ソフトウェア工学 87－2
(1992. 9. 29)

プログラムと仕様書の統合管理による開発保守支援システム

上原 三八　園部 正幸　吉野 利明　直田 繁樹　石崎 あゆみ　川辺 敬子

富士通研究所
川崎市中原区上小田中1015

あらまし
本稿では、事務処理システムの保守支援環境を対象として、プログラムとドキュメントを関係付けて管理し、その上に支援ツールを構築するアプローチについて述べる。保守作業では、ソフトウェアを理解するために多くのドキュメントへ効率良くアクセスし、かつ定型的なレビューや承認などを行なう必要がある。これを支援するための、プログラムから仕様書を生成しこれらを関係付けて管理する機能、GUIを用いたレビュー支援機能、プロセス自動化機能などについて述べる。関連管理機能の実現は、従来のファイルシステムの上にマッピング層を設け、各種の関係付けをマッピング層に与える情報で制御する方式で行なった。
和文キーワード　データ統合、レビュー支援、ソフトウェア保守、プロセス支援

Supporting Software Maintenance by Integrating Documents and Programs

Sanya UEHARA, Masayuki SONOBE, Toshiaki YOSHINO, Shigeki SUGUTA, Ayumi ISHIZAKI, and Keiko KAWABE

Fujitsu Laboratories Ltd.
Kamikodanaka 1015, Nakahara, Kawasaki, 211 JAPAN

Abstract
To support maintenance of business application systems, this paper describes an approach which (1)inter-relates and integrates programs and documents and (2)builds various support tools on it. Software maintainers have to efficiently access various documents and perform routine works. Some features to support them are discussed, such as generation of a specification from a program, inter-relating documents, review support, and process support. Inter-relating documents is realized by introducing a mapping layer which interprets various inter-relating information.

英文 key words　data integration, review support, software maintenance, process support

1．はじめに

本稿では、事務処理システムを対象とした開発環境の研究について、特に保守支援に焦点を当てて述べる。長期にわたり多人数で保守拡張されるシステムの開発環境を調査した結果、以下の問題があることが分かった。

・開発後も「絶え間ない機能拡張」があるが、大量の設計文書が紙のため、変更必要個所の調査・修正といった作業が繁雑を極める
・「集団型開発」のため、調査・意思決定・意思疎通の作業が多過ぎる
・全工程一貫した支援環境がなく、バラバラなツールを苦労しながら使っている

これと関連して、保守担当者からの要望を以下のようにまとめることができる。

(a)*保守・修正に役立つ設計資料の計算機化（電子化）*

現在の支援ツールはプログラム向けの管理ツールがほとんどである。最近、設計図がCASE製品で定義できるようになりつつあるが、表記法は上流の一部に限られていている。プログラムなどから抽象度の高い設計資料を一部自動生成するなどのリバースエンジニアリング［1, 2］が必要である。

(b)*図でシステム全体を見通したり、必要に応じて細部が見られるインタフェース*

(c)*関連した修正個所など、ドキュメント間のリンク付け機能*

従来のプログラム個別の修正管理機能ではなく、プログラムおよびドキュメント間で関連した修正個所を管理できると良い。この情報は例えば、修正後のエラー発生時に行なわれる調査の際に使用される。

(d)*種々の一覧表・管理表の自動生成*

プログラムやドキュメントなどを分類して一覧表で管理する作業が自動化されると良い

(e)*システム中の機能グループの管理機能*

通常、プログラムやモジュールを基本とした管理のみがされている。しかし、システムを理解し保守を行なう上で、複数のプログラム中に組み込まれている、トランザクションやサービス単位の処理を把握する必要がある。

(f)*障害時の原因探索や、変更の波及を調べる支援機能*

以上の現状分析に基づき、ドキュメントの情報を関係付け、その上に支援ツールを構築するアプローチにより、システムを設計・試作した。本稿では、そのアプローチおよび上記(a)(c)(d)の支援機能とその実現法について述べる。

2．システム概要

プログラムとドキュメントを関係付けて統合管理するアプローチと、システムの機能を説明する。

2．1 関連管理

通常の計算機環境では、プログラムやドキュメントはファイル名で個別管理されているため、ドキュメント間の関係付けは人間によって行なわれる。例えば、プログラムprog1のファイル名はprog1.pg、詳細仕様書のファイル名はprog1.spなどである。しかし、プログラム数が増えたり、プログラムを分類して管理しようとすると、ディレクトリを作る必要があるのでパス名も管理対象になり複雑になってしまう。保守作業で必要となる、大きな、しかも他人の作ったソフトウェアを理解するためには、多くの情報に効率良くアクセスできる必要がある。そこで、ユーザにはＧＵＩを通して関連ドキュメントを効率良くアクセスできるインタフェースを提供する。そのために必要となる関係付け情報は極力自動生成することとした。例えば、プログラムの呼びだし（call）関係を図形表示してドキュメントを選択できるようにするために、呼びだし情報をプログラムから自動的に抽出し管理する。

関連管理の基本的な考え方は、与えられたプログラムとドキュメントの内部および外部の情報間に対してハイパーテキストの概念を適用することである。しかし本システムの特長は、関係付けに必要な情報やドキュメントを自動生成すること、また、３章で述べるように既存ファイルシステムとの整合性を考慮した実現法を行なっていることである。

図１はユーザインタフェースの画面イメージである。関連情報を表示させるための操作の基本は、そのオブジェクトの上にマウスカーソルを持っていきクリックすることである。複数の関連情報がある時は、それらがメニュー表示される。図１では、一覧表の上で文書名を指定するとその文書が表示され、モジュール構造図の上でモジュール名を指定すると対応するプログラムまたはドキュメント（メニュー表示される）が表示され、また、データ名を指定するとその定義情報が

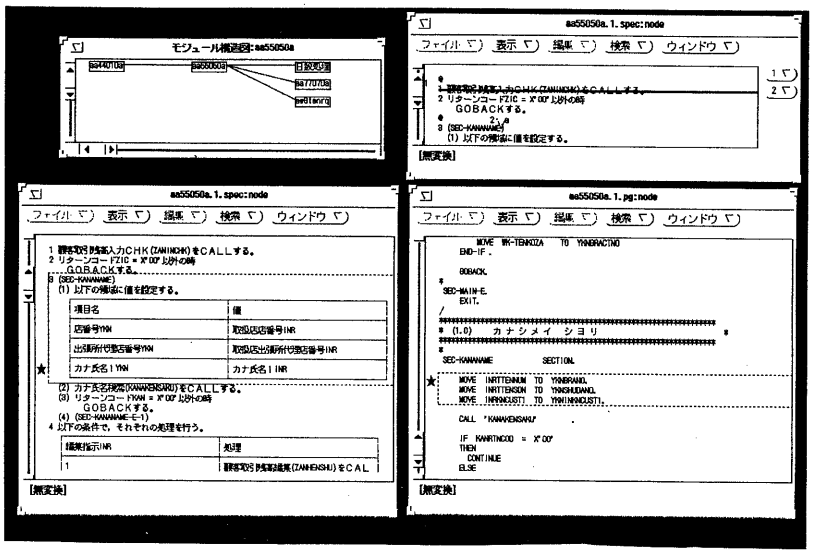

図1 ユーザインタフェースの画面イメージ

ディクショナリより検索され表示される。

2.2 仕様書生成

仕様書が手書きの場合は、書き換え修正などの管理・保守コストは極めて高い。また、電子化されている場合でも、プログラムとの一致性、記述内容の過不足や標準化に問題がある。この問題を解決するために、プログラムを解析し、抽象度の高い仕様書を生成する［3,4］。図2は仕様書生成のパターンの例である。ここでは、英字名から日本語名への変換にはディクショナリを用いる。

図2(a)は、WK-で始まる作業変数の利用目的を解析し、その目的がデータの「分解」であることを認識して、訳出した例である。事務処理で作業変数の利用頻度がもっとも高いのが、この「分解」、そして帳票作成やレコード作成などに現れる「合成」である。実際のプログラム上では、作業変数への代入や参照の関係は条件文や分岐文などにより複雑なため、制御フローとデータフロー［5］を解析して行なう。

(b)は、スイッチとして使われている作業変数SW-ATENDを、その使用目的を解析して「ファイル終端」という訳語で置き換え、そしてその変数への代入文を訳出せずに消去している。これは、プログラマが通常とる戦略（決まり言葉、cliche［6］）をシステムが知識として持つことにより処理する。(c) はプログラムを表に変換する例で

```
01 WK-ACTENNO.
  02 WK-BRANO  PIC  9(2).
  02 WK-ACTNO  PIC  9(4).
  .....
MOVE INBRANO TO WK-BRANO.        店口座番号の
MOVE INACTNO TO WK-ACTNO.          先頭の2桁に店番号
MOVE WK-ACTENNO TO                 最後の4桁に口座番号
                 OUTACTENNO.     を設定する。
```

(a) 作業変数（合成）の消去例

```
 PERFORM SEC-READ                ファイルの終端まで以下の処理
   UNTIL SW-ATEND=ON .           を行なう。
 CLOSE AMOUNT-FILE .               使用量ファイルを読み込む。
  .....                              ファイル終端の時
SEC-READ SECTION.                      繰り返しの先頭に戻る。
 READ AMOUNT-FILE.                 料金に使用量*10を設定。
 AT END                            料金を書き出す。
   MOVE ON TO SW-ATEND           使用量ファイルをクローズする。
   GO TO SEC-READ-END
 END-READ .
 COMPUTE FEE=AMOUNT*10.
 WRITE FEE.
SEC-READ-END .
 EXIT .
```

(b) 作業変数（スイッチ）の消去例

```
EVALUATE JUSHO
 WHEN "CHIBA"
  PERFORM SEC-KENZEI
 WHEN "TOKYO"
  PERFORM SEC-TOZEI
 WHEN OTEHR
  CONTINUE
END-EVALUATE .
EVALUATE KINMUSAKI
 WHEN "TOKYO"
  PERFORM SEC-WR-KINMU
 WHEN OTHER
  CONTINUE
END-EVALUATE .
```

	"CHIBA"	"TOKYO"	その他
現住所	県民税の処理実行	都民税の処理実行	—
勤務先	—	勤務先出力処理実行	—

(c) 分岐文からの表への変換例

図2 仕様書生成のパターン例

ある。

この様にプログラムを整理したレベルの仕様書は、プログラムの理解を容易にし、保守性を向上させる、と期待される。これは、事務処理ではプログラムの８割程度が業務を表現したものであることから、業務の内容と照らし合わせながら保守・拡張する作業に役立つのである。

２．３ 連動スクロール

仕様書とプログラムが同時に表示されても、その対応をとることが容易であることが望ましい。そこで、プログラムと仕様書の対応関係を生成し、対応個所を表示するように自動的にスクロールする機能を実装した。図１下側のプログラムと対応する仕様書のウィンドウ中に破線の矩形が表示されているが、これは矩形中の情報が対応することを表している。連動スクロール・モードでは、一方のウィンドウをスクロールすると、関連ウィンドウも対応個所を表示するように自動的にスクロールする。★は、その位置の情報を連動させることを示し、他方のウィンドウも★の位置に対応する情報を表示する（テキストの末端が近辺にあり、その位置に表示できない場合を除く）。この★の位置はウィンドウ中であれば自由に変更することができる。対応個所が一対一でなく、多対多の場合は、表示中の前方ないし後方に対応ドキュメントがあるマークが表示され、そこへジャンプスクロールさせることができる。

連動するための情報は、一般には解析ツールによって自動的に生成する。仕様書を生成する場合は、自動的にプログラムとの対応関係をステートメント単位に生成しそれを利用する。

図３は複数の連動スクロールを行なっている例である。中央のドキュメントは左右のドキュメントと連動している。右のウィンドウをスクロールすることにより、中央のウィンドウがスクロールする。ここで、さらにそれに合わせて左のウィンドウがスクロールされるモードと、されないモードがある。（ただしもし、左と右が対応する連動スクロールモードとなっていても、循環はさせない。）連動スクロールモードには、関連ドキュメントを表示させると自動的に入る。あるいは、無関係なメニューで表示させた場合で連動していない時は、連動コマンドを選んで、２つのウィンドウを選択する。なお、複数のウィンドウ間での対応付けを分かり易くするため、矩形の破線の色を、異なる色で表示するようにする。

連動する情報を修正する場合は、ユーザは明示的にその関係を指定する。追加に際しては、既存の対応している枠の中に入れる場合と新たに関係を定義する場合がある。即ち、矩形の境界にある文字と文字との間に文字列を挿入する場合には、どちらの矩形内かそれとも新たな矩形内かで３通りの場合があり、ユーザが指定する。

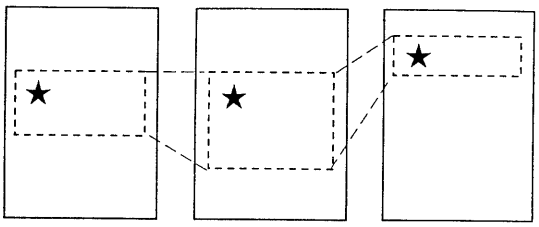

図３ 複数ウィンドウ間の連動スクロール

２．４ HyperReView

複数のドキュメントにわたる修正個所を関連付けて管理する機能と、レビューを支援する機能である。

（１）修正内容表示

プログラムやドキュメントの修正内容を、修正前のテキストの上に赤色の校正記号で表示する機能である。プログラムの修正履歴を差分で管理するツールは一般的であるが、ここではGUIを利用して差分の内容を理解する作業を効率化しようとするものである。図１の右上のウィンドウはその差分を表示している例である。

修正差分を獲得するには（専用の)エディタ内で獲得する方法と、ファイル比較で行なう方法とがある。前者は実際にユーザが修正した内容を一字一句その順番で正確に記録できる利点があるがエディタ毎に組み込まなければならない。本システムでは、XView［7,8］のエディタ上にこのような差分を獲得し表示する機能を開発した（図１参照）。また、後者のファイル比較の方法にはUNIXのdiffを使うことが考えられるが、UNIXは行単位の差分であること、そしてプログラム以外では良い結果が得られないどの欠点があるため、diffを改善した修正差分の獲得方法を開発した［9］。

（２）しおり

ドキュメント中の文字列または一点に対してし

おりをつける機能を設けた。しおりは複数のプログラムやドキュメントにわたって複数個定義できる。このしおりには以下の機能が付加されている。

ドキュメントの修正差分の獲得時に、自動的にしおりが付けられる。これにより、大きなドキュメントでも、その修正個所を即座に確認できるようになる。また、関連するしおりをグルーピング化してまとめて１つのしおりにすることができる。さらに、レビュー作業を支援する機能として、しおり毎に修正内容をキャンセルする機能、複数のレビュー者による修正内容をマージする機能がある。

しおりには、コメントをつけることができる。コメントには、なぜその様に修正するのか（修正しおり）、あるいはその個所がどのような意味を持つのか（参照しおり）などを記す。また、レビューの際には修正者とレビュー者とのやりとりも記述される。しおりには番号が付けられるので、その順序で参照するナビゲーション機能がある。これを用いて、ドキュメントの重要な個所を特定の順序で、コンメト付きで参照することができる。

図４は、HyperReViewの機能を使って複数のドキュメントとプログラムを参照しながらプログラムを修正する例である。障害作業ドキュメントの作成、参照個所と修正個所へのしおりの挿入（一部自動的）、作業終了の記入、といった軌跡を記録している。後日、この一連の障害修正作業内容を、参照したドキュメントを含めて再現することができる。

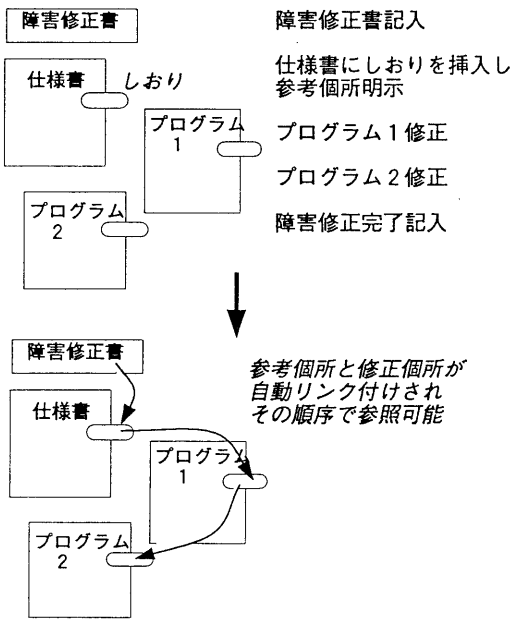

図４ 障害修正内容の自動リンク付け

２.５ プロセス自動化

複数の人間が共同作業を行なっている開発現場では、多くの定型的な作業手順（プロセス）が存在する。この手順を自動化することで、以下の支援が行なえると考える。

・適切なツールの自動起動
・作業のガイダンスおよびチェック
・作業者間のコミュニケーション支援
・作業報告書の作成・登録（メール発行含む）
・作業状況の管理

なかでも、作業者間での情報のやりとりの自動化、障害修正や機能拡張の際の定型ドキュメントへの記入と管理の自動化は、効果的と期待される。

具体的には、障害発生→修正案件作成→設計書修正（←→レビュー／承認）→プログラム修正（←→レビュー／確定）→テスト／本番登録→修正案件終了報告、といったプロセスで、企画、設計、プログラム開発、運用の各部署の担当者とレビュー者間に、情報が流れかつ作業日が記録される。これを自動化する機能を設計した。本システムでは、各作業者がメニュー中の「依頼作業」のボタンをクリックすると、依頼されている作業の一覧が表示される。例えば、設計者が設計書の修正を完了すると、システムは設計者に確認した上で、レビュー担当者にレビュー依頼を自動発行する。そうすると、レビュー担当者が「依頼作業」のボタンをクリックしたときに表示される一覧に、その依頼された作業が挿入される。このように、作業プロセスの自動化により、ドキュメントと作業状況を総合的に管理することができる。

２.６ 管理表の自動更新

本システムではモジュールを分類して管理する機能を備えている。例えば、オンライン／バッチ、業務、科目、．．の分類が可能である。これにより、プログラムの検索は、名前とモジュール構造図からの検索の他に、管理表による検索が可能となる。表１は、管理表の例である。縦

に業務と科目、横にモジュール階層番号（若い数字が上位モジュール）をとったものであり、表中の数字はモジュールの個数を表している。この数字上をマウスでクリックすることにより、モジュール名の一覧表を表示させてプログラムまたはドキュメントを検索することができる。

表1 モジュール管理表の例

業務	科 目	3	4	5	6	7	計
預金	当座預金	1	50	62	18		131
	普通預金	1	48	65	22	3	139
	別段預金	1	52	52	16	4	125
	通知預金	1	35	38	15		89
	定期預金	1	55	57	20		133
	積立預金	1	38	39	16	1	95
	預金共通	2	101	104	20	65	292

この管理表は、仕様書またはプログラムを登録すると自動的に更新される。新規のモジュールであればモジュール数と、場合によっては、縦や横の欄が拡張される。このモジュールの分類の登録には2通りの方法を採用している。一つは、モジュール名を登録する時に、ガイドされるメニューによってユーザ自身が定義する方法である。もう一つは、モジュール名に分類の情報が規約化されている時は、その規約をあらかじめシステムに外付けで（ファイルで）与えておく方法である。規約とは、モジュール名の何番目の文字が何を意味するかといったものである。これらの分類はUNIXファイルの木構造で実現しているが、一般のユーザはそれを知る必要はない。

3．関連管理の実現法

関連管理、HyperReView、および連動スクロールといった、ドキュメントを関係付けて管理する実現法を説明する。

3．1 概要

本システムでは、ハイパーテキストのようなノード間の関係の他、HyperReViewや連動スクロールなど多様な関係付けを実現する必要がある。また、通常のハイパーテキストシステムが提供する専用データベース方式と異なり、既にユーザが運用している既存ファイル・システムと連携できる実現法をとる。

（1）仮想構造

通常ハイパーテキストでは、定められた物理的構造に合うように文書を加工するのに対して、ここでの考え方は、文書はそのままにしておき、文書間の関係付けを自由に行なえるようにする、という違いがある。図5に示される概念的な実現法から分かるように、従来のファイルシステムの上にマッピング層を設けて、あたかも文書が関連付けられているかのごとく仮想的にシミュレートするというものである。文書の関連付けはマッピング層に与える情報のみを変化させることにより動的に変更させることが可能となる。また、実文書の一部が共有されている場合は、一方を修正することで他方にも反映させることが可能である。本実現法の特長は、

・ユーザが扱う情報は仮想文書なので、実文書とのマッピング如何により、部分的に重複する仮想文書など幅広い表現が可能
・既存ファイルシステム上に構築されてきたツール等の資産が損なわれることなく、2つのファイルシステムの利点を得ることが可能である。

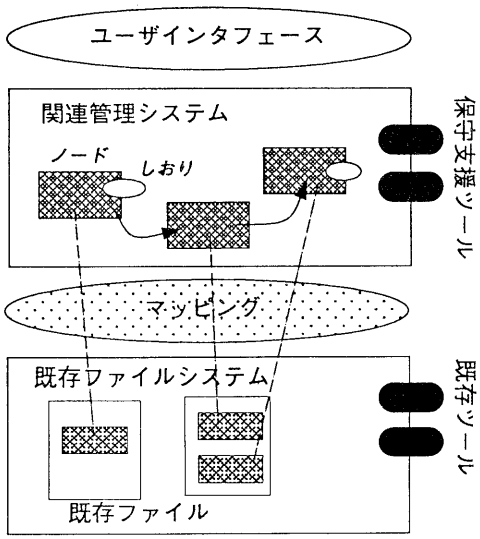

図5 関連管理の実現

（2）マッピング

マッピングは以下の機能を実現する。

・仮想文書を実文書のどの部分から合成して作るかを定義する。即ち、仮想文書と実文書と

の対応付けを行なう
・ハイパーテキストにおける、リンク元とリンク先のテキストの対応をとる。
c.表示中のテキストをマウスで指定することで行なわれるサービスやメニュー表示を定義する

（3）パッケージ

一個以上のノードの集まりでパッケージを形成する。パッケージによってノード名にスコープを持たせることができるので、ノードの命名法が柔軟になる。すなわち、同種の情報を持つノードにはパッケージが異なれば同じ名前を命名できる。

同一パッケージ内のノードをローカルノードと呼び、他パッケージから参照するノードをグローバルノードと呼ぶ。グローバルノードを指定するには、パッケージ名：ノード名 とする。ローカルノードを指定するには単にノード名だけで良い（図6参照）。

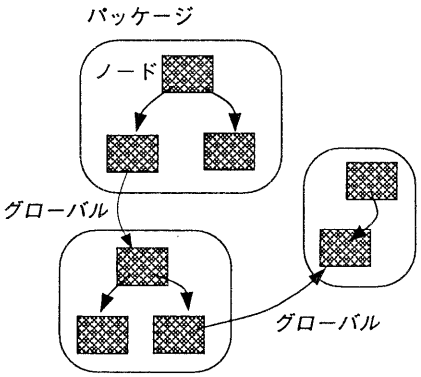

図6 パッケージとノードの関係

3.2 構造

3.2.1 実現構造

図7で示すように、ノードとパッケージの他、主なデータ構造はオブジェクトで表現される。パッケージ情報、マッピング情報、連動情報、しおり情報、ウィンドウ情報は、オブジェクトの集合である。マッピング情報中の個々のオブジェクトはそれぞれ1つのマッピングを示すが、連動情報中の個々のオブジェクトは二つの文書間の連動情報を持つ。ウィンドウ情報のオブジェクトは一つのウィンドウに関する情報を管理する。同一文書でも複数のウィンドウで表示することがあるため、ウィンドウ情報オブジェクトは集合となっている。

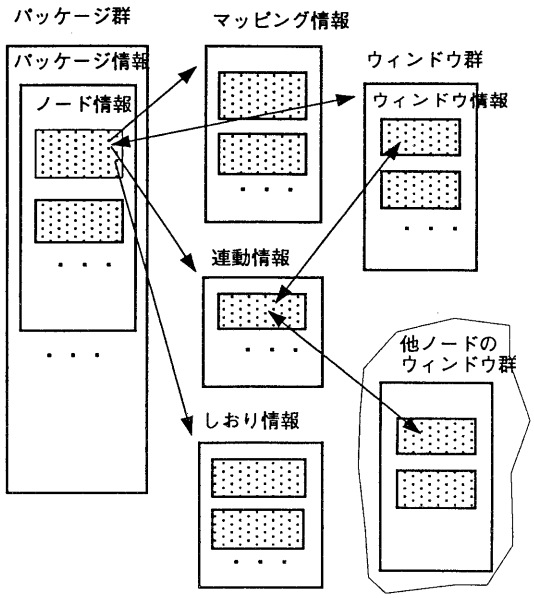

図7 関連管理の実現構造

3.2.2 外部ファイル形式

関連情報を定義する関連定義ファイルの構造を説明する。この関連定義ファイルと実文書とが一緒になってはじめて関連文書が実現する。従って一般には、関連文書への変更は、関連定義ファイルと実文書との両者への変更となる。関連定義ファイルには、パッケージ定義、しおり定義、連動定義がある。通常、1パッケージには、1設計文書もしくはプログラムを対応させる。

（1）パッケージ

図8(a)にパッケージの関連定義ファイルの構文、(b)に定義例を示す。パッケージ定義は1個以上のノード定義からなる。パッケージで使用する実文書のファイル名fidをすべてパッケージ名の直後に記述する。この中の先頭のfidはノード定義中にファイル名の記述が無い場合、そのノードのfidとなる。

ノード定義は、内容定義とマッピング定義の繰り返しからなる。内容定義は、fid（ノードの場合は一つに限る）と領域定義からなる。マッピング定義は、領域からアクション（プログラムのcall）への対応付けである。

文字列の指定はゾーン指定と呼び、行とカラム（省略可）のペアからなる。カラムが省略さ

```
パッケージ定義 = [ファイル定義] ノード定義...
ノード定義 = {内容定義 [マッピング定義] } ...
内容定義 = [ファイル定義] [領域定義] ...
マッピング定義 = 領域->アクション
ファイル定義 = @file {"fid" [,"fid"] ...}
アクション = N.ノード指定 | P.パッケージ指定 |
             A.アクション名 | M.メニュー名 |
             NOP | DEF
```

(a)関連定義の構文則

```
pac1
@file {"textfile"}
<node1>
@content {
 Z(1;10)
 Z(20,2;30,2)
}
@map {Z(1,4;1,8)->N.node2
 ...
}
<node2>
 ....
```

(b)関連定義例

図8 パッケージの関連定義

れた場合にはその行全部を指定されたものとする。例えば、1から10行目を表すZ(1:10)は1行目の1文字目から10行目の40文字目Z(1,1:10,40)と同一である。NOPは何もしない意味ではなく、ウィンドウマネジャーに制御を渡す、の意味である。P．パッケージは、パッケージ内の先頭ノードを表示すること、DEFはデフォルト表を参照すること、アクションはアクション表（外付けで定義、内容はルーチンのcallからなる）に従った処理を行なうとの意味である。この他に、ノード毎に定義できる情報として、ウィンドウのバックグラウンドの色（@color{ }）やデフォルトのウィンドウサイズ（@winsize{ }）がある。

（2）HyperReView

HyperReViewは複数文書にわたり挿入された、順序付けられた文書変更プラン（エディタ・コマンド）とコメント／メモを含む。その定義内容は、パッケージ毎にグループ化される。それぞれのグループの中で複数のしおり情報が差分情報を伴って定義される。各しおりの内容は、ノード名、メモの内容、文書の修正コマンド列からなる。定義概要を図9に示す。cmdの中は、変更差分である。

（3）連動情報

2つのパッケージの文書間の対応関係を定義する。これは、図10に示すように、実ファイル名と連動する情報とのペアの繰り返しである。

```
package:パッケージ名
1 {node:ノード名
   memo {文字列}
   cmd {
      以下のコマンドの列
        削除はD 領域指定
        挿入はI 点指定 文字列
        移動はM 領域指定 点指定
        複写はC 領域指定 点指定
  } }
2 {....      }
....
package: ....
```

図9 HyperReViewの定義内容

```
fid1          fid2
Z(1;10)       Z(1;2,10);
...           ...
```

図10 連動情報定義例

（4）モジュール構造図

モジュールの呼び出し情報は、呼び出しているモジュール名の一覧と呼び出されているモジュール名の一覧をモジュール毎に作る。これは、プログラムの修正がある毎に変更しておき、ユーザの表示要求があった時点でそれらの情報を集めてモジュール構造図を表示する。

4．評価と課題

本システムはSunワークステーション上で、Xウィンドウシステム向けのツールキットXView[7, 8]を利用し、C++で開発した。なお、仕様書生成機能については、COBOLパーサはyaccで、本体はCommon Lispで開発した。

プログラムとドキュメントをディレクトリ構造ベースでなく、管理表や一覧表のGUIでアクセスするインタフェースは、多くのプログラムやドキュメントを抱え、UNIXやPCに不慣れなエンドユーザにとって、分かり易いと評価されている。また、赤色での差分表示や連動スクロールも使い易いとの評価を得ている。今後、実際の作業を想定した試用と評価を開発保守担当者と共同で行なって行く。

仕様書生成にかかる時間は、約4,000行のプログラムで1分、40,000行のプログラムで10分程度である（SparcStation2 での elapse time）。現

在、多くの実プログラムを用いた生成結果の評価を行ないながら、生成表現の改良を行なっている。

以上の他に今後検討すべき課題として、ドキュメントとプログラムへのアクセス権をモジュールの分類と開発組織の上に定義実装すること、より汎用的なプロセス自動化支援のための、記述力のあるプロセス記述言語とその自動化システムの開発がある。

5．まとめ

本稿では、種々のドキュメントとプログラムを関係付けて管理するシステムの機能と実現法を述べた。過去の障害修正個所の検索やレビュー作業を支援するためのユーザインタフェースについても述べた。保守作業環境を改善するためには、多くのドキュメントを基にシステムを理解することや、煩わしいと思える定型的な作業を極力自動化して支援することが重要と思われるが、本研究はそのような目標の実現を目指すものである。この目標は、開発作業についても共通している部分があると思われる。

ドキュメントやプログラム等を関連管理する基盤と関連して、最近、PCTEを含むレポジトリ［10, 2］への関心が高い。筆者らは、現実のユーザニーズを解決するためにどのような機能を実現すべきかの検討を積み重ねることにより、今後のレポジトリ構築の要件を明確にしていきたい、と考える。

参考文献

［1］E.J. Chikofsky and J.H. Cross II : Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software, pp.13-17, Jan. 1990.

［2］C. McClure : The Three Rs of Software Automation - Re-engineergin, Repository, Reusability, Prentice Hall, 1992.

［3］吉野利明ほか：事務処理プログラムからの仕様書抽出, 第6回人工知能学会全国大会, 1992.

［4］上原三八: 保守拡張のためのリエンジニアリング技術, 人工知能学会セミナー、1992.

［5］A.V. Aho, R. Sethi, and J.D. Ullman : Compilers - Principles, Techniques, and Tools, Addison-Wesley, 1986.

［6］C. Rich and R. Waters : The Programmer's Apprentice: A Research Overview, IEEE Computer, pp.10-24, Nov. 1988.

［7］D. Heller: XView Programming Manual, O'Reilly & Associates, Inc., 1990.

［8］XView プログラマーズ・ガイド（国際化版），日本サン・マイクロシステムズ株式会社, 1991.

［9］川辺敬子ほか: 分散開発環境におけるレビュー支援, 第45回情報処理学会全国大会, 1992.

［10］I. Thomas: PCTE Interfaces: Supporting Tools in Software-Engineering Environments, IEEE Software, pp.15-23, Nov. 1989.