ソフトウェアエンジニアリング・データベース KyotoDB の設計と実現†

鰺坂恒夫†† 松本吉弘††

CASE ツールを統合，相互接続する開放型プラットフォームであるソフトウェアエンジニアリング・データベース KyotoDB の設計と実現について述べる．KyotoDB は，ソフトウェア開発保守におけるプロセスとプロダクトを統合的に管理する点で，リポジトリを越える．プロダクトの要素間の意味的関係を作成し利用しようとするとプロセス抜きにはできないし，プロセスの管理，支援はプロダクトに関する情報なしにはありえない．これを実現しているのは次の4つのクラスを複合構造化したオブジェクト・データモデルである．すなわち，単位作業の管理と支援，プロダクトの意味構造の保持，協調活動の管理と支援，プロジェクト全体の管理をそれぞれ行うオブジェクトである．これをカーネルとし，KyotoDB ユーザエージェントを介してツールとのデータ統合，制御統合を実現している．データ統合については，オブジェクト・ストリームと呼ぶ局所的なオブジェクト構造を反映した意味移転言語を提案している．これは KyotoDB のバックエンド・データベースとの結合にも用いる．KyotoDB はまたソフトウェア開発保守のためのグループウェアとしても機能する．複数の単位作業 (役割記述)，複数のプロダクトの要素間の意味的関係をもとに，メッセージの記述やその選択規則，機能記述をコミュニケーションモデルとしてカーネルに内包しており，有機的な協調活動支援を可能とする．

1. はじめに

ソフトウェア開発保守の計算機支援を指してCASEと呼ぶようになってすでに数年が経過する．この間，構造化分析／設計やオブジェクト指向分析／設計といった方法論が拡張または提案され，そしてそれを支援する CASE ツールが抱き合わせで提供されて方法論の普及を促してきた[1)~5)]．しかし，CASE ツールを統合するデータベースやプラットフォームが標準的に使われる環境にはまだなっていない．KyotoDB はそのひとつの試みであり，次に述べる２つの先進的な設計思想に基づいている．

第一にソフトウェア開発保守におけるプロセスとプロダクトを統合的に管理することである．プロダクト，すなわち種々の仕様図面や文書，プログラム記述等を格納，管理するのが CASE データベースに対する第一義的な要求である．しかし，そのプロダクトに対して働きかける作業者やツールのプロセスをあわせもたなければ効果的な支援はできない．逆にプロダクトの中身に言及せず，プロセスの記述と制御に重点を置いたアプローチだけでは，きめの細かい支援ができなくなる．このようなプロセスとプロダクトの有機的統合という特徴から，KyotoDB をリポジトリではなく，ソフトウェアエンジニアリング・データベース (SEDB) と呼んでいる．

第二は開放型CASE環境である．これからのCASE環境はチーム内や組織内で閉じた環境であってはならない．種々の環境で使われる様々なツールは，SEDB を含むツール・プラットフォームを通して適切に相互接続し連係しなければならない．計算機ネットワーク環境ですでに常識的となった「開放性」は，CASE環境においては次のような効果をもたらす．すなわち，ソフトウェア開発保守の作業機能や作業フェーズをつないで切れ目のない（シームレスな）支援を可能にするとともに，一方ではプロセスとプロダクトの流通，再利用を促して，生産性と品質の向上に資する．

以上の２点は続く２つの章で今少し詳しく論じる．KyotoDB の基礎となる理論的考察はすでに示されている[6)]ので，４章以降の本論文ではその設計と実現を通じて，上記２点の議論を展開する．

2. ソフトウェアプロセス＋ソフトウェアプロダクト＝ソフトウェアエンジニアリング

CASE環境におけるリポジトリとは，ソフトウェア開発保守の過程で認識される対象ソフトウェアのデータや手続きの名前，属性，それらの間の関係を管理するものと定義される[7)]．プロダクトの管理単位がファイルでなく要素単位であり，その属性をあわせて管理する点で，ライブラリとは異なりデータディクショナ

† Design and Implementation of KyotoDB: A Software Engineering Database by TSUNEO AJISAKA and YOSHIHIRO MATSUMOTO (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

りを超えるものである[8]が，要素間の意味的関係を管理するところが最も難しく，ここで成功しているものがない．

要素間の意味的関係は，その作成にも利用にも作業プロセスの定義と実働が深く関わる．したがって，プロセスをあわせもつ SEDB でなければ，これを有効に扱うことはできない．すなわち，設計，仕様化における判断や決定の履歴の記録と参照，およびそれに基づく意味的関係の作成支援，あるいは意味的関係を利用した変更の波及，伝播解析や協調活動支援は SEDB をもつ環境で可能となる．

一方，「ソフトウェアの問題はそれ自身よりもそれを生産するプロセスにある」[9]という認識もすでに定着しており，これからの CASE 環境ではプロセスを直接的に維持管理しなければならない．しかし，ソフトウェアプロセスを進行させるイベントの発生源の多くはプロダクトやその要素に由来するため，プロセスをプロダクトと切り離して独立に持っていても有効な支援はできない．KyotoDB を含む環境ではプロセスを構成する各単位作業 (Unit Workload) の定義，支援，管理を中心として分散協調的にプロセスが進行するが，単位作業の始動，終結条件はプロダクトの状態に基づくものである．またソフトウェアプロセスの重要な部分を占める協調活動も，プロダクト間関係に基づく協調活動モデルに従って進行すべきものである．さらに生産管理も，プロダクトの状態を表す指標をもとにして実効のあるものとすることができる．

このように，プロダクトを有機的に管理しようとするとプロセスが必要になるし，プロセスの管理と支援はその対象とするプロダクトとその要素に関する情報なしにはありえない．プロセス中心，プロダクト中心のどちらにも偏ってはいけないわけである．KyotoDB では5章に述べるデータモデルで，このプロセスとプロダクトの統合を果たしている[10]．

3. 開放型 CASE アーキテクチャ

CASE ツールを統合，相互接続するためには，ツールとツール・プラットフォームが次の3つのインタフェースにおいて開放的でなければならない．

- オペレーティングシステム (OS) とのインタフェース
- ユーザインタフェース (UI)
- SEDB とのインタフェース

図1は個々のツールからみた，自分をとりまく環境である．どのインタフェースにおいても，ツールおよびツール・プラットフォームの双方が，それぞれに固有の論理とデータ構造から分離した緩衝領域を用意し，その界面で相互接続するしくみになっていなければ，真に開放的ではありえない．図1の中心部は個々のツールに固有の論理とデータ構造を示し，SEDB や User Interface と記した領域は同様にそれらに固有の論理とデータ構造を示す．これらにはさまれた領域が，双方から独自性を捨てて歩み寄るべき意味領域である．図では歩み寄った位置を示す点線がその領域のちょうど中央に描かれているが，例えばもし SEDB の国際標準が定まったとすれば，この点線は SEDB の境界線まで外側に移動することになる．なお，SEDB や UI 自身もより下位層のプラットフォームすなわち OS の上に浮かぶものであり，一番外側の円がそれを示している．

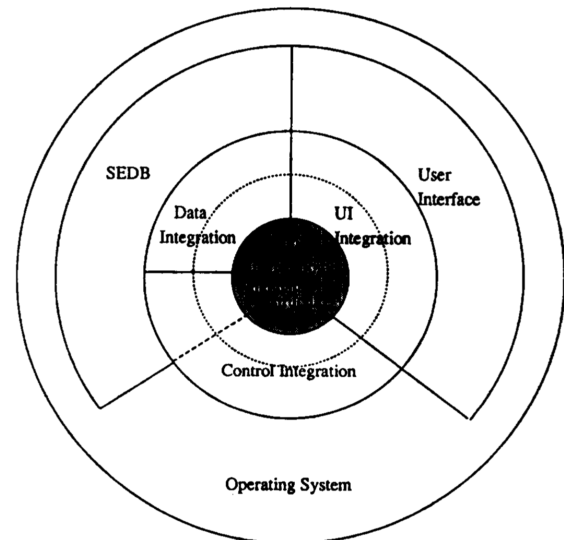

図1　ツール・インタフェース
Fig. 1 Tool interfaces.

OS とのインタフェースには，ツールの起動，終了，制御情報の受渡し，ファイルシステムとのインタフェースなどが含まれ，制御の統合を行うものである．OS とツールの間には，システムコールやコマンドより高い抽象レベルをもつ緩衝領域を設定することが考えられる．PCTE[11]の狙いのひとつはここにあると思われる．しかし，Unix オペレーティングシステムの提供するサービスは，やや抽象度は低いものの標準性は高いため，KyotoDB に接続するツールは現在のところこれを直接利用することを想定している．図1でいえば，制御統合の領域の点線が OS のほうへ大

きく拡がっている状況である．

緩衝領域を介して相互接続するという観点から UI を見ると，サーバクライアントモデルを早くから採用しているウィンドウシステムは，正にこの方式を先取りしていると言える．ただし，ツール固有の論理とデータ構造が，異なるウィンドウシステムとうまくプラガブルになっているとは言いがたい．KyotoDB は個々のツールの UI には全く立ち入らないが，KyotoDB のユーザエージェント（後述）とツール群は同じ UI サーバ（現状では X-window）の上で稼働しなければならないので，KyotoDB としてもこの問題に無関係ではない．さらに，UI はプロセスの実働におけるプラットフォームになるため，なおさら無関心ではいられない．UI そのものがプロセスを規定し制約を与えることのないよう，プロセスの解釈・実働化機構と独立でなければならない．（この問題は本論文の範囲ではこれ以上論じられない．）

SEDB とのインタフェースは特に CASE 環境に固有の問題であるが，CASE ツールと SEDB を CASE 環境に固有のワンセットのものとして強結合してしまっては，開放的生産環境にならない．このインタフェースとして，データ統合，すなわち STL[12]や CDIF[13]のような，CASE データ意味移転[14]の手立てが必須となるのである．これらはCASE ツールの扱うプロダクトの構造や要素の意味をニュートラルな言語で表現し，異なるツール間でのデータのやりとりを可能にするものである．このような緩衝領域を設けることによって，SEDBは一般的な情報資源辞書システムの標準IRDS[15),16]に準拠するなり，KyotoDB のように独自のデータモデルによるなど，自由に設計，開発ができるし，一方個々の CASE ツールも SEDB の方式に独立に設計，開発ができる．KyotoDB ではこの意味移転の手立てとして，オブジェクト・データモデルを表現するオブジェクト・ストリームを用いている．6 章で詳細を述べる．

KyotoDB のように，プロダクトだけでなくプロセスも統合的に管理する SEDB では，ツールはプロセスプログラムにおけるオペレータであり，OS の機能を使いながらツールの制御も行う．図 1 で緩衝層の制御統合が SEDB に一部かかっているのはこれを表現しており，KyotoDB ではそのカーネルからの指示に従ってユーザエージェントがこの制御を行う．

4. KyotoDB の全体構成と動作概要

KyotoDB は次の 5 つのサブシステムから図 2 に示すように構成される．

（1）　KyotoDB カーネル

複雑な構造と関係をもつソフトウェア・プロダクトとプロセスを管理するため，オブジェクト指向データモデルを採用する．次の 4 つのクラスのオブジェクトが複合構造をつくる．

- Project : プロジェクト全体を管理するオブジェクト
- Plan : プロセス記述と管理データを格納するオブジェクト
- Product : プロダクトを格納するオブジェクト
- Coordinator : プロダクト間関係を保持し協調活

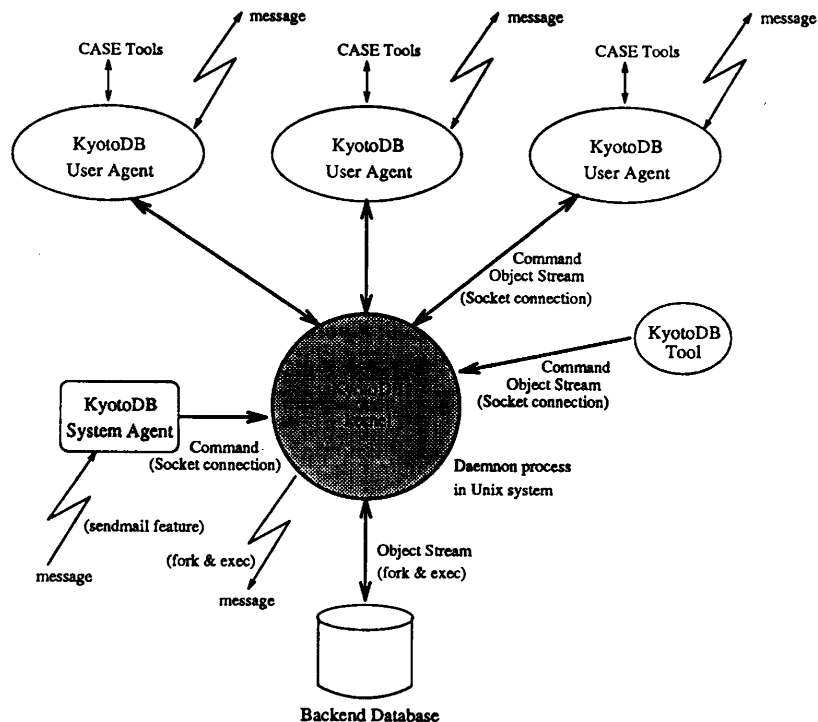

図 2　KyotoDB の構成
Fig. 2　KyotoDB components.

動支援の中核となるオブジェクト

通常のデータベース・システムで意味モデルを定めるスキーマは，Project オブジェクトを頂点（オブジェクト・マネージャ）とするオブジェクト構造として与えられ，データ操作を実現するのは複合オブジェクトを渡るメソッド連鎖である．KyotoDB カーネルは C++ で書かれている．

（2） KyotoDB ユーザエージェント

KyotoDB カーネルとツール群およびユーザとのインタフェースをとる．ツールとのインタフェースについては，図1で SEDB に隣接する緩衝層（データ統合と制御統合の一部）にあたる．データ統合はオブジェクト・ストリームを，制御統合は既定のコマンド（ともに6章で述べる）を用いている．作業者から見ればユーザエージェントも1つのツールであって，KyotoDB への login/logout やユーザ主導のコマンドの発行，作業者への同期的メッセージの伝達，非同期的メッセージの処理を行う．C++ で書かれている．

（3） KyotoDB システムエージェント

作業者からの非同期的メッセージを解析し，KyotoDB カーネルに伝える．Cで書かれ，Unix のメッセージ配送エージェントである sendmail の別名処理機能からパイプで起動される．6.2.2 項および7章を参照．

（4） KyotoDB ツール

KyotoDB に接続し個々の機能を果たす CASE ツールは KyotoDB の一部ではなく，マルチベンダ環境を構成するものであるが，KyotoDB に対する固有のツールとして，プロセス計画のためのツールがある．Cで書かれている．

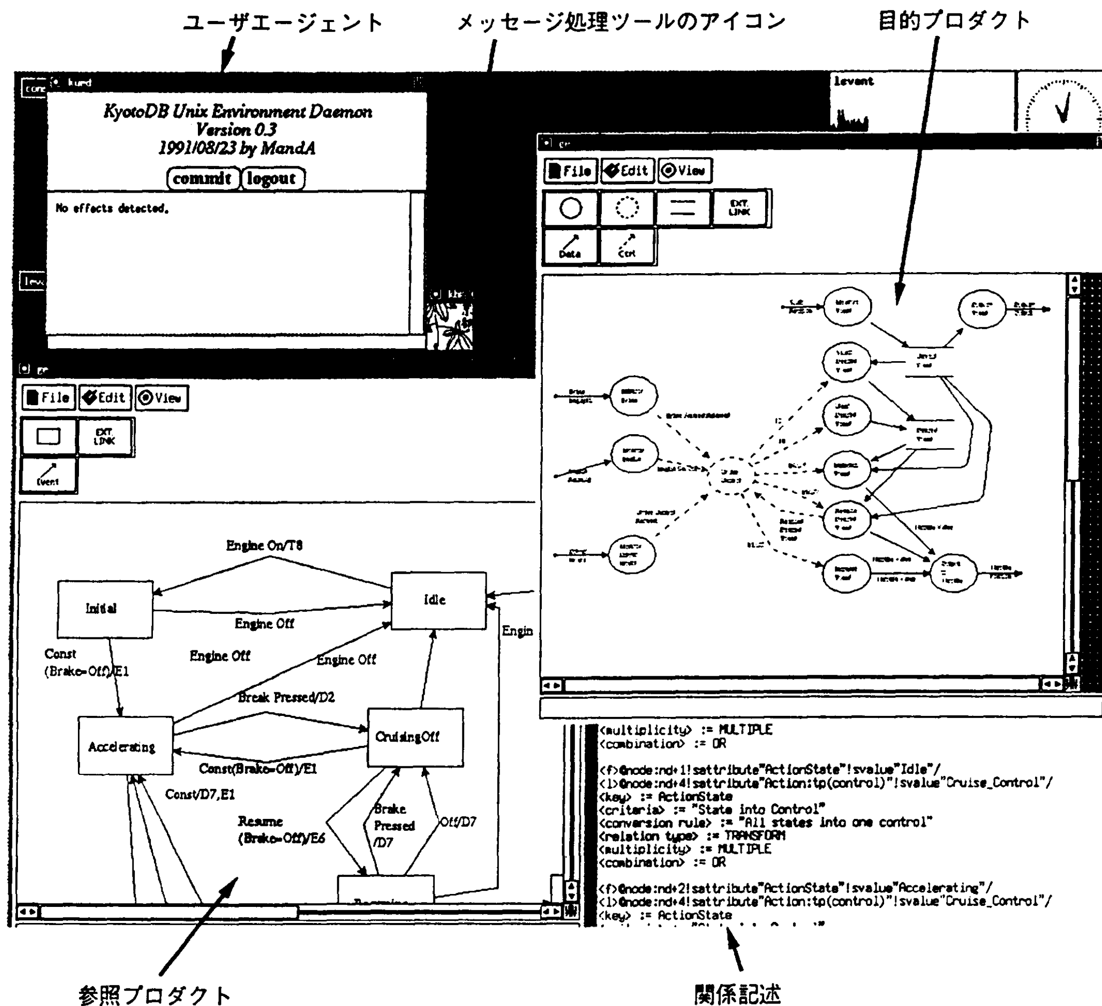

図 3 KyotoDB 環境における作業例
Fig. 3 KyotoDB environment—Example.

（5） KyotoDB バックエンド

KyotoDB カーネルのオブジェクト群を永続化して保存するデータベース．現状の KyotoDB では Unix のファイルシステムを直接的に利用している．プロジェクトデータの再利用のための検索機能も必要である．Cで書かれている．

ここで KyotoDB を用いたソフトウェア開発保守作業の典型的なシナリオの概略を述べる．プロジェクトの一番初めでは，サーバとなるワークステーションで，管理者のプロファイルだけを入力して空のKyotoDB カーネル（存在するオブジェクトは1個のProject のみ）を立ち上げる．次にいわゆるプロセス計画フェーズがあって，開発すべきプロダクトとそれを担当する人員を割り当てた計画書が作られ，これをオブジェクト・ストリームに変換して KyotoDB に入力する（上記（4）のツールが6.2.1項で述べる制御コマンドを用いる）．この段階でプロジェクトに必要なPlan と Coordinator が生成される．プロダクトや関係記述はそのファイル名や操作するツールが指定されるだけで，内容はまだ空である．

この状態になると各作業者が KyotoDB に login することができる．作業者が各自のワークステーションに login するとユーザエージェントが立ち上がり，次にこれを使って KyotoDB に login すると，プロセス計画で指定されたとおりツールは自動的に起動される．（ただし自分が参照すべきプロダクトが未完成のときは，作業開始を拒否される．）作業者はこのツールで参照プロダクトを見たり，作業用のプロダクトや関係記述の編集を行う．（図3）ユーザエージェントは作業中のファイルを監視しており，セーブされてタイムスタンプが変わると，そのファイルをオブジェクト・ストリームに変換するプログラムを起動してカーネルに送り込む．

編集したプロダクトは，適当な時点で作業者の判断によりコミットされる．これは Coordinator による検査を受ける．波及効果がなくそのまま認められる場合，波及効果が見つかって協調活動を引き起こす場合，関係記述の不備のため波及解析ができず延期される場合がある．

5. KyotoDB カーネル

5.1 KyotoDB カーネルの構造

KyotoDB カーネルはオブジェクトベースであるから，複合と継承からなる構造をもつ．KyotoDB の場合，通常オブジェクト指向技術で重視される継承階層よりも，複合構造のほうがより本質的である．複合構造に従って，起動されるメソッド連鎖が統制され，上位オブジェクトが下位オブジェクトの自己反映的な役割を果たす．

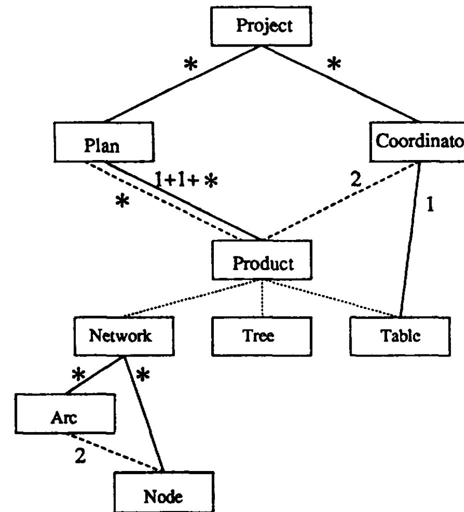

図 4 KyotoDB カーネルのオブジェクト構造
Fig. 4 Object structure of KyotoDB kernel.

複合構造には，実装上はどちらもポインタであるが，概念的に構成複合（上位のオブジェクトが下位のオブジェクトを構造要素として保持する）と参照複合（参照のためのポインタとして保持する）の2種類がある．図4に KyotoDB カーネルの構造を示す．実線が構成複合，破線が参照複合，点線が継承構造を示し，図の上が複合，継承ともに上位である．実線，破線につけたラベルは複合構造要素数を示す．

Project は複数の Plan, Coordinator を構造要素として保持する．Plan は作業中および承認された状態の目的プロダクト（この Plan で開発保守すべきプロダクト）それぞれ1個と複数の作業中の関係記述を要素として持ち，複数の参照プロダクトを参照複合で保持している．Coordinator は2個のプロダクトを参照し，1個の関係記述を要素として保持している．Network, Tree, Table は Product のサブクラスである．Product 以下の複合構造は，図4では Network についてのみ示してあり，複数の Node, Arc を構成複合で保持，Arc はその元，先にあたる Node を参照複合で持っている．

5.2 オペレーティングシステムと KyotoDB カーネル

図1にも示されているように SEDB 自身もなんら

かのプラットフォーム，すなわちオペレーティングシステムの上に構築される．KyotoDB カーネルは Unix システムにおける1つのデーモンプロセスとして立ち上げられる．1つの Project オブジェクトを生成し，該当するバックエンド・データベースのデータがあればそれを読み込んだ後，KyotoDB ユーザエージェント，システムエージェント，KyotoDB ツールからのソケットを介した接続とデータ入力を待ち受ける．

5.3 Project オブジェクト

KyotoDBカーネルと外界とのインタラクションは，すべて Project オブジェクトを通じて行われる．この意味から Project はメタオブジェクトの性格をもっている．KyotoDB カーネルに入力されるデータはすべて次章で述べるKコマンド言語に従うものであり，Project オブジェクトのパージング・メソッドに渡される．その結果，

・login/logout 処理メソッド
・Plan や Product の格納メソッド
・システムエージェントからのメッセージを処理するメソッド

などが起動される．いずれも核心の処理は他のオブジェクトのメソッドを呼ぶことによって達成される．

KyotoDB カーネルから外への出力も Project オブジェクトが窓口となる．そのためのメソッドは次のとおりである．

・ユーザエージェントに対してコマンドを発信（ソケットに書き込み）するメソッド
・Coordinator による関係解析などの結果，作業者に対して非同期的にメッセージを送る（sendmail を起動する）メソッド
・バックエンド・データベースとのデータのやりとりを行う（バックエンド・マネジャを子プロセスとして起動する）メソッド
（バックエンドからの読み込みは現状ではカーネル起動時に1度だけ行われ，すべてのオブジェクトが読み込まれる．書き出しはオブジェクトのいずれかが変更される度に行われ，一貫性を保っている．）

他の Project オブジェクトのメソッドは，

・指定されたプロダクトを担当している作業者名
・指定されたプロダクトに関係している Coordinator オブジェクト

など，Plan や Coordinator から来る問い合わせに答えるメソッドである．これらはさらに，下位オブジェクトのメソッドに頼る．

Project オブジェクトのインスタンス変数は，構成複合を維持する Plan, Coordinator へのポインタのほか，管理者のプロファイルを保持する．

5.4 Plan オブジェクト

Plan オブジェクトは，1つのプロダクト単位を開発保守する1人の作業者のプロセス，すなわち単位作業を支援，維持管理する．そのインスタンス変数は次のとおりである．

・作業名，作業概要
・参照プロダクト（参照複合の Product オブジェクト）
・プロジェクト内で承認された目的プロダクト（1個）
作業者が独自に編集中の目的プロダクト（1個）
編集中の関係記述（参照プロダクトと同数）
(以上は構成複合の Product オブジェクト)
・プロセス記述
・作業者名，作業者のパスワード
・作業者スキル
・作業量見積り
目標作業時間，目標開始時刻，同終了時刻
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

Plan オブジェクトの主要なメソッドは2つある．1つは enact であって，Project の login メソッドから login に成功すると直ちに呼び出される．これはプロセス記述を解釈，実働化するものであるが，現状では参照プロダクトのブラウザと目的プロダクトおよび関係記述のエディタをユーザエージェントに起動させるにとどまる．ただし，参照プロダクトはこの単位作業を開始する前提条件となっているため，どの参加プロダクトも承認された状態になければ，作業は開始されない．ブラウズ／エディットに用いるツール，およびそのツールに合う可視ファイルは Product オブジェクトで指定されている．

もう1つの主要メソッド commit は，編集した関係記述をリリースする（Coordinator に登録する），あるいは編集したプロダクトを完成した目的プロダクトとしてリリースするため，作業者の判断によりユーザエージェントを通じて発行されたコマンドを処理する．後者の場合，該当する Coordinator（この目的プロダクトを参照している Plan の数だけある）の波及

解析メソッドを呼ぶ.

ほかに,

- 複合連鎖で呼ばれるインスタンス格納/保存メソッド
- 自分のもつインスタンス変数に関して Project からの問い合わせに答えるメソッド

などのメソッドがある.

5.5 Product オブジェクト

バイトストリームとしてファイルに格納してあるプロダクトでは，決して意味単位の識別，追跡はできない．プロダクト要素間の意味的関係を管理するためには，仕様図面や文書等のプロダクトを，その構成と個々の意味内容が識別できる細粒度の単位に区分し，アクセスできなければならない．ソフトウェアを構成するプロダクトの大部分は，次の３つの構造で意味識別の可能な形に表現，格納できる.

- ネットワーク構造
  例：状態遷移図，データ／制御フロー図，システムブロック図
  単位データ：ノードの型，ノードの値，アークの型，アークの値，接続関係
- 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表現など）
  単位データ：ノードの型，ノードの値，接続関係
- 表構造
  例：決定表，入出力対応表，メモリマップ，種々の一覧表
  単位データ：属性名，属性値

これらの構造が図４に示したように Product を継承する型となっている．Network は図４のとおり，Node, Arc オブジェクト（それぞれ型と値をインスタンス変数として持つ）を構成複合で保持し，接続関係は Arc がその元と先の Node を参照複合で持つことにより表現される．Tree は再帰的な構成複合構造で構成される．Table は原子データを直接保持するオブジェクトを２次元配列で持つ．ノードがまた構造をもつ場合は，プロダクトを分け，それらの関係を管理することで対処する．スーパクラスである Product では，プロダクト種別，および自分をブラウズ／エディットするツールやそのツール固有の可視ファイルの所在（マシン名およびパス名）を保持している.

Product オブジェクトでは文献12)に示された分類に従うと，ソフトウェア・プロダクトの意味的関係の解析に必要な以下の情報を保持しており，表現にかかる情報は個々のツールの側に任せている.

- 概念情報：アクション，データなどソフトウェアを構成する素子的な概念に関する情報
- 関係情報：概念情報間の関係に関する情報
- 局面情報：概念，関係情報の部分集合で，状態遷移，データフローなど特定のビューでソフトウェアをみたときの情報

主要なメソッドはインスタンス格納／保存，および要素検索／参照である．前者は Plan オブジェクトの動作に従い，後者は関係記述の作成や Coordinator における波及解析に利用される.

5.6 Coordinator オブジェクト

Coordinator オブジェクトは２つの Product 間の関係を管理する．各 Plan オブジェクトの参照プロダクトの数をすべての Plan に渡って総計した数だけある．２つの Product を参照複合で持ち，それらの関係記述（Table 型 Product）を構成複合で保持する．関係記述は関係するプロダクト要素（アーク，ノード等）の対応表で，要素データの属性，接続関係や値による対応の同定，変換規則，他の関係との関係などが記述されている.

最も重要なメソッド findripple は Plan の commit メソッドから起動される．承認を得ようとしている編集中のプロダクトについて，最も最近に承認された版との差分を抽出して関係するプロダクトと比較し，関係記述を参照しながら，波及解析を行う．波及が見つかると，協調活動を起動してその中核として動くが，これについては７章で述べる.

6. KyotoDB とツールとの相互接続

6.1 データ統合―オブジェクト・ストリーム―

３章で述べたように，SEDB とツールの相互接続のためには，CASE データの意味を移転させる何らかの言語が必要である．現在 KyotoDB で用いているのは，その複合オブジェクト構造を忠実に反映しストリーム化したものである．これは，ツールとの接続とともに，永続オブジェクトを保持するバックエンドとカーネルとのデータのやりとりにも共通して使えることを考慮したものである．STL や CDIF のように，実プロダクトに対応し固定した言語要素で情報パケットを組むものではないので，プロダクトだけでなくプロセスやプロジェクト構造全体，あるいはその一部を取り出して移転することもできる．プロダクトの部分

```
<Ostream> ::= <boo> <class name> : <oid> { <Ostream> } <eoo>    ; 構成複合
            | <bol> <class name> : <oid> : <loid>              ; 参照複合
            | <boa> <type> <var name> <value>                  ; 原子データ
            | <del> <class name> : <oid>                       ; オブジェクトの消去

    <boo> ::= @    ; begin of object
    <eoo> ::= /    ; end of object
    <bol> ::= &    ; begin of object link
    <boa> ::= !    ; begin of atomic data
    <del> ::= *    ; delete object

    <oid> ::= <インスタンス変数名> { + <index> }
    <loid> ::= { ^ } <domain>
        <domain> ::= <oid> { . <oid> }
            /* ＫｙｏｔｏＤＢの場合, <loid> は次のいずれか.  */
            /*  ^ <oid> . <oid> (Product の参照複合)       */
            /*  ^ <oid>         (node の参照複合)          */
    <type> ::= d | c | s | f
            /* データ型を示す1バイト. 順に整数, 文字, 文字列, 実数.  */
    <var name> ::= <インスタンス変数名>
    <value> ::= " <文字列> " { ; " <文字列> " }
            /* 繰り返しは整数, 実数の配列の場合.  */
```

図 5 オブジェクト・ストリームの構文定義
Fig. 5 Definition of object stream.

に関しては，STL や CDIF の意味要素をインスタンス値として定義することにより，それらからオブジェクト・ストリームへの変換が可能である．

オブジェクト・データモデルに従う世界とそうでない世界でデータをやりとりするためには，アドレスの保存，復元を解決することが最も重要である．**図5**で定義するオブジェクト・ストリームでは，オブジェクトの局所性を保つため，外界におけるオブジェクトの同定をユニバーサルな ID によらず，ドメイン・アドレッシングのような局所化の可能な方法をとっている．また，構成単位としては複合構造オブジェクトのどの部分でも独立して扱えるようになっている．

〈boo〉…〈eoo〉で1つのドメイン（オブジェクト）を構成し，〈oid〉は上位ドメインにおけるそのオブジェクトのアドレスである．例えば，ある Network オブジェクトのデータを外部から表現する場合，

@Project : cruise@Plan : plan+5@Network : product…@Node : node+12…/…///

のようになる．ここで cruise は Project オブジェクトの（それが複数ある場合を想定しての）識別名であり，plan+5, product などはインスタンス変数 plan〔5〕, product などを指示している．〈loid〉はそれを含むドメインのいくつか（KyotoDB の場合1つ）上からの相対ドメインアドレスで解決できる．参照複合においても構造的な階層関係は存在するので，この方法で一般性は失わない．

図6にオブジェクト・ストリームの実例を示す．1つのプロジェクトの構造の一部を示しており，Plan, Coordinator はそれぞれ1つだけ現れている．それらの内部もかなり省略されているが，目的プロダクトの内部は一部示されている．

オブジェクト・ストリームによるプロダクトの意味記述（すなわち Product オブジェクトの構成）は，局面情報に含まれる構造（ネットワーク構造など）を一般化してオブジェクト構造とし，その中の特定のインスタンス変数で概念や属性を表している．例えば状態遷移図，データ／制御フロー図のようなネットワーク型プロダクトは，**図7**のように記述される．オブジェクトをもつインスタンス変数 nd, ac, source, sink で構造を形づくり，特定の文字列値をもつインスタンス変数 attribute で意味要素を識別する．

6.2 制御統合

KyotoDB ではプロセスもプロダクトと有機的に関係づけて管理しており，作業やツールの制御を行う．これをカーネルとの間にたってインタフェースするのが KyotoDB ユーザエージェントである．カーネルから見て双方向の制御コマンド言語を定めており，ユーザエージェント，システムエージェントや KyotoDB

```
@Project:cruise
 !smanagername"yhm"
 @Plan:plan+1
  !sworkname"make DCFD"
  @Network:product
   !sviewfile"sirius:/mnt/KyotoDB/demo/dcfd.cv"
   !stoolpath"/mnt/KyotoDB/bin/canadcfd"
   @node:nd+0!sattribute"Action:tp(data)"!svalue"Mntr_Brake"/
   @node:nd+4!sattribute"Action:tp(control)"!svalue"Cruise_Ctrl"/
   @node:nd+15!sattribute"DataStore"!svalue"Desired_Speed"/
   @arc:ac+0!sattribute"Carrier:ct(controlflow)"!svalue"Brake"/
       &node:source:^nd+0&node:sink:^nd+4/
   @arc:ac+26!sattribute"Carrier:ct(dataflow)"!svalue"Thrtl_Pos"/
       &node:source:^nd+18/
  /
  &Network:reference+0:^plan+0.product
  @Table:relation+0
   !sviewfile"sirius:/mnt/KyotoDB/demo/rel.std-dcfd"
   !stoolpath"/usr/local/bin/GNU/nemacs"
  /
  !smembername"ajisaka"
  !dquant"1500"
  !dcost"250"
 /
 @Coordinator:coordinator+0
  &Network:former:^plan+0.product
  &Network:latter:^plan+1.product
 /
/
```

図 6 オブジェクト・ストリームの実例（一部）
Fig. 6 A part of object stream example.

```
ネットワーク型プロダクトのインスタンス変数定義
  { @node:nd+<n>!sattribute"<attribute>" [ !svalue"<name>" ] / }
  { @arc:ac+<n>!sattribute"<attribute>" [ !svalue"<name>" ]
            [ &node:source:^nd+<n> ] [ &node:sink:^nd+<n> ] / }
        <n> ::= 0 | 1 | 2 | ...
        <name> ::= <文字列>
        <attribute>: 下記参照

状態遷移図，データ／制御フロー図の要素の属性名（IEEE P1175/D7 準拠）
  for STD
      node "ActionState"
      arc  "Event"
  for DCFD
      node "Action:tp(data)"
      node "Action:tp(control)"
      node "DataStore"
      arc  "Carrier:ct(dataflow)"
      arc  "Carrier:ct(controlflow)"
            notes: tp stands for 'has transformation purpose'
                   ct stands for 'is carriertype'
```

図 7 ネットワーク型プロダクトのオブジェクト・ストリーム
Fig. 7 Definition of object stream elements for network type products.

ツールからカーネルへのKコマンド，カーネルからユーザエージェントへのUコマンドがある．

6.2.1 Kコマンド

• plan

プロセス計画フェーズで作成された Plan 群をオブジェクト・ストリームで KyotoDB ツールからカーネルに送り込む．管理者の識別名とパスワードも引数に含まれている．カーネルは立ち上げ時に管理者プロファイルを入力しており，これによって権限を検査する．

• login

ユーザエージェントは起動するとまず login ウィンドウを開く．これに従って作業者が login すると，ユーザ名とパスワードをもってこのコマンドをカーネルに送る．

• logout

ユーザエージェントのlogoutボタンをクリックするとこのコマンドが送られる．カーネルはその作業者に対応する Plan の作業管理指標データをバックエンドに保存する．

• store

ユーザエージェントはUコマンド edit（後述）で送られてきたエディット対象ファイルを監視しており，タイムスタンプが変わる（セーブされる）と，そのファイルをオブジェクト・ストリームに変換するプログラム（これはツールのデータフォーマットごとに必要）を起動して，このコマンドでカーネルに送り込む．これにより，ツールが扱う表現情報も含んだ可視ファイルとカーネル内のプロダクトの情報との一貫性を保証して

いる．

カーネルにおいて作業者の識別は Project オブジェクトで管理しているソケット番号とユーザ名の対応表により，またファイルの識別（目的プロダクトか関係記述か）はコマンドの引数による．

・commit

ユーザエージェントの commit ボタンをクリックするとこのコマンドが送られる．ファイルの識別（目的プロダクトか関係記述か）はコマンドの引数による．

・reply

作業者が発信したメッセージのうち，カーネルのもつ情報に影響を与えるものが，システムエージェントを経由して送られてくる．

6.2.2 Uコマンド

・loginresult

login の成功，不成功を通知する．

・edit

エディタ起動コマンド．起動するツールとそのツール固有の可視ファイルの所在（マシン名およびパス名）を引数で送る．

・browse

ブラウザ起動コマンド．引数は edit と同様．

・commit

カーネル内のコミット動作（「編集中の目的プロダクト」を「承認させた目的プロダクト」にコピーする）に対応して，可視ファイルのコピーを行わせる．

・refresh

新たにコミットが成功した場合，それを参照しているブラウザを立ち上げ直し，最新の参照プロダクトを見せるようにする．

・report

login しているユーザに同期的にメッセージを伝える．非同期的な通信は，次章で述べるように電子メイルシステムによる．

7. グループウェアとしての KyotoDB

前章で述べたUコマンド report や refresh もグループウェアの性格をもっているが，作業者間のコミュニケーションと協調活動は，非同期性を必要とする場面が多いため，主として電子メイルによっている．協調活動の発端となるのは，Plan の始動条件，参照関係にあるプロダクトの状態や関係解析などに基づいて発信されるカーネルからのメッセージ(電子メイル)である．カーネルにはコミュニケーションモデル要素[17]が内包されている．役割記述は Plan オブジェクトに，メッセージ記述やその選択規則と機能記述はすべて Coordinator オブジェクトに含まれている．

作業者側にはユーザエージェントに含まれる（から起動される）メッセージ処理ツールがあって，作業者のメイル・スプールから KyotoDB に関わるメッセージを拾い上げて表示する．さらにメッセージの内容によって，

・応答メッセージのテンプレートを用意し，編集窓を開ける．

・指定されたプロダクトを表示する．

・指定されたシェルコマンドを実行する．

といったグループウェアの機能をもつ．

作業者が返信した応答メッセージが，波及効果の解決などカーネルのもつ情報に影響を与える場合，そのメッセージは関係する作業者とともに KyotoDB にも宛てられ，システムエージェントがこれを受けとってカーネルに送る．これは Coordinator で処理され，さらに次の協調活動を引き起こす．

KyotoDB がグループウェアとして動く典型的な例をいくつか以下に示す．

・作業者が KyotoDB に login した際，自分が参照すべきプロダクトが未完成の（コミットされていない）ときは，その旨レポートされ，そのプロダクトを担当している作業者にメッセージが送られる．複数の参照プロダクトの1つでも完成したものがあれば作業は開始できるが，すべて未完成であれば作業は開始されない．

・目的プロダクトをコミットしようとする際，そのプロダクトに関わる関係記述が未完成であればコミットは成功せず，その関係記述を担当している作業者にメッセージが送られる．

・目的プロダクトをコミットしようとする際，波及効果が発見されると，関係する作業者すべてに協議を促すメッセージが送られる．このメッセージはまたコミットされようとしているプロダクトを表示させる．

・協議の結果，変更が認められる場合，それを通知する応答メッセージは関係する作業者とともにシステムエージェントにも送られる．カーネルは波及を受けた側の作業者に refresh コマンドを発行し，コミットを試みた作業者に対しては，不在の場合を考慮してメイルによりUコマンド commit と同等の効果を与える．（Uコマンド commit は，波及効果がなくコミットが直ちに成功した場合に用いられる．）

8. 評　　価

KyotoDB はプロセスとプロダクトの統合，および開放的 CASE プラットフォームという2つの目的を実現した．オペレーティングシステムとユーザインタフェースに対する標準的な仮定 (Unix と X-window) さえ満たされれば，接続するツールを選ばずに稼働できるので，組織独自の環境の使い勝手には一切影響を与えない．新たなツールを接続する場合に唯一手を加えなければならないのは，そのツールのデータフォーマットからオブジェクト・ストリームに変換するプログラムを用意することだけである．そのプログラム名はプロセス記述の中のツールに関する記述から導かれ，ツール自体や KyotoDB 自体に手を入れる必要はない．

一方，このようにして KyotoDB に接続されたツールは，それによって開発保守されるプロダクトが細粒度で KyotoDB に管理される．プロダクトは単に電子化されただけの孤立したものではなくなり，異なるプロダクト要素間の意味的関係が管理されて，ソフトウェア開発保守の作業フェーズをつないでシームレスな支援を可能にする．また，プロセス定義や協調活動モデルがカーネルの複合オブジェクト構造の中でプロダクトと有機的に組み合わされて，個々の作業者のプロセスやグループの活動に対する支援のサービスが付加される．

これらプロセスおよび協調活動に関連するサービスそのものの性能は，カーネル内の処理も軽く，またカーネルから分離され作業者ごとに用意されたユーザエージェントやメイルシステムを用いるので，Plan の数が 20 程度あるいはそれ以上になっても問題はない．KyotoDB 固有の機能の性能が最も影響を受けるのはプロダクトの規模であろう．とくに現状ではカーネルのメモリ管理およびバックエンド・データベースのチューニングが不十分である．

9. おわりに

KyotoDB は現時点では今すぐ生産現場に持ち込めるほどに充実するには至っていないが，次世代のソフトウェア生産環境に関する研究を実証的に進めるためのベースラインとして使えるようになっている．これを使って推進，発展させるべき研究課題のいくつかを以下に挙げる．

- プロダクトの意味構造について，局面情報にひきずられずに，概念情報と関係情報を基礎となる情報として管理し，それらから観点写像によって個々の局面情報を導出するしくみ[18]．
- 抽象レベルが高くかつ一般的な制御統合．PCTE の用意するプロセス群はまだオペレーティングシステムのレベルに近く，また別のツール統合標準化の試みである ATIS[19]でもツールの起動制御のみにとどまっている．
- 細粒度プロセスの記述と実働．
- コミュニケーションモデルの記述と実働．
- プロダクト間の差分抽出（静的な抽出あるいはプロダクトに対する操作列解析によるもの）および関係解析，波及解析のアルゴリズム．
- 細粒度プロセスとプロダクトの状態に基づく正確な管理指標の実用．
- プロセス計画フェーズの支援．
- 自己反映アーキテクチャの強化，およびそれによるプロセス変更支援．

謝辞　KyotoDB カーネルは，日本アイ・ビー・エム(株)より提供頂いたオブジェクト指向言語処理系 COB により当初開発された．KyotoDB に接続するツール例として作成したプロダクト・エディタは，日本電気(株)より提供頂いたユーザインタフェース構築ツール鼎および「ゆず」を用いている．オブジェクトベースの研究には三菱電機(株)より提供頂いたデータベース・システム HDM を利用している．以上，ここで謝意を表します．また，KyotoDB および接続ツールの開発には京都大学工学部情報工学教室の沼田賢一（現在，富士ゼロックス(株)），山下薫（現在，東洋エンジニアリング(株)），沢田篤史，満田成紀，西尾修一，沢田宏，宗続俊彦，菅原達也，京都高度技術研究所の藤田充の各氏に協力を頂いた．ここに謝意を表します．

参考文献

1) Ward, P. T.: The Transformational Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, *IEEE Trans. Softw. Eng.*, Vol. SE-12, No. 2, pp. 198-210 (1986).
2) Gomaa, H.: Structuring Criteria for Real Time System Design, *Proc. of 11th Int'l. Conf. Softw. Eng.*, pp. 290-301 (1989).
3) Rumbaugh, J. et al.: *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs (1991).
4) 佐原　伸，中谷多哉子，藤野晃延：オブジェク

ト指向 CASE の動向, 情報処理学会「オブジェクト指向ソフトウェア技術シンポジウム」, pp. 87-116 (1991).
5) 本位田真一, 伊藤 潔: OOA/OOD の上流 CASE, 日本ソフトウェア科学会「90年代の CASE」, pp. 15-29 (1991).
6) Matsumoto, Y. and Ajisaka, T.: A Data Model in the Software Project Database KyotoDB, *JSSST Adv. Softw. Sci. Tech.*, Vol. 2, pp. 103-121 (1990).
7) 竹下 亨: CASE の発展と見通し, コンピュータソフトウェア, Vol. 7, No. 3, pp. 2-22 (1990).
8) 竹下 亨: CASE 概説, 共立出版, 東京 (1990).
9) Levy, L. S.: A Metaprogramming Method and Its Economic Justification, *IEEE Trans. Softw. Eng.*, Vol. SE-12, No. 2, pp. 272-277 (1986).
10) 松本吉弘: 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, 情報処理学会論文誌, Vol. 30, No. 9, pp. 1154-1163 (1989).
11) ECMA: Portable Common Tool Environment (PCTE) Abstract Specification, Standard ECMA-149 (1990).
12) IEEE Computer Society's Task Force on Professional Computing Tools: A Standard Reference Model for Computing System Tool Interconnections, P 1175 Trial-Use Standard (1991).
13) EIA CASE Data Interchange Format Technical Committee: General Rules for CDIF Syntax and Encodings, EIA-PN 2389 Partl, Draft, 1.02 (1990).
14) 鯵坂恒夫: 情報資源辞書と意味移転言語, 日本ソフトウェア科学会「90年代の CASE」, pp. 133-145 (1991).
15) ISO/IEC: 10027 Information Resource Dictionary System (IRDS) Framework (1990).
16) ISO/IEC JTC 1/SC 21: N 4895 Information Resource Dictionary System (IRDS) Service Interface, Working Draft, Revision 11 (1990).
17) Uta Pankoke-Babatz (ed.): *Computer Based Group Communication—The AMIGO Activity Model*, Ellis Horwood, Chichester (1989).
18) 鯵坂恒夫, 大森 匡, 松本吉弘: 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とその実現, オブジェクトテクノロジーの高度応用に関するObase ワークショップ論文集, pp. 89-96 (1992).
19) ANSI X3H4: Information Resource Dictionary System ATIS—A Tools Integration Standard, Working Draft (1990).

(平成4年2月10日受付)
(平成4年9月10日採録)

鯵坂 恒夫(正会員)
1956 年生. 1980 年京都大学理学部物理学系卒業. 82年同大学院工学研究科情報工学専攻修士課程修了. 85 年同博士課程研究指導認定退学. 同年, 京都大学工学部情報工学科助手. 90 年より同助教授. 工学博士. ソフトウェア工学, データ工学, 情報システム工学に関する研究に従事. ソフトウェア設計自動化, プログラム自動生成などの研究を経て, 最近はとくに開放型 CASE 環境に関する研究を行っている. 日本ソフトウェア科学会, システム制御情報学会各会員.

松本 吉弘(正会員)
1932 年生. 1954 年東京大学電気工学科卒業. 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 年京都大学工学部情報工学科教授(情報システム工学講座担任), 現在に至る. この間, 工業用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, ソフトウェア開発支援環境の研究などに従事. 日本電機工業会進歩賞, 発明協会全国発明表彰発明賞, 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.「計算機制御システム」,「ソフトウェア工学」など著書13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.