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

ソフトウェア要求定義・分析における オブジェクトマネージメントシステム

加藤 康人
(株)PFU

ソフトウェアの要求定義・分析を支援するツールとして、プロトタイピング手法を用いたPrototyper's WorkBench(PWB)の研究開発をハワイ大学と共同で行った。このPWBのオブジェクトマネージメントシステム(OMS)として筆者らはオブジェクト指向をベースとした情報格納庫であるHyperMedia系システム(ActiveBook)をSmalltalk-80上に実装した。本論文ではActiveBookの機能とソフトウェア要求定義・分析における有用性について考察する。

Object Management System for Software Requirement Analysis

Yasuhito Kato
PFU Limited

Prototyper's WorkBench(PWB) is integrated tool environment to support softoware requirements analysis and specification using prototyping approaches. The information repository, ActiveBook based on Object oriented and Hyper media techniques is developed for the Object Management System of PWB. ActiveBook is implemented in Smalltalk-80 environment. This paper describes the functionalities of ActiveBook and its availability for software requirement analysis and specification.

1 はじめに

コンピュータの運用環境は複雑化かつ大規模化しており、ソフトウェアに求められる機能も高度化かつ多量化の傾向にある。このような環境の変化にともなうソフトウェアへの要求定義は非常に難しく曖昧なものになり、初期の段階での完全無欠な要求定義の作成は不可能といえる状況にある。この解決策として要求定義段階でシステムの動作イメージを提示し、ユーザの意図を再確認する手法としてプロトタイピング方式が注目されている。ハワイ大学の宮本教授と共同で研究開発した Prototyper's Work-Bench(PWB) は、このプロトタイピング方式を用いて要求定義・分析を支援し、ソフトウェアの品質と生産性を向上させることを狙ったツールである [1]。

このPWBで扱うプロダクト (ユーザからのソフトウェア要求やPWBのツールを用いて記述したシステムのモデルなど) やそれらのプロダクト間の関連をすべて電子化して格納し、それらを任意に操作し管理できるオブジェクトマネージメントシステム (OMS) として、オブジェクト指向をベースとした情報格納庫である HyperMedia 系システム (ActiveBook) を実装した。本稿ではまずPWBを用いたソフトウェアの要求定義・分析の手順について説明し、次にそれらを支援するために必要な OMS の機能について、そして ActiveBook においてそれらの機能がどう実現されているかを説明し、最後にその有用性、今後の課題について述べる。

2 PWBによる要求定義・分析

PWBの特長は、ユーザの要求が100%明確でない段階でもそこでいったん要求定義を行い、それで満足するシステムモデル (プロトタイプ) を迅速に構築し、そのモデルの動きをユーザと開発者が確認しながら両者の合意のもとに仕様を固めていくことである。これによりシステム開発の前段階でユーザの要求を正しく反映でき、また要求定義に含まれている種々の問題点を洗いだすことにより、要求定義の不明確事項が開発の途中段階や終了直前段階まで検出されないといった状況を排除し、それによりソフトウェア開発のロスタイム / ロスコストの発生を最小限に食い止めることができる。

PWBを用いた要求定義・分析の手順を次に示す。

1. 要求の文章化とモデル化

   ユーザからのシステム要求を文章化し、それらの要求をどの視点から分析すべきか、どこまで記述するかを検討した後、システムのモデル化を行う。PWBではシステムを6つの異なる視点から分析し、モデル化することができる。この時点で曖昧な文章は形式化されシステムの静的な側面からの分析を行う。

   PWBの6つのモデル (視点) を次に示す。

   (1) FM(Function Model)

   システムに必要とされる機能、属性、支援機構を記述

   (2) FSM(Functional Structure Model)

   システムの各機能間の依存関係、機能とオブジェクトの関係を記述

   (3) UIM(User Interface Model)

   システムとユーザとのインタフェース、入出力画面 (データ) を記述

   (4) EOPM(External Operational Profile Model)

   システムを外部から見て様々なイベントに対しシステムの状態がどのように遷移するかを記述 (状態遷移図)

   (5) IOPM(Internal Operational Profile Model)

   システム内部のプロセスの制御構造を記述 (ペトリネット)

   (6) PM(Performance Model)

   システムの性能、リソースを記述 (キューイングネットワーク)

システム要求をこれらのモデルにモデル化するためのツールがモデリングエディタである。モデルは、ERA(Entity - Relation - Attribute)モデルをベースとし、Entity(ものごと)をアイコン、Relation(ものごとの関係)をアイコン間の矢印線、Attribute(ものごとや関係の性質)をテキストとして図形表現される。

2. モデル(プロトタイプ)の実行
   複数の視点からモデル化した個々のモデルで関連のあるモデルをお互いに関連付け、実行(アニメーション)する。モデルの動きを見ることにより、システムの全体像や全体と部分の関係、システムの振る舞いなど動的な側面からの分析を行う。

3. モデル(プロトタイプ)の洗練
   上記の過程において要求に対する漏れ、矛盾などが検出されればそれらを排除し、ユーザと開発者が満足するシステムのモデル化が行えるまでモデルを修正/追加し洗練する。

PWBを用いた要求定義・分析手順を図1に示す。

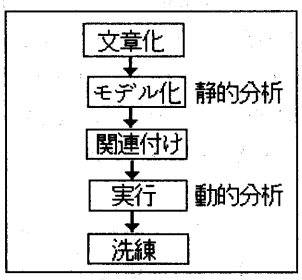

図 1: PWBを用いた要求定義・分析手順

3 求められるOMSの機能

PWBによる要求定義・分析を支援するためのオブジェクトマネージメントシステムを考える上で重要なことは、より曖昧性の高い様々な形態の情報、つまりより抽象度の低い情報(種々の文章や絵、図が混在したシステムの要求書やそれらをモデル化したシステムモデル群など)をどのような枠組の中に格納するのか、そしてそれらを扱うためにどのような操作を可能にするかである。これらを従来のデータベースの枠組の中で扱うには無理があり、柔軟なデータ構造が扱え、ユーザビューが動的に変化できるようでなければならない。また、マンマシンインタフェースはビジュアルなオブジェクトが操作でき、オブジェクトはマルチウインドウを使って複数個のオブジェクトを表示でき、操作できる必要がある。以下にOMSに求められる機能について述べる。

3.1 多種のプロダクトの管理

PWBで扱うプロダクト(情報)は、文章(テキスト)、絵、種々のシステムモデル記述(ダイアグラム)、モデルの実行結果(グラフ)など多岐に渡る。また、それらのプロダクトを編集するための各種のツール(テキストエディタ、イメージエディタ、モデリングエディタなど)が存在する。これらのプロダクトやツールを統合して統一的に蓄積、操作できるためのオブジェクト指向をベースとしたマルチメディア環境が必要となる。

3.2 プロダクトの関連付け

システムのモデル化を行う段階で重要なことは、任意の段階でシステムの部分をより詳細化して(トップダウンに)モデル化したり、逆にある部分をより抽象化して(ボトムアップに)モデル化したりといったことが柔軟に行える必要がある(モデルの階層構造化)。また、PWBではシステムの外部から見た振る舞いをモデル化したり、またシステム内部から見た構造をモデル化するというように1つのシステムを視点を変えてモデル化することができる(マルチビュー)。この様子を図2に示す。

複数のモデル間に関連があればそれを明確に関連付けられなければならない。このためにはPWBのプロダクトをノードとみなし、それらの間にリンクを張りノードとリンクからなるネットワークで構成するようなハイパーメディア環境により、システム全体のモデル化が行え、ノードに付加されたリンクを辿って自由に各プロダクトにアクセスできることが必要となる。

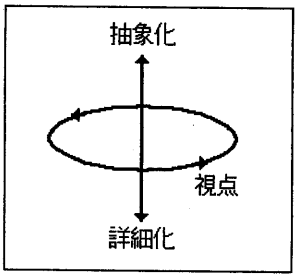

図 2: モデル化のマルチビュー

3.3 柔軟なデータ構造とユーザインタフェース

PWBでは、システムをモデル化し静的 / 動的分析を経てユーザと開発者がお互いに納得できる要求仕様を固めていく。ここで重要なことは、ある程度の段階でシステムのモデル化を行いそれらをもとにモデルを追加したり、修正したりして徐々にプロトタイプを作り込んでいく過程である。ここではまず断片的なプロダクトが存在し、それに対してプロダクトを追加、修正したり、それらの関連を付けたり、それらのプロダクトの構造化や分類が容易に行えなければならない。プロダクトの構造化は、あらかじめ枠組を決めておきそこに格納する方法と、断片的なプロダクトから枠組を構成するような2つの方法が行えるような柔軟な構造が必要である。またプロダクトあるいは構造化、分類されたプロダクト群は視覚化されていて、直接そのプロダクトに対し操作できる (ダイレクトマニュピレーション) ことが必要である。視覚化されたプロダクト自身もユーザが直感的にその意味を認識できるようにカスタマイズできることも重要となる。

4 ActiveBook

PWBの情報格納庫であるHypermedia系システムActiveBookをSmalltalk-80上に実装した。以下にこのActiveBookの機能についてPWBで扱うプロダクトを中心に説明する。

4.1 プロダクトの構成

ActiveBookで扱うプロダクトはシートとエリアという基本単位で構成される。シートの属性にはPWBの要求定義・分析に必要なテキスト (文章、キャラクタベースの文字列)、イメージ (ビット列からなるデータ、イメージスキャナから読み込んだデータなど)、モデル記述 (モデリングエディタで図形言語を用いて記述したシステムモデル) がある。シートの属性によりそのシートに記述できる内容が決まる。エリアとはある領域を表し、その属性としてシンプルエリア (シート上のある領域)、ボタン (それ自身がある機能を持ったアイコンエリア) や各種のツール (ツールを起動するためのツールアイコン) などがある。後述のプロダクト間の関連付けはこれらのシートあるいはエリアの間で行うことができる。
また、シートやエリアの基本単位をあるまとまりとして扱うための上位構造として本、キャビネットあるいは部屋 (プロジェクト) がある。つまり、本は何枚かのシートの集まりであり、キャビネットは何冊かの本やいくつかのボタン、ツールなどの集まりであり、部屋 (プロジェクト) は本、キャビネットなどの集まりから構成される。この上位構造を用いて各種プロダクトを構造化、分類してActiveBookに格納、操作することができる。

4.2 プロダクトの生成

ActiveBookで扱うプロダクトは視覚化されたアイコンで表現される。システムにはユーザが自由にアイコンを定義でき、カスタマイズできるようにtoolboxが用意されている。このtoolboxを使って定義したアイコンに対し、シート、本、キャビネット、部屋(プロジェクト)などの属性を与えることにより実際にプロダクトが生成される。キャビネットには何冊かの本を格納

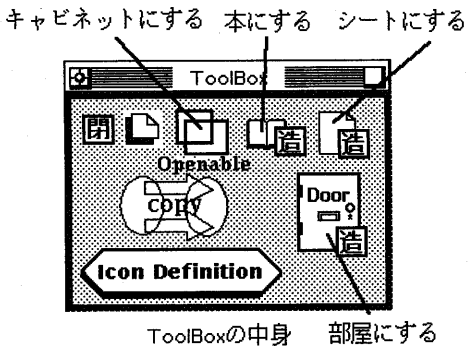

図 3: toolboxの中身

でき、本は何枚かのシートの集まりとして構成される。ユーザはあらかじめこれらのキャビネットや本を作っておき(作ったばかりのキャビネットや本には何も入っていない)、本に何も書いていないシートを何枚か挿入しておきモデリングエディタなどのツールを使って内容を記述していくこともできるし、先にツールを使って内容を記述した後でそれらをシートとし、分類してまとめて本にしてキャビネットにしまうこともできる。

また部屋をいくつか用意しておき、ある部屋では画面設計などユーザインタフェースのモデル化を行うために必要なツールやそのプロダクトを、別の部屋には内部構造のモデル化やアニメーションを行うために必要なツールやそのプロダクトを置いておくことにより、それらのプロダクト(本やツール)のウィンドウを閉じたりしなくても部屋を移るだけでもとの作業を行うことができる。プロダクトのウィンドウを開き過ぎて混乱することもない。もちろん別の部屋とのプロダクトやツールのやりとりは自由に行えるので、モデルを実行したり、関連付けを行うときには関連するプロダクトを1つの部屋にまとめて作業を行えばよい。このように柔軟度の高いデータ構造の上に自分に適したビューを実現することが可能である。図4にSmalltalkで実現したプロダクトのClass構造を示す。プロダクトはすべて*Area*というクラスであり、キャビネットの場合はそのインスタンス変数である*subAreas*に本やキャビネットが格納され、本の場合にはシートが格納される。

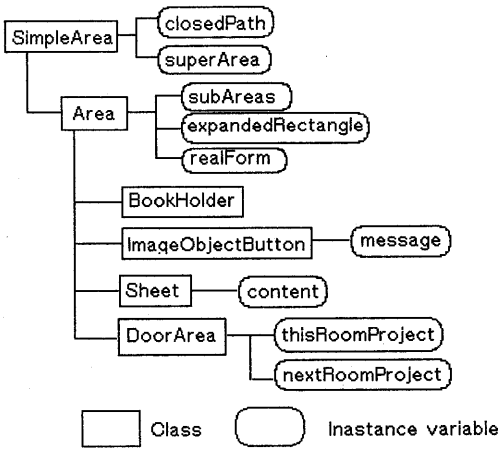

図 4: プロダクトのClass構造

4.3 プロダクトの操作

各プロダクトには固有の操作があり、そのプロダクト(のアイコン)に直接マウスで触れるという統一的な動作によりその操作を実行できる。たとえば、キャビネットをクリックするとその中に格納されている中身のビューが現われ、本をクリックすれば本が開き、部屋をクリック(ノック)すればその部屋に入れるなど。本をキャビネットにしまったり、プロダクトを別の部屋に移動するには、その本やプロダクトをマ

ウスでドラッグしてキャビネットや部屋のアイコンの上に重ねれば (置けば) よい。

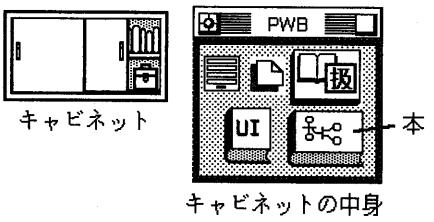

図 5: キャビネットと中身

4.4 シートの作成

本の中に格納するシートの属性には、テキスト、イメージ、モデル記述などがあり、シートの作成はどの属性のシートを生成するかを指定して行い、そのシートの内容の記述には各種のツールを使用する。各種のツールはそのシートの属性により自動的に選択され起動される (たとえばモデル記述用のシートに対してはモデリングエディタが自動的に起動される) ので、シートの内容を記述するためにどのツールを使うかを意識する必要はない。PWB のモデルの実行機能を使ってモデルを動かしてみて、必要なところを修正してすぐまた実行したり、ということが容易にできる。また、直接ツールを指定し、モデルなどを記述した後でそれらをシートに変換して扱うことも可能である。とりあえずモデリングエディタで画面の設計などを行っておき、それらをシートに変換し、あとでまとめて本にするというようなことも可能である。

本の中のシートは、その中の 1 枚を本から剥したり、それを別の本に挿入したりといった操作も直接そのシートを別の本の任意のページに重ねる (置く) ことで行うことができる。この場合もシートの属性 (テキスト、イメージなど) を意識せず同じように扱うことができる。もちろん本ごと別の本に重ねて 1 冊の本にまとめることも可能である。図 6に Smalltalk で実現したシートの Class 構造を示す。

Sheet のクラスのインスタンス変数である *content* にはそのシートの属性により CachedText、CachedForm、CachedModel のいづれかのインスタンスが格納される。クラス *CachedThing* はこれらの抽象化クラスであり、実際のデータはそのインスタンス変数 *thing* に入る。データはキャッシングメカニズムにより管理され、必要に応じて外部との入出力が行われる。要求仕様を固める段階で何度もモデルの修正を繰り返し洗練していくことを考えると、必要なデータはメモリ上におき、キャシュから追い出すときにデータが変更されていればディスクに書き出すことは合理的である。扱うメディアを追加する (シートの属性を増やす) 場合は、この CachedThing のサブクラスとして定義することで可能となる。

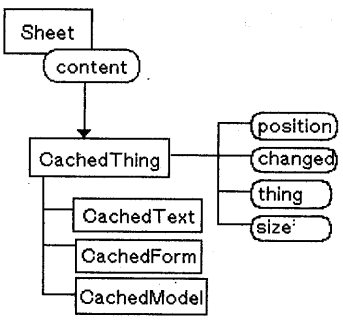

図 6: シートの Class 構造

4.5 プロダクト間の関連とその操作

生成したプロダクト間の関連付けにはそれらのプロダクト間にリンクを張ることで行う。リンクを張る対象となるプロダクトの種類には、基本単位であるシートとエリアがある。これらのシートやエリアの間にある種類のリンクを張ることにより、それらのプロダクト間の関連付けを行い、お互いに参照することが可能となる。関連付けを行うためのリンクの種類には次の 4 つがある。

1. 分解 / 抽象、部分 / 組み立て
   PWB のモデルを階層的に構造化する。
   個々の機能やプロセスとそれらを詳細記述したモデル間の関連付け (分解 / 抽象) や、個々のデータやファイルとそれらを詳細記述したモデル間の関連付け (部分 / 組み立て)

2. 実行制御、データ受渡し
   PWB のモデル実行時の関係要素間を関連付ける。
   モデル実行時の起動ポイント / 同期ポイントやデータの受渡しポイントなどの関連付け

3. 検索結果、インデックス
   フィルタリングによる検索結果、インデックス生成機能などによりシステムが自動的に参照関係をつける。

4. ユーザ定義
   ユーザが任意のエリア、シート間に自由に参照関連をつける。リンクの意味付けはユーザが行える。

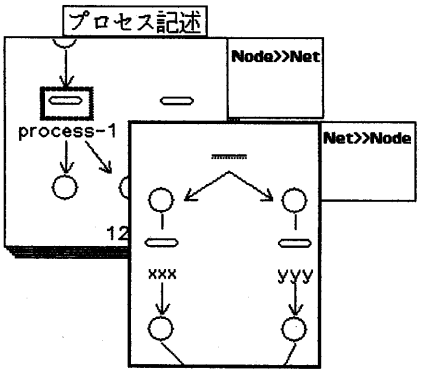

図 7: プロダクト間の関連

リンクを張る操作はまずもととなるプロダクトを指定し、対象となるプロダクトを直接マウスでクリックした後、それらを関連付けるリンクの種類を指定する。分解 / 抽象、部分 / 組み立て、実行制御、データ受渡しなどのリンクの種類は、PWB のモデルを構築する上で意味のあるプロダクト間での関連付けのみが可能となる。つまり、意味のない要素の分解 (プロセスからファイルなど) や要素間のデータ受渡しの関連付けはできないようシステム側で制御する。

階層構造化された PWB のモデル群に対してその階層構造を木構造として参照するビューを提供する。この木構造の各ノード (エリア) と実際のシートにはシステムが自動的にリンクを付け (検索結果のリンクの種類と同様)、ユーザはそれを利用してモデルの全体木構造から 1 つのシート (モデル) を選び出すことができる。また、その構造をもとに幾つもの本に分散して格納されているモデルをまとめて一冊の仮想的な本を自動的に創り出すことができる。このときに階層のレベルを指定できるので、たとえば 3 階層までのモデル群だけまとめて本にするようなことも可能である。

関連付けられたリンクはそのプロダクトのウインドウの右隅に現われる。関連先のプロダクトを参照するにはそのリンクをマウスでクリックすることにより行う。リンクを辿ってきたプロダクトには、どのリンクを辿ってきたかがわかるようマークがされるため、そのプロダクトに

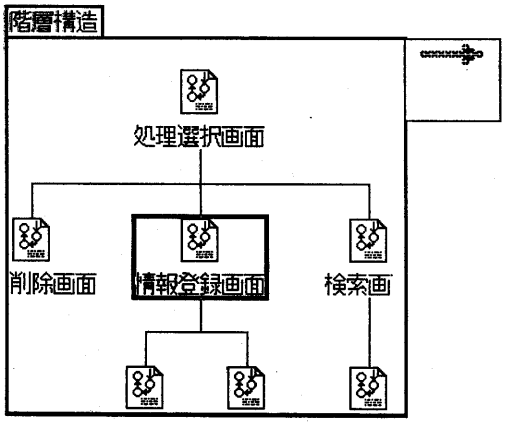

図 8: モデルの木構造

複数からリンクが張られていてももとのプロダクトに戻ることが可能である (リンクの双方向性)。
リンク情報の管理はプロダクトの管理とは独立に実現されており、関連付けられたプロダクトがどこに格納されているかを意識する必要はない。つまり、関連付けられているシートが今読んでいる本と別の本にあったり、その本が別のキャビネットにあっても自動的に関連する本が開かれその内容を見ることができる。
また、シートの差し替えなどもリンクの構造を気にする必要はなく、シートをある本から別の本に移し替えてもそのシートのリンク情報は保持されている (上位構造とリンク情報の分離管理)。

4.6 プロダクトの参照

ActiveBook内にあるプロダクトはビジュアルなマンマシンインタフェースを使って自由に参照することができる。参照の方法には次のようなものがある。

- ブラウジング (本のメタファを利用)
  実際の本を扱うようにその本の中身 (シート) をぺらぺらめくったり、本の厚みを利用してだいたいこのあたりのページといったやり方や2ページおきにシートをめくるという方法で参照する。またシートにタグを付けておき、そのタグを使ってそのページを参照したり、次のタグまでぺらぺらめくることもできる。試行錯誤的に目的に沿ったプロダクトに接近していくときに有効である。

- ナビゲーション (リンクを辿る)
  プロダクトに付加されたリンクを辿ることにより、関連プロダクトを参照する。リンクをプロダクトを探索するための通り道として利用するやり方である。

- フィルタリング (フィルタを利用)
  ある特定の関係があるプロダクトだけを取り出すようなフィルタを用いて、その結果から探し出す。プロダクトの階層関係から仮想的な本を創り出したり、プロダクト間のリンクの属性やそのパターンから該当するリンク情報を抽出しそれをもとに参照する。フィルタを利用してたとえば、モデルの中からデータの受渡しを行っているものだけを抽出するなど。

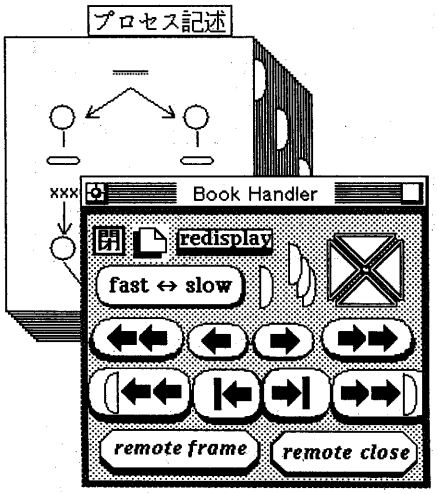

図 9: 本の操作

これらの参照のやり方は相互に関連しながら行えることが重要である。フィルタを利用してその結果を本にすれば本のメタファが利用できるし、本やフィルタを利用して起点となるプロダクトが見つかれば、そこからリンクを辿って関連プロダクトを参照できる。しかも、これらの操作がすべてビジュアルに行えるのである。

5 まとめ

ActiveBookでは、ビジュアルなマンマシンインタフェースとそのカスタマイズ、プロダクトの動的な構造とその操作、プロダクト間の関連とその操作などを実現した。これによりソフトウェアの要求定義・分析という段階で、如何

にそのシステムのモデル(プロトタイプ)を迅速に構築し、それらを検証でき、ユーザとシステム開発者の相互が満足のいく要求仕様の作成が支援できるかについて説明した。PWBのプロダクト(システムモデル)は、従来の紙ベースの仕様書とは異なり、それ自体が実行可能な仕様書としてコンピュータ内に存在する。それらのプロダクトは本の単位、またはキャビネットの単位で電子メディアとして取り出せ、ネットワークなどを経由して他のマシンに取り込めば、即実行することが可能である。

これからの課題としては、ネットワーク/分散環境を意識した運用管理あるいはマルチユーザにおける協調活動の支援機能の実現が挙げられる。また、大規模なプロダクトを光ディスクなどに格納する場合のプロダクトの高速なブラウジング方法、共有のプロダクトと個人のプロダクトとの境界や関連付けの機構、なぜモデルを修正したか、どのような手順で行ったかなどの作業プロセスをも含んだプロダクトのバージョン管理なども解決しなければならない問題として考えている。

参考文献

[1] 宮本勲"PROTOTYPER'S WORKBENCHの概念設計"ソフトウェア・シンポジウム1985