マルチメディア通信と分散処理 37 － 6
（１９８８．５．２０）

LOTOSのグラフィカル表現に関する一考察

高橋 薫　辻 宏郷　白鳥 則郎　野口 正一

東北大学電気通信研究所

分散システムの形式的仕様化、特に、OSIアーキテクチャの形式的仕様化を目的として、仕様記述言語LOTOSがISOで開発されてきている。LOTOS仕様の表現方法として、現在、一次元表現(テキスト表現)が規定されているが、仕様の持つ意味や読解性を向上させる観点からは、二次元的な表現(グラフィカル表現)を併せて備えることが望ましい。本論文では、このような観点から、LOTOS仕様のグラフィカル表現を提案する。具体的には、(1) プロセス定義の階層構造を図式的に分かりやすく反映させる表現、(2) テキスト表現との対応性および理解性を考慮した仕様のグラフィカル表現、を考察する。また、これらの表現を支援するためのソフトウェア環境について議論する。

Graphical Representation for LOTOS Specifications

Kaoru TAKAHASHI, Hirosato Tsuji, Norio SHIRATORI, and Shoichi NOGUCHI

Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Sendai-shi, 980 Japan

LOTOS is a formal description technique developed within ISO for the formal specification of distributed systems and in particular the OSI architecture. In this paper, we propose graphical representations for LOTOS specifications, to enhance the understandability and readability. We give two graphical forms - (1) a graphical representation reflecting the hierarchical definition of LOTOS processes, and (2) a graphical representation aimed at the enhancement of understandability of a LOTOS specification. And, we discuss a software environment for supporting these representations.

1. はじめに

システムの仕様化は、システム開発における最も基本的で重要な要件であり、仕様化に際しては、厳密で正確な記述を可能とする形式的記述法(FDT)の採用が望まれる。このような背景の下で、分散システム、特に、OSIアーキテクチャの形式的仕様化を目的としたFDTであるLOTOS (Language Of Temporal Ordering Specification) がISOによって開発されている[1]。

LOTOS仕様の表現方法としては、現在、線形的な表現(テキスト表現)が規定されている。しかしながら、LOTOSテキスト仕様の分かりにくさを回避し、仕様記述された内容を理解性良く、また、読解性に優れた形式で表現するためには、二次元的な表現(グラフィカル表現)を併せて導入することが必要である。これによって、仕様記述言語のテキスト表現に精通していない利用者にとっても、視覚的な面から、仕様の理解度、更には、言語自身の理解度を高めることが期待される。フローチャート表現やSDLグラフィック表現[4]などがこのような例となっている。

以上の観点から、本論文では、テキスト表現との親和性、および、理解性·読解性に優れたグラフ記号の積極的な活用を基本としたLOTOS仕様のグラフィカル表現を提案する。また、仕様におけるLOTOSプロセスの階層的定義の構造を視覚的に的確に反映させる表現、即ち、プロセス構造図式表現を導入する。そして、これらのグラフィカルな表現を利用者に支援·提供するためのソフトウェア支援環境を与える。

以下では、**2.**および**3.**で、それぞれ、上述したプロセス構造図式表現そしてグラフィカル表現を与える。そして、**4.**でこれらの表現を支援するソフトウェア環境について述べる。

2. プロセス構造図式表現

LOTOS仕様におけるプロセス定義の構造を視覚的に分かりやすく表現することは、プロセス定義とプロセス定義の間の関係やプロセス名のスコープ等を明確化することに寄与する。本節では、与えられたLOTOSテキスト表現におけるプロセス定義の記述構造から、それを反映させるプロセス構造図式表現を導出するための方法を与える。

図1に示すように、プロセス構造図式はラベル付きの箱とそれらの間の連結枝によって表現される。ラベルとしては、トップレベルのプロセス定義を表す"specification"とそれ以外のプロセス定義を表す"process"を用いる。箱の中には、プロセスの名前および仮パラメータリストが記される。連結枝は、枝の下位に示されるプロセス定義が、枝の上位に示されるプロセス定義に含まれるということを意味する。このような図式表現によって、プロセス定義の記述構造が容易に理解できる。図2に、この表現に対する構文を示す。構文は、グラフィカル·シンボルを終端記号として含むBNF記法を用いて定義され、構文定義に現れる非終端記号$XXXX_S$は、この表現に特有なものを表す。Sをサブスクリプトとして持たない非終端記号YYYYは、LOTOSの一次元構文(テキスト表現)[1]における非終端記号YYYYに対応する。

与えられたLOTOS仕様のテキスト表現から、対応するプロセス構造図式を導出するための方法を図3に与える。これは、

#S# ： {LOTOSテキスト表現}

→ {プロセス構造図式表現}

によって表され、#S#はテキスト表現の構文における各非終端記号からの写像として定義される。

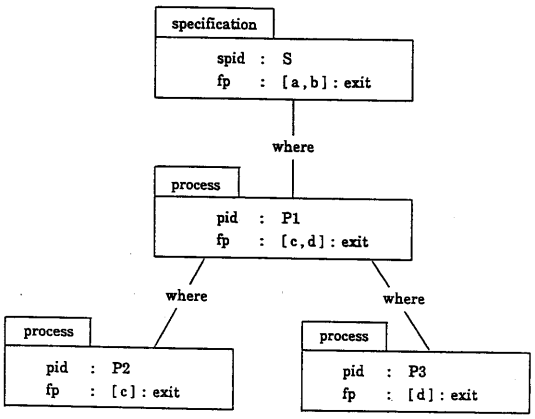


図1　プロセス構造図式表現(例)

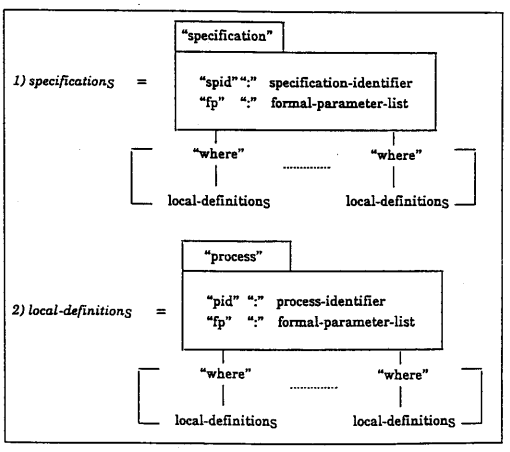


図2　LOTOSプロセス構造図式表現の構文

3. 仕様のグラフィカル表現

本節では、LOTOSのグラフィカル表現とその意味を与える。本論文で提案するグラフィカル表現は、テキスト表現との親和性を高める観点から、構文的に一対一対応の設計となっている。従って、互いの表現からの相互変換を可能としているところに特徴がある。グラフィカル表現の意味は、グラフィカル表現の構文からテキスト表現の構文への変換を与える関数によって表される。

以下、**3.1**と**3.2**で、それぞれ、グラフィカル表現の構文とその意味を与える。**3.3**では、グラフィカル表現の例を示す。

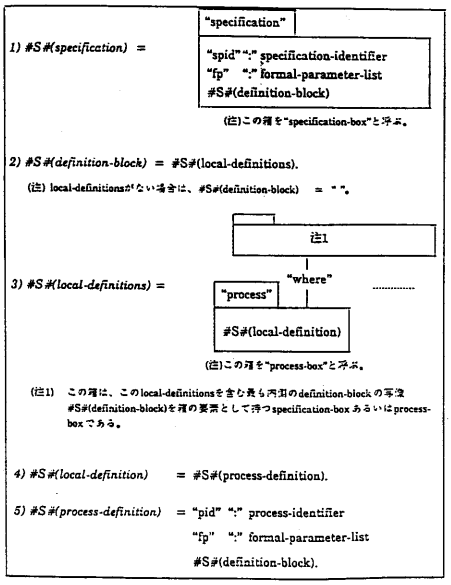
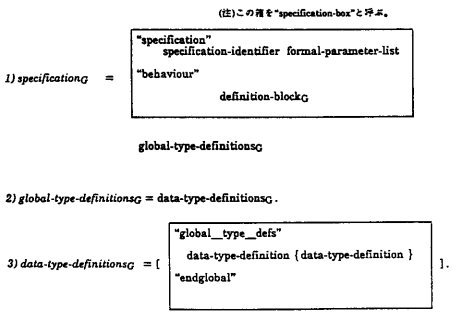


図3　写像#S#

3.1 グラフィカル表現の構文

LOTOS仕様のグラフィカル表現を、グラフィカル・シンボルを終端記号として含むBNF記法を用いて以下のように定義する。構文定義に現れる非終端記号$XXXX_G$は、グラフィカル表現に特有なものを表す。Gをサブスクリプトとして持たない非終端記号YYYYは、LOTOSのテキスト表現の構文[1]における非終端記号YYYYに対応する。



5) local-definitions$_G$ = local-definition$_G$ { local-definition$_G$ }

(注) これらの箱を"process-box"と呼ぶ。これらの箱は、他のどんな箱にも含まれない新たな箱である。

6) local-definition$_G$ = process-definition$_G$.

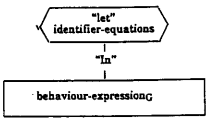
7) process-definition$_G$

= "process" process-identifier formal-parameter-list "(" of-id-sequence ")"
"behaviour"
definition-block$_G$.

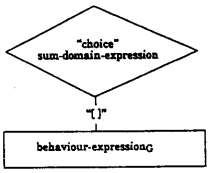
(注) of-id-sequenceは、このプロセスを一意に識別するプロセス階層のパスを表す。つまり、of pid1 of pid2 of — of spid の形をとる。ここで、pid1は、このprocess-definition$_G$を含む最も内側のdefinition-block$_G$を持つprocess-definition$_G$のprocess-identifierであり、pid2はpid1を持つprocess-definition$_G$を含む最も内側のdefinition-block$_G$を持つprocess-definition$_G$のprocess-identifierである。以下同様。最後は、specification$_G$のspecification-identifier spid となる。

8) behaviour-expression$_G$ = local-definition-expression$_G$
| sum-expression$_G$
| par-expression$_G$
| hiding-expression$_G$
| enable-expression$_G$.

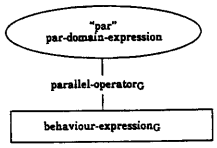
9) local-definition-expression$_G$ =



10) sum-expression$_G$ =

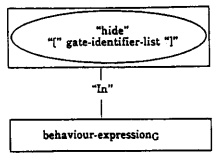


11) par-expression$_G$ =

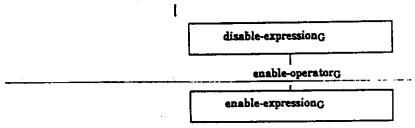


12) parallel-operator$_G$ = "|"
| "||"
| "|[" [gate-identifier-list] "]|".

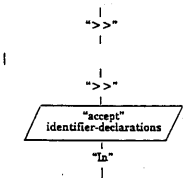
13) hiding-expression$_G$ =



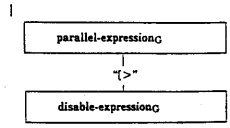
14) enable-expression$_G$ = disable-expression$_G$



15) enable-operator$_G$ =

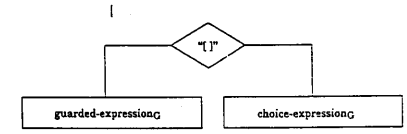


16) disable-expression$_G$ = parallel-expression$_G$

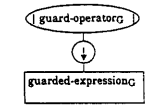


17) parallel-expression$_G$ = choice-expression$_G$

18) choice-expression$_G$ = guarded-expression$_G$

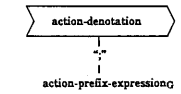


19) guarded-expression$_G$ = action-prefix-expression$_G$

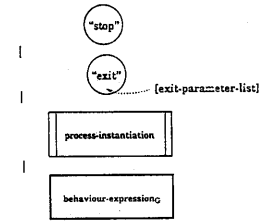


20) guard-operator$_G$ = guard.

21) action-prefix-expression$_G$ = atomic-expression$_G$



22) atomic-expression$_G$ =



23) 注意

specification$_G$によって生成されるLOTOSグラフィカル表現は、上から下もしくは左から右へと読まれる。

3.2 グラフィカル表現の意味

前節で与えたLOTOSグラフィカル表現の意味は、グラフィカル表現の構文からテキスト表現の構文への変換を与える関数【.】によって定義される。以下にこの関数を示す。【.】＝.の場合は、単に.と書かれる。

1) 【*specification*$_G$】 = "specification" specification-identifier formal-parameter-list
【global-type-definitions$_G$】"behaviour"【definition-block$_G$】
"endspec"

2) 【*global-type-definitions*$_G$】 = 【data-type-definitions$_G$】

3) 【*data-type-definitions*$_G$】 = 空 --- data-type-definitions$_G$がない時
= data-type-definition --- その他の時
—
data-type-definition

4) 【*definition-block*$_G$】 = 【behaviour-expression$_G$】 --- behaviour-expression$_G$だけの時
= 【behaviour-expression$_G$】【local-definitions$_G$】
--- その他の時

(注) local-definitions$_G$の存在は、このdefinition-block$_G$を持つspecification-boxあるいはprocess-boxに示されるspecification-identifierあるいはprocess-identifierをidとし、そのboxのof-id-sequenceをofidseq (specification-boxの時は、空)とした時、"of" id ofidseq をof-id-sequenceとして持つboxがあるかどうかによって判定できる。

5) 【*local-definitions*$_G$】 = "where" 【local-definition$_G$】
—
【local-definition$_G$】

6) 【*local-definition*$_G$】 = 【process-definition$_G$】

7) 【*process-definition*$_G$】 = "process" process-identifier formal-parameter-list
":=" 【definition-block$_G$】 "endproc"

8) 【*behaviour-expression*$_G$】 = 【local-definition-expression$_G$】
--- local-definition-expression$_G$の時
= 【sum-expression$_G$】 --- sum-expression$_G$の時
= 【par-expression$_G$】 --- par-expression$_G$の時
= 【hiding-expression$_G$】 --- hiding-expression$_G$の時
= 【enable-expression$_G$】 --- enable-expression$_G$の時

9) 【*local-definition-expression*$_G$】 = "let" identifier-equations
"In" 【behaviour-expression$_G$】

10) 【*sum-expression*$_G$】 = "choice" sum-domain-expression
"[]" 【behaviour-expression$_G$】

11) 【*par-expression*$_G$】 = "par" par-domain-expression
【parallel-operator$_G$】【behaviour-expression$_G$】

12) 【*parallel-operator*$_G$】 = "||" --- "||"の時
= "|||" --- "|||"の時
= "|[" "]|" --- "|[" "]|"の時
= "|[" gate-identifier-list "]|" --- "|[" gate-identifier-list "]|"の時

13) 【*hiding-expression*$_G$】 = "hide" gate-identifier-list
"In" 【behaviour-expression$_G$】

14) 【*enable-expression*$_G$】 = 【disable-expression$_G$】 --- disable-expression$_G$の時
= 【disable-expression$_G$】
【enable-operator$_G$】【enable-expression$_G$】
--- その他の時

15) 【*enable-operator*$_G$】 = ">>" --- ">>"の時
= ">>" "accept" identifier-declarations "In"
--- その他の時

16) 【*disable-expression*$_G$】 = 【parallel-expression$_G$】 --- parallel-expression$_G$の時
= 【parallel-expression$_G$】"[>"【disable-expression$_G$】
--- その他の時

17) 【*parallel-expression*$_G$】 = 【choice-expression$_G$】 --- choice-expression$_G$の時
= 【choice-expression$_G$】
【parallel-operator$_G$】【parallel-expression$_G$】
--- その他の時

18) 【*choice-expression*$_G$】 = 【guarded-expression$_G$】 --- guarded-expression$_G$の時
= 【guarded-expression$_G$】
"[]"【choice-expression$_G$】 —その他の時

19) 【*guarded-expression*$_G$】 = 【action-prefix-expression$_G$】—action-prefix-expression$_G$の時
= 【guard-operator$_G$】【guarded-expression$_G$】
—その他の時

20) 【*guard-operator*$_G$】 = guard "—"

21) 【*action-prefix-expression*$_G$】 = 【atomic-expression$_G$】— atomic-expression$_G$の時
= action-denotation
";"【action-prefix-expression$_G$】—その他の時

22) 【 "stop" 】 = "stop"

【 "exit" 】 = "exit"

【 "exit" 】 = "exit" exit-parameter-list
exit-parameter-list

【 process-instantiation 】 = process-instantiation

【 behaviour-expression$_G$ 】 = "(" 【behaviour-expression$_G$】 ")"

3.3 例

LOTOS仕様のグラフィカル表現の簡単な例を図4に与える。図4に対して、【.】を適用して得られるテキスト表現、即ち、意味は図5の通りである。この例は、記述対象のシステム(即ち、"System")が、インタラクションポイントinpで、環境からソートboolの値を入力し、インタラクションポイントoutpで、そのまま出力することを示している(記述の仕方は冗長ではあるが)。例から分かるように、データの記述は、本来のテキスト表現と同じ形式を採用している。また、プロセス(behaviour)の記述は、従来のSDL/GRやフローチャート表現でなじみの深い記号などを積極的に取り込んだものとなっている。

この例からも分かるように、グラフィカル表現とテキスト表現の間には一対一の対応関係があり、相互の表現変換が可能となっている。

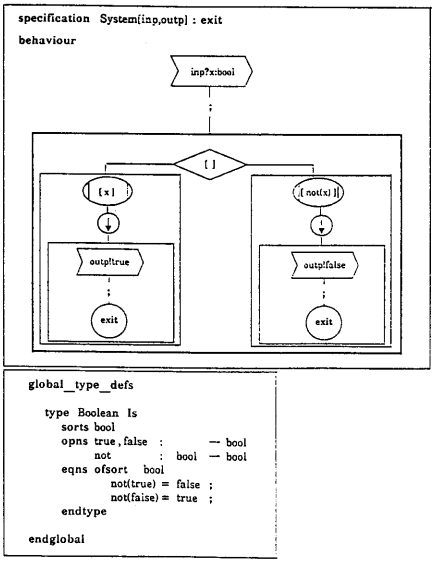


図4 グラフィカル表現(例)

```
specification     System[inp,outp] : exit
  type Boolean Is
    sorts    bool
    opns     true, false  :        → bool
             not          : bool   → bool
    eqns     ofsort  bool
               not(true)    = false ;
               not(false)   = true  ;
  endtype
  behaviour
        inp?x:bool;  ( [x]      → outp!true ; exit
                     [] [not(x)] → outp!false; exit )
endspec
```

図5 得られるテキスト表現

4. 支援環境

前節までで述べたLOTOS仕様の2つの図的な表現を、計算機支援によってユーザに提供するための環境を与える。

本支援環境では、LOTOS仕様のテキスト表現を入力とし、上述のプロセス構造図式表現およびグラフィカル表現を出力する。従って、本支援環境は、テキスト表現における仕様の記述構造や仕様それ自身のより理解性・読解性に優れた表現を自動的に生成・提示する機能をユーザに提供するものであり、これにより、仕様の持つ内容や意味をユーザに対してより的確に分かりやすく伝えるための有効な環境を備える。

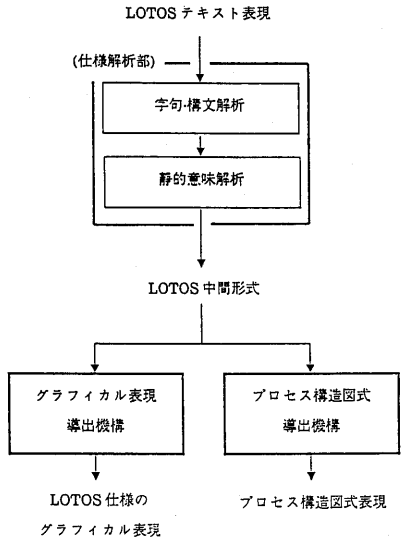


図6 支援環境の構造

図6に示すように、支援環境は以下の3つの機能から成る。

(a) 仕様解析部

仕様のテキスト表現をファイルとして入力し、LOTOSの規則に従い、字句解析、構文解析、そして静的意味解析を行う。エラーがある場合には、その旨を通知する。エラーのない仕様に対しては、以下で述べる2つの機能の実行に必要な中間表現をファイルとして生成する。

(b) プロセス構造図式導出機構

中間表現を入力とし、対応するプロセス構造図式表現を生成・表示する。

(c) グラフィカル表現導出機構

中間表現を入力とし、対応するグラフィカル表現を生成・表示する。グラフィカル表現は、前述した通り、本来のテキスト表現と一対一の対応にある。従って、その意味を与える関数【.】の

逆関数をアルゴリズムとして実現することによってテキスト表現からグラフィカル表現の導出を行っている。

本支援環境は、東芝AS3000ワークステーション上にC言語を用いてインプリメントされている。上記(b)、(c)の機能はマルチウィンドー環境の下で実行される。

図7は自動販売機のシステムのLOTOSテキスト仕様(左上のウィンドー)に対して機能(b)を実行し、対応するプロセス構造図式表現を導出・表示したものである。図8は、同じ仕様に対して機能(c)を実行し、対応するグラフィカル表現を導出し表示したものである。ここで、右下のウィンドーは、トップレベルのプロセスのbehaviourを示している。同様に、左上のウィンドーおよび左下のウィンドーは、それぞれ、サブプロセス"Vending_machine"および"Devil"を示している。ウィンドー中の見えない部分は、スクロールや優先表示などのウィンドー操作機能を用いることによって可視化可能となる。

5. むすび

本論文では、LOTOS仕様の理解性・読解性の向上を目的とした2つの図的表現を与えた。また、これらの表現をユーザに支援するための支援環境についても述べた。現在、支援環境の一環として、グラフィカル表現を基本として仕様それ自身を実行する機能(Specification Executor)の開発を行っており、それについては、別途報告する予定である。

文　献

(1) ISO : "LOTOS (Formal description technique based on the temporal ordering of observational behaviour)", ISO/DIS8807 (1987).

(2) J. P. Briand, M. C. Fehri, L. Logrippo and A. Obaid : "Executing LOTOS specifications", Proc. of 6th Int. Workshop on Protocol Specification, Testing and Verification, pp.3-1-3-19 (1986).

(3) ISO : "Proposed new question on graphical representation for LOTOS", ISO / TC97 / SC21 / WG1 T40 (June 1987).

(4) CCITT : "Functional specification and description language (SDL)", CCITT Recommendations Z.100-Z.104 (1984).

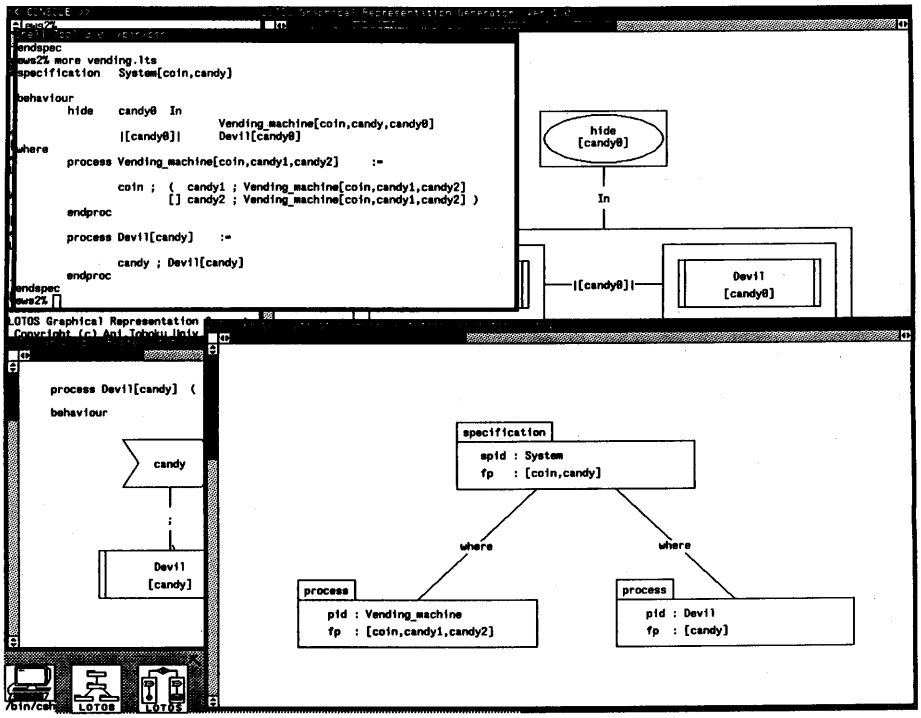


図7　プロセス構造図式表現の出力例

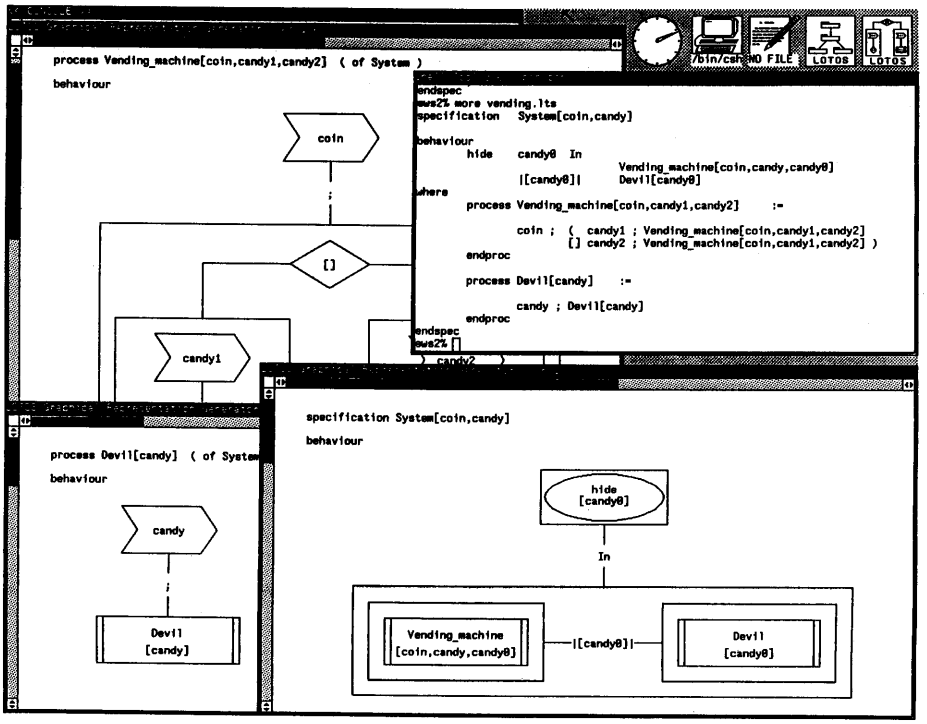


図8　グラフィカル表現の出力例