データベース・システム　79－4
プログラミング言語　26－4
（１９９０．　９．　７）

ソフトウェアエンジニアリング・データベースＫｙｏｔｏＤＢのオブジェクトモデル

鯵坂恒夫，沢田篤史，山下薫，松本吉弘

京都大学工学部

ＫｙｏｔｏＤＢは，ソフトウェア開発のプロセスと生産物を管理するソフトウェアエンジニアリング・データベース（ＳＥＤＢ）と，それを中心とする開発支援環境である．ＫｙｏｔｏＤＢの目標と全体構成を示した後，生産物の要素相互間の意味関係を管理するために必要となる，仕様書等生産物のデータ化について述べる．この静的対象に対して，動的対象であるプロセスを記述，管理するための Unit Workload の構成について次に述べる．このようなＳＥＤＢのデータモデルとしてオブジェクトモデル，とくに複合オブジェクトと構造継承が妥当であることを議論する．

Object Model for Software Engineering Database KyotoDB

Tsuneo Ajisaka, Atsushi Sawada, Kaoru Yamashita, Yoshihiro Matsumoto

Department of Information Science
Kyoto University

Sakyo Kyoto 606 Japan

KyotoDB is a CASE environment with a software engineering database (SEDB) for software development processes and products. Objectives and overall configuration of KyotoDB are firstly described. To navigate from product to product along semantic relationship between artifacts, KyotoDB must store products as structural objects not as textual ones. In addition to those static objects, KyotoDB also stores and manages dynamic objects, i.e. process descriptions. An object model especially with complex objects and structure inheritance is discussed for a data model of KyotoDB.

1．はじめに

ＫｙｏｔｏＤＢは，ソフトウェア開発のプロセスと生産物を管理するソフトウェアエンジニアリング・データベース（ＳＥＤＢ）である．あるいは，それを中心とする開発支援環境全体を総称することもある．ＫｙｏｔｏＤＢの目標とするところは次の３つである．[1]

●生産物とその要素相互間の意味的関係を管理

生産物とは仕様書，プログラム，テストケース等のソフトウェア構成要素である．意味的関係の管理とは，ファイル単位の版管理，構成管理ではなく，各生産物に含まれる意味内容を抽出し，それらの関係を種別化して管理することをいう．これにより，変更の波及，伝播解析や，設計，仕様化における決定，判断の履歴管理を可能とする．ただし，この関係記述も生産物のひとつであって，自動的に生成されるものではない．

●静的対象（生産物）と動的対象（プロセス）を統合管理

生産物を作っていくプロセスの記述，およびそのプロセスにおける進捗度や資源消費などの指標を，生産物と同じ環境に置くことにより，生産技術と生産管理を効果的に融合する．

●協調作業の支援

生産物とその相互関係，および作業プロセスが一元的に管理されれば，開発メンバ間の協調活動のモデル化が容易になり，グループウェアが実現できる．

2．ＫｙｏｔｏＤＢの全体構成

ＫｙｏｔｏＤＢを中心とする環境は大きく分けて３つの部分から構成される．

○ユーザインタフェース

開発メンバ各人に専用のワークステーションにウィンドウシステムがあって，Smalltalk のＭＶＣモデル[2] でいう view と controller にあたる部分が存在する．現在のシステムでは鼎[3] を利用している．

○ツール群

テキストエディタ，グラフエディタ，言語処理系，デバッガ等の一般的なツールの他，関係追跡ツール，プロセス・スケジューラ，協調作業（コラボレーション）ツール[4] などが含まれる．

○データベース核

関係記述を含む生産物とプロセス記述を管理するオブジェクトベースである．

ＫｙｏｔｏＤＢ初期プロトタイプではＣＯＢ[5]を用いて作成している．

以下本稿ではデータベース核を構成するためのデータモデルを，ユーザインタフェースやツール群との関係も考慮しながら検討を進める．

３．生産物のデータ化

生産物の要素相互間の意味関係を管理するためには，仕様書等の生産物をデータ化しなければならない．すなわち，生産物の意味内容を識別できる単位に区分し，アクセスできなければならない．

ソフトウェア構成要素となる生産物の大部分は次の３つの構造でデータ化できる．

○木構造

例）モジュール構成図，テキスト（格文法による構文木）

内部データ）接続関係，ノードの型，ノードの値

○ネットワーク構造

例）データフロー図，状態遷移図，システムブロック図

内部データ）接続関係，ノードの型，ノードの値，アークの型，アークの値

○テーブル

例）決定表，入出力対応表，メモリマップ，種々の一覧表

内部データ）属性名，属性値

ただし，これらの構造の内部に現れるノード値や属性値がまた３つの構造のいずれかである場合もありうる．

ソフトウェア生産物の多くを占める自然語テキストやプログラムテキストは，バイトストリームとしてファイルに格納してあるだけでは，決して意味関係の追跡のできないものとなる．全くの自然語文書では形式化は難しいが，各組織で定型のフォーマットに従った仕様書やプログラムテキストならば，システム内部で木構造データとして扱える．格文法[6]に基づいて格（ケース）ラベルをノードの型としてもつ構文木にしておくと，識別すべき意味単位の追跡に役立つと思われる．内部形式の表現の一例を図１に示す．

上記の３つの構造ではデータ化しにくいものとしては，画面イメージデータやタイミングチャートなどが挙げられるが，今後に譲る．

４．プロセス記述

ＫｙｏｔｏＤＢにおけるプロセス記述は，Unit Workload (UW) と呼ぶ開発メンバ１人に割り当てられる作業単位を基礎とし，ＵＷのネットワークとしてプロジェクト

```
/* Data Definitions */
d0000:u_command(CONSISTS CONSTRUCT:{retrieve|upd_comment|reserve});
D0000:key(CONSISTS CONSTRUCT:{title|author|keyword*});
D0020:book(CONSISTS CONSTRUCT:[title, author, keyword, contents,
                               abstract, comment, book_id]);
D0030:book_list(CONSISTS CONSTRUCT:book*);

/* User Process */
m0010:u_monitor(ASSERTED STATE:s0000; RULE:r0000;
                         START:c001A; STOP:c001B);
m0011:u_monitor(INPUTS DATA:u_command; FROM:ui);
m0012:u_monitor(INPUTS DATA:uid; FROM:environment);
m0013:u_monitor(ACTIVATES MODULE:do_retrieve; DATA:[u_command, uid];
                          CONDITION:c0010);

s0000:process_table(HAS PROCESS:server);
r0000:shell(HAS VARIABLE:uid);
c001A:shell(ACCEPTS COMMAND:adlib);
c001B:u_monitor(ACTIVATES MODULE:do_retrieve);
c0010:u_command(ISINCASE CASE:retrieve);

m0020:do_retrieve(ASSERTED STATE:s0000; RULE:r0000;
                           START:c001B; STOP:c002B);
m0021:do_retrieve(GETS DATA:[u_command, uid]; FROM:u_monitor);
m0022:do_retrieve(INPUTS DATA:key; FROM:ui);
m0023:do_retrieve(ACTIVATES MODULE:u_call_server;
                            DATA:[u_command, uid, key);
m0024:do_retrieve(GETS DATA:rc_ret; FROM:u_call_server);
m0025:do_retrieve(STORES DATA:rc_ret);
m0026:do_retrieve(GETS DATA:book; FROM:u_call_server;
                       CONDITION:c0020);
m0027:do_retrieve(GETS DATA:err_msg; FROM:u_call_server;
                       CONDITION:c0021);
m0028:do_retrieve(STORES DATA:book.book_id; CONDITION:c0020);
m0029:do_retrieve(OUTPUTS DATA:book; TO:ui; CONDITION:c0020);
m002A:do_retrieve(OUTPUTS DATA:err_msg; TO:ui; CONDITION:c0021);

c002B:do_retrieve(OUTPUTS DATA:(|| book err_msg);
c0020:rc_ret(ISINCASE CASE:0);
c0021:rc_ret(ISINCASE CASE:1);
```

図1　格文法による仕様の表現の一例

全体を表す．（図2参照）ＵＷの内部データは次の通りである．

作業名，作業概要，
参照生産物（前提条件），目的生産物（終了条件），
作業手順（プロセスプログラム），
作業者名，作業者スキル，
作業量見積り，目標作業時間，目標開始時刻，目標終了時刻，
許容コスト，目標品質レベル，目標再利用率，
実作業量（進捗度），実作業時間，
実消費コスト，達成品質レベル，達成再利用率

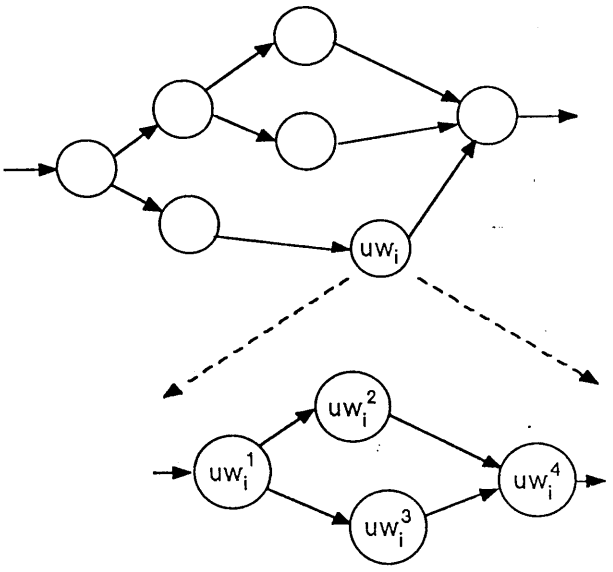

図2　UWネットワーク

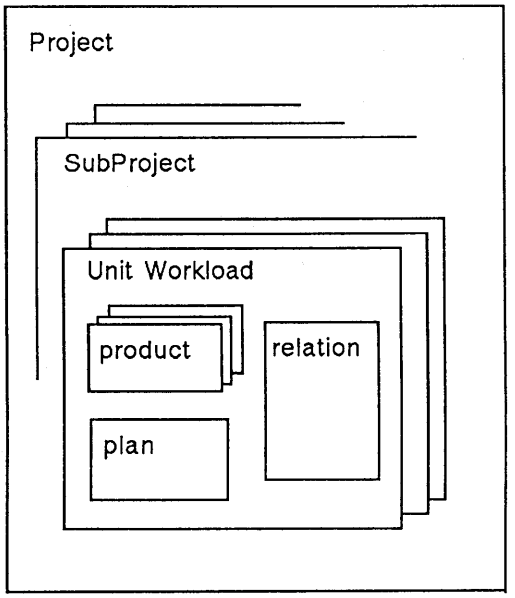

図3　UWオブジェクトの構造

下2行のデータがプロセスの実働とともに変化するものであり，その上2行のデータと対応している．もちろん目的生産物もプロセスの進行にしたがってできあがっていくものである．各目的生産物にはいくつかの版を設けることができて，関係する他の開発メンバの担当する生産物との整合性が，コラボレーションによってとれる前のものも扱えるようにする．プロセスプログラムにはツールの呼び出しや標準規格，部内規格，手法（知識）などの利用がスクリプトで記述される．

プロジェクト計画においてはUWの定義，UWネットワークの作成の後，各メンバの性向，能力，コストに関するデータをもとに，各UWにメンバを割り当てるスケジューリングが必要である．これは PERT と異なり，時間条件とコスト条件を同時に満足しなければならず，また1人のメンバを時間的重複のない限り複数のUWに割り当てることもできるので，複雑な最適化問題となる．

5．オブジェクトモデル

以上に述べたような，ソフトウェア開発における生産物とプロセスを管理するSEDBは，どのようなデータモデルに基づくものであるべきだろうか．まず，生産物もプロセスも十分複雑な構造を持っており，またその構造要素相互の連関も複雑であることから，リレーショナルモデル等従来の単純なモデルでは苦しい．複雑なデータ構造を扱い，その要素へのアクセスを制御できるという初期的目標から，オブジェクトモデルが解として浮かぶが，ここでそれを改めて検証してみる．

まずオブジェクトモデルの特徴を次のように整理し，それに従ってデータモデルの妥当性を考える．

(1) カプセル化
(2) インスタンス生成
(3) メッセージ通信
(4) 複合構造
(5) 継承構造

(1) カプセル化

仕様書，文書，図表等ソフトウェア開発における生産物は，それぞれに固有の意味をもっているから，内部に含まれるデータとそれに対する操作を強結合しておき，モジュール性を高める意義は大きい．また，データの安全性の点から，情報隠蔽の必要性もある．

(2) インスタンス生成

ＳＥＤＢのオブジェクトはプロジェクトの進行とともに蓄積されていく性質のもので，生成，消滅を繰り返すものではないから，とくにこの特徴が恩恵をもたらすものではない．むしろ，オブジェクトのパシステンスを効率的に実現することが不可欠である．ＫｙｏｔｏＤＢ初期プロトタイプでは，Unixのファイルシステムを直接的に使った初歩的なメソッドを用意している．

(3) メッセージ通信

ＳＥＤＢにはプロセスを記述した動的なオブジェクトがあり，ここからメッセージが発信され，静的な生産物オブジェクトが受信するというモデルは，概念的に適している．Smalltalk のような統合的環境にシステム全体を置くのならば，２節に述べたユーザインタフェースやツール群とも統一的なプロトコルで通信できる．一方，Unix のようなツールキット的環境では，インタフェースやツール群とＤＢ核との間にしかけが必要になる．

(4) 複合構造

４節で述べたように，ソフトウェア開発プロジェクト全体はＵＷネットワークで表される．これは３節にあげたネットワーク構造で表されれるが，このネットワークのノードはまたネットワーク構造である場合も多い．プロジェクト全体がサブプロジェクトあるいは開発チームのネットワークとして構成されることがあるからである．生産物オブジェクトも同様にオブジェクトの入れ子，すなわち複合構造（part-of 階層）になることはごく普通である．さらにＵＷ自身もその内部データとして生産物オブジェクトを抱える構造になっている．（図３参照）

複合構造が最も効果を発揮するのは，メソッドの作り込みによるデータの一貫性保持である．外側のオブジェクトが受けたメッセージにより，内側のあるオブジェクトのあるメソッドを起動する，同時に複数のオブジェクトに働きかけなければならない，

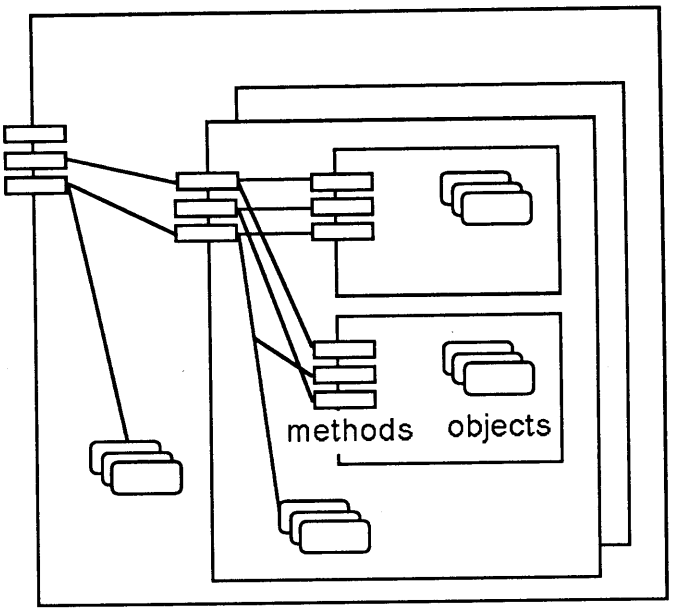

図4　複合オブジェクト構造

などのアクセス制御を構造的に作り込むことができる．（図4参照）

また，ソフトウェア開発の活動では，ＵＷと生産物の関係やプロジェクト構成など，動的に変化する要素が大きいが，複合オブジェクトの動的束縛によりこれに対応することができる．

(5) 継承構造

オブジェクトモデルでは複合構造の階層と直交して，継承の階層（is-a階層）がある．ＫｙｏｔｏＤＢのオブジェクトの多くはまず，３節にあげた３つの構造のいずれかとそれに作用する挿入，削除，変更等の基本的な機能を継承する．もう少し具体的な層でも，例えば仕様書に共通する章だて，フローグラフに共通な型とそれに対する操作，プロジェクトとチームに共通な構成とコミュニケーション機能などの継承がありうる．ＳＥＤＢのオブジェクトベースでは，オブジェクト指向プログラムの場合と異なり，構造継承が機能継承より以上に本質的な階層構造を作ると思われる．[7]

以上のように，オブジェクトモデルはＳＥＤＢのデータモデルとして必要なしくみを数多く提供しているといえる．しかし，肝心の意味関係の扱いについて，オブジェクトモデルは有効なからくりを与えてくれない．このモデルが本来オブジェクトの独立性を基礎として成り立っていることから，当然の帰結とも考えられる．現状では関係記述オブジェクトを作ってＵＷに束縛し，問い合わせメッセージを発するという素朴な構成となっている．（図３，図５参照）

オブジェクトモデルの特性を生かした意味関係の扱いとして，関係を各生産物オブジェクトに作り込んでしまうことが考えられる．このとき，双対リンクの一貫性をと

```
d0000.CONSTRUCT.retrieve(IMPLEMENTS ENTITY:S0000.COMMAND.retrieve);
m0022.DATA.key(IMPLEMENTS ENTITY:S0000.PARAMETER.key);
c0020.CASE.0(IMPLEMENTS ENTITY:S0000.RESULT.success);
c0020.CASE.1(IMPLEMENTS ENTITY:S0000.RESULT.fail);
c0020.CASE.0(IMPLEMENTS ENTITY:D0010.EVENT.hit);
m0010(DERIVED FROM:design_paradigm15);
    ......
m001*.u_monitor->m002*.do_retrieve(ACCOMPLISHES Requirement00)
```

図５　関係記述の一例

るため，やはり複合オブジェクトの構造が有効であろう．あるいは，関係を保持する部分をオブジェクト環境とは分離したリポジトリとすることも考えられる．関係追跡の効率とシステム全体の一貫性を考慮しながら，さらに検討，実験を行う必要がある．

６．おわりに

オブジェクトモデルによってＳＥＤＢを構築すると，このモデルが本来もつ局所性により管理が分散されることになり，集中管理が好まれない組織に向いているのではないだろうか．オブジェクトモデルは複合構造と継承構造がうまく織り成されてはじめて有効性を発揮する．ＳＥＤＢを開発する側も，集中管理的な従来のソフトウェア・アーキテクチャのイメージを払拭することが，成功のカギであるように思われる．

参考文献

[1] Y. Matsumoto and T. Ajisaka: A Data Model in the Software Project Database KyotoDB, Advances Softw. Sci. Tech. JSSST, Vol.2, 1990. (to appear)

[2] G.E. Krasner and S.T. Pope: "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Parc Place Systems, 1988.

[3] 暦本，菅井，森，内山，垂水，杉山，秋口，山崎：Ｘウィンドゥ上のマルチメディアユーザインタフェース構築環境：鼎，情報処理学会第３０回プログラミングシンポジウム，pp.105-115, 1989.

[4] 山下薫，沢田篤史，鯵坂恒夫，松本吉弘：ＣＡＳＥ環境における協調活動支援ツールの試作，日本ソフトウェア科学会第７回ソフトウェア研究会，SW-90-7-7, pp.43-48, 1990.

[5] 上村務，横内寛文，吉田幹，大平剛，J.M. Lucassen: ＣＯＢにおけるオブジェクト指向機能，コンピュータソフトウェア，６巻１号，pp.4-16, 1989.

[6] E. Charniak and Y. Wilks (eds.): Computational Semantics, Fundamental Studies in Computer Science, Vol.4, chapter II Linguistics, North-Holland, 1976.

[7] 青木淳：オブジェクト指向プログラミング環境におけるインテグレーション過程，第１０回ソフトウェアシンポジウム，1990.