. Conf. on Management of Data, 1986, pp.379-386.

[6] LSI Logic's big bag of ASIC design tools, Electronics, Feburary 5, 1987, p.59-61.

[7] Tool Set Links All Stages of Full-Custom IC Design, Electronics (July 1,1985), 60-62.

[8] Tsichritzis,D.C. and Lochovsky,F.H. : Data models, Prentice-Hall,Inc. 1982.

[9] Woelk,D., Kim,W. and Luther,W. : An object-oriented approach to multimedia databases, Proc. Int. Conf. Manage. Data (SIGMOD), 1986, pp.311-325.

[10] Woo,C.C and Lochovsky,F.H. : Supporting distributed office problem solving in organizations, ACM Trans. Office Inf. Syst., 4,3 (July, 1986), pp.185-204.

〔11〕植村：データベース・システムの基礎，オーム社，1979.

〔12〕宇田川，他：関係演算による回路図の妥当性チェック，第３２回情処全大（1986年3月）

〔13〕加藤，他：パターン情報処理とマルチメディアデータベース，電総研彙報，Vol.51-4 (1987)

〔14〕川越，他：CAD/CAMへのマルチメディアデータベースの応用，情報処理，Vol.28-6 (1987)

〔15〕上林：マルチメディアデータベースの技術課題，情報処理，Vol.28-6 (1987)

〔16〕小島，他：拡張可能なデータモデルに基づくオブジェクト指向データベースの機能について，信学技報，DE87-5 (1987)

〔17〕近藤，他：オブジェクト指向データの特徴と記憶・利用方式，信学技報，DE87-1 (1987)

〔18〕田中，他：オブジェクト指向データベースにおける仮想化について，信学技報，DE87-1 (1987)

〔19〕穂鷹：データベース要論，共立出版，1978.

〔20〕増永：オブジェクト指向マルチメディアデータモデル，信学技報，DE87-4 (1987)

〔21〕中村：エンジニアリング・データベース，情報処理，Vol.25.No.4.pp.349-354 (1984).

〔22〕各種ＣＡＤシステムで実用化が始まったエンジニアリング・データベース，日経エレクトロニクス，1985年12月16日号，pp.143-160.

〔23〕設計合理化の鍵を握るＣＡＤデータベース，日経ＣＧ，1987年12月号，pp.10-22.