ソフトウエア工学 93－9
（1993．7．8）

プロセスプログラミングにおけるリフレクション機構の有効性

鵜飼 孝典

(株) 富士通研究所 国際情報社会科学研究所
〒261 千葉県千葉市美浜区中瀬 1-9-3
e-mail: ugai@iias.flab.fujitsu.co.jp

形式的仕様記述言語 LOTOS の動作式部分に リフレクション機構を導入し、拡張した言語として RLOTOS (Reflective LOTOS) がある。RLOTOS では、メタレベル とオブジェクトレベル の概念が存在する。メタレベルシステムはオブジェクトレベルの制御システムに関する情報を扱い、オブジェクトレベルの記述をデータとして扱い、実行するインタプリタが存在する。このインタプリタを制御することでオブジェクトレベルの動作を変更する。本論文では、RLOTOS の概要を述べ、次に RLOTOS を用いて要求獲得段階のプロセスの記述を行い、リフレクション機構がソフトウェアプロセスの記述において有効であることを示す。

Process Programming with the Reflective Facilities

Takanori Ugai

International Institute for Advanced Study of Social Information Science,
FUJITSU LABORATORIES LTD.
1-9-3, Nakase 1-Chome, Mihama-ku,Chiba-shi,Chiba, 261, Japan
e-mail: ugai@iias.flab.fujitsu.co.jp

We have proposed the formal specification language RLOTOS (Reflective LOTOS) which is an extension of LOTOS. It has *reflective computation* facilities and layered architecture which contains *object level* and *meta level* as the other reflective languages. On meta level there is an interpreter which executes the object level description, and we can change the behaviour of the object level by controlling the interpreter. In this paper, we present the outline of RLOTOS, and then we make some examples of process programming using its reflective facilities.

1 まえがき

LOTOS (Language Of Temporal Ordering Specification)[1] は OSI(Open System Interconnection) の参照モデルに基づき、通信プロトコルを記述する目的で作られた形式的仕様記述言語であり、並列性や非決定性、同期、割り込みなどの記述を行なうための構文を持つ。また LOTOS は ISO (International Standard Organization) において標準化され、多くの実際例の記述が行なわれている。さらにラベル付き遷移規則によって操作的な意味が与えられており、LOTOS の記述を直接実行させることもできる。そのため、通信プロトコルばかりでなく、他のシステムの記述にも用いられ [2, 3]、その適用性が検討されている。リフレクションの機構は計算途中においてその計算の過程や計算状態へのアクセスや変更を可能にするために、オブジェクト指向言語 [4] や、論理型言語 [5] に導入された例が報告されている。RLOTOS は LOTOS の実行可能性に注目し、リフレクションの機構を導入した言語 [6] である。RLOTOS では他のリフレクション機構をもった言語と同様にメタレベルとオブジェクトレベルの 2 つのレベルの概念を持つ。オブジェクトレベルは通常の意味でのプログラム であり、メタレベルではオブジェクトレベルの記述やその実行に関する情報を扱う。RLOTOS でのリフレクション機構の導入は、オブジェクトレベルとメタレベルに情報を分離して記述することにより、了解性の高い記述を行うことが出来るようにするために行われた。文献 [7] では RLOTOS を用いていくつかのシステムの仕様記述を行った例が報告されている。LOTOS によるソフトウェアプロセスの記述に関する研究は文献 [8] などで行われている。このなかでも議論されているように LOTOS は並列性や非決定性、同期、割り込みなどの記述を行なうための構文を持つために、部品に分割し、それぞれ独立に記述することが可能である。そのため、内部動作に関わらず、それぞれの部品のインターフェイスのみの整合性を維持すればよいという特徴を持つ。またソフトウェアプロセスモデリングにおけるリフレクションの研究として文献 [9] が報告されている。本論文で記述したソフトウェアプロセスは小規模ではあるが、実際のソフトウェアの作成時の要求獲得段階である。本論文では、RLOTOS の概要を述べ、次に LOTOS を用いて全体の概要の記述を示し、さらに、RLOTOS を用いて詳細化を行ない、最後にリフレクション機構のソフトウェアプロセスの記述における有効性について議論を行う。

2 RLOTOS のリフレクティブ機構

RLOTOS は LOTOS の実行可能性に着目し、その動作式に関するメタプログラミングを可能にするようにリフレクション機構を導入した言語である。RLOTOS におけるリフレクション機構は記述能力の向上を目的としたものではなく、オブジェクトレベルとメタレベルに分離して記述することによって、仕様の了解性を向上させることを目的としたものである。

2.1 LOTOS

LOTOS の記述は、動作に関する記述を行なう部分とデータに関する記述を行なう部分からなる。動作に関する記述部分はプロセス代数で、データに関する記述部分は多ソート代数でそれぞれ意味づけされている。動作に関する部分は外から観測される振舞いとして定義され、外部から観測可能なイベント間の順序によって LOTOS のプロセスを定義する。それぞれのプロセスは一般に複数のプロセスからなる階層構造になっており、イベントを用いてプロセス間でデータを受け渡すことが出来る。LOTOS では表 1 にあげた構成子を用いて動作式を記述する。

2.2 RLOTOS

図 1 は RLOTOS の記述とその動作に関する概念構造を示す。メタレベルには LOTOS のイ

Constructor	機能
a;B B1>>B2	順次実行
B1 [ ] B2	選択実行
B1 || B2 B1 ||| B2 B1 $\|[g_1,\ldots,g_n]\|$ B2	同期 並列実行 ゲートによる同期
B1 [> B2	割り込み
stop	停止
exit	終了

a: アクション
B,B1, B2: 動作式
$g_1,\ldots g_n$: ゲート

表 1: 動作式 の構成子

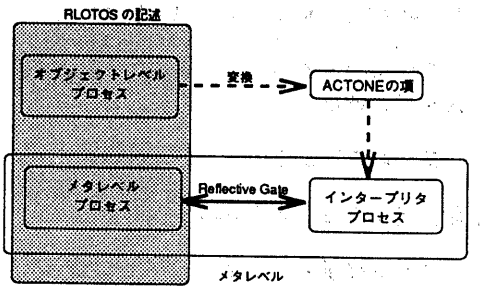

図 1: RLOTOS の概念構造

ンタープリタプロセスが存在し、RLOTOS のオブジェクトレベルとして記述された LOTOS 記述を解釈実行する。そのインタプリタプロセスは、オブジェクトレベルの記述を LOTOS で扱うことの出来る抽象データ型言語 ACT ONE の項表現に変換したものを入力とする。図 2 のように、メタレベル上のインタプリタプロセスの出力ゲート *outg* を通して出力される項が、RLOTOS によって記述された動作式が示すイベントの動作系列に対応づけられている。

RLOTOS では、リフレクティブプロセスとリフレクティブゲートという特別のプロセスとゲートを用いてメタレベルへのアクセスを行うことによって、リフレクティブ機構を用いた計算を行う。リフレクティブプロセスからインタプリタプロセスへのアクセスは LOTOS の同期の概念を用いたプロセス間の通信によって行

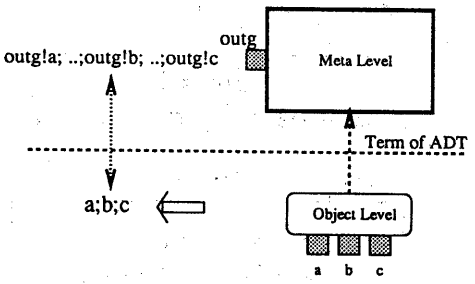

図 2: オブジェクトレベルとメタレベルでのイベントの対応関係

われると考えることができる。

RLOTOS では、LOTOS のラベル付き遷移規則に基づいた操作的意味論にしたがって、図 3 に示した 3 種類のリフレクティブゲートと呼ぶ特別なゲートが存在する。ユーザはこれらのリフレクティブゲートを用いてインタプリタを制御する。これらのリフレクティブゲートはそれぞれ次のようなデータを扱う。

currentg 現在の behaviour expression の状態

nextg 次に起こり得るイベント

restg nextg で渡されたイベントが起こった後の behaviour expression の状態

メタレベルでのインタプリタプロセスとリフレクティブプロセスは、次のプロトコルにしたがってこれらのリフレクティブゲートを使って通信を行う。

1. リフレクティブプロセスは、インタプリタプロセスから currentg を通して、現在の behaviour expression をソート Bexp の項として受け取る
2. インタプリタプロセスから nextg を通して、次に起こり得るイベントをソート Act の項として受け取り、このイベントが起こった場合の次の状態を restg からソート Bexp の項として受け取る

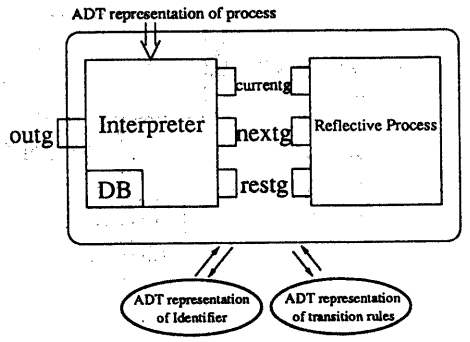

図 3: リフレクティブプロセスとリフレクティブゲート

3. リフレクティブプロセスは必要に応じて受け取った値から次に起こるイベントと次の状態を再計算する

4. 逆にリフレクティブプロセスからインタプリタプロセスへ **nextg** を通して、実際に次に起こすイベントをソート **Act** の項として渡す

5. リフレクティブプロセスからそのイベントが起こった後に移るべき状態の behaviour expression をソート **Bexp** の項としてインタプリタプロセスへ **restg** を通して渡す

6. インタプリタプロセスは渡されて来たイベントを outg を通して出力し、オブジェクトレベルでそのイベントが起こる。その後、次の状態についてこのプロトコルを繰り返す。

3 LOTOS によるソフトウェアプロセス

本論文で示すのは、要求仕様の獲得部分の実際の例を、[8] のモデルを一部変更し、記述したものであり、プロジェクトへの参加者の動作に関する部分である。

全体を参加者の相互作用 (interaction) として記述し、参加者それぞれの振舞をプロセスとして仕様化する。

3.1 記述対象

本稿で記述するプロセスは次のようなものである。

1. システム提供者 (client) からの要求仕様の提起
2. システム利用者 (user) からの要望の提起
3. システム提供者へのインタビューによる要求仕様の獲得
4. 設計者 (designer) による要求仕様の分析
5. システム利用者へのインタビューによる要求仕様の獲得
6. 要望の解析による要求仕様の洗練
7. 設計者によるシステムの提案
8. システム提供者の承諾
9. 要求仕様の決定

それぞれの段階で任意回の繰り返しがあり、client の要求も任意の時点で割り込むこととする。

3.2 通常動作の LOTOS による記述

本節では上記のプロセスの概要を LOTOS で記述した例を示す。以降の節でそれをもとに、RLOTOS のリフレクティブ機構を用いて詳細化、追加を行う。

今回記述を行うプロセスでは、参加者は次のようになっていることが分かる、

- システム提供者 – システムの発注を行い、仕様に関する決定権を持つ
- システム利用者 – システムの利用者であり、発注されるシステムについて要求を述べることができる

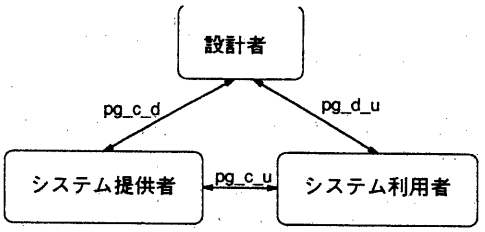

図 4: 記述対象プロセス

- 設計者 – システム提供者、システム利用者からの要求を獲得、分析し、設計を行い、プレゼンテーションを行い、発注を受ける。

本プロセスでは以上の 3 人が図 4のように話し合いを行なって、要求仕様を確定する。

- 全体

```
specification soft_process
          [pg_c_u,pg_u_d,pg_d_c]:exit:=
type event is
sorts Event
opns request:     -> Event
     interview:      -> Event
     analyze:        -> Event
     ack:            -> Event
     presentation:   -> Event
endtype
behaviour
  get_requirement[pg_c_u,pg_u_d,pg_d_c]
process get_requirement
         [pg_c_u,pg_u_d,pg_d_c] : exit :=
   (client[pg_d_c]|[pg_d_c]|
        designer[pg_d_c,pg_u_d])|[pg_u_d]|
        user[pg_u_d]
endproc
endspec
```

pg_c_u はシステム提供者とシステム利用者のやりとりが行なわれるイベントを示し、pg_u_d はシステム利用者と設計者のやりとりが行なわれるイベントを示し、さらに、pg_d_c は設計者とシステム提供者のやりとりが行なわれるイベントを示す。システム提供者とシステム利用者、設計者は独立に記述され、それぞれのやりとりをするイベントで同期して、全体のプロセスの動作となる。Event 型の sort は、参加者間のやりとりの内容を示す。例えば、pg_c_u ! request はシステム提供者とシステム利用者のあいだで要求が起こったことを示す。

- システム提供者

```
process client[pg] : exit :=
     (pg !request ; client[pg])
  [] (pg ?present:Event ;pg !ack ; exit)
endproc
```

ゲート pg を通して、client は外界とデータを交換する。pg !request は client が要求を出すことを示し、pg?x:Event はプレゼンテーションを受けることを示す。client は任意回要求を出し、プレゼンテーションを受けて、了解する。

- システム利用者

```
process user[pg]:exit:=
    (pg!request; client[pg])
endproc
```

利用者は聞かれると次々と要求を言う。

- 設計者

```
process designer[pga,pgb]:exit:=
    hide pgc in
    get_req[pga] >> get_req[pgb]
    >> do_interview[pga,pgc]
    >> do_interview[pgb,pgc]
    >> do_present[pga]
    exit
where process get_req[pg]:exit:=
         pg ? x:Event ;(exit[]get_req[pg])
      endproc
      process do_interview[pg,pgc]:exit:=
          pg ?interview:Event;pgc !analyze;
               (exit[]do_interview[pg,pgc])
      endproc
      process do_present[pg]:exit:=
        pg!presentation;pg?answere:Event;exit
      endproc
endproc
```

設計者は client からの要求を聞き、user からの要求を聞き client へインタビューを行い、それを分析し、user へインタビューを行い、それを分析、client に提案を行って、承諾をもらって終了する。このプロセスの中で pga は client と話をするためのイベント、pgb は user と話をするためのイベント、pgc は自分自身のイベントを示す。

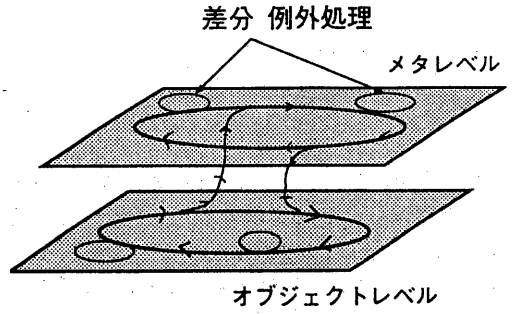

図 5: 差分記述, 例外処理の分離

4 RLOTOS によるソフトウェアプロセス記述

本節では、今回行なった記述の中で RLOTOS の特徴が顕著に現れた 3 つの部分について述べる。1 点目は段階的に詳細化していく際に、追加部分をメタレベルに記述した部分である。これは雛型を作っておき、それからの差分のみを記述する際に有効である。2 点目は、例外処理をメタレベルで記述した部分である。この部分では、通常動作に関する記述に変更を加えることがないため、保守性の向上にも有効である。最後は、オブジェクトレベルの振舞いの動的な変更の部分であり、その動作を環境によって変化させる。つまり、通常の記述方法では繁雑になる大域的な環境へのアクセスを容易に記述できる。

4.1 差分記述

今回の記述対象となったプロセスでは、client はプレゼンテーションを受けた後さらに追加で要求を出すことができる。この追加分をメタレベルとして、図5のように分離して記述する。

```
process meta-level[currentg,nextg,restg]
                                :noexit:=
  currentg?x:Bexp;nextg?y:Act;restg?z:Bexp;
  [isout(y,presentation)] ->
       nextg!y;(restg!pre(proc(add_client,z))
                [] (restg!z))
  meta-level[currentg,nextg,restg]
endproc
process add_client[pg_c_d] : exit :=
  pg_c_d !request; pg_c_d !presentation;
  pg_c_d !ack ;exit;
endproc
```

RLOTOS のメタレベルで制御できるのは、外から観測されるイベントである。したがって add_client では client と desinger のやりとりに関するイベントの操作を行ない、プレゼンテーションの前に request を一回入れている。

4.2 例外処理の分離

Behaviour expression の任意の時にある Behaviour expression が割り込み、その時点に戻ってくるような Behaviour expression の例を示す。

```
process client[pg]:exit:=
  pg!;exit
endproc
```

の任意の時点で

```
process work[pg]:exit:=
  pg!request;exit
endproc
```

が割り込み、割り込んだところに戻ってくることを LOTOS で記述すると

```
process foo[pg]:exit:=
  pg;(work[e,f][]exit)>>(b;(work[e,f][]exit))
>> ..略..
>>d;(work[e,f][]exit)
endproc
```

とプロセス foo を書き直すことになり、a;b;c;d;exit が正常動作であり、bar が割り込みによる処理であることを理解することが困難になる。これを RLOTOS で記述するとプロセス foo に変更を加えることなく、

```
process meta-level
                [currentg,nextg,restg]:noexit:=
  currentg?x:Bexp;nextg?y:Act;restg?z:Bexp;
  nextg!y;(restg!pre(proc(work,z))
        [](restg!z)
  meta-level[currentg,nextg,restg]
endproc
```

と記述でき、プロセス work が割り込むか割り込まないかの非決定的選択による動作を用いて直接表現できる。また記述量も少ないものとなる。今回は、client からの要求が任意の時点で起こることを、これによって記述した。

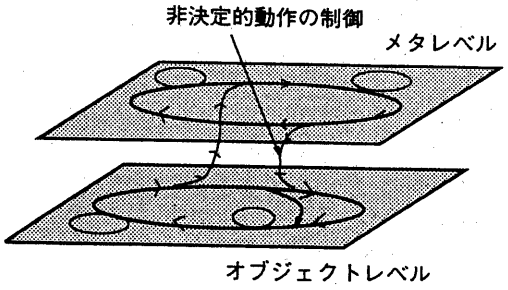

図 6: 非決定的動作の制御

4.3 同一名モジュールの動的制御

マクロ機能を持つ多くの言語では、そのマクロに関する制御はその言語とはことなる構文規則を持つ。また多くの場合、マクロを実行時に環境に合わせて展開することはできない。

例えば interview というプロセスは、designer は client に対しては short_presentation をしたあと hearing をする。user に対しては quiestion をして hearing をする。これを非決定的選択で記述すると相手を選ぶことができないので、図6のように相手にしたがってその動作を切替えることをメタレベルによる制御と考える。

```
process meta-level
          [currentg,nextg,restg]:noexit:=
  currentg?x:Bexp;nextg?y:Act;restg?z:Bexp;
  [getname(y,pg_c_d)] ->
     nextg!putval(y,short_presentation);
                                  restg!,z
  [getname(y,pg_c_u)] ->
     nextg!putval(y,presentation);restg!,z
  [others] ->
        nextg!y;restg!,z
  meta-level[currentg,nextg,restg]
endproc
process interview[pg]:exit:=
  (pg!short_presentation;pg!hearing)
[](pg!question;pg!hearing)
endproc
```

この例では、非決定的動作をメタレベルのプロセスによって決定的な動作に変えている。

5 考察

RLOTOS をはじめとするリフレクション機構を備えた言語では、オブジェクトレベルとメタレベルの概念を用いることができる。メタレベルでは、オブジェクトレベルの動作を大域的に把握し、動作の制御を行うことができる。したがって、エラー処理などの例外動作に関する部分をメタレベルに記述することは、書かれた仕様を分かりやすいものとし、非常に有効である。また例外処理を加えることによって通常動作への変更を行う必要が無いために、保守性を高めることが可能になる。さらに RLOTOS では実際のエラー処理はオブジェクトレベルに記述し、その制御に関する記述のみをメタレベルに記述することが可能であるために、通常のリフレクション機構を備えた言語で言われるメタプログラミングの繁雑さが軽減される。

差分記述に関しては、通常のソフトウェアの開発では、前段階で要素であるものを後からプロセスに詳細化する際に、外から観測されるイベントに関するプロセスを分離して記述することができる。

ただし RLOTOS のリフレクション機構はプロセス毎にメタプロセスを持つものではないために外部から観測されるイベントに対する操作のみが行なえる。そのため差分記述をプロセス毎に行なうことは困難であり、今後、プロセス毎にメタプロセスをもつようなモデルのリフレクション機構が必要であると考えられる。

例外処理の記述に関しては、LOTOS では割り込みから戻って来ることは言語デザイン上、仮定されていない。しかし、いくつかのソフトウェアでは割り込みからの復帰機能が必須のため、RLOTOS による記述が非常に有効である。特に、例外処理などでプロセスを越えた情報を利用する場合には、非常に有効である。ソフトウェアプロセスでは、障害による割り込みと、それを復旧して通常の業務に戻ることは、日常的に行なわれるため、このように記述できることは非常に有効である。

同一名モジュールの動的制御として、非決定的動作をメタレベルのプロセスによって動作

を決定的なものに変える例を示した。これはLOTOSでプロセスに分離することによって外部と無関係にそれぞれの動作を記述するが、特別な場合として外部の情報が必要な場合がある。ソフトウェアプロセスでは、おなじプロセスでも相手によって多少動作を変えることは例外処理と同様に日常的な事である。

6 あとがき

本論文では、まずRLOTOSのリフレクション機構について述べた。そしてそれを用いてソフトウェアプロセスの記述を行った。オブジェクトレベルとメタレベルの概念を用いて記述することで、差分プログラミングの可能性を示し、例外処理を通常処理から分離して記述できることを示した。これによってリフレクションの機構がプロセスプログラミングにおいても有効であることが示された。

今後は、リフレクション機構を備えた言語を用いた際の記述技法や開発支援環境、ソフトウェアプロセス支援環境の整備が課題である。

謝辞

本論文の一部は、筆者が東京工業大学に在学中に行った研究に基づきます。RLOTOSについて助言をいただいた佐伯元司助教授、広井武氏に感謝の意を表します。

参考文献

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

[2] 大蒔他. 図書館の問題とエレベータの問題のlotosによる仕様記述. 情報処理学会ソフトウェア工学研究会, Vol. 64,, 1989.

[3] 大蒔, 二木. 形式的仕様記述言語LOTOSの仕様経験. 情報処理, Vol. 31, No. 10, pp. 1440–1413, 1990.

[4] Takuo Watanabe and Akinori Yonezawa. Reflective Computation in Object-Oriented Concurrent System and Its Applications. In *Proc. of Fifth IWSSD*, pp. 56–58, 1989.

[5] Jiro Tanaka. An Experimental Reflective Programming System written in GHC. *Journal of Information Processing*, Vol. 14, No. 1,, 1991.

[6] 鵜飼孝典, 広井武, 佐伯元司. 仕様記述言語LOTOSにおけるリフレクション：RLOTOS. 情報処理学会研究報告, Vol. 92, No. 20, pp. 1–10, 1992.

[7] 広井武, 佐伯元司. リフレクション機構を持った仕様記述言語RLOTOSとその応用. 情報処理学会研究報告, 1992.

[8] 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*, pp. 90–104, 1991.

[9] Sergio Bandinelli and Alfonso Fuggetta. Computational Reflection in Software Process Modeling: the SLANG Approach. In *Proc. of 15th ICSE*, pp. 144–154, May 1993.