ソフトウェア工学 83−14
（1992. 2. 7）

ＬＯＴＯＳによるソフトウェアプロセス記述を支援するツール

吉田光輝　　佐伯元司

東京工業大学 工学部 電気電子工学科

本論文では，形式的仕様記述言語ＬＯＴＯＳを用いて，ソフトウェア開発プロセスを記述するための支援ツールについて述べる．ソフトウェアプロセスを，1) プロセス中で実行されるタスク，2) タスクを実行するための資源，とに分けてモデル化し，それらを並列に通信しあうProcessとしてＬＯＴＯＳ言語で記述する．記述するための方法論を提案し，そのための支援ツールをＸウィンドウシステム上に作成した．記述の際には，７種類の表，ダイアグラム，チャート図を中間生成物として作成していく．支援ツールは，これらの生成物を作成するための７種類のエディタからなっている．

Supporting Tool for Describing Software Processes by using LOTOS

Mitsuteru Yoshida　　Motoshi Saeki

Department of Electrical and Electronic Engineering
Tokyo Institute of Technology

This paper presents a supporting tool for describing software processes. Software processes are described with a formal specification language LOTOS. Our model to describe software processes consists of two parts — one is for tasks which are performed in a software process and another is for resources which perform the tasks. Tasks and resources are modeled as processes concurrently communicating with each other. We have developed a method to describe software processes based on our model. In our method, seven kinds of tables, diagrams, and charts are produced as intermediate products. Our tool supports the constructions of the products, and contains seven structured editors for them.

1 はじめに

ソフトウェアプロセスを形式的に記述することは，

1. 開発に携わる人が自分自身のプロセスを曖昧さなく理解できる，
2. プロセスに従って開発者を navigate することにより，効率的なソフトウェア開発支援を行なうことができる [3]，
3. 検証や検査，シミュレーションなどの手法と組み合わせることにより，そのプロセスの特性を分析し，資源の割り当て方，管理方法の決定に役立てることができ [7]，またプロセス自身の欠点を見つけられ，その改善にも役立てることができる，

といった点で有用と思われる．特に，2.，3. の目的を達成するには，記述されたプロセスが何らかの形で実行できることが不可欠である．

Osterweil も述べているように [9]，ソフトウェアプロセスもソフトウェアと見なすことができる故，形式的に記述する際には，ソフトウェアを形式的に仕様化する際の問題点と同じ問題が生じる．すなわち，いかにしてソフトウェアプロセスを形式言語で記述するかである．現在までに，種々のソフトウェアプロセスのモデル化手法や記述言語が研究されてきた [12],[8],[5]．我々は，形式的仕様記述言語 LOTOS(Language of Temporal Ordering Specification)[4] をソフトウェアプロセス記述に用い，モデル化のための方法論として，構造化分析法 [1]，オブジェクト指向分析法 [11] を用いた手法を提案した．本稿では，この手法に基づいてソフトウェアプロセスを記述する際の支援ツールについて述べる．なお，本稿ではソフトウェアプロセス記述の例として，共通問題 [6] を使用した．

2 ソフトウェアプロセスのモデル化手法

2.1 記述のためのモデル

ソフトウェアプロセスを異なった，相補的なビューで捉え，各ビューからの記述を行なう手法がある [8]．本手法では，ソフトウェアプロセスを機能的な側面と構造的な側面の 2 つに分けて捉える [10]．前者はソフトウェアプロセス中で実行されるタスク，タスクによって生成されるプロダクトを捉え，後者はタスクを実行する資源（開発者，プロジェクトチーム，コンピュータ，ツールなど）を捉える側面である．タスクの実行順序，例えば作業のスケジュールや開発者の割り当てといったタスクは他のタスクよりも先に行なわれるといったことも記述しなければならない．このようなタスクの振舞いとは別に，タスクを実行する資源の振舞いも考えられる．例えば，ユーザ，顧客，設計者といった人たちが協同であるシステムの要求仕様を作成するタスクを行なっていたとしよう．彼らはお互いにインタラクションを行ないながら，タスクを実行していく．このような資源の振舞いはタスクの振舞いと分けてモデル化し，記述するのが自然であろう．以上をまとめると，我々の記述モデルは図 1 のように表される．

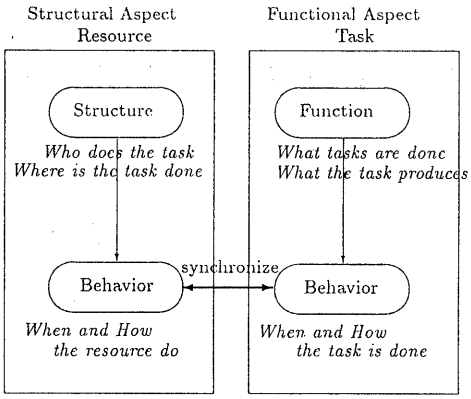


図 1: ソフトウェアプロセス記述のモデルの概略

2.2 記述の方法論

前節のモデルに従って，LOTOS 言語でソフトウェアプロセスを記述するための方法論を図 3に示す．この方法論は，機能的側面 (Functional Aspect)，つまりタスクの記述を行なう部分と，構造的側面 (Structural Aspect)，つまり資源の記述を行なう部分の 2 つに分かれる．前者は構造化分析法，後者はオブジェクト指向分析法をベースにしている．プロセス記述を完成させるために 7 種類の表やダイヤグラム，チャート図等を作成していき，これらの記述から LOTOS 記述の雛型がシステマティックに生成される．

2.3 タスク部の生成物

プロセス記述者は最初に，どのようなタスクがあり，どのようなプロダクトが取り扱われるかをリストアップする．抽出したタスクとプロダクトを表としてまとめたものが Task & Product List である．次にタスクの入力となるプロダクト，出力プロダクトを求め，プロダクトの流れをダイヤグラムで記述する．これが Product Flow Diagram である．この Product Flow Diagram は Structured Analysis の Data Flow Diagram と同様に階層的に記述することもできる．つまり，1 つのタスクを複数のサブタスクとその間の product Flow に分解して記述することもできる．タスクとプロダクトの関係に注意しながら，タスクの実行順序をチャート図 Time Chart で記述する．タスクは，逐次的，並列的，あるいは選択的に実行されることがある．Time Chart は，タスクの複雑な動作を完全に記述するためのものではなく，記述者に直観的な理解を与えるために，簡単なものになっている．

2.4 資源部の生成物

プロセス中の participant とそれらの間の関係を抽出し，Entity Relationship Diagram の形式で表現する．この Diagram を Participant Relationship Diagram と呼ぶ．この Diagram と Task List をもとに，プロセス開始時の Task の Participant への割り当てを決める．割り当ては表形式

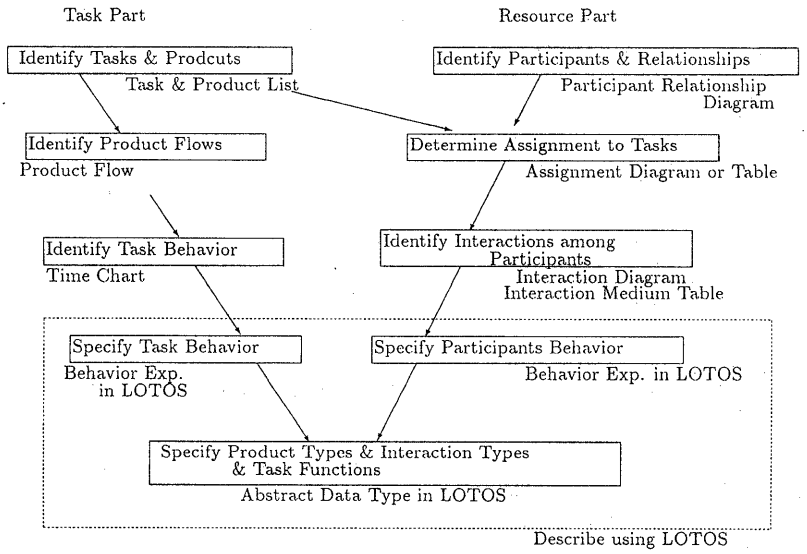


図 3: ソフトウェアプロセス記述のための方法論

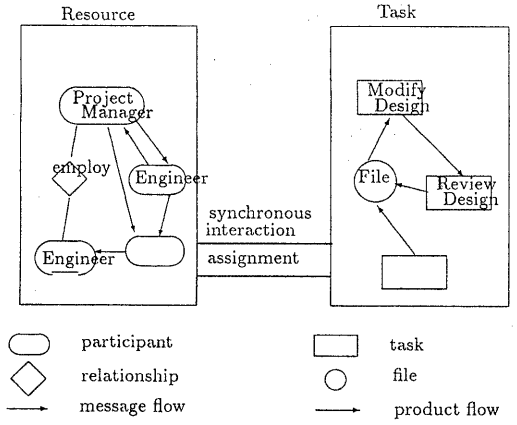


図 2: Resource パートと Task パートの例

(Assignment Table と呼ぶ) で記述される．プロセス実行中に割り当てが動的に変化する場合，その様子はこの表に反映される．最後に，抽出した Participant 間で起こるインタラクション，例えば生成されたプロダクトの受渡し等，をダイヤグラム (Participant Interaction Diagram と呼ぶ) で記述する．各インタラクションは，E-mail，手渡し，口頭等の手段で行なわれる．このダイヤグラムが資源の物理的な結合を表わすものとなっている．どのプロダクトがどういう手段で受け渡されるかは表の形でリストアップする．この表は Interaction Medium Diagram と呼ばれる．図2に Resource パートと Task パートの一例を示す．

2.5 LOTOS 記述への変換

以上の過程で得られた7種類の表，ダイヤグラム，チャート図から LOTOS 記述の雛型を自動的に生成する．LOTOS 記述の雛型とは，LOTOS の Process, Process の gate 引数，Process 間の結合，Abstract Data Type である．

Product Flow Diagram 中のタスク，ファイルは，すべて LOTOS の Process に変換される．各々の持つ入力，出力プロダクトは，Process の gate を介して渡される．各 Process は，入出力プロダクト受渡し用の gate で同期させるように連結して，雛型を生成する．例えば，図 5 の Product Flow Diagram より生成される Process Develop_Change_and_Test_Unit の雛型について考えてみよう．図中の長方形はタスク，円はファイルを表し，プロダクトフローの黒丸印側がそのフローの destination であることを表す．タスク Schedule_and_Assign_Tasks を表す Process は，ファイル file から Original Project Plan を受けとり，AAP へ Requirement change，AFP へ Notification，file へ Updated Project Plan を出力する．使用する gate は，順に g_4，g_3，g_5，g_6 である．gate 名は支援ツールによって自動的に付加される．この部分の雛型は，

```
process Develop_Change_and_Test_Unit[pg,...]
:  noexit := hide g_1, g_2, g_3,g_4,···,g_28,g_29 in
   ((((((((((((((((Schdule_and_Assign_Tasks
    [pg,sg,g_3,g_4,g_5,g_6] |[g_3]|
    AAP[g_3]( *** )) |[g_4,g_6]|
    file[g_4,g_6]( *** )) |[g_5]|
    AFP[g_5,g_7]( *** )) ···
    ............
endproc
```

となる．雛型中の***はそのファイルの初期値を表している．

Time Chart よりタスクを制御するための Process schedule を生成する．Time Chart は，タスクの実行順序を規定しているので，それに従ってタスクを起動 (wakeup)，停止 (sleep) させる Event を起こすように Process schedule が作られる．この Process と Product Flow Dia-

gram から得られた雛型が1つの gate sg で連結される．図6の Time Chart より，以下のような雛型が生成される．例えば，Schedule_and_Assign_Tasks が終了してから，Modefy_Design，Modefy_Code，Monitor_Progress，Modefy_Test_Plans タスクが開始される．これら4つのタスクは，並列に行っても構わない．この順序関係は，LOTOS コードでは，Schedule_and_Assign_Tasks が sleep した後，残りの4つのタスクが wake_up するように表現される．

```
process schedule[sg] :  exit
   := (sg !Schedule_and_Assign_Tasks !sleep;
      (sg !Modify_Design !wake_up ; exit |||
      sg !Modify_Code !wake_up ; exit |||
      sg !Monitor_Progress !wake_up ; exit |||
      sg !Modify_Test_Plans !wake_up ; exit)
      >> schedule[sg] )
      []
      (sg !Modify_Test_Plans !sleep;
      (sg !Modify_Unit_Test_Package !wake_up ;
      exit) >> schedule[sg] )
endproc (* schedule *)
```

Participant Interaction Diagram からも，Product Flow Diagram の場合と同様に，各 Paricipant を LOTOS の Process にして，雛型を生成する．このようにして生成された Resource パートの雛型は，Task パートのそれと gate pg で連結される．この gate は，Participant をどのタスクに割り当てるかを表すデータの受渡しに使用される．

図4に生成される LOTOS Process の階層構造とそれらがどの中間生成物から得られるかを示す．

3 記述支援ツールの方針と機能

支援ツールは，前節で述べたような中間生成物の作成を支援し，LOTOS 記述の雛型を生成するだけでなく，実際に LOTOS 記述を実行させ，ユーザが記述の誤りを発見したり，プロセス自身の不具合を発見したりすることも支援する．このために以下のような，ツールが持つべき機能に関する基本方針を立てて設計を行なった．

- 表，ダイヤグラム，チャート図といった種々の構造をした中間生成物を作成しなければならないため，これらの入力編集が容易に行なえるエディタが必要である．
- 階層構造の管理
- 中間生成物を論理的に関連づけて蓄積，管理できる機構が必要である．
- LOTOS 記述の実行が行なえ，実行による記述のチェックが行なえる．実行の様子が記述時に作成した Product Flow Diagram 等の中間生成物に視覚的に反映される．
- シミュレーションによる，資源の負荷などの分析が行なえる．

記述支援ツールは，

1. Chart Editor : 7種類の表，ダイアグラム，チャート図の入力，編集に使用する．各図ごとに，異なるウィンドウが用意されており，入力，編集は，各々のウィンドウで行なう．さらに，入力された図を元に，LOTOS コードの雛型を自動的に生成する機能も持っている．
2. Text Editor : LOTOS コードを入力，編集するために使用する．
3. Simulator : LOTOS コードを実行する．実行のスナップショットは，Chart Editor によって描かれた図の一部に出力され，ユーザに対する視覚的サポートが行なわれる．

の3つのパートから成る．このうち，本ツールでは，2. の Text Editor には Emacs を，3. の Simulator には SEDOS/LOTOS[2] といった既存のツールをそのまま利用した．次節で Chart Editor の機能について，その詳細を述べる．ツールを起動すると，初期メニューが表示され，ユーザは 'draw'，'edit'，'exec' の3つのコマンドのうちのどれかを選択する．これらは，各々，Chart Editor，Text Editor，Simulator を起動させるコマンドである．

4 Chart Editor の機能と構造

7つの図に合わせて，7種類のエディタが用意されている．各々，図表の要素の入力，消去，再描画，ファイルへの保存/ファイルからの読み込みができる．ユーザは，初期メニュー中の draw コマンドを選択すると，7種類の図表のうちのどれを入力，編集を行なうかを選択させるメニューが表示され，選択された図表に応じた Editor が起動される．本ツールは，マルチウィンドウシステムの上に実現されている．

4.1 図の入力，編集

本節では図，つまり Product Flow Diagram，Assignment Diagram，Time Chart，Participant Interaction Diagram の4つの入力，編集方法について述べる．これらの4つのエディタは，図形要素 (Task, File, Participant) を新たに入力するコマンドと図形要素を消去する destroy コマンドが共通に用意されている．さらに，Product Flow Diagram，Participant Interaction Diagram 用のエディタには，図形要素間に関係をつける (Task 間の Product の流れ，Participant 間の Interaction) コマンド relation が用意されている．

図の入力方法を Product Flow Diagram を例にとり，説明する．図5に Product Flow Diagram の例をあげる．ユーザは，どの図形要素，つまり task か file かのどちらを入力するかをメニューから選択し，マウスボタンをクリックする．マウスカーソルを配置する場所まで移動し，クリックする．その後，task 名，もしくは file 名をキーボードから入力する．Product Flow を入力するには，まずメニューより，relation を選択し，出力側の図形要素 (source)，入力側の図形要素 (destination) の順にクリックする．最後に Flow 名をキーボードより入力する．メニュー中の destroy コマンドは，図形要素の削除に使用する．

Product Flow Diagram は階層構造を持っているため，その階層構造中を移動するコマンドとして layer が用意されている．メニューより layer を選択後，表示されているダイアグラム中の task をクリックすると，その task に下の階層が作られ，新しい Product Flow Diagram 作成用のウィンドウが開かれる．その task がすでに下の階層を持っている場合は，その階層の Product Flow Diagram が読み込まれ，編集対象となる．

Time Chart は，task の動作可能時間を両方向矢印で表現したもので，その入力，編集は以下のようにして行なう．まずメニューより task コマンドを選択し，task 名を入力した後，その task の起動時刻，終了時刻の順に入力してい

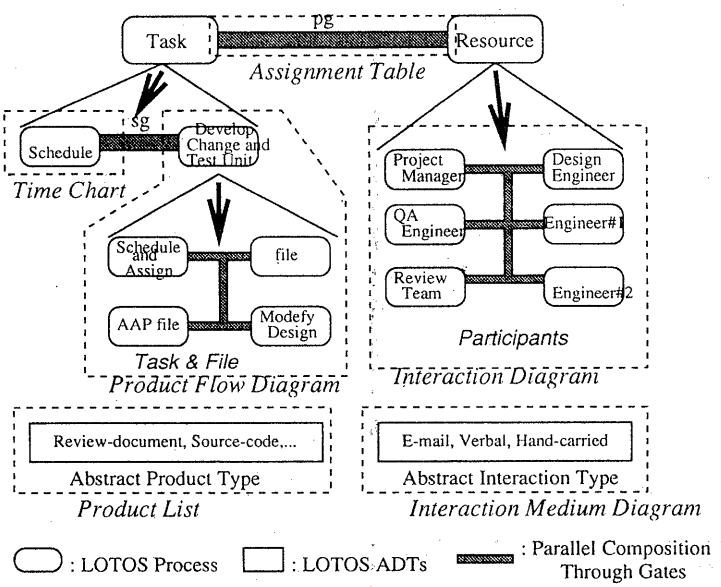


図 4: LOTOS Process の構造

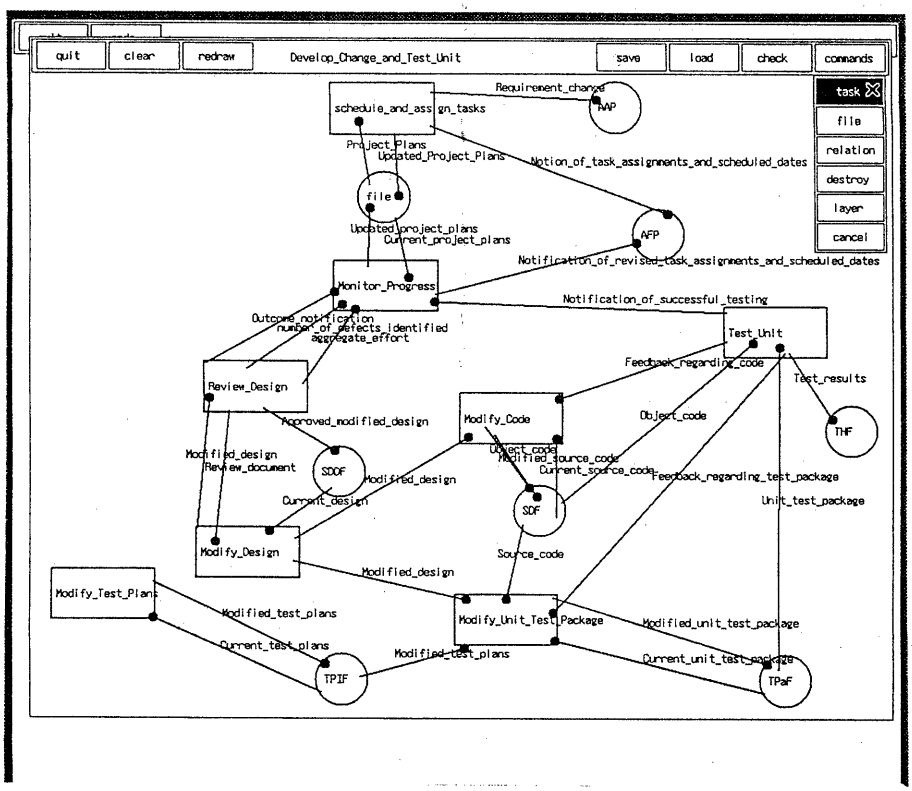


図 5: Product Flow Diagram の画面例

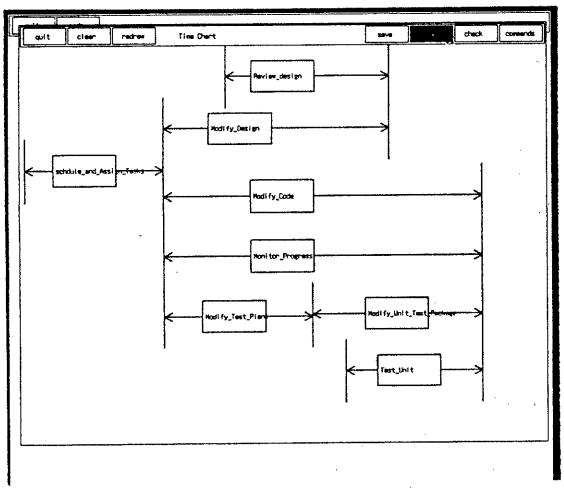


図 6: Time Chart の画面例

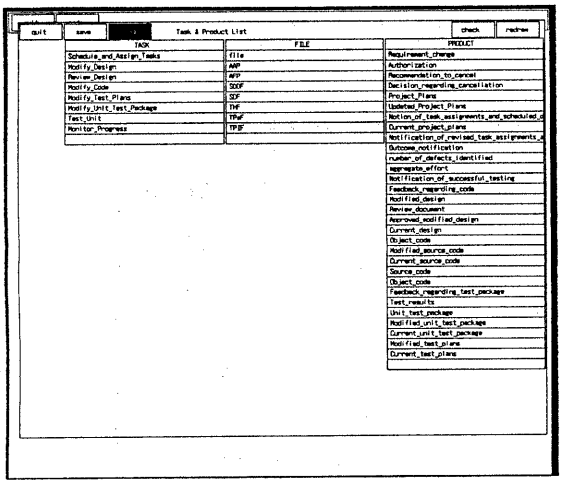


図 7: Task&Product List の画面例

く．時刻の入力は，マウスカーソルを画面中の適切な位置に移動し，クリックすることによって行なう．すでに入力してある task の起動時刻，もしくは終了時刻の近傍をクリックすると，入力した時刻はそれらと同じものとなる．Time Chart 用のエディタの画面例を図 6に示す．

4.2 表の入力，編集

この節では，表，つまり Task&Product List，Participant Relationship Diagram，Interaction Medium Diagram の 3 つの入力，編集について述べる．初期メニューよりこれらのエディタを起動すると，表の構造に従ってカラム名とスロットが表示される．例えば，Task&Product List の場合は，図 7のように，TASK, FILE, PRODUCT の 3 つのカラムが表示される．入力は該当スロットにマウスカーソルを移動し，キーボードから行なう．CR を入力すると，新たにスロットが最下段に追加される．

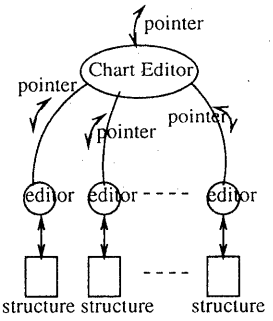


図 8: Chart Editor プログラムの構造

4.2.1 プログラムの構造

7 つのエディタごとに Task&Product List や Product Flow Diagram などの中間生成物は構造体で管理している．それぞれのエディタは，C 言語の関数で書かれ，実行が終了すると中間生成物を表す構造体へのポインタを出力値として返す．また，Chart Editor が終了する時はこれら 7 つの中間生成物のポインタが返される．このようなプログラムの構造を図 8に示す．

4.3 図形式のデータ構造

Product Flow Diagram を例にとり，図形式の中間生成物がどのようなデータ構造で保持されているかを説明する．タスク，ファイル，プロダクトフロー それぞれを一つの構造体で表している．これらは入力順にポインタ (next) で接続され，線形リストを構成している．入力順に接続したのは，図形要素の消去によるポインタのはりかえを容易にするためである．

プロダクトフローの構造体は下図のように，LOTOS コードへ変換する際に使用する gate 名（エディタが自動的につける），その source となっているタスクもしくはファイルを表す構造体へのポインタ，destination となっている構造体へのポインタを保持している．

プロダクトフロー名
gate 名
source を指すポインタ
destination を指すポインタ
...（図中の座標データなど）
次の図形要素へのポインタ (next)

図 9に Product Flow Diagram の入力例と，エディタによって作られるデータ構造をあげる．Product Flow Diagram 中の番号は入力順を示す．タスク A，B，C，プロダクトフロー flow1,flow2 がこの順で入力され，各々の構造体が入力順にポインタ next によって連結される．flow1，flow2 の構造体中にこれらの source，destination となっている構造体へのポインタが保持されている．

また，画面上での位置に関する情報を保持するために，タスク，ファイル，プロダクトフローを表す構造体はそれぞれ表示位置に関する座標を持っている．Participant Interaction Diagram も全く同様な構造をしている．さらに Time chart においては，これらのデータ以外に，そのタス

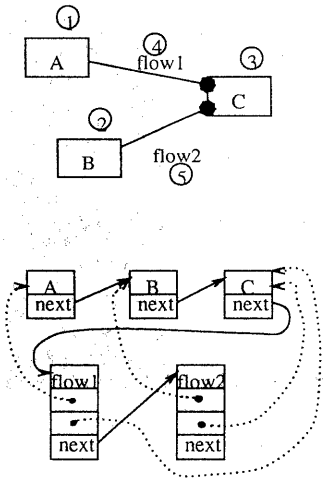


図 9: Product Flow Diagram のデータ構造

クの開始時刻と終了時刻を表すを表すデータを，矢印の終点のX座標でその構造体中に保持している．

4.4 表形式のデータ構造

図 10(a) の Task&Product List を例にとって説明する．表中の 1 つのエントリーを 1 つの構造体で表現する．図中の Task&Product List では，a1,a2,··· が Task であることを表している．また，この表では横方向のエントリ間の関係はない，つまり a1 と b1 との関係はない．従って，縦方向の関係のみをアクセスできるようなデータ構造とすればよいため，図 10(b) のような単純な構造を用いた．ポインタ var は，表中の項目 (Task, File, Product など) をつなぐためのもので，ポインタ next が表中のエントリを項目ごとにたどっていくためのものである．例えば，Task にはそのエントリ (Task 名)a1, a2, ···が線形リストの形で連結されている．

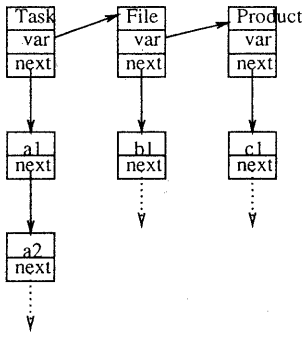
4.5 LOTOS 記述の Simulator

LOTOS 言語にはラベル付き遷移システムによって Operational な Semantics が与えられており，その Semantics に従って LOTOS 記述が実行される．本ツールでは，Simulator として ESPRIT プロジェクトで開発された LOTOS インタプリタである SEDOS LOTOS INTERPRETER(hippo と呼ばれる) を使用する．ソフトウェアプロセスを実行することにより，プロセス記述自身の誤りを発見したり，プロセスをあらかじめシミュレートすることにより，そのプロセスの不具合を発見し改善に役立てることができる．本支援ツールでは，LOTOS 記述の誤りの発見を容易にするため，Product Flow Diagram や Participant Interaction Diagram に実行の様子を表示する機能を組み込んだ．例えば，Product があるタスクから別のタスクへ流れていったときに，Product Flow Diagram 中の該当 Product Flow の表示を変え，実行状態をユーザに知らせるといった機能である．

Simulator プロセスと Chart Editor プロセスの入出力を

Task	FILE	Product
a1	b1	c1
a2	b2	c2
···	···	···

(a) Task& Product List



(b) データ構造

図 10: Task&Product List のデータ構造

パイプで連結し，Simulator からの出力を Chart Editor 側で解析する．出力が LOTOS の Event 発生を示すものであれば，Event が発生している gate を出力メッセージから抽出し，該当する Diagram やチャート図に表示する．表示は各エディタプログラムで行っている．図 11に実行例と Product Flow Diagram の表示を示す．Simulator は，タスク Schedule_and_Assign_Tasks が file より original_test_plan を読みとっている箇所を実行している．Simulator の画面上には，そのプロダクトフローに対応する gate g_4 で Event が生じていることが表示されている．Product Flow Diagram 上では対応するプロダクトフローの表示が反転している．

5 おわりに

本論文では，LOTOS によるソフトウェアプロセスの記述支援ツールについて述べた．本ツールでは，7 種類の表やダイアグラムといった中間生成物を管理するが，これらの生成物間の無矛盾性のチェックを行う機能は組み込んでいない．また，Simulator の出力を視覚的に表示する機能しかなく，シミュレート結果をプロセス改善やプロセスの実施に役立てる機能としてはまだ不十分と思われる．例えば，資源の割り当てについては，実行中に 1 つの資源にどんなタスクが割り当てられているかを表示し，資源の過負荷を検出したり，資源のスケジュールや管理のしかたをチェックしたりする機能等が有用であろう．実際のプロセス記述経験を積み，これらの機能を増強していくことが今後の課題である．

参考文献

[1] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, 1978.

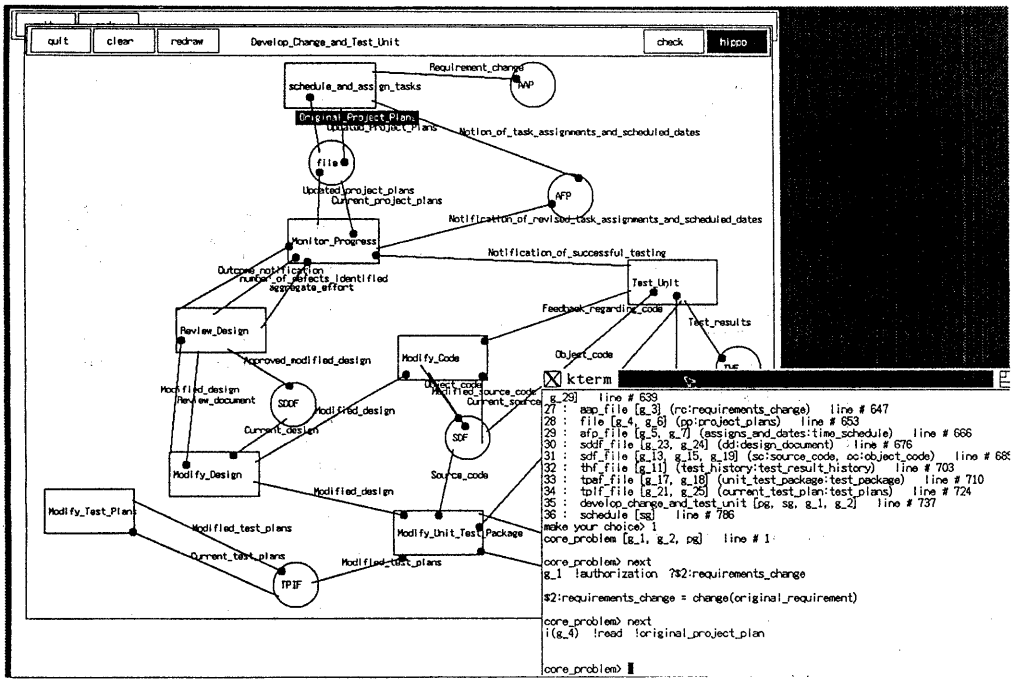


図 11: 実行例と Product Flow Diagram

[2] P. Eijk. The Design of a Simulator Tool. In *The Formal Description Technique LOTOS*. North-Holland, 1989.

[3] K. Inoue, T. Ogihara, T. Kikuno, and K. Torii. A Formal Adaption Method for Process Descriptions. In *Proc. of 11th ICSE*, pages 145–153, 1989.

[4] ISO 8807. *Information processing systems — Open Systems Interconnection — LOTOS — A formal description tecnique based on the temporal ordering of obsevational behaviour*, 1989.

[5] T. Katayama. A Hierarchical and Functional Software Process Description and its Enaction. In *Proc. of the 11th ICSE*, pages 343–352, 1989.

[6] M. Kellner, Feiler P., A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach. Software Process Modeling Example Problem. In *Proc. of 1st International Conference on the Software Process*, pages 176–186, 1991.

[7] M.I. Kellner. Software Process Modeling Support for Management Planning and Control. In *Proc. of 1st International Conference on the Software Process*, pages 8–28, 1991.

[8] M.I Kellner and G.A. Hansen. Software Process Modeling : A Case Study. In *Proc. of 22st Hawaii Conferrence*, pages 175–188, 1989.

[9] L. Osterweil. Software Processes Are Software Too. In *Proc. of 9th ICSE*, pages 2–13, 1987.

[10] M. Saeki, T. Kaneko, and M. Sakamoto. A Method for Software Process Modeling and Description using LOTOS. In *Proc. of 1st International Conference on the Software Process*, pages 90–104, 1991.

[11] S. Shlaer and S.J. Mellor. An Object-Oriented Approach to Domain Analysis. *ACM SIGSOFT Software Engineering Notes*, 14(5):66–77, 1989.

[12] L. G. Williams. Software Process Modeling : A Behavioral Approach. In *Proc. of 10th ICSE*, pages 174–186, 1988.