ソフトウェア工学 64－17
（１９８９．２．３）

交換システム向き視覚的ソフトウェア開発手法

池田 功＊　山根 智＊　石井直子＊　新家 誠＊＊
＊富士通株式会社　＊＊富士通関西通信システム株式会社

従来より、ソフトウェア開発の効率化施策は行われているが、ソフトウェアライフサイクルの各工程毎の局所的な効率化に留まっている。このために、各工程間で効率化手法が不連続であり効率化した成果が後工程に伝播できないために、従来の効率化施策では充分でなかった。より一層の効率化を達成するためには、開発環境の統合化を基本とした全ライフサイクルの統合的な開発環境の改善が必要である。

本稿では、交換ソフトウェアを対象とした設計工程及び試験工程を通した全工程に渡って、連続的な統一ビューを提供できる新しいビューの手段とこれをベースにしたソフトウェア開発環境を提案する。

The Method of Visual Software Development for Switching System

Isao Ikeda　Satoshi Yamane　Naoko Ishii　Makoto Shinka
FUJITSU LIMITED
1015,Kamikodanaka,Nakahara-ku,Kawasaki 211,Japan

The efficiency of software development has ever improved in the insufficient local optimization of software life cycle.For this,as the efficiencies are unified between software life cycles and the effectiveness of the efficiency is not able to be transferred to the later processes,the existing efficiency has never sufficient. In order to achieve more efficiency, it is necessary to improve systematically software development environment containing all life cycles based on the integration of development environment.

This paper presents the means of view of software and the software development environment based on this view which provide the successive and unified view in all processes through design and verification process for switching system.

1．バックグラウンド

1．1　基本的な考え方

我々は、交換ソフトの視覚的ソフトウェア開発手法を実現するために、現状のソフト開発プロセスをソフト表現即ちソフト視覚化の観点から図1に示す様に捉えた。

図1に示す様に、ソフトの実体をダイレクトに把握するのが困難なため、その姿を様々なセマンティックから間接的に可視化するための種々のチャートやダイアグラム等の表現様式が開発され適用されてきた。これらのチャートやダイアグラム等をここではソフトをそのセマンティックレベルに対応して覗くためのビューと呼ぶことにする。これらのソフトビューの多くはライフサイクルの特定の工程を可視化するものであり、必ずしも相互に連続するものではなかった。しかし最近の開発環境統合化の動向に伴い、これらのビューの不連続性が人間の知的作業の増幅という可視化の目的を阻害している事が顕在化しつつある。（翻訳ギャップの存在）　また、これらの処理や機能、性能等を検証する試験においても、表現様式と同じビューを設定することが困難であった。（解釈ギャップの存在）　また自然言語等で記述するために個人の能力等によって記述や解釈に曖昧さを含んでいるために、この曖昧さに起因する誤りが混入する。（曖昧さの存在）

以上の背景から開発環境の統合化に対応した全開発過程を統合するビューが必要であり、近年のWSの発展を利用して従来の制約を乗り越えた新しいソフトの可視化環境を提案する。

1．2　問題点

従来のソフトビューは段階的詳細化の過程に沿って不連続な階段状をなしているため、セマンティックレベルの推移に応じてビューを切り換えて見る必要があり、以下のセマンティックギャップ等の問題がある。

(1)翻訳ギャップ

要求仕様からソフトウェアアーキテクチャを決定し、機能分割や処理論理の決定といった作業過程において人間が多段階の翻訳を行う。まず、自然言語で記述した要求仕様（交換サービス仕様）に基づいて機能仕様を作成する。この際、交換動作は発呼等の処理要求の監視と通話接続等の処理実行を繰り返して進められていき、呼の状態が発呼、呼出、応答、切断、終話と変化していく過程では同じ入力信号に対しても処理は一定ではなく、呼の状態に依存するこのために、交換ソフトでは状態遷移の概念を導入して、種々のセマンティックレベルに応じた状態遷移図により機能仕様を明確に表現している。次に、実現すべきソフトウェアの構造を決定して、処理論理をＨＩＰＯ等のチャートで記述し、インプリメントを高級言語で記述する。つまり、自然言語から高級言語までの抽象度の異なった様々な表現で種々の設計プロセスを記述するために、各翻訳プロセス間でセマンティックギャップが生じて、思考の中断や情報の欠落を引き起して生産性や品質の低下を招いている。

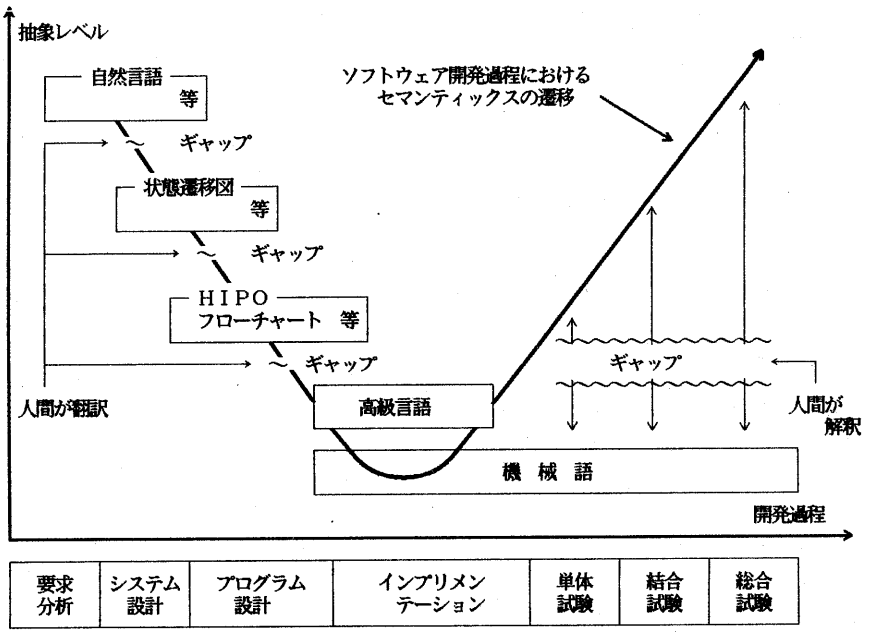

図1．ソフトウェア表現に着目したソフトウェア開発過程

(2)解釈ギャップ

交換システムでは、有限のリソースを用いて複数の呼を同時に扱う一方、サービス基準を満たすため、接続要求に対して一定時間以内にこれを処理する必要がある。このため実時間多重処理やタイミング条件などがきびしくそれらが充分満足されているかどうかを確認するための実マシンにおけるターゲット試験が必須となる。ところが、ターゲット試験では、試験手順や確認手順等をすべて機械語レベルで実施しなければならない。このために、試験が進むにつれて試験項目等の確認内容（自然言語）と確認手段（機械語レベル）との間にセマンティックギャップが広がり、試験の信頼性や効率の低下を招いている。

(3)曖昧さ

各設計プロセスは、プログラミング言語を除いて曖昧な自然言語の記述に基づいて段階的に作業を進める。この際、自然言語には冗長性等の曖昧さがあるために、人により又、時間によりその解釈が一意でなくなり誤りが混入していた。つまり、状態遷移図による機能仕様やチャートによる処理論理の記述に誤りが混入して、後工程に誤りが拡散的に伝播している。また、試験プロセスでは、試験項目や手順を自然言語で記述しているために、解釈や記述に同様な曖昧さが生じて、誤りが発生し、試験作業の信頼性の低下を招いている。

1．3　解決のアプローチ

以上の問題を解決するためにはソフトの開発過程全般にわたるビューの連続性を実現し、人間の思考を中断させないことが重要である。我々はこのような観点から総合的なソフトの可視化を達成するため、交換ソフト向けの新しいソフトビューモデルとして図2に示すV字モデルを提案する。

ソフト仕様は、大きく外部仕様と内部仕様に分けることができる。すなわち、交換システムの各種サービスや様々な要求条件を明記した仕様（外部仕様）とそのサービス仕様を交換システムがどの様に実現するかを明記した仕様（内部仕様）である。

このソフト仕様が持つ二つの側面（仕様）に対応し、ソフト開発の全過程でソフトをただ二つのビューにより表現する。

外部仕様　＝　WHATビュー
内部仕様　＝　HOWビュー

さらに、HOWビューをソフト仕様の詳細度の面から、機能仕様やモジュール構成等のソフトェアアーキテクチャの作成（システム仕様）及び処理論理の作成（プログラム仕様）でとらえて、アーキテクチャビュー／実行論理ビューで表現する。

システム仕様　＝アーキテクチャビュー
プログラム仕様　＝実行論理ビュー
　　→ HOWビュー

WHATビューとアーキテクチャビュー、実行論理ビューの各々でソフトビューを統一して形式化及びシンボル化し、さらに各ビューをスムースに結合して連続性を保つことによりソフトの全ライフサイクルのビューの連続性を実現し、人間の思考を中断させないでソフト開発の効率化を図ることが可能となる。

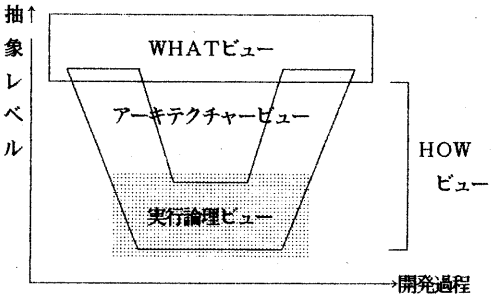

図2．V字モデルにおけるビュー

2．ソフトウェア視覚化手法

本稿では、ソフトウェア開発効率を向上させるために最も効果的で直接的に必要な実行論理ビューに焦点を当てる。

2．1　視覚化の考え方

実行論理ビューは、アルゴリズムとデータ構造に分けて捉えることができ、それぞれの目的に応じた表現方法を採り入れるべきである。

(1)アルゴリズムの表現

我々はアルゴリズムに対して、実行文の集まりをどのような順序で実行するかという制御の流れを表現する制御構造と手続き型言語のアルゴリズムの本質であるマシンの内部状態（データ）を直接変化させる効果を持つ実行ステートメントに分解する。また、人間が思考し、それを表現する時、自然言語表現及び図形表現が理解しやすい。この観点より、構造は図形表現、ステートメントは自然言語表現が適している。これより、アルゴリズムを明確に表現するためには以下の要求条件を満足させることが重要と考える。

・制御構造の図形による視覚化
・実行ステートメントの自然言語による形式的記述

我々は、これらの要求条件を満足する表現として、図3に示す様に、制御構造には富士通で制定した構造表記法で制御構造の図形化により制御の範囲及び階層を明示するＹＡＣⅡ（Yet Another Control Chart Ⅱ）、実行ステートメントには形式的な自然言語表現を採り入れた。

(2)データ構造の表現

交換ソフトウェアでは、とりわけ、データ量が膨大であり、データ構造が複雑である。なぜならば、交換機は局ごとにサービス機能等の違いがあり、各局毎にプログラムを作ることは不可能であるために、プログラムの一般化を図る目的で、処理を記述するアルゴリズムとその実行に際して具体的な値を与えるデータを明確に分離して、局条件依存性等をデータで吸収する必要があるためである。また、交換機は経済化の要求や限られたハードウェアの処理能力の条件化で、極めて多数の呼を同時に処理するといった実時間多重処理であるため、交換ソフトではメモリ節約と処理効率が重要となる。このように、交換ソフトでは、膨大かつ複雑なデータに関してメモリ節約と処理効率を考慮した設計が必要である。

また、データ構造の表現はフィールドの位置関係等の構造情報及びそのフィールドの型や説明等の属性情報の二つの要素からなると考えられる。アルゴリズムと同様に、理解の容易性の点から構造情報は図形表現、属性情報は自然言語表現が適している。さらに、メモリ節約及び処理効率を実現するためには、交換マシンアーキテクチャを考慮して、フィールドをどのように定義すればメモリ削減できてかつ高速な命令セットでアクセスできるか検討してデータ割り付けを決定しなければならない。このために、構造情報はデータ割り付けの図形表現が必要である。

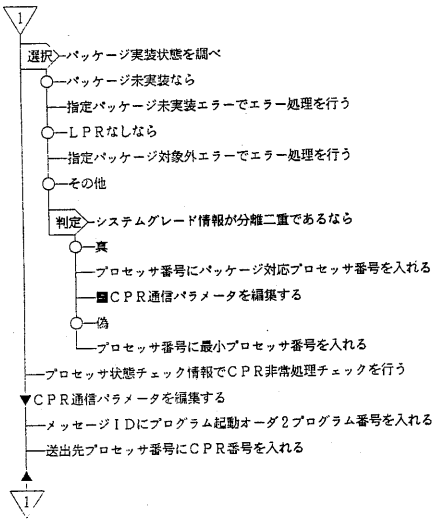

図3　アルゴリズムの視覚化表現（ＹＡＣⅡ）

以上よりデータ構造の表現には以下の要求条件を満足させることが重要であると考える。

・交換マシン上でのデータマッピングの図形表現

・各データフィールドの意味や値等の自然言語表現

我々は、これらの要求条件を満足する表記として、図4に示す様なデータ構造図と表によるデータ表記（データ図）を採り入れた。データ図は、データ構造を直観的かつ明確な図形表現とし、各データ及びデータ要素の意味を自然言語表現にしたものである。

２．２　視覚化による問題解決

以上の視覚化の考え方より、アルゴリズムはＹＡＣⅡビューで、データはデータ図ビューで表現する。我々はこの視覚化手法に基づいて、プログラム設計から検証まで統一的ビューで表し、セマンティックギャップを埋めて、曖昧さを解決した。

(1)翻訳ギャップの克服

ＹＡＣⅡ／データ図からプログラムを図5に示す様にジェネレータで自動生成することにより、プログララム設計とインプリメンテーションとのセマンティックギャップを埋める。また、プログラム設計作業における人間の思考プロセスに沿ったスムースな設計プロセスとするため、ＹＡＣⅡ／データ図表記に段階的に詳細化できる表現を採用した。

(2)解釈ギャップの克服

プログラム設計の機能及び処理の検証を行う交換マシン上のターゲット試験において、試験の確認内容を試験手順書及び設計書のビューに変換する。即ち、従来の確認内容である機械語情報を処理論理に関してはＹＡＣⅡビューに

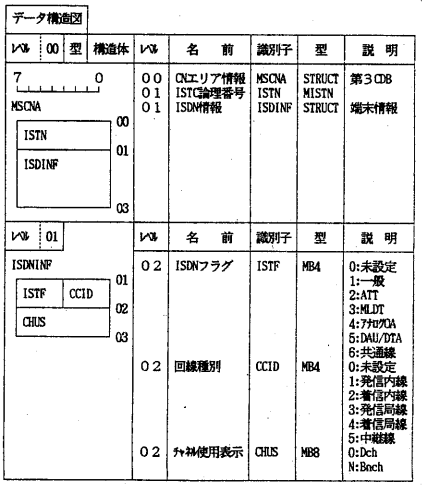

図4　データ構造の視覚化表現（データ図）

データ構造に関してはデータ図ビューに変換する。これにより、図6に示す様に試験手順と結果ダンプの確認ビューをプログラム設計のビューとその検証のビューを統一して、設計と試験のセマンティックギャップを埋める。

(3)曖昧さの解消

従来の自然言語によるプログラム設計では、自然言語が本来持ち合わせている不完全さ及び非形式性により、同一の文でも人により解釈が異なるとかいった曖昧さを含んでいた。しかし、プログラム設計をＹＡＣⅡ／データ図で記述し、その形式化された自然言語及び視覚化の効果によって記述の曖昧さを解消して誤りの混入を除去する。

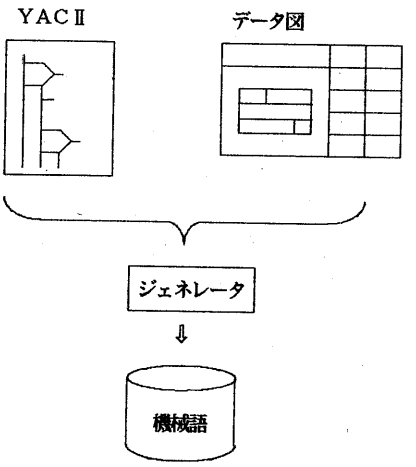

図5　プログラムの自動生成

3．統合的視覚化開発環境

以上の視覚化手法に基づいて図7に示す様にＹＡＣⅡ／データ図の構成管理やファイルジェネレーションを行うホストマシン、最終試験段階であるターゲット試験を行う交換マシン及び各種操作を行うワークステーション（設計ＷＳ／試験ＷＳ）をローカルエリアネットワーク（ＬＡＮ）に接続し、ソフトウェアビューをＹＡＣⅡ／データ図に統一した高度な統合的視覚化開発環境を実現した。

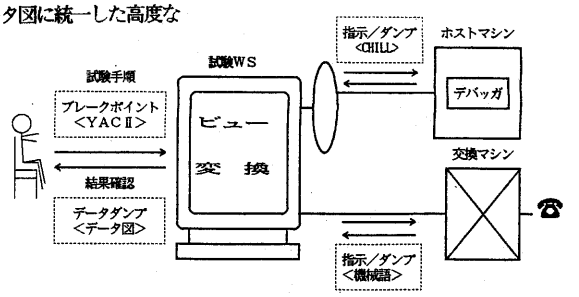

図6　試験ビューの変換

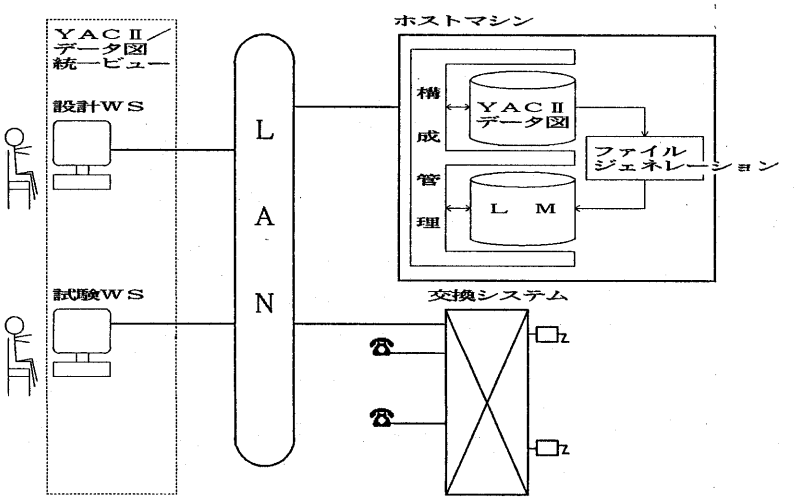

図７．交換システム統合視覚化開発環境

3．1　統合的視覚化開発環境の実現技術

統合的視覚化開発環境を構成する特徴的な技術を以下に示す。

(1)ＹＡＣⅡ／データ図の表現技術

①設計表現技術

ＹＡＣⅡ／データ図は従来の手書きの設計書をベースとした設計記述言語であり、記述は自然言語が望ましい。しかし、自然言語が本来有する意味の曖昧さ及び解析処理における問題点を回避するために形式化した日本語を用いる。

また、人間がソフトウェアを設計する過程は、抽象度の高い仕様から段階的に詳細化して目的のプログラム要素を作成する。このような人間の設計プロセスに沿った設計記述法を提供するために、本設計記述言語は図８に示す様な段階的詳細化記述を実現する。ここでは、概要記述の構造体を詳細記述のフィールドまでブレークダウンする記述過程を示す。

さらに、データ図はフォーム形式を採用しており、キーワードの日本語化及びデータ指定の形式化を実現しているので入力及び理解の容易な手段を提供する。

②内部表現技術

ＹＡＣⅡ及びデータ図は、図形情報を含んでいるために、従来のテキスト言語と同様なコンパイル技術では、プログラムを生成できない。これを解決するためには、図形情報が表現するセマンティックスを抽出してテキスト形式で表現しなければならない。

我々は、図９に示す様な三つの機構を実現して図形情報をテキスト形式に変換した。まず、図形情報を分割機構によってフィールド等の意味のある最小の言語要素である論理的単位に分割し、次に順序性変換機構によって図形情報の順序性からテキスト形式の順序性に変換する。また、表現形式変換機構によって図形情報のセマンティックスを論理的にコード化してテキスト形式としての保存形式に変換する。

また、従来のテキスト言語と同様に、論理的単位をベースとして、汎用的ソースコードコントロールシステムによって履歴管理を実現できるという利点も生じる。

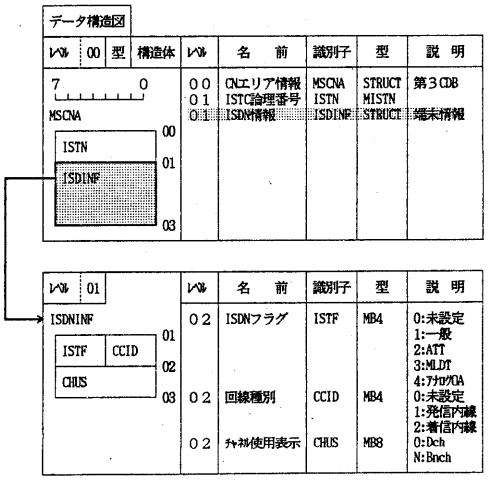

図８　データ図の段階的詳細化記述例

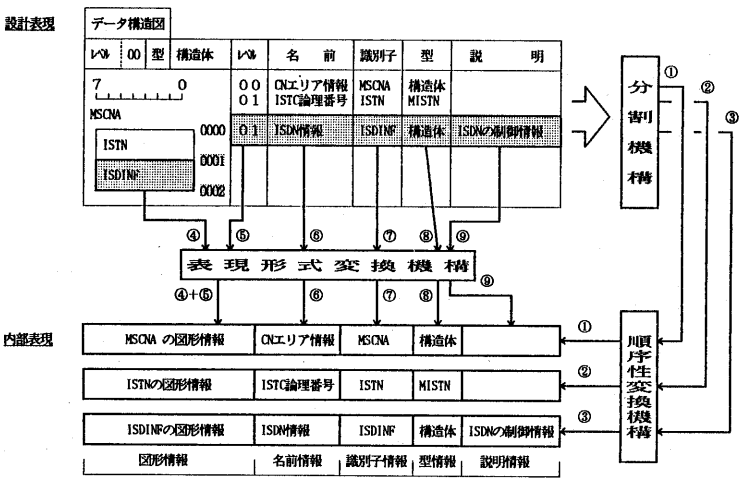

図9　データ図のテキスト表現形式への変換メカニズム

⑵統一ビューを高度なＭＭＩで実現する技術

高度なＭＭＩの実現には以下の三つの機能が必要である。

・ソフト開発の目的に叶っていること

ソフト開発は本来、データとアルゴリズムを相互に参照しながら設計する様な作業が多い。つまり、複数の情報を同時に参照しながら作業を進めなければならず、これを支援する必要がある。

・操作性が良いこと

良好な操作性を実現するためにはエディタ等のツールの使用手順を簡単化して良好なレスポンスが必要である。

・あまり知識のない人でもできること

あまり知識のない人でも作業できるようにするにはパラダイム機能が必要である。

これらの機能を満足するために図１０に示す様なＭＭＩをＷＳ上に実現したので以下にその特徴を示す。

①マルチウィンドゥによりデータ構造を参照しながらアルゴリズムを設計するといった作業を支援する

②エディタ等の人間が操作する必要のあるツールの使用手順を簡略化するためにコマンドをアイコン化してマウスインターフェースを実現した。

③ソフト開発を支援するために入力すべき情報をメニュー形式で指示するパラダイム機能を実現した。

⑶試験支援技術

試験支援には以下の二つの機能が必要である。

①試験の確認内容を設計レベルに引き上げること

試験の確認内容である機械語のダンプ情報をＹＡＣⅡ／データ図に変換する必要がある。

②試験の実施内容を設計レベルに引き上げること

試験の実施内容である機械語をＹＡＣⅡ／データ図に引き上げる必要がある。

これらの機能を実現するために、ターゲットマシン非依存性を有しているＹＡＣⅡ／データ図より実行可能形式を生成するファイルジェネレート時にＹＡＣⅡ／データ図とマシンコードとの対応情報を生成する。この対応情報を利用して、マシンコード情報とＹＡＣⅡ／データ図情報を相互変換する。これにより、試験作業（クロス試験及びターゲット試験）においてＹＡＣⅡ／データ図ビューのＭＭＩを提供する。

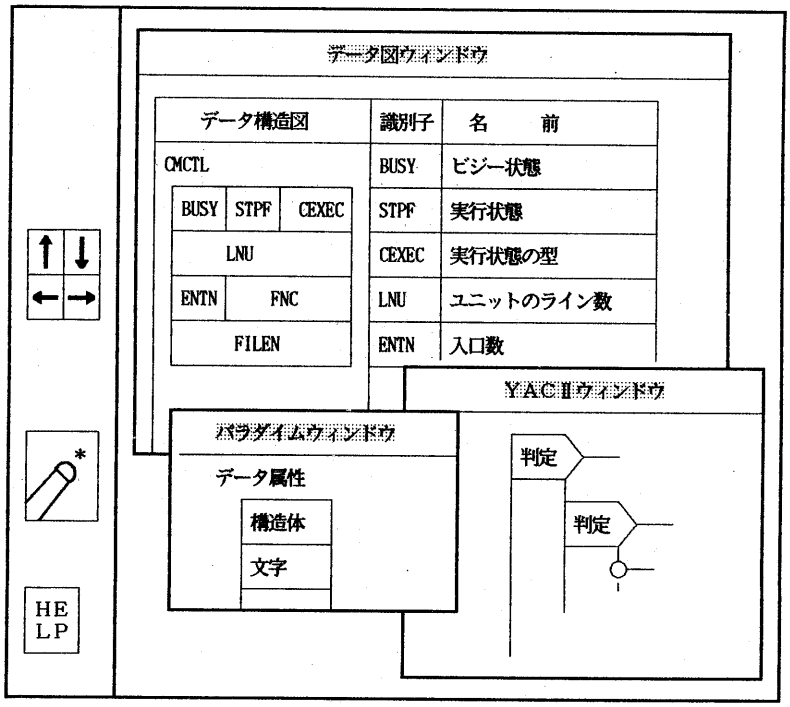

図１０　統一ビューの高度なＭＭＩの実現

(4)既存ソフト資産の環境移行技術

既存ソフト資産の視覚化環境への移行には以下の二つの機能が必要である。

①既存のソフト資産のＹＡＣⅡ／データ図への自動変換

②自動変換したソフトの日本語化／部品引用自動変換

これらの機能を満足するために以下の解決策を講じた。

①既存ソフト資産を自動変換してＹＡＣⅡ／データ図を生成するためには、前述の図形情報のテキスト形式と既存ソフトとの対応関係を定義しなければならない。テキスト形式はフィールド等の意味のある最小の論理的単位毎のレコードの集合なので容易に既存ソフトと対応づけできる。図１１に示す様に自動変換機構によって既存ソフトの各構成情報をデータ図テキスト構成情報に変換できる。

②自動変換の高度化のためにデータの日本語化及び部品引用による変換を実現する。我々は従来より手書きの設計書においてデータを日本語定義していたので、日本語変換機構を利用してデータ図テキストの名前情報欄及び説明欄にデータの日本語情報の組み込みを実現した。また、ＹＡＣⅡ／データ図の部品としてパラメータの有無によって各々マクロ部品及び組み込み部品がある。自動変換機構によってこれらの部品引用したＹＡＣⅡ／データ図テキストを生成する手段を提供する。

５．結論

我々は、アルゴリズム及びデータ構造のＹＡＣⅡ／データ図による視覚化に基づいて、交換ソフトウェア指向の統合的視覚化開発環境について述べた。現在、ＰＢＸ等の開発に適用しており以下の効果を上げている。

①ソフトウェア生産性の向上

②ソフトウェア設計品質及び検証品質の向上

③ソフトウェアメンテナビィリティの向上

今後は上流工程であるアーキテクチャビュー及びWHATビューの視覚化手法を確立して、それらを統合する開発環境を構築していく予定である。

本システムを開発するにあたり、終始御指導いただいた通ソ開）ソフトウェア技術部藤本部長、第一開発課井上課長、藤井調査役に感謝致します。

参考文献

〔１〕井上他 "CHILL GRAHIC REPRESENTATION AND PROGRAMMING SUPPORT ENVIRONMENT", CHILL CONFERENCE '86 1986.

〔２〕豊島他 "VISUAL DEVELOPMENT ENVIRONMENT FOR EMBEDDED SOFTWARE", MIMI'88 1988.

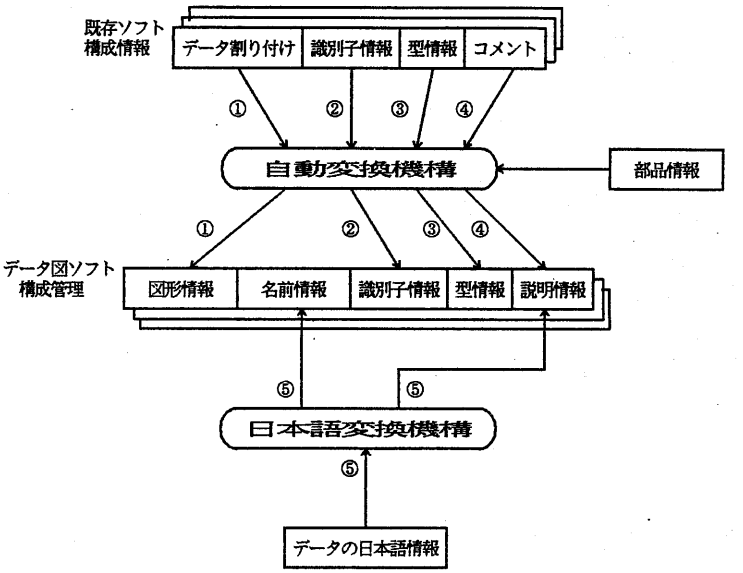

図１１　既存ソフトの環境移行メカニズム