Vol. 30 No. 9 情報処理学会論文誌 Sep. 1989

# ソフトウェアエンジニアリングデータベース SEDB/OKBL のデータモデルについて†

松本吉弘††

ソフトウェアエンジニアリングを支援する計算機環境で用いられるデータベースにおいて，オブジェクトを基底とし，オブジェクト間の意味関係を表現できるデータモデルのひとつを提案し，このデータモデルを基にした実験的ソフトウェアエンジニアリングデータベース SEDB/OKBL のデータ定義，操作などの方法を示すことによって，その有用性を論じる．

## 1. まえがき

ある目的をもったソフトウェアに関して，そのライフサイクルにわたって，ソフトウェア技術者の遂行するエンジニアリング活動のことを，ソフトウェアエンジニアリングとよぶ．ソフトウェアエンジニアリングを支援する計算機環境（computer-aided software engineering environment：以下 CASE と略称する）に関して，多くのアプローチがなされている．ソフトウェアエンジニアリングデータベース（software engineering database：以下 SEDB と略称する）は，CASE のなかに置かれ，ソフトウェアエンジニアリングに必要なデータを統合し，記憶し，管理する装置である．

CASE は，なんらかのソフトウェアエンジニアリングパラダイム[15]に従って，エンジニアリングを支援する．CASE への期待にはつぎのようなものがある．

イ）仕様，図式などの記述，編集，記憶
ロ）記述の誤り検査，美麗印刷による文書性向上
ハ）実現仕様の要求に対する正当性検証
ニ）修正に付随する波及範囲を示し，修正を容易にすることによる保守容易性向上
ホ）異なる視点から記述された対象／関係の統合
ヘ）プログラムのテスト
ト）再利用増進，知識の可視化，意志伝達容易化
チ）作業標準化，準自動化
リ）原価管理，工程管理，構成管理など管理業務の合理化

以上のような期待を集める CASE のための SEDB のデータモデルは，少なくともつぎのような要素から構成する必要があろうと考えられる．

a）静的対象：実世界に存在する実体，設計者が頭に描く抽象実体，仕様書，プログラムテキストのようなソフトウェア構成要素，これら構成要素内に含まれる記述単位などを表現する．

b）操作的対象：設計作業，プログラミング作業，原価管理，工程管理のような活動の手順や軌跡を表現する．従来の SEDB では，ここで述べる操作的対象の概念は，SEDB を利用する利用者の応用プログラムやコマンドのなかで表現されるものとされてきた．以下で述べる SEDB では，これをデータモデルそのものの一部であると見なし，利用者の負担を軽減すると共に，利用者作業の合理化，効率化を図ろうとしている．

c）以上の対象間に存在する依存関係（dependency）：上記の各対象の間の意味関係を表現する．

本稿では，まず SEDB に関する研究の歴史と動向，および本研究の背景について述べる．つぎに，筆者の提案するデータモデルの思想を，実験的 SEDB である SEDB/OKBL の記述を用いて説明する．さらに，その実現方法について触れ，CASE としてこれを操作，実行した場合の有用性について考察する．

## 2. 本研究に対する背景

CASE に関する研究は 1970 年代から始まったが，その流れはおおよそ 4 つに分類される．

（1） コマンドを利用したシステム：オペレーティングシステムの上に，ユーザの目的に合致したコマンド言語処理体系を築き，コマンドを実行するとき希望のツールが選択され，ツールに被処理体が渡され，処理が行われる．unix™（AT & T，ベル研開発のオペレーティングシステム）のほか，初期の CASE はほ

† A Data Model Applied in Software Engineering Database SEDB/OKBL by YOSHIHIRO MATSUMOTO (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科

とんどこの方式をとっている.

(2) 構造規則を利用したシステム：記述文面に関する構造規則を属性文法などを用いてあらかじめ定義し，文面を作っていく際に文法に基づいて必要なツールを呼び出し，文面作成を支援するようなシステムで，Cornell Program Synthesizer[5] などがある.

(3) 原言語仕様を拡張したシステム：最終的に実行する原言語の仕様を拡張した CASE 言語体系を構築し，その翻訳実行過程で必要なツールを呼び出して原言語で記述された文面を操作，処理する．たとえば，原言語である Mesa に対して作られた Cedar[6] がある.

(4) データベースを中心としたシステム：記述される文面や図式そのものはユーザ仕様に従い，記述に含まれた要素や，要素間の関係をデータベースに記憶し，管理し，設計，製作，試験，保守に利用するようなシステム．PCTE (portable common tool environment) に基づいたシステム[1]や，Ada Programming Support Environment などは，この例である.

(2), (3)は，programming-in-the-small の支援に重点を置いたシステムであり，他の2つは programming-in-the-large および ライフサイクルの支援を期したシステムである．SEDB は，とくに(4)に関して論じられてきた．本稿においても，(4)を対象として論じる.

1980年代の初頭から，SEDB に関して多くの会合がもたれた[1)~4)]．この経過を要約すると，つぎのとおりである.

米国における CASE は，(2)，(3)に代表されるようなツールを主体にして発達してきた．多くの大学でこの種のシステムが開発されている．これに対して，日本では応用ソフトウェアを大規模に生産している大企業が，もっぱら社内設備として CASE を開発してきた．筆者らが開発し，1977年以来実用してきたSWBシステム[12]はその一例である．これらの CASE は，ソフトウェアのライフサイクル全体にわたって，一貫性のある技術および管理支援を行うために，上記の(1)をベースにし，それに(4)の性質を加えたものであった[14]．

ヨーロッパでは，欧州経済共同体 (EC) が支援する ESPRIT (European Strategic Programme for Research and Development in Information Technology) において，前述の PCTE が作成された．ここでは，移植性，共通性という概念が強く打ち出されており，米国の Common Ada Programming Support Environment (APSE) にも影響を与えた．英国では，PCTE や文献1) の思想に基づいて，IStar[7] のようなシステムが作られている.

当初，階層，網，関係型データモデルがソフトウェアエンジニアリングにおけるデータモデルに合致するかどうかの検討が種々行われた．その結果は，文献4), 8), 9) や，CAD (計算機支援設計) に関する各種の学会発表などを通じて良く知られているとおりである．従来の3つの基本的データモデル，すなわち階層，網，関係型のデータモデルでは，ソフトウェアエンジニアリングが必要とする，とくにデータ間の意味関係を十分表現できないと考えられている．関係型を例に挙げれば，ひとつのタプル内で，配列，構造，束ね（たとえば，直積集合）などの表現が可能でないこと，有向グラフのようなデータモデルをデータの冗長記憶を避けながら実現することが難しいこと，手続き属性 (procedural attribute) が表現できないこと，などが問題として挙げられている.

一方，新しく生まれたオブジェクトモデル（たとえば文献 10)に対しては，generalization/specialization を基底とする is-a 階層の表現は容易となったが，aggregation, interaction, composition, cross-product, summarization などによって分類される association[18] を含む関係表現は十分できないこと，これらを可能とするファンクショナルデータモデル，またはセマンティックデータモデル[11]については，まだ十分成熟していないこと，などが指摘されている．しかし，筆者はセマンティックデータモデルが，以下に述べるような要件をある程度満たすものと期待して，本研究を行った.

以下，なぜ SEDB に関してセマンティックデータモデルの適用を考えようとするかを述べる.

(1) ソフトウェア要求定義において，実世界の問題，とくにそのもつ意味をモデル（たとえばデータフローモデル，状態遷移図など）によって表現することが必要となるが，計算機内部表現がセマンティックデータモデルに基づいていれば，これらのモデル記述を内部表現へ，より合理的に写像できる.

(2) ソフトウェア構成要素（文書，プログラムなど）相互的に存在する関係，または制約をセマンティックデータモデルでは素直に表現できる．担当者が，ひとつの要素の一部を修正する場合に，それと関係をもつ他の要素の関連部分を表示し，修正することを

CASE が促すためには，制約となる関係の的確な保持と利用が必要となる．ここでいう関係には，たとえば，次のようなものがある．

＊同一のライフサイクルフェースにおけるソフトウェア構成要素（たとえば，要求定義フェースでは，機能ごとの要求仕様書）の間に存在する関係（interaction），たとえば共に参照する入出力名，共に利用する共通機能など．プログラムモジュール間で互いに参照するデータまたは手続きの参照関係もこれに入る．

＊異なるライフサイクルフェースで作られるソフトウェア構成要素相互間の構成依存関係（composition），たとえば，要求仕様書上のある機能がどのプログラムとどのプログラムを連接して実行することによって実現されるか，など．

＊同一性質のソフトウェア構成要素（たとえば，仕様書）が共通にもたねばならない性質（generalization），構成（aggregation）に関する規則のようなもの．

（3） セマンティックデータモデルでは，ダイナミックモデル（トランザクションともよばれる）[19] の表現を可能とする．これが，SEDB に対して次のようなことで役立つ．ソフトウェアエンジニアリングデータモデルにおいては，1章のb）で述べた立場から，ソフトウェア構成要素のような静的，受動的（他の対象の状態を変化させる力のない）対象だけではなく，ソフトウェアエンジニアリングの各作業単位（以下単位ワークロード (unit workload) と称する）のような手続き属性をもった操作的，能動的（受動的対象の状態を変化させ得る）対象も表現せねばならない．単位ワークロードとは，1人の担当者に与えられた仕事単位のことである．筆者の SEDB では，この単位ワークロードもデータモデルのなかではひとつのオブジェクトとして表現される．単位ワークロードに相当するオブジェクトには，たとえば，この単位ワークロードに関して，その担当者が行う操作に関するプロセス，すなわちトランザクションをすべてカプセル化して格納してある．担当者は，端末に向かって単位ワークロードオブジェクトを呼び出し，必要なトランザクションを起動し，これと対話しながら，ソフトウェアの生産を進めることができる．

セマンティックデータモデルに以上のような可能性をもとめて実験を進めている SEDB/OKBL (object-oriented knowledge-based language) を通じて，SEDB に関する筆者の考えを述べる．

## 3. ソフトウェアエンジニアリングを表すデータモデル

プロジェクトは，あるひとつのソフトウェアのライフサイクルに携わる複数の人々によって形成される．プロジェクトにおいて形成されるソフトウェアエンジニアリングは，時間によって変化するプロセスである．このプロセス（以下，単にプロジェクトとよぶ）は，工程進捗度，計上原価，成果物完成度，品質評価進捗度，などのような管理指標を状態変数としてもつ操作的，能動的オブジェクト（2章(3)で述べた）と，製作中の文書やプログラムのようなソフトウェア構成要素の文面そのものの状態を状態変数としてもつ静的，受動的オブジェクト（同じく2章(3)で述べた）によって表すことができる．

前者は，エンジニアの活動そのものを手続き属性の集合としてカプセル化したオブジェクト（以下，活動オブジェクトという）であり，これに対して後者は，活動を受け入れてソフトウェア構成要素の状態を管理するオブジェクト（以下，静的オブジェクトという）である．活動オブジェクトは，前章で述べたトランザクションを手続き属性としてもつ．たとえば，ある活動オブジェクトAが，前述の単位ワークロードのような仕事aを表しているとすると，仕事aを実行する人が，その仕事のなかで静的オブジェクトに対して実行するトランザクションをすべてまとめて，活動オブジェクトAのなかにはカプセル化してある．ひとつのトランザクションとは，利用者が通常，ソフトウェア構成要素に関して行う操作の手順であり，ツールの起動，ツールへのパラメータの設定，ツールと構成要素格納ファイルとの連結などから成っている．活動オブジェクトでは，このトランザクションを手続き属性としてもち，利用者は活動オブジェクトとの対話を通じてトランザクションにパラメタ値を与え，実行する．一方，活動オブジェクトはトランザクションの実行を監視することによって，利用者の作業状況を測定し，このオブジェクトが状態変数としてもつ管理指標の値を更新する．

静的オブジェクトは，各ソフトウェア構成要素に対応して作られ，活動オブジェクトが実行するトランザクションによって，ソフトウェア構成要素に対して利用者が行う操作に関して，ソフトウェア構成要素の状態を管理する．ここで行う管理とは，関係制約を用いて行う他の構成要素との整合性の検査，トランザクシ

ョンが要求する他の構成要素内の関係部分の同定，版の管理，アクセス履歴の管理などである．

データモデルの例を**図 1**，**図 2**，**図 3** に示し，以下，具体的に説明する．図 1 と図 2 は，文献 11) と同様の記法を用いて書かれており，三角はクラス，丸はインスタンス，二重線の矢はクラスとサブクラスの関係(矢の先端がクラスで，根元はサブクラス)，実線の矢は意味をもった関係を表す．実線の矢はそれぞれ意味記述を伴い（図には記されていない)，矢の方向はその意味記述に従って定める．意味記述では，主格となるオブジェクトから目的格になるオブジェクトへ向かって矢を描く．図では矢として記述されるが，この関係の意味はデータモデルにおいては関係型オブジェクトによって表現される．

図 1 において，最上層のクラスがソフトウェア工場であり，これは活動，静的の両性を備えている．これには図に示すようなサブクラスがある．人事管理，など管理に関するものは活動オブジェクトである（詳しく述べないので，図 2 では点線で記してある)．プロジェクトは，単位ワークロードという活動オブジェクト，およびソフトウェア構成要素という静的オブジェクトをサブクラスとしてもつ．前者は，前章で述べたようなトランザクションを手続き属性としてもち，CASE 利用者の活動を監視し，利用者が始動するトランザクションによってツールを動かした

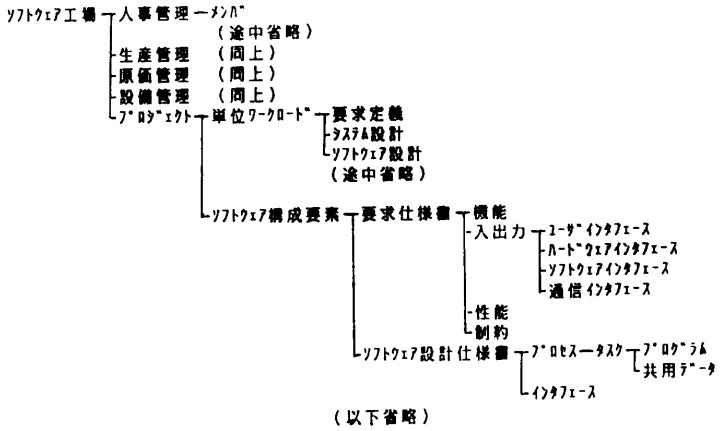



**図 1** クラスの ISA 階層の例
Fig. 1 An example of ISA hierarchy on some classes.

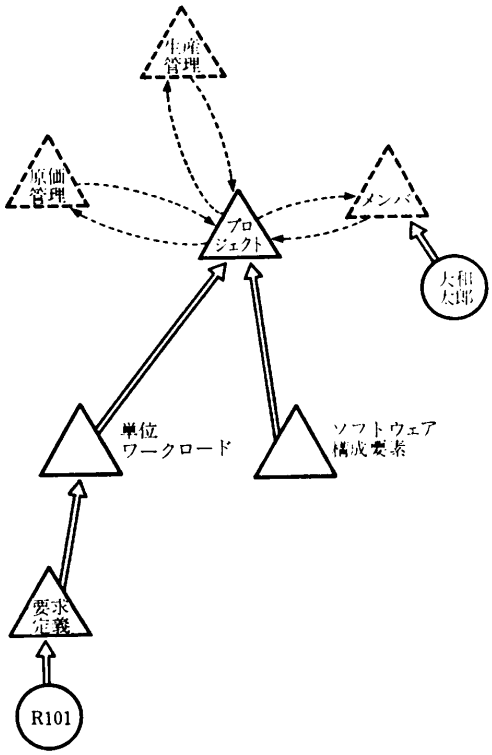
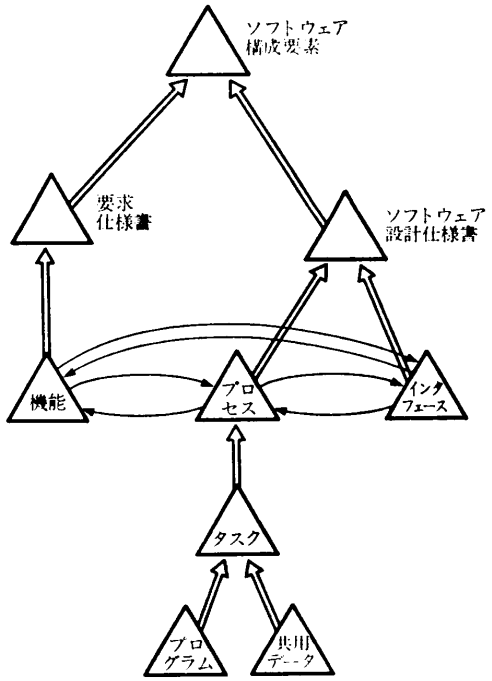



**図 2** SEDB のデータモデルにおける活動モデルの一部
Fig. 2 A part of activity model in the SEDB data model.




注) 「機能」は機能要求仕様書の略記であり，「プロセス」はプロセス設計仕様書の略記である．「インタフェース」…「共用データ」も同様の略記．

**図 3** SEDB のデータモデルにおける静的モデルの一部
Fig. 3 A part of static model in the SEDB data model.

り，ソフトウェア構成要素オブジェクトへのアクセスを行う．このオブジェクトは，管理指標に関するものを内部変数としてもち，CASE 利用者が実行するトランザクションを監視し，作業の履歴や出来高を計量して，管理指標と実績との比較を行う．たとえば，単位ワークロードの出来高と目標量との差や，現在生産性と目標生産性との比較を行って CASE 利用者へ表示する．後者，すなわちソフトウェア構成要素オブジェクトは，製作するソフトウェア構成要素，そのものの仕様，その性質，および他のソフトウェア構成要素との関係（以下に示す）を内部データとしてもち，ソフトウェア構成要素の入っているファイルへのアクセス，他のソフトウェア構成要素の内容との整合性 (integrity) を関係オブジェクトを作用させることによって管理する．

スーパクラスとサブクラスの間の関係は階層である．本稿では，階層を単なる is-a の関係とはせず，つぎのような意味をもった階層であると考える．（“／”の左がスーパクラス，右がサブクラス）

＊より一般的対象／より特殊な対象：たとえば，仕様書／設計仕様書の関係である．プロジェクトでは，一般に要求仕様書，設計仕様書，プログラム仕様書など複数種類の仕様書が作られるが，これらの間には一般的属性が存在する．この一般的属性を仕様書クラスに入れる．図2の単位ワークロードと要求定義との関係はこれに当たる．

＊全体／構成要素：たとえば，プログラムモジュールという構成単位が，説明書ファイル，ソーステキストファイル，オブジェクトコードファイルから成るとした場合，プログラムモジュールはクラスであり，他はサブクラスである．図3の要求仕様書とソフトウェア設計仕様書との間にはこのような関係が存在し，スーパクラスのもつある属性は要求仕様書が継承し，他の属性はソフトウェア設計仕様書が継承する．

＊共有物をもった要素の集合／集合の要素：たとえば，ある機能に関する要求仕様書がいくつかの設計仕様書によって実現されるとき，また，設計仕様書に書かれたある思想がいくつかのプログラムモジュールによって実現されるとき，これらのものの間には implements/implemented_by という関係が存在する．このような関係は新しいオブジェクト（タイプコンストラクタともよばれる[11]）として表現され，この関係型オブジェクトはこれに関係するソフトウェア構成要素によって共有される．このような場合，この関係型オブジェクトを含む関連のオブジェクトをサブクラスとし，集合をクラスとする．図3で機能，プロセス，インタフェース間に存在する関係はこのような意味をもったものである．

＊クラス／インスタンス：is-a 階層であり，図2のメンバと大和太郎のような関係に用いる．

## 4. データモデルの記述

図4は，単位ワークロードというオブジェクトの定義記述の例を示している．定義記述は，2行目の型名記述，3行目の上位型名記述，4から 15 行目にわたる内部属性記述，それ以下の関係属性記述から成る．内部属性は，このオブジェクトのもつ状態変数（印刷可能型ともよばれる）へのアクセスを定義する．たとえば，5行目の記述は，このオブジェクト型が文字列データとの間に，“単位ワークロード名”という関係をもっていることを表している．このオブジェクト型

```
001 定義：
002 オブジェクト型：単位ワークロード；
003   上位型：プロジェクト；
004   内部属性：
005    （単位ワークロード名　文字列）
006    （被操作許諾対象名　（集合　文字列））
007    （操作許諾対象名　（集合　文字列））
008    （単位ワークロード現在状態　（集合　（文字列　実数））
009    （目標原価　実数）
010    （実績原価　実数）
011    （目標工数　実数）
012    （実績工数　実数）
013    （目標生産性　実数）
014    （実績生産性　実数）
015    （プロセスステータス
        . . . . ）；
016   プラグ属性：
017    （担当メンバ名　ﾜｰｸ/ﾒﾝﾊﾞ対照）
018    （操作対象要素名　ﾜｰｸ/構成要素対照）
019    （前ワークロード名　ワークロード計画）
020    （後ワークロード名　ワークロード計画）
021    （ファイル操作（構成要素名　版名　操作目的）　ソフトウェア構成要素）
022    （ファイル削除（構成要素名　版名）　ソフトウェア構成要素）
023    （ファイル生成（構成要素名　版名）　ソフトウェア構成要素）；
024   ソケット属性：
025    （ワークロード開始（単位ワークロード名）　SEDB [..]）
026    （ワークロード中断（単位ワークロード名）　SEDB [..]）
027    （ワークロード継続（単位ワークロード名）　SEDB [..]）
028    （ワークロード完了（単位ワークロード名）　SEDB [..]）
029    （実績原価　原価管理 [..]）
030    （実績工数　工程管理 [..]）
031    （実績生産性　SEDB [..]）
032    （プロセスステータス　SEDB [..]）
033    （プロセスプログラム実行（単位ワークロード名）　SEDB [..]）
034     . . . . . .
035     . . . . . ；
036 定義終；
```

図4 データ定義記述の例（“単位ワークロード”の定義）
Fig. 4 An example of data-definition description (the definition of “Unit Workload”).

からインスタンスが生成されるとき，具体的な単位ワークロードの名が，この文字列に束縛される．

“プロセスステータス”というのは，この単位ワークロードの現在の状態を表す変数である．実世界の単位ワークロードは，事前条件が満たされて，はじめて開始できる．たとえば，「単位ワークロード R 101 を開始するためには，タービン，発電機，…の仕様書がそろっていなければならない」というような事前条件が満たされていなければならない．この事前条件は，R 101 の直前にあるひとつ，または複数の単位ワークロードすべてが完結することによって満たされる．インスタンス R 101 の実行開始に当たって，この事前条件の検査を行う必要がある（この条件検査は，後に述べるワークロード開始という関係属性のなかに含まれる）．

また一方，R 101 が完結する場合には，あらかじめ定めた事後条件を満たさなければ完結したとはいえない．たとえば，入出力リストが全部満たされること，というようなことが事後条件である．プロセスステータスというのは，このような条件の，現時点における満たされ度合を分かりやすく表示するための変数のデータ構造である．この部分の記述では，構造，配列，などの表現が必要となる．

16行目から始まる関係属性は，プラグ属性とソケット属性に分かれる．下記の記述が行われるとき，オブジェクトAのプラグ属性は，オブジェクトBのソケット属性に接続しているという．

```
定義：                    定義：
オブジェクト型：A；       オブジェクト型：B；
  ……；                     ……；
  プラグ属性                ソケット属性
  (X(a b c …) B)            (X(a b c …)
                                  A […])
  ……；                     ……；
定義終；                  定義終；
```

ここで，Xは関係名であり，a，b，c，…は引数である．プラグ属性は，オブジェクトが活動中に他のオブジェクトの内部変数へアクセスする性質を表し，ソケット属性は，オブジェクトが非活動中にアクセス要求を受けてこれに応える性質を表す．活動オブジェクトは必ずプラグ属性をもち，静的オブジェクトは活動オブジェクトへ接続するプラグ属性をもたない．プラグの性質は，たとえば 17 行目にあるように，他のオブジェクト（たとえば，ワーク／メンバ対照）へ，“担当メンバ名”というメッセージを送出して，自分を担当するメンバ名を返してもらうことである．ソケットの性質は，たとえば，29 行目にあるように，“原価管理”というオブジェクトから“実績原価”というメッセージを受けて，内部属性である実績原価の値を“原価管理”に返すことである．

ソケット属性には，25行目のような記述がある．これは(…)のなかの最後の要素（この場合“SEDB”というオブジェクト）のもつプラグ関係属性によって駆動される機能を表している．[…]が表す文は，属性スクリプトというものである．25 行目についていえば，SEDB から，“ワークロード開始”というメッセージを受け取って，属性スクリプトに記述された手続きを実行する．この記述には OKBL の Lisp 式を用いる．この Lisp 式では，プラグ属性として定義された関係名をデータとして用いることができる．

**図 5** は，図 3 で示した“機能”というクラスの定義の例を表している．111 行目からのソケット属性記述が，図 4 の 21 から 23行目までのプラグ属性に対応する．

図 4 にあったワーク／メンバ対照，ワーク／構成要素対照，ワークロード計画などのようなオブジェクトは，関係型オブジェクト，またタイプコンストラクタ[11]とよばれるものである．たとえば，ワーク／メンバ対照では，単位ワークロード名とメンバ名との直積の部分集合（ひとりのメンバが複数の単位ワークロードを担当する）が格納される．ワークロード計画では，網状に接続された単一ワークロードの集合がデータ構造として格納される．

```
101 定義：
102 オブジェクト型：機能；
103   上位型：要求仕様書；
104   内部属性：
105   （機能名  文字列）
106   （被操作許諾対象名  （集合  文字列））；
107   プラグ属性：
108   （構成要素名（機能）  構成要素内訳）
109   （依存プロセス名  機能／プロセス依存関係対照）
110   （依存インタフェース名  機能／インタフェース依存関係対照）；
111   ソケット属性：
112   （ファイル操作（構成要素名  版名  操作目的）  単位ワークロード [..]）
113   （ファイル削除（構成要素名  版名  操作目的）  単位ワークロード [..]）
114   （ファイル生成（構成要素名  版名）  単位ワークロード [..]）
115     . . . . . .
116     . . . . . .；
117 定義終；
```

**図 5** データ定義記述の例（“機能”の定義）
Fig. 5 An example of data-definition description (the definition of “Function”).

## 5. SEDB/OKBL の実現

### 5.1 OKBL

SEDB/OKBL におけるデータ定義は，OKBL[13),16)]という言語による記述に変換する．OKBL は，Franz Lisp で書かれたオブジェクト指向型言語であり，ABCL[17)] をその源とする．OKBL によるオブジェクトの記述形式を説明するために，簡単な例をまず掲げる．数を数えるカウンタという概念は，整数の記憶とこれに対する操作（たとえば，カウントアップ，カウントダウン，カウント問い合わせ）が複合されたものである．OKBL によるこのカウンタの記述はつぎのとおりである．

```
[: Class  カウンタ
   (: InstanceVariables    [回数  := 0])
   (: InstanceScripts
      (: receive  [: カウントアップ]
         [回数 := (1+  回数)])
      (: receive [: カウントは？]
      ^回数))]
```

もっとも外側にある [⋯] に囲まれた部分がオブジェクトを表し，このオブジェクト名はカウンタである．第1行目では，さらに“カウンタ”をクラス名として定義する．カウンタは種々の目的に対して特定化することができる．たとえば，“投球数カウンタ”のようなオブジェクトを生成することができる．

カウンタから投球数カウンタというインスタンスを生成するためには，継承する InstanceVariables および InstanceScripts をあらかじめ指定してから，つぎのような文を実行する．

```
[投球数カウンタ  = =
                [カウンタ <= [:new]]]
```

### 5.2 SEDB/OKBL から OKBL への変換

4章で説明した定義は，オフライン時に，OKBL による表現に変換され，実行時は OKBL によって解釈実行される．以下，例を用いて説明する．図4に示した定義は，**図6**に示すような OKBL の記述に変換される．図4の5から 15 行目に書かれた内部属性データは，図6の8行目のインスタンス変数によって実現される．これらは，図6，3行目を実行することによってインスタンス化され，6行目で初期化される．図4の 16 から 24行目に書かれたプラグ属性は，それぞれ他のオブジェクトへのメッセージ送出文に変換される．たとえば，第17行目は，

```
[X := [ワーク/メンバ対照 <= [: 担当メンバ名
      単位ワークロード名]]]
```

のように変換される．これが実行されると，ワーク／メンバ対照（オブジェクト）の内部変数である担当メンバ名が当該オブジェクトの内部変数 $X$ に束縛される．図4の25行目に書かれたソケット属性は，図6，20行目に示すようなスクリプトに変換される．ここで示した“;”から始まる行は注釈記事であるが，本来はここへ属性スクリプト内に記述された Lisp 式を入れる．プラグ属性を通して得られた上記 $X$ のようなデータは，この Lisp 式のなかで使用できる．

```
001 [:Class 単位ワークロード
002  (:ClassScripts
003   (:receive [:new *単位ワークロード名 *被操作許諾対象名 *操作許容対象名
                    *単位ワークロード現在状態 *目標原価 *実績原価 *目標工数
                    *実績工数 *目標生産性 *実績生産性 *プロセスステータス]
004    (:var OBJ)
005    [OBJ := [%Self <= [:new]]]
006    [OBJ <= [:初期化 *単位ワークロード名 *被操作許諾対象名 *操作許容対象名
                    *単位ワークロード現在状態 *目標原価 *実績原価 *目標工数
                    *実績工数 *目標生産性 *実績生産性 *プロセスステータス]
007    ^OBJ))
008  (:InstanceVariables
       単位ワークロード名  被操作許諾対象名  操作許容対象名
       単位ワークロード現在状態  目標原価  実績原価  目標工数
       実績工数  目標生産性  実績生産性  プロセスステータス)
009  (:InstanceScripts
010   (:receive [:初期化 *単位ワークロード名 *被操作許諾対象名
                    *操作許容対象名 *単位ワークロード現在状態 *目標原価
                    *実績原価 *目標工数 *実績工数 *目標生産性
                    *実績生産性 *プロセスステータス]
011    [単位ワークロード名 := *単位ワークロード名]
012    [....
019     ....])
020   (:receive [:ワークロード開始 :asked_by SEDB]
        ;入力者名が被操作許諾対象名に合致していなければ，拒絶のメッセージ
        ;をを返す．拒絶しない場合，操作可能な構成要素名，版名を
        ;[:操作対象要素名（単位ワークロード名）]メッセージを用いてﾜｰｸ/構成
        ;要素対照に問い合わせし，
        ;[:ファイル読み出し  ....]メッセージを用いてソフトウェア構成要素
        ;から作業しようとするファイルを取り出してメモリへロードする．
        ;作業開始時刻，その他を記録する．
       ^'準備完了  :"準備完了"というメッセージを返す)
021   (:receive
       ......)
022  )
023 ]
```

**図6** “単位ワークロード”の OKBL による表現の初めの部分
Fig. 6 The beginning part of “Unit Workload” described by using language OKBL.

## 6. SEDB/OKBL の操作と実行

SEDB/OKBL の操作手順の一部を図7に示す．SEDB は，unix™ の上で作動する OKBL で書かれているため，2行目に示すように OKBL をよぶ．3行目の“〉”は，OKBL でのプロンプトである．3行目で，SEDB というオブジェクト（ユーザとのインタフェースを司る）に“[：…]”で囲まれたメッセージを送る．〈使用者の名前〉の部分には，名前記号を直接入れる．SEDB が，この名前を許諾した場合は，4行目に示すように“SEDB”を返してくる．次に，5行目で，“R 211”という名前の単位ワークロードの作業を開始したい旨のメッセージを送る．これに対して，SEDB は，6，7行目に示すような文を返してくる．8行目で，正しい名前を入れてやると，9行目でこれが受け入れられる．このとき，ユーザは単位ワークロード“R 201”というオブジェクトの中に入ったことになる．このワークロードは，既に前から手をつけていたものであるとし，それを継続するとして，10行目に示すようなメッセージを送る．ここで，“R 201”は，作業時間登録などの処理を行い，11行目の文を返す．ユーザは，“F 2011”のファイル第1.0版を継続して操作したいとし，12行目のメッセージを“R 201”へ送る．ファイル名が13行目で確認された後，14行目に示すように，“R 201”の中からエディタをよぶ．15行目でエディタがよばれたことを確認して，ユーザはエディタの操作に移る．エディタの操作を終了した後は，再び“R 201”へ制御が戻される．

ここで例を挙げたように，プロジェクトメンバが製作するソフトウェア構成要素を格納したファイルは，それぞれに1対1に対応するオブジェクトによって管理されている．また，ユーザの作業単位，すなわち単位ワークロードのそれぞれは必ず専属のオブジェクトに対応づけられる．ユーザが SEDB をアクセスするときは，必ず単位ワークロードのいずれかにアクセスせねばならない．単位ワークロードを介さずには，構成要素の入ったファイルにアクセスできないようになっている．ひとつのプロジェクトには，通常，複数のメンバが参加するため，SEDB には複数のトランザクションが同時に発生することになる．ここで，トランザクションTとは，メンバTが SEDB へアクセスし，ツール操作やファイル操作などによって，いくつかのオブジェクトの内部変数に変化を与え，メンバTが目的を果たすまで終ることのない一連の処理のことをいう．ソフトウェアエンジニアリングにおけるトランザクションには，次の2つの特徴がある．

```
001  $ unix
002  % okbl
003  > [SEDB <= [:メンバ名　<使用者の名前>]]
004  SEDB
005  > [SEDB <= [:作業開始　R211]]
006  R211はあなたの単位ワークロードではありません
007  あなたの担当すべきものは，R201です
008  > R201
009  R201
010  > [R201 <= [:ワークロード継続]]
011  R201
012  > [R201 <= [:ファイル操作（F2011 1.0 継続）]]
013  F2011 version1.0
014  > <エディタの名称>
015  <エディタの名称>
```

図7 ワークステーションから SEDB を操作する際の対話例

Fig. 7 An example of command/response series displayed on a CRT of the SEDB workstation.

（a） トランザクションが何日かにわたって完結せず，途中で頻繁に中断（夜間の作業休止），継続（朝の作業再開始）を繰り返す．

（b） 並行トランザクションが起こり得る．

第1の問題は，トランザクションの実行ステータスを中断時に保存し，継続時にそれぞれを用いて実行再開できるようにする必要を提起する．第2の問題について，ソフトウェアエンジニアリングにおける並行トランザクションの性質を通常のオンライントランザクションシステムのそれと比較して考えた結果，SEDB/OKBL ではつぎのように考えている．

（イ） 利用者一人の SEDB に対するアクセス，すなわちひとつのトランザクションに対してひとつの SEDB/OKBL を専有して割り当てる．なお，この専有された SEDB/OKBL が，プロジェクト全体が使用する SEDB の中で，ひとつの並行タスクを形成するようにする．利用者の一人一人が1台のワークステーションを専有する現状からみて，自然な成り行きと考える．

（ロ） 各トランザクションが，互いに共有するデータをアクセスする必要に遭遇したときには，互いに情報を交換し，同時にアクセスすることのないようにする．あるトランザクションが長期にわたって共有データを排他的にアクセスするときには，ユーザ間の電子メイルによって人為的に問題を解決できるようにする．

（ハ） トランザクションがきわめて長期間にわたって継続することが考えられるが，この間，オブジェクトの内部状態は，互いに一貫性のない状態に置かれる

可能性がある．とくに，関係型オブジェクトの状態がそのような状態になる．このようなことを考慮して，必要なオブジェクトには，一貫性がとれているかどうかを表示するフラグを立てるようにし，一貫性検査を無駄に行わないようにする．

## 7. むすび

SEDB が必要とする要件が，従来の網，関係型データモデルでは満たされない点を考え，データ自身に手続き属性を与えるオブジェクトの思想と，データ間の複雑な意味関係を表現するセマンティックデータモデルの思想を用いて，この要件を満たす方法について研究した．

本稿で述べた方法に従えば，具体的につぎのような点で，有利性が期待できると考える．

（a） 従来の SEDB では，融合が適切に行えなかったプロジェクト管理データとソフトウェア構成要素が，同一 SEDB 上で，一元的に共存し得るようになる．

（b） 大型のプロジェクトが作成する文書では，品質向上，保守性向上などの見地から，要素間の関係 (interaction) 表示，および一貫性保持が重要視されている．データ間の複雑な関係制約を相当程度表現できる本方式では，この問題の解決に向けて，一歩前進できる．

（c） 活動オブジェクトによってトランザクションを管理，および監視させ，利用者の生産活動を支援することによって，ソフトウェア生産プロセスの品質向上，生産性向上が期待できる．

## 参考文献


1) Commission of the European Community: Requirements for Software Engineering Databases, Final Report, Imperial Software Technology Ltd. and Imperial College/DoC (June 1983).
2) Riddle, W. E. et al. (ed.): Software Environments Workshop Report, *ACM Softw. Eng. Notes*, Vol. 11, No. 1 (1986).
3) *Proceedings of the ACM SIGSOFT /SIGPLAN Software Engineering Symposium on Practical Software Development Environments* (Apr. 1984).
4) Proceedings of the ACM SIGSOFT /SIGPLAN Software Engineering Symposium on Practical Software Development Environments, *ACM SIGPLAN Notices*, Vol. 22, No. 1 (1987).
5) Teitlebaum, T. et al.: The Cornell Program Synthesizer: A Syntax-directed Programming Environment, *Comm. ACM*, Vol. 24, No. 9, pp. 563-573 (1981).
6) Teitelman, W. et al.: A Tour through Cedar, *Proc. 7th International Conf. on Software Engineering*, pp. 181-195 (Mar. 1984).
7) Dowson, M.: Integrated Project Support with IStar, *IEEE Softw.*, Vol. 5, pp. 6-15 (1987).
8) Bernstein, P. A.: Database System Support for Software Engineering—An Extended Abstract, *Proc. 9th International Conf. on Software Engineering*, pp. 166-178 (Mar. 1987).
9) Penedo, M. H. et al.: "PMDB"—A Project Master Database for Software Engineering Environments, *Proc. 8th International Conf. on Software Engineering*, pp. 150-157 (Aug. 1985).
10) Maier, D. et al.: Development of Object-oriented DBMS, *Proc. OOPSLA '86 ACM*, pp. 472-482 (Sept. 1986).
11) Hull, R. et al.: Semantic Database Modeling: Survey, Applications, and Research Issues, *ACM Comput. Surv.*, Vol. 19, No. 3, pp. 201-260 (1987).
12) Matsumoto, Y. et al.: SWB System: A Software Factory, in Hunke, H. (ed.), *Software Engineering Environments*, pp. 305-318, North-Holland (1981).
13) 松本，オブジェクトモデルに基づいたソフトウエア設計，*bit*, Vol. 17, No. 10, pp. 19-30; Vol. 17, No. 11, pp. 67-72; Vol. 17, No. 12, pp. 70-82 (1985).
14) 松本，田中：データ構造指向の要求分析技法，情報処理，Vol. 27, No. 2, pp. 160-170 (1986).
15) 松本：ソフトウェアに対する要求の形成，情報処理，Vol. 28, No. 7, pp. 853-861 (1987).
16) 松本：オブジェクト指向型操作的仕様に関する一考察，情報処理学会論文誌，Vol. 29, No. 3, pp. 273-280 (1988).
17) Yonezawa, A. et al.: An Object Oriented Approach for Concurrent Programming, Research Report, C-63, Dept. of Information Science, Tokyo Institute of Technology (1984).
18) Su, Stanley Y. W.: Modeling Integrated Manufacturing Data with SAM*, *IEEE Comput.*, Vol. 19, No. 1, pp. 34-49 (1986).
19) Peckman, J. and Maryanski, F.: Semantic Data Models, *ACM Comput. Surv.*, Vol. 20, No. 3, pp. 153-189 (1988).


(昭和63年10月 3 日受付)
(平成元年 6 月13日採録)

**松本 吉弘**（正会員）

1932年生．1954年東京大学工学部電気工学科卒業．同年(株)東芝入社．1974年東京大学より工学博士の学位受領，1985年同社理事，1989年1月京都大学情報工学科教授，現在に至る．この間，計算機制御システムの開発，ソフトウェア生産技術の開発などに従事．日本電機工業会進歩賞，発明協会全国発明表彰，科学技術庁研究功績者表彰，アメリカ電気電子技術者協会（IEEE）フェロー表彰を受賞．筑波大学非常勤講師，IEEE Transactions on Software Engineering の editor も務めている．「計算機制御システム」など著書10点．電気学会終身員，IEEE フェロー，日本ソフトウェア科学会会員．