SDSP - System Design/Development Standard Procedure

椿 正明
千代田化工建設（株）

1．SDSPの目的

ソフトウェアの大型化とともに、良いソフトウェアをいかに効率的に開発するかが重大な問題になってきている。DBMSなどの強力なツールが与えられても、これをいかに位置づけ使いこなしてゆくべきか、なお解決すべき問題は多い。SDSPは大規模なアプリケーションシステムの設計、開発に関する技術の標準化、パッケージ化をねらうものである。したがってMBA社のPRIDE[5]、IBM社のBSPおよびIPT[3]（HIPO, SP etc）などと同様の、いわばメタソフトウェアとも言うべきものである。ソフトウェア生産の全工程を対象とするので、通常論じられることの多いデータベース設計よりも広範なものと言えよう。

このような設計、開発の標準化は当然製品ソフトウェアの標準化や品質保証をもたらすべきものであるが逆に製品の標準化を可能にする技術的素地の上に成り立つものである。SDSPは当社で開発されたDBMS、DPLS[4][5][6]（Database, Program base, Language base, Support）を適用するためのユーザーシステム計画、設計、開発担当者――教育の過程から生れたものであるが、ここでは汎用性、独立性の高いデータベースによるデータ構造の標準化、ダイナミックリンク機能によって実現されたプログラムベースによるプログラム構造の標準化、ランゲージベースによるユーザ言語の標準化がその素地を用意した。SDSPは決してDPLS環境を前提とするものではないが、データ中心のアーキテクチャを志向するので、データ独立性の高いデータベース環境が好都合ではある。

DBMSとは単に強力なデータマネジメントツール以上のものである。なぜならソフトウェアのアーキテクチャをプログラム中心から180度転換してデータ中心に変革させる必然性を持っているからである。しかしこの大きな変革を大多数のアプリケーションシステム設計者が納得しこれのものとしてゆくには、なおしばらくの時日を必要とするようである。実際供給されたDBMSの不備によるところも多いであろうが、すでに開発されたデータベースシステムには、規模は大きくとも、相変らずプログラム中心の設計思想によるものが多いように思われる。新しい酒は新しい革袋に入れられなければならないというわけで、DBMSを使用するからには、データ中心の設計手法を確立しなければなるまい。データ中心とは現実世界（の情報）中心ということであり、グローバルに情報を扱う立場からすれば本来あるべき姿でもあろう。SDSPはこのような本来あるべきソフトウェアの設計・開発技術を志向するものである。

2．SDSPの特徴

SDSPの特徴――必らずしもSDSPに固有という意味ではない――は次の7項目にまとめられる。

(1) ドキュメントの段階的出力にシステム開発は内容の規定されたドキュメントを9段階にわたって出力することによって完成する。

(2) ユーザによる明確な要求定義に要求仕様は単純かつあいまいさの残らない、定まった文法（あるいは画法）をもつDesign Languageによって記

述する。これはたとえほとんどシステム設計者が作成するとしてもユーザ部門の責任で発行される。

(3) 逐次開発：システムのスコープをあらかじめ見定め一発開発的に開発するのでなく，とりあえず運用すべき部分だけ設計・開発し，この繰返しにより大規模システムを構築してゆく。どんなシステム設計の達人でもそのスコープは読みきれるものではない。だから既開発部分が追加変更に影響されないように設計しておこうという立場をとる。

(4) データ中心のアーキテクチャ：データは現実世界の表現であって公共の資源であり，特定のプログラムに従属専有されるべきものではない。したがってデータの設計がシステム設計の中心テーマとなる。

(5) Conceptual Approach：データの設計にあたっては処理手順（プログラム）に影響されない，データ自身の特性によって決定されるConceptual Model（スキーマ）の確立をめざす。より物理的なデータ構造は2次的な問題と考える。

(6) DD/Dの活用：設計されたデータの構造はデータベースに記述されるのでDD/Dサポートが受けられるが，さらにプログラム，ユーザ，ハードウェアを包含するシステム環境すべてをDD/Dの対象とし，ソフトウェアの設計開発に関するCADを実現する。

(7) データ管理フレーム：データ中心のアーキテクチャから必然的にデータの生成，更新，保守あるいはライフサイクルに応じてサブシステムが形成される。すなわちデータ管理によってサブシステムのフレームが決められる。

3. 各論

3.1 9段階の出力ドキュメント

9段階にわたって出力されるドキュメントは図-1のとうりである。ここではソースプログラムもソースデータも出力ドキュメントの1つと見なされる。トップダウンを志向し，ユーザースマニアルはソースプログラムはおろか，システム詳細設計以前に決定されるべきとしている。システム計画担当者はユーザ部門の代表であり，システム開発におけるユーザの参加と責任を重視している。

出力ドキュメント	発行責任者
(1) システム計画仕様書	システム計画担当者
(2) システム設計仕様書	システム設計者
(3) ユーザースマニアル	システム計画担当者
(4) システム詳細設計仕様書	プログラマ
(5) データ管理マニアル	データベース管理者
(6) ソースプログラム	プログラマ
(7) ソースデータ	システム計画担当者
(8) システムテスト報告書	プログラマ
(9) システム監査報告書	システム計画担当者

図-1 出力ドキュメントと発行責任者

各ドキュメントの内容はそれぞれ決められており，図-2に(1)，(2)の例を示す。

3.2 情報処理フローチャート

ユーザによる明確な要求定義はシステム計画仕様書に記載されるが，その中心となるものが情報処理フローチャートである。情報処理フローチャートは誰が如何なる情報処理を何時行うか，そのときの入出力情報はどこから来てどこへ行く

(ステップ1) システム計画仕様書		
(発行元ユーザセクション)		(発行年月日)
(責任者)	(主担当者)	(希望納期)
(システムの名称)		

項目番号	項目	頁	備
(1-1)	システムの目的		
(1-2)	情報処理フローチャート 1)		
(1-3)	蓄積情報		
(1-4)	指令形式(例示) 2)		
(1-5)	適用範囲		
(1-6)	関連システム 3)		
(1-7)	関連法規類準		
(1-8)	使用すべきデータ源		
(1-9)	使用すべき処理モデル源		

(ステップ2) システム設計仕様書		
(システム設計セクション)		(発行年月日)
(責任者)	(主担当者)	(Job No. & JCL Catalog No.)
(システムの名称)		

項目番号	項目	頁	備
(2-1)	スキーマダイヤグラム 1)		
(2-2)	属性仕様 2)		
(2-3)	指令形式(定式化) 3)		
(2-4)	レポートリスト 4)		
(2-5)	サブプログラムリスト 5)		
(2-6)	適用業務用サブプログラムリスト 6)		
(2-7)	工程表		
(2-8)	予想マンアワー		
(2-9)	その他		

図-2 出力ドキュメントの内容(例)

かを示すものである。これはブロック図やいわゆるポンチ画のあいまいさ、文章による表現の不統一、難解さ、整合性チェックの難しさ、試行錯誤にともなう書き直しの煩雑さを解決するために考案されたチャート記法である。一種のDesign Languageであり、次のような簡単な文法(画法)をもつ。

(1) あるIntelligence(人、セクション、コンピュータシステム)に記憶される情報の状態を横線 ——→ (人)、═══→ (コンピュータシステム)で表わす。

(2) 情報処理アクティビティを縦線 ↑ で表わす。

(3) 生成された情報とその媒体により次のように表わす。

コンピュータへの入力 (現実世界の情報)　(制御情報)
人への入力 (コンピュータの出力)　C (チェックやコメント)
(人の出力)
コンピュータ内入出力

(4) 情報の加工を伴わない移動を ——→ ↑ ↓ などで表わす。ただしIntelligence Aの情報の増減をもたらさない入力を A ——→ のように斜めに入力させ、またAのチェックを必要とするとき A ——C——→ のように書く。

(例) 部Aは部Bからレポートrを受けとり、これにもとずいて入力データ r を作成しDB(1)に投入する。次にこれにもとずいてDB(2)の情報を参照しつつ処理Pを行なわせ結果をDB(1)に記憶する。次に入力 r と処理Pの結果を出力データ R として出力させこれをチェックして部Bに送付する。こ

れを情報処理フローチャートによって書くと図-3のようになる。

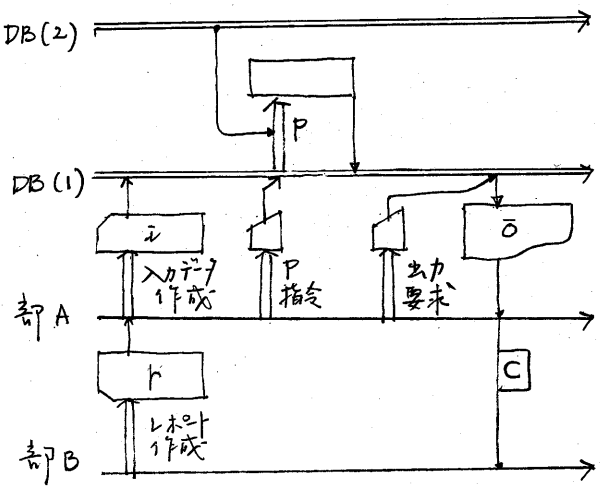


図-3. 情報処理フローチャート例

システム計画仕様書はシステム設計・開発の指針を定めるものであり、ユーザの要求を疑義なくフォーマルに記述したものでなければならない。これをあいまいにしたままシステム設計に入ることがエラーや手直しの原因になっていることはしばしば指摘されるところである。近頃Requirement Engineeringが話題となっているゆえんでもあろう。実際ユーザとシステム設計者は異った世界に住んでおり、同じ言葉を違った意味に用いていることも多く、同床異夢とならないよう、ユーザとシステム設計者が疑義なく効率よく通じ合えるDesign Languageが必要である。情報処理フローチャートはその1つのアプローチと考えられるであろう。

3.3 データ中心のアーキテクチャ

プログラムAはAをシステムの中心と考え、プログラムBはBをシステムの中心と考える。ではAとBを統合したシステムではA, Bいずれが中心となるのか。これがシステムの大型化、統合化につれて顕著になって来たプログラム中心のアーキテクチャの矛盾である。プログラム設計の考え方が(1)プログラム中心 (2)プログラムとデータの分離 (3)データの統合と構造化、プログラムの分割と構造化 (4)データ中心 (5)データ独立、と発達してきたのもこの矛盾を解消するためであろう。赤ん坊はまず自己中心的世界観を持ち、成長するとともに社会を中心としてそこに自らを位置づけるようになるが、プログラマの分身であるプログラムもプログラム中心から、現実世界を表わすデータ中心に成長してゆくもののようである。データは1つの大規模な統合体となるのでデータの設計が重要になるが、プログラムは分割され小規模となるので、プログラムの設計はデータ中心のアーキテクチャにおいてむしろ易しくなるもののようである。

3.4 データのConceptual Model

データ中心のアーキテクチャは必然的にデータとプログラムの独立性を要求するため、プログラムからのConceptual Viewすなわちデータの Conceptual Modelが重要な課題となる。これは物理デバイスの制約から独立しているばかりでなく特定の応用プログラムのもつLocal Viewからも独立した最も論理的でグローバルなモデルである。このような論理的なモデルを用いるとき、スペースを与えるための物理的モデルは不可欠であるから、必然的にデータモデルはSenko[75], Sundgren[80]、石田[9]らの主張するような多段階モデルとならざるを得ない。勿論ユーザあるいは応用プログラムからはConceptual Model(の1部)だけが見えており、より物理的なモデルはTransparentになる。

このようなConceptual Modelの素材となる概念として、SDSPではInforma-

tion Algebra[10]の提案したEntity, Attribute, Valueおよび分類のためのEntity Typeという概念を用いる。近年データベースのセマンティクスやモデリングの研究が進むにつれ、この考え方は最も理論的かつ実用的なものとして注目されつつあり、Moulin[11]らはこのモデルがRDB, CODASYL, ANSI-SPARCにおけるConceptual Modelとして注置づけられることを述べている。

Conceptual Modelは本来データ自身のもつ特性の解析によって決められるべきものであり、それをどう処理するか(How)すなわちプログラムとは独立に——どんなデータを扱うか(what)は勿論プログラムから決ってくるが——決まるものである。したがってConceptual Modelにおいては原則として効率の議論は発生しない。CODASYLやIMSのモデリングにおいて効率の議論が入ってくるのは、そのデータモデルが完全に論理的でないことの何よりの証拠であろう。効率の議論はシステムのスコープを決め応用プログラムを想定してはじめて成立する。したがってこれに左右されるデータモデルはシステムのスコープの変化に応じきれない一発開発志向を強要するモデルといえよう。効率の議論は応用プログラムに見えない論理的なモデルにおいて考慮すれば足りると思われる。

Conceptual Modelは現実世界についての共通のViewを規定するものであり、以後応用プログラムは、これによってデータにアクセスするので十分広い視野をもって正しく設計されなければならない。1,2個の特定プログラムのLocal Viewに左右されると正しい情報構造を見落すことになり易いので、できるだけ異なった3個以上の応用プログラムの立場から検証するのが望ましい。

Conceptual Modelし設計の結果は、システム計画仕様書における蓄積情報(図-2参照)すなわち洗い出されたAttributeの集合であり、その結果はスキーマダイヤグラムおよび属性仕様である(図-2参照)。スキーマダイヤグラムはEntity Type(同種Entityの集合)とこの間の関係を表示する。属性仕様は各Entity Typeに属する個々の属性の属性(属性名, バイト長, 属性タイプ, データタイプetc)を規定する。各Entity Typeに属する属性はいつまでも追加の可能性が残っているから、スキーマダイヤグラムが決定されればConceptual Modelはほぼ人と設計されたと見なしてもよい。したがってConceptual Modelの設計とは、「Attributeの集合をEntity Typeごとに分類し、Entity Type間の関係を図示すること」と言っても大きな誤りはない。

さてここでAttributeの集合をどんな基準でEntity Typeに分類すればよいのであろうか?それは人によらず唯一の解を持つのであろうか?これは現在のところ——将来もそうかもしれないが——設計者のセマンティクスあるいはデータ観に依存しており、唯一の解を持つ保証はない。しかし多くの場合同一の解を得るあるいはその解について同意を得ることができるようである。Mijares[12]はこれに関しいくつかのアルゴリズムを提案しているが筆者も以下にやゝセマンティクスに依存するが実用的なアルゴリズムを提案しよう。RDBと関連して議論されて来た正規型の問題はEntity Typeの同定と深い関係を持つようであり、とくにFagin[13]による第4次正規型との関係は興味深い。正規型の議論はセマンティクスを含まない厳密な議論であるが、その出発点である第1次正規型の創成にはセマンティクスが必要であろうから、厳密な議論にも限界がある。

SDSPのConceptual Modelの基本概念はEntity, Attribute, Valueであって、しばしば見られるように、Entity間の関係を表わすためのAssociationあるいはRelationshipをこれらに加えるものではない。すなわちAssociationをもつ

Entityとして扱う。したがってn:mのネットワーク対応関係は2項関係をキー(Identifier)とするEntity Typeによって表現される。なお1:mのトリー対応関係は子Entityに親Entityを記述する属性を配置して表現するのが普通である。

最も一般的なEntityの対応関係は、関係するEntity Typeの数とトリー/ネットワークの別により生ずる図-4の4種であろう。それぞれについてオカレンス表現およびスキーマダイヤグラム表現を示す。

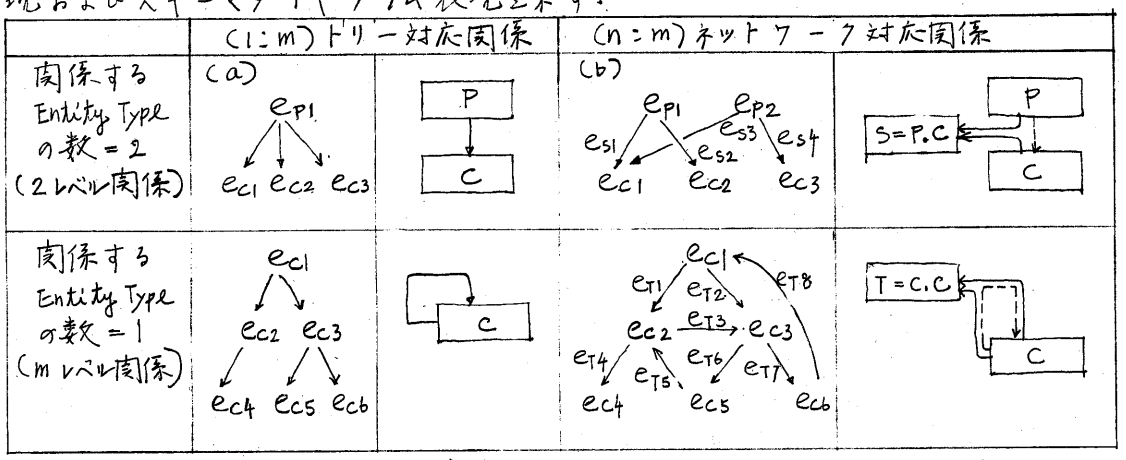
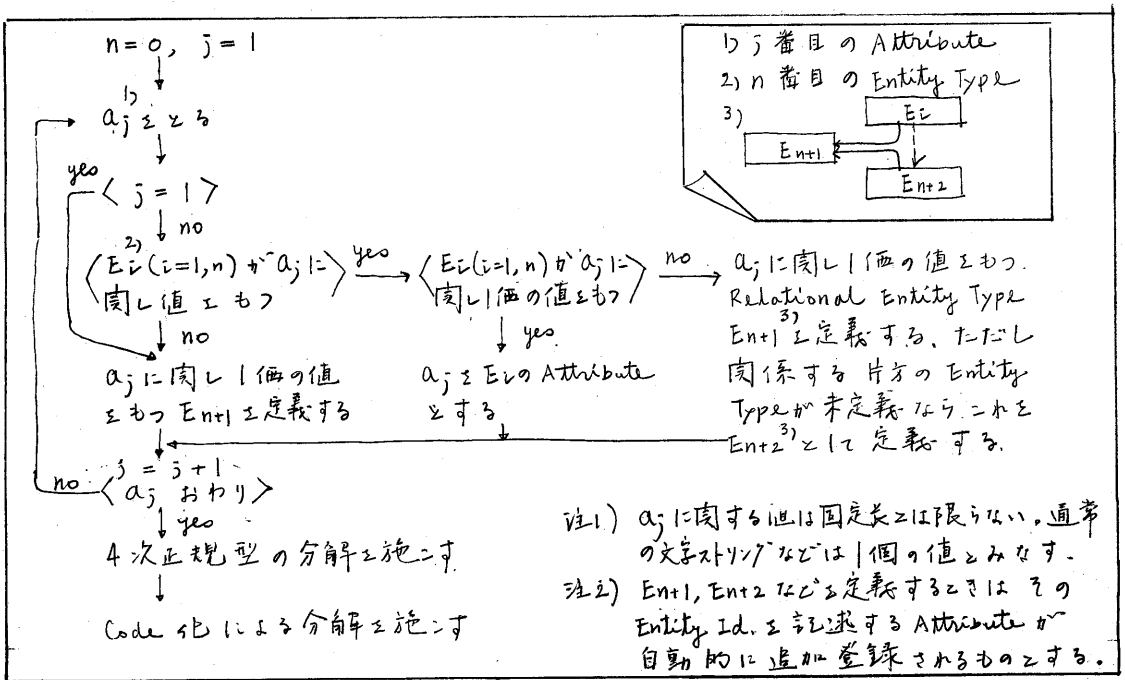


図-4 代表的なEntityの対応関係

スキーマダイヤグラムにおける箱はEntity Typeを表わしている。図-4に見られるようにネットワーク対応関係の場合は関係を表現するためのEntity Type——図-4のSやT, Relational Entity Typeと呼ぶ——を設ける。以上を考慮してSDSPでは図-5のようなEntity Type同定のアルゴリズムを提案している。



注1) ajに関する値は固定長とは限らない。通常の文字ストリングなどは1個の値とみなす。

注2) En+1, En+2などを定義するときはそのEntity Id.を記述するAttributeが自動的に追加登録されるものとする。

図-5 Entity Type同定のアルゴリズム

スキーマダイヤグラムはConceptual Modelの表現手段としてきわめてすぐれたものと思われる。物理的なファイルフォーマットに慣れたユーザにはとっつきにくいきらいがあるが、それはConceptual Modelの抽象性によるものであり、データの基本設計におけるDesign Languageとして決定的なものと思われる。

3.5 DD/D

ソフトウェア開発におけるDD/Dの重要性は今後ますます高まってゆくであろう。データベース化によってデータの設計がシステム設計の主要部分を占めることもその一因であるが、単にデータに限らず、プログラム、ユーザー、デバイス端末等のハードウェアを含む全システム環境がDD/Dの対象となってくるからである。あらゆる応用分野にデータベースが応用されつつあるとき ソフトウェアの設計、開発やコンピュータシステムの運用が例外であるはずはあるまい。元来DD/D情報のかなりの部分が、バラバラにではあるがすでにOSによって扱かわれており、これとDDLの情報とによって DD/D情報の大部分は改めて入力することなく自動的に収集できるはずである。理想的には、DD/D情報にもとずいて稼動するいわばDD/D Machine すら考えられてよいであろう。

実際PRIDEではDD/Dがソフトウェア開発ツールの主柱と成しており、そのためのプログラムとしてPRIDE LOGIKが用意されている。SDSPではDPLSのDD/D機能を用いるが、すでに (1) Attribute (Conceptual Model) (2) サブファイル (3) データベース (4) ユーザとパスワード (5) ロードモジュール (6) サブプログラム (7) ソースステートメント (8) ソースデータ (9) コマンド (10) コマンドマクロ (11) データセット (12) プログラマ などに関する情報を扱っている。現在はDPLSおよびこのもとに開発された応用システムの情報のみを管理しているが、これ以外のシステムのリソース管理、ソフトウェア開発に利用することに何の障害もない。実際DD/D情報の1つであるプログラム間のCALL-CALLEDの関係はDPLSのVersion-upにおいてプログラムのインタフェースを変更する際、その計画や作業において非常に有効に利用されてきた。

3.6 データ管理フレーム

SDSPではサブシステムへの分割にデータ管理フレームすなわちデータ管理の観点を導入する。これはプログラム中心からデータ中心へのシステムアーキテクチャの変化に呼応するものである。データ管理フレームでは、同じ管理者によって、同じ管理プログラムによって、同じタイミングで登録、更新される一群のデータがサブシステムを定める。したがって一年に一回程度改訂される、標準部の管理する標準データは、これを管理するプログラムとともに1つのサブシステムを形成するであろう。3～5年のライフを持つプロジェクトの設計、管理データを収容するプロジェクトデータベースはそれぞれ1つのサブシステムの中心となるであろう。DD/D情報を管理するDD/Dシステムも1つのサブシステムとなる。

もちろんこのようなデータ管理フレームのほかに応用分野の固有技術にもとずく分割が必要となることがあろう。この場合サブシステムは同じ応用分野のプログラム群と対応づけられるであろう。データとプログラムの分離が徹底するならば、データ管理フレームによってデータのサブシステムが、プログラム管理フレームによってプログラムのサブシステムがそれぞれ定義されることもあり得よう。

なおサブシステムへ分割する基準として、PRIDEではタイムフレーム、BSPで

はデータクラスにもとずくビジネスプロセスを提案している。これらの基準とデータ管理フレームとの関係は明らかにされていないが、ビジネスプロセスによる分割は図-6のような情報システムネットワークを前提としており、ある程度大規模なレベルではプログラム中心のアーキテクチャとなっているのが特徴的である。

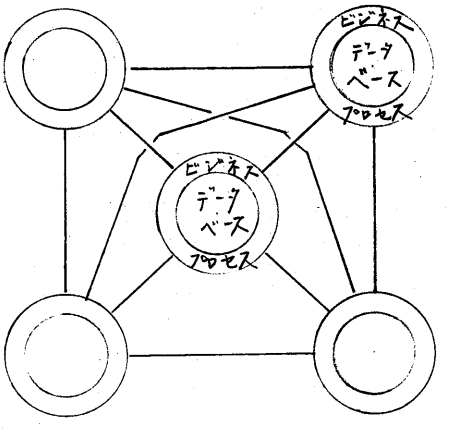


図-6 BSPの情報システムネットワーク

4. 結び

ソフトウェアの開発，保守，運用を効率化するために高級言語，モジュール化，SP，データベース，DD/D，フォームレート，Design Language など種々の工夫が施されて来たが SDSP はこれらを体系化してソフトウェア設計，開発，運用の各フェーズで効果的に適用してゆこうとするものである。とくに強調していることは、次の2点である。

(1) 設計とはフォーマルなドキュメントを出力することである
(2) データ中心のアーキテクチャを構成する

SDSPは今後とも種々の応用を通じて改良され、データベースおよびDD/Dを中心として発展してゆくものと思われる。

引用文献

1) 日本能率協会：システム開発プロセスの改善，EDP リサーチレポート 1975年3月1日号
2) IBM：Business Systems Planning, Information Systems Planning Guide
3) IBM：Improved Programming Technologies：Management Overview
4) Tsubaki, M.：DPLS－Database, Dynamic Program Control & Open-Ended POL Support, Proc. of Workshop on Data Bases for Interactive Design, Waterloo, Ontario (1975)
5) Tsubaki, M.：Multi-Level Data Model in DPLS－Database, Dynamic Program Control & Open-Ended POL Support, Presented at International Conference on Very Large Databases, Frammingham, Mass.
6) 椿正明：プログラム管理およびユーザ言語管理機能を補強したDBMS ――DPLS，情報処理 Vol 17 No10, pp921 (1976)
7) Senko, M.E, et al：Data Structures and Accessing in Database Systems IBM Systems Journal, Vol12, No1 pp30 (1973)
8) Sundgren, B：Conceptual Foundation of the Infological Approach to Databases. Proceeding of IFIP TC2 Working Conference (1974)
9) 石田喬也：データベースのセマンティクスとインプリメンテーション，情報処理学会，データベース研究会資料 74-10 (1974)

10) CODASYL Development Committee: An Information Algebra-Phase I Report. ACM, Vol. 5, No. 4, pp 190 (1962)
11) Moulin, P et al: Conceptual Model as a Data Base Design Tool, Modelling in Data Base Management Systems. (North Holland 1976)
12) Mijares, I & Peebles, R.: A Methodology for the Design of Logical Data Base Structures (North Holland 1976)
13). Fagin, R: Multivalued Dependencies and a New Normal Form for Relational Databases. (To appear in TODS)