リフレクション機構を持った仕様記述言語のための記述支援

鵜飼 孝典

(株) 富士通研究所 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 を用いて仕様を記述するための総合環境について述べる。とくに本環境では、メタレベルでデータとして表現されるオブジェクトレベルの動作をユーザに理解しやすい形で提供できるようになっている。

An Environment for A Reflective Specification Language

Takanori Ugai

Institute for 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 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 an environment for the formal specification development in RLOTOS. The environment provides some useful tools for representing the meta information which is the behaviour of object level.

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ために、LOTOS のイ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。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, 8] 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。

本論文では、RLOTOS においてオブジェクトレベルの動作がメタレベルでどのように表現されているかを述べ、つぎに RLOTOS を用いて仕様を記述するための総合環境について述べる。RLOTOS ではメタレベルの表記とオブジェクトレベルの表記が異なるためにその対応関係を把握することが困難になっている。これを解決するために本環境では、メタレベルで表現されるデータとオブジェクトレベルでの対応する動作式を相互変換するツールを備えているためにユーザは全体の動作をより容易に理解することが可能になる。

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。また、イベントを通してプロセス間で渡されるデータは、代数型言語で等式によって表現する。

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。

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

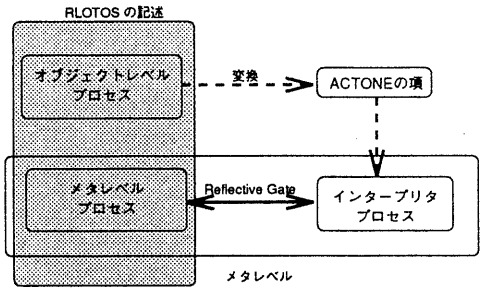

図 1: RLOTOS の概念構造

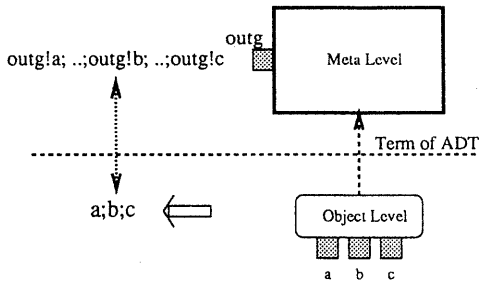

図 2: オブジェクトレベルとメタレベルでのイベントの対応関係

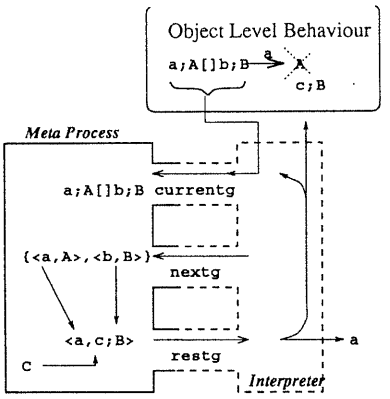

図 3: リフレクティブプロセスとリフレクティブゲート

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 interpriter プロセスによって計算された次に起こり得るイベントとそのイベントが起こった後の状態の対の集合

restg メタプロセスによって必要に応じて再計算された次に起こり得るイベントとそのイベントが起こった後の状態の対,

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

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。

図 3 では $a;A$ または $b;B$ がおこる動作式に対しメタプロセスでは、前者から a をとり次に起こるイベントとし、後者の B の前に c というイベントを加えて a が起こった後の状態にするという計算を行なっている。

2.3 動作式のデータによる表現

2.3.1 イベントの項による表現

リフレクションの機能を持った言語では、メタレベルに関する記述もまたその言語自身で記述され、オブジェクトレベルの記述はメタレベルではデータとして扱われる。LOTOS では ACT ONE の項がデータを表現する。ここでは LOTOS の記述の ACT ONE の項による表現を定義する。[8] で述べられている様にして、動作式を表 2 のように抽象データ型の Exp の型の項で定義する。LOTOS の動作式を抽象データ型によって表現する方法については文献 [9] でも検討されている。

Constructor	ACT ONE による表現
a;B	pre(a, B^*)
B1 ⟩⟩ B2	ena($B1^*$, $B2^*$)
B1 [ ] B2	cho($B1^*$, $B2^*$)
B1 || B2	sync($B1^*$, $B2^*$)
B1 ||| B2	para($B1^*$, $B2^*$)
B1 |[$g_1, \ldots, g_n$]| B2	gen($B1^*$, *gate-set*, $B2^*$)
B1 [⟩ B2	dis($B1^*$, $B2^*$)
stop	act_stop
exit	act_exit

a:	アクション
B,B1, B2:	動作式
$g_1, \ldots g_n$:	ゲート
gate-set:	ゲートの集合の ACT ONE 表現
a:	アクション a の ACT ONE 表現
B^*,$B1^*$, $B2^*$:	動作式 の ACT ONE 表現

表 2: 動作式 の項による表現

例えば、 動作式 "$((a;b;exit)[](c;b;exit))$ ||| $((a;b;exit)[](c;d;exit))$" は次のような項によって表現される。

$$para(cho(pre(a, pre(b, act\_exit)),\ pre(c, pre(b, act\_exit))),\ cho(pre(a, pre(b, act\_exit)),\ pre(c, pre(d, act\_exit))))$$

(ただし a,b,c,d を
アクションの ACT ONE 表現とする)

アクションの ACT ONE 表現は次のように定義する。

A !*value* ?*variable*:*type*...[*guard*]
⇓
atr(A, oup(*value*)+inp(*variable*, *type*)+..., *guard*)

atr:	Name, SymbolList, Symbol → NameId
oup:	Symbol → Symbol
inp:	NameId, SortId → Symbol
NameId :	ゲートの名前の型
SortId :	変数の sortname の型

"atr" という構成子は、ゲートの識別子、入力の変数の識別子、出力の値の式からアクションの項表現を構成するために使われる。例えば、$a!x$ というアクションは "atr(a, outp(x), true)" という項によって示される。

2.3.2 遷移規則の項に対する操作による表現

遷移規則 "$B_1 \xrightarrow{a} B_2$" では $a' \in head(B_1')$ と $rest(\{a'\}, B_1') = B_2'$ と定義される。ただし、B_1'

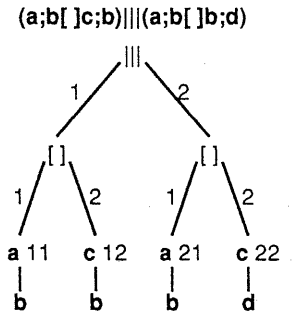

図 4: イベントの出現位置の番号づけ

と B_2' はそれぞれ B_1、B_2の抽象データ型による表現であり、a' はイベントの名前を示す a と B_1 中の出現位置の対である。

関数 "*head*" は動作式にあらわれる同じ名前のアクションを区別するために図 4 のようにその出現位置を示す番号を付ける。関数 "*head*" と "*rest*" はアクションの名前を示す "NameId" 型の項とその出現位置を示す "IDNumber" 型からなる "Act" 型の項を操作する。関数 "*head*" は次に起こり得るイベントの集合を示す "ActSet" 型の項を返す。関数 "*rest*" は関数 "*head*" が返す集合の要素である "Act" 型の項と動作式を示す "Exp" 型の項を入力し、"Exp" 型の項を返す。このようにして関数 "*head*" と "*rest*" は LOTOS のラベル付き遷移規則に対応して "Exp" 型の抽象データ型上に定義される。以下にその一部を示す。

```
type Exp is ...
sorts Exp
opns
  pre     : NameId,Exp -> Exp
  para    : Exp,Exp -> Exp
  cho     : Exp,Exp -> Exp
  (* constructors
           for behavor expression *)
  head    : Exp,Number -> ActSetSet
  rest    : ActSet,Exp -> Exp
eqns
forall E1,E2:Exp, A:Act,
  I:NameId, S1,S2:ActSet, N:Id_Number
ofsorts ActSetSet
  head(pre(I,E1),N) =
    InsertSet(Insert(num_atr(I,N),{}),{}) ;
          (* rule for action prefix *)
  head(cho(E1,E2),N) =
        Union(head(E1,N+1),head(E2,N+2)) ;
  head(para(E1,E2),N) =
        Union(head(E1,N+1),head(E2,N+2)) ;
      (* rule for interleaving operator *)
  ....
ofsorts Exp
 rest(InsertSet(A,{}),pre(I,E1)) = E1 ;
        (* rule for action prefix *)
 IsIn(S1,head(E1)) =>
       rest(S1,cho(B1,E2)) = rest(S1,E1);
 IsIn(S1,head(E2)) =>
       rest(S1,cho(E1,E2)) = rest(S1,E2);
 IsIn(S1,head(E1)) =>
        rest(S1,para(E1,E2))=
                  para(rest(S1,E1),E2);
 IsIn(S1,head(E2)) =>
        rest(S1,para(E1,E2))=
                  para(E1,rest(S1,E2));
      (* rule for interleaving operator *)
  ....
endtype
```

例として 2.3 節で示した動作式 $(a; b; exit$ [] $c; b; exit)$ ||| $(a; b; exit[]c; d; exit)$ に "*head*" "*rest*" を適用した結果を示す。簡単化のために "A" と "id-numver" からなる "Act" 型の項 *num_atr*(*A*,*id-number*) は "A(*id-number*)" と表記する。ここで "A" はアクションの名前でありその動作式中の出現位置を "id-number" とし、"0" は "Id_Number" 型の初期値を示す項である。また、集合を示す構成子も省略し、{} と表記する。

```
head(para(cho(pre(a,pre(b,act_exit)),
      pre(c,pre(b,act_exit)),
      cho(pre(a,pre(b,act_exit)),
      pre(c,pre(d,act_exit)))), 0) =
            {a(11),a(21),c(12),c(22)};
rest(a(11), para(cho(pre(a,pre(b,act_exit)),
      pre(c,pre(b,act_exit))),
            cho(pre(a,pre(b,act_exit)),
            pre(c,pre(d,act_exit))))) =
      para(b,cho(pre(a,pre(b,act_exit)),
      pre(c,pre(d,act_exit))));
```

2.4 RLOTOS による記述の際の問題点

RLOTOS では、ユーザが記述するメタプロセスでオブジェクトレベルの表現が抽象データ型に変換されているために

- 中置式から前置式に変換されている
- オペレータがソートのオペレータに変換され、名前が変更されている

という繁雑さがある。

さらにメタプロセスを含む全体のプロセスの動作を把握しにくいという問題を持っている。

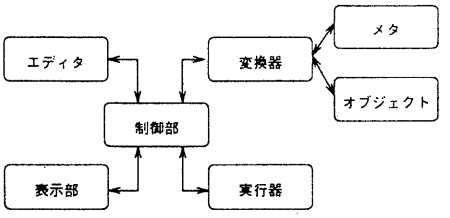

図 5: 全体構成

3 RLOTOS 記述支援環境 – RLOTOSE

本節では RLOTOS 記述支援環境 RLOTOSE (RLOtos TOtal Support Environment) について述べる。

本環境では、前節で述べた規則にしたがってオブジェクトレベルとメタレベルの情報を相互変換し、表示することによってメタレベルでデータとして表現されているオブジェクトレベルの動作をユーザにわかりやすい形で提供できる。

3.1 全体構成

本環境は、図 5 のようにエディタ、**表示部**、**実行器**、**変換器**、**制御部**の 5 つの部分からなっている。表示部は、実行器、変換器の結果を制御部を通して受け取り表示する。エディタは、仕様の入力を行なう部分であり、キーボードからの入力の他に表示部からマウスによるカットアンドペーストによって入力することが可能である。実行器は、エディタで入力された RLOTOS による仕様を実行する実行系である。制御部は、以上の 4 つの部分を接続し、それぞれの部分とデータのやりとりを行なう。

ユーザは、図 7 のようにエディタとオブジェクトレベルとメタレベルの相互変換の結果の表示と実行の経過を表示する 3 つの部分を操作する。

以下の節でそれぞれについて説明する。

3.2 メタレベルとオブジェクトレベル(変換器)

図 6 のようにこの環境には、仕様中の静的なメタレベルの情報をオブジェクトレベルの形で表示したり、逆に開発のためにオブジェクトレベルの形で入力した情報をメタレベルの表現に変換する変

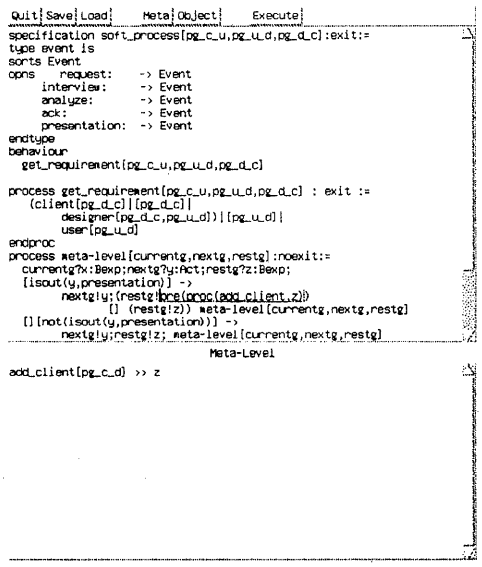

図 6: メタレベルとオブジェクトレベルの変換

換器を備えている。この変換器は、2.3節で述べた動作式のデータによる表現を双方向に変換する。さらにこれを利用して実行器でステップ実行しているときに、メタレベルの様子をオブジェクトレベルの表現で表示することによって、ユーザは実行の様子を容易に理解することができる。

図中の上半分はエディタであり、下半分が変換器による変換結果を表示する部分である。図では、エディタで入力されたメタプロセスの記述の一部をオブジェクトレベルの表現に変換し表示している。

3.3 実行器

図 7 に実行器を利用して記述した仕様を実行させる様子を示す。図では実行時にメタレベルの状態を実行器によって表示させ、それをオブジェクトレベルの表現に変換している様子を示している。

本環境中の実行系は、オブジェクトレベルとメタレベルからなる仕様を実行することができる。現在のところメタメタレベル、メタメタ.. レベルを含む仕様の動作はサポートしていない。

RLOTOS ではメタレベルを構成するインタープリタを LOTOS 言語自身で記述しそれを制御する機構も LOTOS の構文上で定義されている。従って、これらの機構の実現には図 8 のように LOTOS

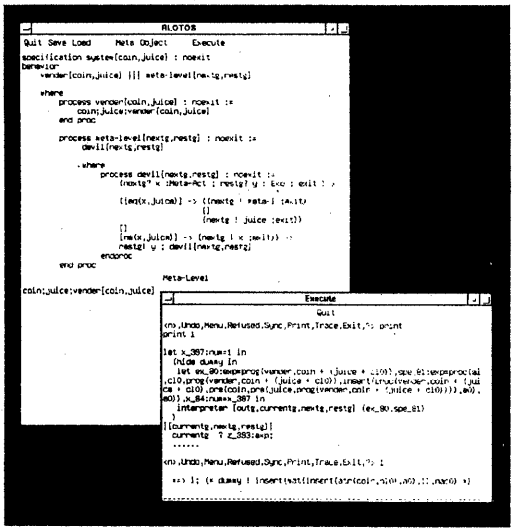

図 7: 実行の様子

の実行形を利用することができる。RLOTOS インタープリタはまず、RLOTOS の記述からメタプロセスを構成し、メタレベルのインタプリタプロセスへの入力がオブジェクトレベルの項による表現になるように変換する。そして、メタレベルのプロセスを LOTOS の実行形によって実行する。そこから出力されるイベント列からオブジェクトレベルのイベント列を次のように計算する。オブジェクトレベルのイベントは、メタレベルでは outg に出力される値となる。オブジェクトレベルのイベントに付加される値は outg に出力される値の内部構造に含まれる。同様に考えるとオブジェクトレベルのイベントはメタメタレベルでは、outg から出力される値の内部構造に含まれる。

3.4 表示部

表示部は先に述べたように変換器の結果を表示する部分と実行器の動作を表示する部分からなる。エディタを含めてマウスによるカットアンドペーストによってデータ交換を行なうことが可能である。

4 RLOTOSE を使った開発

RLOTOS によって開発を行なう場合

1. オブジェクトレベルの記述

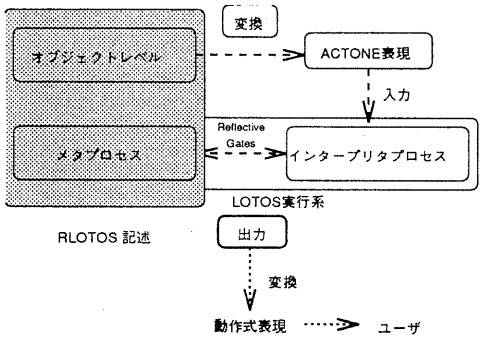

図 8: 実行器

2. メタプロセスの記述

の 2 つのフェーズに分けることができる。

RLOTOSE を利用した場合、以下のような手順で開発を行なう。

1. オブジェクトレベルの記述はエディタと実行器を用いて記述を行なう
2. RLOTOSE を用いてメタプロセスを記述するとき、ユーザはオブジェクトレベルの記述形式で式を表現し、メタレベル形式に変換してメタプロセスの記述に利用する
3. 記述後、実行器を使ってメタプロセスを含む全体の動作を確認する
4. 必要に応じて対話的に動作中のメタレベルの情報をオブジェクトレベルの形式で確認する

5 LOTOS の開発環境

LOTOS による仕様記述の支援のためのツールには、文献 [10] であげられているようにシミュレータ、エディタ、仕様からの試験系列の導出、他の FDT への変換、仕様の自動インプリメンテーションなどが提案されている。また、これらを含む総合環境 [11] もフリーソフトウェアとして手に入れることができる。

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、RLOTOS 内部で動作式がど

のようにデータとして表現されているかについて述べた。さらにこの動作式とデータによる表現の相互変換器を利用した開発支援環境について述べた。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それに沿った形での開発支援環境を構築していくことが課題となる。

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] Motoshi Saeki, Takeshi Hiroi, and Takanori Ugai. Reflective Specification : Applying A Reflective Language To Formal Specification. In *Proc. of Seventh IWSSD*, pp. 204–213. IEEE Computer Society, Dec 1993.

[9] Kazuhito Ohmaki, Koichi Takahashi, and Kokichi Futatsugi. A LOTOS simulator in OBJ. In E. Vazquez J. Quemada, J. Manas, editor, *Formal Description Techniques III*, pp. 535–538. Elsevier Science Publishers B.V. (North-Holland), 1991.

[10] 高橋薫, 神長裕明, 白鳥則郎. Lotos 言語の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, pp. 35–46, 1 1990.

[11] A. Manas. The TOPO Implementation of the LOTOS Multiway Rendezvous. Technical report, Dpt. Telematics, Technical Univarsity, 1991.