拡張リレーショナル・データベース ADAM のバージョン管理機能とその実現†

宇田川 佳久††

本文は，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．この過程で多くのバージョンが作られる．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．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メタ情報，例えば，構成要素名や生成・検証・引用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．また，データ・ディレクトリの性能を測定し，約 250 個の引用関係から特定の引用関係を検索するのに要する時間が，数ミリ秒であることを確かめた．

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[6),20)~22)]．

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．すなわち，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),6),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，応用プログラム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),18)]．

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も，事務用データベースと同様の効果を期待することができる．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） 設計オブジェクトのいろいろな表現を管理する機能

設計オブジェクトは，設計フェーズごとに異なる表現をもちうる．例えば，LSI の回路は，論理設計のフェーズでは論理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をもつ．設計データベースは，これらの表現を管理

† Design and Implementation of Version Control Mechanisms for an Extended Relational Database ADAM by YOSIHISA UDAGAWA (Information Systems & Electronics Dev. Lab., Mitsubishi Electric Corp.).

†† 三菱電機(株)情報電子研究所

する必要がある．

（9） 設計オブジェクトの階層構造を管理する機能

設計オブジェクトは，通常，複数のモジュールから構成されており，モジュールはサブ・モジュールから構成されている．このように，設計オブジェクトは階層構造を成すが，この階層構造も設計データベースの管理対象となる．

(10) 設計オブジェクトの変更履歴管理機能

設計オブジェクトは，改良を繰り返しながら逐次に作られてゆく．設計オブジェクトの変更履歴は，事務用データベースにおける変更履歴よりも複雑で，頻繁に使われるものである．

設計オブジェクトの階層構造や変更履歴の管理は，設計オブジェクトを単位として行うのが自然であるとの指摘がある[2)~4),8),10)]．このために，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が，“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”という意味での同意が得られている[5),8),9),13)~17),19)]．オブジェクト指向の概念をデータベースに導入することにより，設計オブジェクト単位の操作/管理を容易に行うことができるようになる．

本文で述べる ADAM (Advanced Database with Abstraction Mechanism) と称するデータベースは，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，およびデータ構造と操作演算を一体化しオブジェクトとして管理する機能を付加したもので，ディジタル回路図の管理に適用されている[12)]．

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．これまでに，設計オブジェクトの変更通知をするために2種類の時刻印を使う方法が提案されていた．第3章では，従来の方法に問題があることを述べるとともに，設計オブジェクトの変更通知をするためには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，それに性能について述べている．

2. ADAM データモデルの概要

2.1 オブジェクト指向

設計図をはじめとする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)]．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，階層中の設計オブジェクトを単位とすることが多い．
- 操作演算は，設計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設計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・データベースの一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は，本文

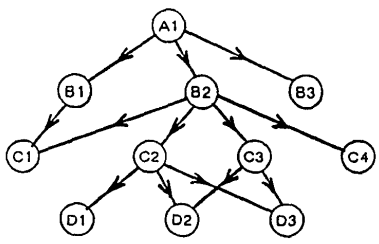

図 1 ADAM データモデルの概要
Fig. 1 Overview of ADAM data model.

第4章で論じられている.

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フェースの記述（仕様）と実現法の記述の対として管理することが有効であると言われている[1),4),12)]. 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”と定義している. つまり，機能あるいは使い方が同じであるが，作り方が違うものをバージョンと呼んでいる. 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うるので，設計バージョンの導出履歴は，一般に木構造となる[3),5)].

ADAM における設計オブジェクト管理機能は，次のとおり.

（A） 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 (Uninstantiated Object) の管理

図2は，ADAM のオブジェクト管理の概略を示している.

2.3 バージョン導出履歴管理の必要性

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まれで，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.

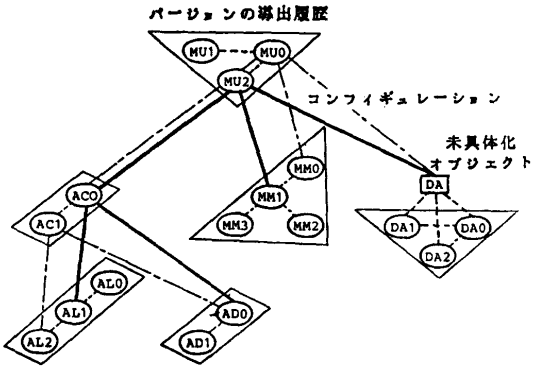

図 2 ADAM のオブジェクト管理の概要
Fig. 2 Overview of ADAM's object management.

（3） インタ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.

2.4 コンフィギュレーション管理の必要性

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）から組み立てられている. 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ジ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見た，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.

2.5 未具体化オブジェクト管理の必要性

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によって，CAD システム全体の性能を向上させることができる. 未具体化オブジェクトは，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. バージョンと未具体化オブジェクトの相違は次のとおり.

ADAM データモデルにおいて，設計オブジェクト

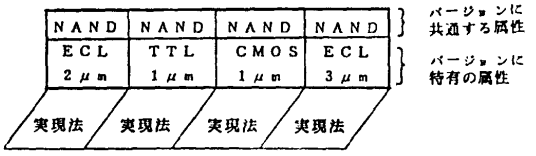

図 3 インタフェースの属性とバージョン
Fig. 3 Attributes and versions.

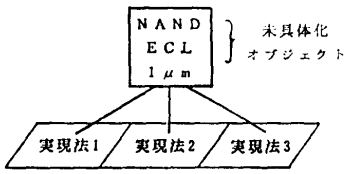

図 4 未具体化オブジェクトと実現法の記述
Fig. 4 Uninstantiated object and implementation description.

はインタフェースの記述と実現法の記述の対として扱われている．概念的には，バージョンはインタフェースを記述する特定の属性（例えば，機能）が同じ設計オブジェクトの集まりと考えることができる（図 3）．これに対し，未具体化オブジェクトは，図 4 に示したように，実現法が特定されていない設計オブジェクト（実現的にはインタフェースの記述）である．未具体化オブジェクトに対応する実現法は，データベースが置かれている計算機とは異なる計算機で作成されていても良い．未具体化オブジェクトは，実現法を特定することによってバージョンと同等の機能を有する設計オブジェクトとなるが，これは要求仕様を満たす実現法を“リリース”することに相当する．

3. 設計オブジェクトの変更通知と時刻印方式

3.1 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おいても，時間的な変更過程におい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．したがって，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，しばしば起こりうる．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）と言う．

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・サイクルをたどる．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．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，一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えないからである．

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一つとなっている．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．

3.2 2時刻印方式による変更通知

Batory ら[1]および Chou ら[3]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. CNT<V. CAT であるとき，バージョンVは設計変更された後に，その変更が受諾（検証）されていることから，実現無矛盾（Implementation Consistent）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集合をJとする．条件

$[\forall X \in J]$ (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

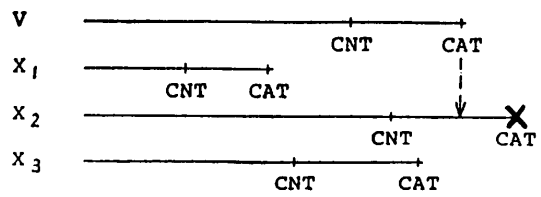

図 5 2時刻印方式の例
Fig. 5 Example of 2-timestamps technique.

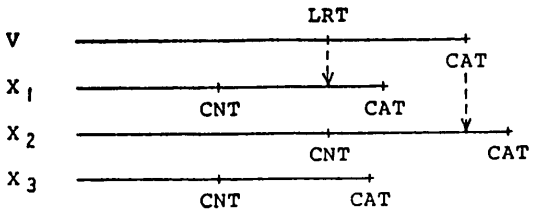

図 6 局所矛盾である例
Fig. 6 Example of local inconsistency.

トが変更されていないことを保証しているものと解釈することができる．また，実現無矛盾であり，参照無矛盾であるとき，すなわち，

(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) であると定義されている．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5**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5の X_2 のようにバージョンの変更後に受諾されない（テストに合格しない）バージョンがあると，バージョンVは妥当でないバージョンを含むものとなり，バージョン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，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．

3.3 3時刻印方式と局所無矛盾

総合無矛盾における問題点を改善するために，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 を導入した．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，次の三つの場合に限られる．

（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6に，局所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．また，図5に示した時刻印をもつバージョンVは局所矛盾となり，このバージョンを受諾することには警告を発するべきであることがわかる．

バージョンが，実現無矛盾，参照無矛盾，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．

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2章，第3章で論じた設計オブジェクトの管理機能は，**図7**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 DDD という名前のリレーションと設計オブジェクトの階層構造を表す CONFIGURATION という名前のリレーションで構成されている．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

バージョンを管理するためには，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

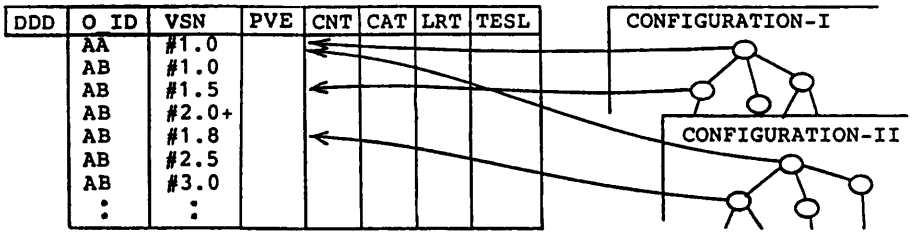

図 7 データ・ディレクトリの概要
Fig. 7 Overview of data directory.

$DPRI(1,30)

NO	O-ID		VSN		PVE	CNT	CAT	LRT	TES
1	PSYS	D	#1.0	+	0	91987/135129	92087/104307	0/ 0	1
2	PSUPP	D	#1.0	+	0	92087/141011	92087/152206	92087/153206	1
3	MSYSA		#1.0	+	0	92087/153305	92087/153032	92087/154116	0
4	MSYSB		#1.0	+	0	92187/ 94008	92187/100152	92187/151708	0
5	P		#1.0	+	0	92087/133157	92087/134009	92087/212251	0
6	PSYS	D	#2.0		1	90987/194449	91087/ 61130	0/ 0	1
7	PSYS	D	#3.0		1	90987/194122	90987/194123	0/ 0	1
8	PSYS	D	#2.4		6	90987/194249	90887/201040	0/ 0	1
9	E2		#1.0	+	0	91887/134048	91987/134040	91987/134311	0
10	DY		#1.0	+	0	91887/140005	91987/133827	91987/134311	0
11	B		#1.0	+	0	90987/140103	91087/104411	91987/134311	0
12	S		#1.0	+	0	91087/ 92311	91087/165809	91987/134311	0
13	E4		#1.0	+	0	91087/ 93402	91087/165524	91987/134311	0
14	PSYS	D	#4.0		0	91987/134345	91987/134358	0/ 0	1
15	PSYS	D	#4.2		14	91987/134421	92187/160108	0/ 0	1
16	MOTOR		*		0	93087/111207	0/ 0	0/ 0	0

図 8 リレーション DDD の例
Fig. 8 Example of relation DDD.

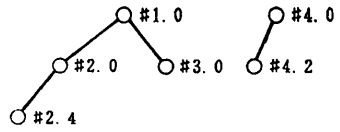

図 9 オブジェクト PSYS のバージョン導出木
Fig. 9 Version derivation tree of object PSYS.

ばならない．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，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，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．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．このようなバージョンの 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．

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．一つのバージョンからは任意の数のバージョンが導出されるが，ある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 フィールドを用意する．図8に示したリレーション DDD は，図9に示したバージョン導出木を含んでいる．Katz ら[5]は，バージョン導出木を一つのルート・ノードをも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9に示したように，複数のルート・ノードをもつ木構造となる．ADAM のバージョン管理の特徴の一つは，複数のルート・ノードをもつ木構造（森構造）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．現在有効バージョンは，任意の時点で，それぞれのバージョンに対してただ一つだけ存在する．リレーション DDD のバージョン

識別名の第8文字目は，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，バージョンが，他のどんな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．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．このような特質のために，コンフィギュレーションに関する情報はリレーション DDD と独立したデータ構造によって管理されている．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ほかに，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．

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．

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（図8）．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 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 にも，同上の時刻が書き込まれる．

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．さらに，コンフィギュレーションに関するデータを検索するのに要する時間の測定結果を示す．

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） 時刻印の妥当性のチェック

以上の機能をインプリメントするのに，FORTRAN 言語で約4,000行を要している．**表1**は，(1)～(5)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．(1)(2)(4)の機能は，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，またはふもとまでたどる機能（$ 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．(5)をインプリメントするためには，オブジェクトを構成する要素オブジェクトを検索する機能（$ REF）が必要となる．

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

表 1 データ・ディレクトリの基本操作コマンド
Table 1 Primitive commands of data directory.

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
Configuration の削除	$ERACF /1
Config. 情報の入力	$CFREAD /1
Config. 情報の出力	$CFWRTE /1
バージョンの具体化	$INST /2
バージョンの未具体化	$DINST /1
実現無矛盾のチェック	$TCIMP /2
参照無矛盾のチェック	$TCREF /2
総合無矛盾のチェック	$TCTOT /2
局所無矛盾のチェック	$TCLOC /2

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 10 回路図と構成要素回路図の一覧
Fig. 10 List of component objects.

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 (NAT 2) を構成要素としている回路図を検索した結果を**図 10** 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 11** は，回路図名が "M 183"，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

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 11 回路図と構成要素回路図の一覧
Fig. 11 List of component objects.

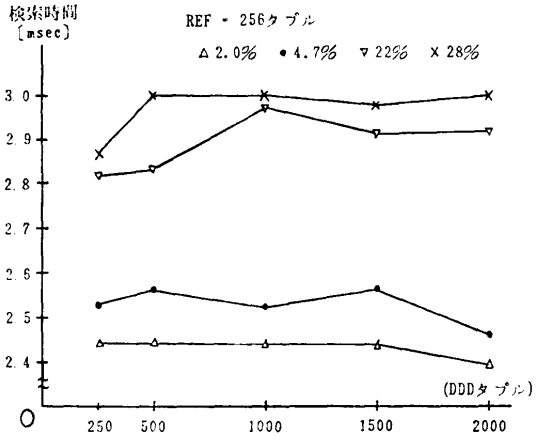

図 12 データ・ディレクトリの性能の一例 COSMO 900 II での測定値
Fig. 12 Performance of data directory (On COSMO 900 II computer system).

良い．

5.4 データ・ディレクトリの性能

図 12 は，コンフィギュレーションに関するタプルを検索するのに要する時間の測定結果を示している．この測定では，コンフィギュレーションのタプルを 256 個に固定し，DDD に関するデータを 250, 500, 1,000, 1,500, 2,000 に変化させている．図 7 に示したように，コンフィギュレーションから DDD には直接ポインタが付けられている．したがって，コンフィギュレーションに関するデータ検索は DDD のタプル数には依存しないものと考えられる．図 12 に示した測定値は，このことを実証している．なお，検索時間は検索結果のタプル数に比例して増加することも判明した．

6. おわりに

本文では，設計図面をはじめとする設計オブジェクトのバージョン（代替案）を管理する機能について論じた．バージョンを管理するための情報は，次のとおり．

（1） バージョンが生成，検証，引用された時刻
（2） バージョンの導出履歴

(3) バージョン間の階層情報(コンフィギュレーション)

これらの情報はバージョンのメタ情報と位置付けることができるものであり，データ・ディレクトリによる管理が適当であることを論じた．本文で述べたデータ・ディレクトリは，(1)，(2)の情報を管理するためのリレーションと(3)の情報を扱うグラフ構造を組み合わせた構成をしている．このデータ・ディレクトリはバージョン間の階層情報を高速に検索することができ(通常数ミリ秒)，類似した設計オブジェクトを検索するのにも役立つことが確認された．

今後の課題として，バージョン管理のための視覚的インタフェースや，バージョンの同時アクセスを可能にする機能を開発することがある．

謝辞 日頃御検討いただく，三菱電機(株)情報電子研究所の皆様に感謝します．また，適切な御指摘をいただいた査読者の方々に感謝します．

参考文献

1) Batory, D. S. and Kim, W.: Modeling Concepts for VLSI CAD Objects, *ACM Trans. Database Syst.*, Vol. 10, No. 3, pp. 322-346 (Sept. 1985).
2) Eastman, C. M.: System Facilities for CAD Databases, *Proc. 17th ACM/IEEE Design Automation Conference*, pp. 50-56 (1980).
3) Chou, H. T. and Kim, W.: A Unifying Framework for Version Control in a CAD Environment, *Proc. 12th Int. Conf. on Very Large Databases*, pp. 336-344 (Aug. 1986).
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 Int. Conf. on Management of Data*, pp. 379-386 (1986).
6) Tool Set Links All Stages of Full-Custom IC Design, *Electronics*, pp. 60-62 (July 1, 1985).
7) LSI Logic's Big Bag of ASIC Design Tools, *Electronics*, pp. 59-61 (Feb. 5, 1987).
8) Spooner, L. D., Milicia, A. M. and Faatz, B. D.: Modeling Mechanical CAD Data with Data Abstraction and Object-Oriented Techniques, *Proc. IEEE Int. Conf. on Data Engineering*, pp. 416-424 (Feb. 1986).
9) Woelk, D., Kim, W. and Luther, W.: An Object-Oriented Approach to Multimedia Databases, *Proc. Int. Conf. Manage. Data* (*SIGMOD*), pp. 311-325 (1986).
10) Wiederhold, G.: Views, Objects and Databases, *IEEE Computer*, Vol. 19, No. 12, pp. 37-44 (Dec. 1986).
11) 植村：データベース・システムの基礎，オーム社，東京 (1979).
12) 宇田川ほか：設計データベースのバージョン管理とその実現，情報処理学会データベース・システム研究会資料，63-2 (1988).
13) 加藤ほか：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, No. 4, pp. 26-44 (1987).
14) 川越ほか：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, No. 6, pp. 730-739 (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, No. 6, pp. 784-791 (1987).
16) 津田ほか：オブジェクト指向データモデル MORE，情報処理学会データベース・システム研究会資料，63-5 (1988).
17) 田中ほか：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，DE 87-1 (1987).
18) 穂鷹：データベース要論，共立出版，東京 (1978).
19) 増永：マルチメディアデータモデル OMEGA におけるオブジェクト指向概念と操作，情報処理学会データベース・システム研究会資料，63-7 (1988).
20) 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).
21) 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.
22) 設計合理化の鍵を握る CAD データベース，日経コンピュータ・グラフィックス，1987年12月号，pp. 10-22.

(昭和63年3月25日受付)
(平成元年3月7日採録)

宇田川佳久 (正会員)

1954年生．1982年東京大学大学院博士課程修了．同年三菱電機(株)入社．現在，同社情報電子研究所主事．エンジニアリング・データベースなどの研究開発に従事．工学博士．電子情報通信学会，人工知能学会各会員．