Vol. 35 No. 7 情報処理学会論文誌 July 1994

リフレクティブウィンドウ操作機構

神田 陽治†

ウィンドウ技術は，現世代ワークステーションのユーザインタフェース技術として重要である．われわれが提案するリフレクティブウィンドウ操作機構は，アプリケーションプロセスから，ウィンドウの状態を安全に操作する仕組みである．ウィンドウの制御データはウィンドウの状態を決めている情報であり，リフレクティブウィンドウ操作機構は，手続き型リフレクションを利用してウィンドウの制御データを安全に操作する．リフレクティブウィンドウ操作機構は，制御データ操作の対象とするウィンドウを探索する能力も備えている．Retrovirus と名付けたプロトタイプを，Xウィンドウシステム上に作成した．Xウィンドウシステムでは，他のXウィンドウシステムの管理するウィンドウへもアクセスできるから，Retrovirus は他のウィンドウシステム上のウィンドウも探し出して，その状態を制御できる．これは，一つのウィンドウシステムに閉じてウィンドウを扱うウィンドウマネージャにはない利点である．本論文では，Retrovirus を使って，グループウェアの各所に分散したウィンドウを集中管理した実施例を示す．

Reflective Window Manipulation Mechanism

Youji Kohda†

Windows are an indispensable device for the user interface of the current-age workstations. The reflective window manipulation mechanism we propose enables application processes to control windows' state in safe. Management data in windows determine the state of the windows, and the reflective window manipulation mechanism can manipulate the management data in safe using procedural reflection. The reflective window manipulation mechanism seeks the target windows by search, whose management data to be manipulated. We have made a prototype of the mechanism on X window system. The X window system can access windows managed by other X window systems. Therefore the Retrovirus can search windows on any other window systems, and control their state. This is an advantage over a window manager that governs windows on just a window system. We show a practical example in which windows of groupwares distributed on several window systems are controlled from the center using the Retrovirus.

1. はじめに

本論文では，リフレクティブウィンドウ操作機構を提案する．これまで，ウィンドウ操作は，ウィンドウマネージャによってユーザに解放されてきたが，アプリケーションプロセスには十分に解放されてはこなかった．リフレクティブウィンドウ操作機構は，アプリケーションプロセスにウィンドウ操作を安全に解放する試みである．

これまでも，制限された方法でなら手段があった．プログラミングの段階であれば，API (Application Program Interface) を介して，ウィンドウ操作をアプリケーションに組み込むことができた．この場合，ウィンドウをどのように操作したいかの詳細があらかじめわかっていないとプログラミングできない．特に，複数のアプリケーションに渡るような操作は，状況に応じた対応が必要なので，あらかじめ予測してプログラミングすることは難しい．別の方法として，プロセスとして活動しているときに，ヒント情報をウィンドウマネージャに必要に応じて渡すことで，ウィンドウ操作のふるまいを実行時に変更することができる．しかし，この場合にも，ヒント情報でできる範囲のことに変更は制限されてしまう．一般に，ヒント情報で変更できることは環境設定的なことに限られ，ウィンドウのふるまいを動的に変更することには向かない．

この問題に対し，われわれは次のアプローチをとる．アプリケーションプロセスとウィンドウの間に，「ウィンドウを操作する能力を持った特別なプロセス」をはさみこみ，アプリケーションプロセスには，この特別なプロセスの助けを借りてウィンドウにアクセスさせる．この枠組をリフレクティブウィンドウ操作機

† 富士通研究所情報社会科学研究所
Institute for Social Information Science, Fujitsu Laboratories

構と呼び，「ウィンドウを操作する能力を持った特別なプロセス」を Retrovirus と呼ぶ．ここに，リフレクティブウィンドウ操作機構の名は，リフレクションの考えを基にしていることを表している．Retrovirus は，アプリケーションプログラムから，API を用いてウィンドウ操作をプログラミングする手間を解放し，ウィンドウ操作を行う共通の枠組を提供する．

すでに，人間の利用者向けの視覚的操作インタフェースを持った Retrovirus を作成している[1]．本論文では，UNIX のシェルスクリプト向けの文字列操作インタフェースを持った Retrovirus を作成した．同時に，ウィンドウ選択機能を拡張し，操作の対象とするウィンドウを探索できるようにした．Retrovirus は，Xウィンドウシステム上で作成され，Xウィンドウシステムでは他のXウィンドウシステムが管理するウィンドウにもアクセスできるので，新しい Retrovirus は，他のウィンドウシステム上のウィンドウも探索できる．ウィンドウマネージャが扱うウィンドウは，一つのウィンドウシステムに閉じているから，これは大きな利点である．この利点を，例えば，グループウェアのウィンドウ管理に生かせる．

グループウェアは共同作業を支援するソフトウェアである．一つのプロセスが複数のウィンドウシステムにまたがってウィンドウを生成する実現方式もあれば，それぞれのウィンドウに責任を持つ複数のプロセスが互いに協調する実現方式もある．共同作業の内容に合わせて，単機能のグループウェアをいくつか組み合わせる場合もあるだろう．このようにグループウェア環境では，出処も活動場所もさまざまなウィンドウが，多数混在することになる．これらのウィンドウを集中して管理できれば便利である．Retrovirus を使えば，各所に散らばるウィンドウのふるまいを統一的に制御できるようになる．本論文では，Retrovirus を使って，実際にグループウェアのウィンドウを集中管理した例を示す．

なお，Retrovirus の名前は，自然界のレトロウイルスから名付けた（レトロウイルスについては，例えば，文献2)参照）．レトロウイルスの遺伝情報本体は RNA 鎖であるが，レトロウイルスは逆転写酵素を用いて，とりついた宿主の DNA 鎖に自らの RNA 鎖を組み込み，宿主の遺伝情報を変更できる．Retrovirus もまた，内部に蓄えていたウィンドウ制御データを，宿主のウィンドウに組み込む能力を持つ．両者の類似から，Retrovirus と名付けた．レトロウイルスを，遺伝子治療に役立てる研究が進められている．Retrovirus についても，ウィンドウ操作に役立つ仕組みの実現を目指している．

2. リフレクティブウィンドウ操作機構

Retrovirus は，リフレクティブウィンドウ操作機構に必要な基本的な機能を実現するものであり，リフレクティブウィンドウ操作機構の参照モデルの役割を果たす（図1）．Retrovirus の機能構成は，Retrovirus が扱う情報の仕様と，Retrovirus が備える操作の仕様で規定される．

2.1 情報仕様

外部からは二つのレジスタを持っているように見える．探索レジスタと情報レジスタである．

探索レジスタ

Retrovirus は，ウィンドウを探索する能力を備えている．探索はいつでも何回でも行うことができる．探索レジスタには，探索に先だって探索条件が書き込まれ，探索後は発見されたウィンドウの探索結果が書き込まれる．アプリケーションプロセスは，Retrovirus の探索レジスタの内容を自由に読み書きできる．これにより，アプリケーションプロセスは，望むウィンドウを必要になるごとに探し出せる．

情報レジスタ

情報レジスタには，一つ分のウィンドウ制御データを蓄えることができる．ここでウィンドウ制御データとは，ウィンドウのふるまいを決定する情報であり，通常はウィンドウシステムの管理下にある．ウィンドウ制御データのうち，どんな情報がウィンドウシステムの外部で利用可能であるかは，用いているウィンドウシステムによって決まる．Retrovirus は内部の情報レジスタに，操作対象のウィンドウのウィンドウ制

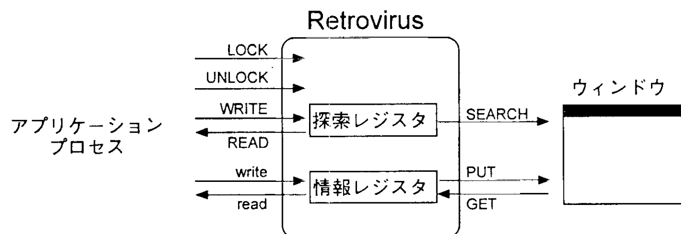

図 1 Retrovirus の機能構成
Fig. 1 The functional structure of Retrovirus.

表 1 Retrovirus の操作仕様
Table 1 The operational specification of the Retrovirus.

区分	操作	内容
第一群	SEARCH	探索レジスタの内容を探索条件としてウィンドウを探し出し，探索結果を探索レジスタに書き出す
	PUT	操作対象のウィンドウのウィンドウ制御データを不可分に読み出したのち，表現変換を施し，情報レジスタに格納する
	GET	情報レジスタのウィンドウ制御データを読み出し，表現変換を施したのち，不可分に操作対象のウィンドウに書き戻す
第二群	WRITE	探索レジスタに指定した内容を書き込む
	READ	探索レジスタの内容を読み出す
	write	情報レジスタに指定した内容を書き込む
	read	情報レジスタの内容を読み出す
第三群	LOCK	鍵をかける
	UNLOCK	鍵を外す

御データを読み出して記録したり，逆に，情報レジスタ内に記録しておいたウィンドウ制御データを，操作対象のウィンドウに書き戻すことができる．さらに，アプリケーションプロセスは，Retrovirusの情報レジスタを自由に読み書きできる．これにより，情報レジスタに保管するばかりでなく，情報レジスタ内のウィンドウ制御データに必要な修正が施せることになる．修正後，操作対象のウィンドウに書き戻すことで，ウィンドウのふるまいを好きなように制御できる．

2.2 操作仕様

表1に，Retrovirus が提供している操作をまとめる．操作は三群に分けられる．

第一群：SEARCH 操作，GET 操作，PUT 操作

第一群は，Retrovirus と操作対象のウィンドウに関わるものである．

SEARCH 操作は，探索レジスタの内容に従いウィンドウを探索し，見つかったウィンドウの探索結果を探索レジスタに書き込む．探索が失敗した場合には，その旨を探索レジスタに書き込む．見つかったウィンドウが，今後の操作の対象ウィンドウとなる．GET 操作/PUT 操作は，情報レジスタと操作対象のウィンドウの間で，ウィンドウ制御データを読み書きする．どちらの操作も，データの一貫性を保つために不可分の操作として行われる．その際，ウィンドウ制御データの表現も変更する．GET 操作のときは，ウィンドウシステムの内部表現から，アプリケーションプロセスが操作しやすい表現へ変換が行われる．PUT 操作のときは，アプリケーションプロセスが操作しやすい表現から，ウィンドウシステムの内部表現へ変換が行われる．

第二群：WRITE 操作，READ 操作，write 操作，read 操作

第二群は，Retrovirus とアプリケーションプロセスに関わるものである．

WRITE 操作/READ 操作は，アプリケーションプロセスとの間で，Retrovirus の探索レジスタを読み書きする操作である．WRITE 操作で，探索レジスタに探索情報を書き込める．READ 操作で，探索結果を読み出せる．write 操作/read 操作は，アプリケーションプロセスとの間で，Retrovirus の情報レジスタを読み書きする操作である．write 操作で，ウィンドウ制御データを情報レジスタへ設定する．read 操作で，情報レジスタのウィンドウ制御データを読み出せる．

第三群：LOCK 操作，UNLOCK 操作

第三群は，Retrovirus の同時使用防止に関わるものである．

LOCK 操作/UNLOCK 操作は，複数のアプリケーションプロセスが同時に Retrovirus を使用するのを防止する操作である．LOCK 操作で，Retrovirus に鍵を掛ける．Retrovirus に鍵がすでに掛けられている場合には，LOCK 操作をしようとしたアプリケーションプロセスは，鍵が開くまで待たされる．UNLOCK 操作で，Retrovirus に掛けた鍵を外せる．

3. リフレクティブウィンドウ操作機構の基礎

リフレクティブウィンドウ操作機構という名前が示唆するように，リフレクティブウィンドウ操作機構は，リフレクションをその基礎に置いている．リフレクションの目標は，「動的自己改変を安全に行う」基

ウィンドウアプリケーションプロセス	⇔	オブジェクトレベルシステム
アプリケーションプロセス	⇔	メタレベルシステム
GET 操作	⇔	REIFY 操作
PUT 操作	⇔	REFLECT 操作
ウィンドウ制御データ	⇔	システム制御データ
write, read 操作	⇔	編集操作
SEARCH 操作	⇔	?
WRITE, READ 操作	⇔	?
LOCK, UNLOCK 操作	⇔	?

図 2 Retrovirus と手続き型リフレクションの間の対応
Fig. 2 The correspondence between the Retrovirus and the procedual reflection.

本的な仕組みの提供にある．リフレクションは，手続き型リフレクション[6]（付録A）とリフレクティブアーキテクチャ[7]（付録B）に区分される．リフレクティブウィンドウ操作機構は，手続き型リフレクションを基礎に置いている．

3.1 手続き型リフレクションとの対応

Retrovirus の操作モデルと手続き型リフレクションの枠組の対応関係をまとめる（**図 2**）．

ウィンドウを使用するアプリケーションを，ウィンドウアプリケーションと呼ぶことにする．オブジェクトレベルシステムに相当するのは，ウィンドウアプリケーションプロセスが生成したウィンドウである．ウィンドウアプリケーションは起動されるとプロセスを生み出し，プロセスは一つ以上のウィンドウを生成して使用する．メタレベルシステムに相当するのは，Retrovirus を介してウィンドウを制御しようとするアプリケーションプロセスである．アプリケーションプロセス自身，ウィンドウアプリケーションプロセスであってもよい．この場合，自分自身のウィンドウを操作対象にすることができる．REIFY 操作に相当するのは，Retrovirus の GET 操作であり，REFLECT 操作に相当するのは，Retrovirus の PUT 操作である．REIFY 操作/REFLECT 操作は，不可分に行われるべき操作で，かつ，その実行過程でデータ表現が変更される．Retrovirus の GET 操作/PUT 操作も，不可分に行われ，かつ，その実行過程でデータ表現が変更される．REIFY 操作によって複製されるシステム制御データに相当するのが，GET 操作によって複製されるウィンドウ制御データである．複製されたウィンドウ制御データの修正は，Retrovirus の write 操作/read 操作によって達成できる．

ウィンドウシステム上では，多数のウィンドウアプリケーションプロセスが同時に活動し，その結果，多数のウィンドウが同時に存在する．すなわち，多数のオブジェクトレベルシステムが同時に存在している．Retrovirus の SEARCH 操作は，多数あるオブジェクトレベルシステムから，操作の対象とするオブジェクトレベルシステムを探索する操作といえる．探索は，WRITE 操作/READ 操作で制御できる．オブジェクトレベルシステムが多数あることは，興味深い可能性をリフレクションにもたらす．例えば，REIFY 操作と REFLECT 操作の間で，操作の対象とするオブジェクトレベルシステムを切り替えれば，あるオブジェクトレベルシステムから別のオブジェクトレベルシステムへと，安全にシステム制御データを転送できることになる．

システム制御データが実際にどんな内容を含むかは，オブジェクトレベルシステムの実行系の実現の仕方によって決まる．逐次型プログラミング言語の実行系では，接続（continuation）と環境（environment）が実行表現として使われることが多い．接続は，計算の残り部分を表し，環境は，計算の現在の状態を表す．接続を扱うことで，オブジェクトレベルシステムの計算の流れを調べたり，変更できる．環境を扱うことで，オブジェクトレベルシステムの計算の状況を調べたり，変更できる．ウィンドウは，逐次型プログラムの計算のように「自発的に動く」ものではないので，ウィンドウ制御データの場合には，接続に当たるものはない．ウィンドウ制御データは環境に相当し，ウィンドウ制御データを扱うことで，ウィンドウ操作の現在の状況を調べたり，変更できる．

3.2 リフレクティブアーキテクチャとの関係

リフレクティブアーキテクチャは，リフレクションのもう一つの実現方法である．リフレクティブアーキテクチャをウィンドウシステムに応用した例として，Silica[3]がある．Silica は，Common Lisp 処理系用のウィンドウシステムであり，Lisp 系言語が有する拡張性を利用して，リフレクティブアーキテクチャを実現している．ウィンドウを扱う操作を，出力管理，入力管理，親ウィンドウ管理，子ウィンドウ管理に切り分け，それぞれの部分を置き換えられるようになっている．置き換えにより，Silica では，ウィンドウシステムが提供する機能を，拡張あるいは再構築できる．

手続きリフレクションとリフレクティブアーキテクチャが相互補完的であるように，Retrovirus と Silica の関係は競合するものではなく，相互補完的といえる．Retrovirus は，読めて書けるウィンドウ制御

データに対して，ウィンドウ単位に手続き型リフレクションを行う．一方で，Silica を使えば，ウィンドウシステム自身を改造し，読めて書けるウィンドウ制御データの種類を増やすことができる．つまり，Silica を使えば Retrovirus をもっと有能にできるし，Retrovirus を使えば Silica ウィンドウのふるまいを個々に容易に扱えるようになる．

表 3 ウィンドウ制御データ項目例
Table 3 Examples of window management data fields.

データ区分	データ項目	内容
ウィンドウ属性	geometry:	ウィンドウの位置と大きさ
	border_width:	ウィンドウの縁の幅
	parentwindow:	親ウィンドウの識別子
	subwindows:	子ウィンドウの識別子
ウィンドウプロパティ	name:	ウィンドウ名
	icon:	アイコン名
	res_class:	アプリケーションのクラス名称
	res_name:	アプリケーションの名称
	state:	ウィンドウのアイコン化情報
	client_host:	アプリケーション起動ホスト名
	command:	アプリケーション再起動方法

4. Xウィンドウシステム上のリフレクティブウィンドウ操作機構

Retrovirus を用いた実施例を示すことで，リフレクティブウィンドウ操作機構の有効性を示したい．グループウェアのウィンドウ管理に Retrovirus を用いた例を示す．Xウィンドウシステム上に開発した Retrovirus のプロトタイプを rv と呼んで，参照モデルとしての Retrovirus と区別する．rv の情報仕様は，Xウィンドウシステム[4] の仕様を考慮して，有用と思われる項目を採用した．他に有用なものがあれば今後追加していく．

4.1 プロトタイプの情報仕様

表 2は，rv で指定できるウィンドウ探索条件をまとめたものである．Xウィンドウシステムでは，他のXウィンドウシステムが管理するウィンドウにも，ネットワーク経由でアクセスできる．この能力を使うことで，rv は他のXウィンドウシステムにあるウィンドウも探索対象にすることができる．通常は，Xウィンドウシステムが動いている計算機名とウィンドウ名の二つを指定することで，ウィンドウを指定する．ウィンドウ名が，生成元であるアプリケーションプログラムや，プログラムの中でのウィンドウの役割を表すように，きめ細かく付けられていることを前提としている．これまで，ウィンドウの名前は積極的な役割を持っていなかったが，本来は出処がわかるようにきめ細かく付けられるべきものである．rv では，ウィンドウの生成元のアプリケーションの種類やウィンドウ識別子などでも，ウィンドウを探索できる．ウィンドウが見つかったのち，ウィンドウ名を指定のものに変更することで，ウィンドウ名のきめ細かい指定も rv で行える．

表 2 探索条件例
Table 2 Examples of search keys.

探索キー	内容
host:	ホスト計算機名
id:	ウィンドウ識別子
name:	ウィンドウ名
icon:	アイコン名
res_class:	アプリケーションのクラス名称
res_name:	アプリケーションの名称

表 3は，rv が読み書きできるウィンドウ制御データをまとめたものである．rv が読み書きするのは，読めて書き戻すことに意味があるウィンドウ制御データである．読み書きできるのは，ウィンドウの属性値とウィンドウのプロパティ値である．前者には，ウィンドウの位置や大きさ，ウィンドウの親子関係といった，ウィンドウの基本的なコンフィギュレーションを決める項目がある．後者には，ウィンドウの名前やアプリケーションプログラムの再起動の方法といったウィンドウマネージャへのヒント情報のほか，ウィンドウの状態などウィンドウマネージャが設定する情報がある．アプリケーション間で合意して，有用な情報をウィンドウのプロパティ値として置くときには，読み書きできる有用な情報として追加してもよい．

4.2 プロトタイプの操作仕様

以下に述べる実施例で，rv を使うアプリケーションプロセスにあたるのは，UNIX のシェルスクリプトである．UNIX のシェルスクリプトと rv の間を結ぶ操作インタフェースとして，rvm というプログラムを作成し，UNIX シェルの特徴であるパイプ機構を生かせるように，rvm と UNIX シェルスクリプトの間の入出力は文字列（バイトストリーム）で行う．

rv は，同時に複数個起動できる．複数個の rv を区別する目的で，rv にはニックネームが付けられる．rvm は，ニックネームをコマンド引数（-n “名前”）の形で指定することで，意中の rv と交信することができる．

UNIX のシェルスクリプトから rv への操作の指示は，rvm へコマンド引数（-c “指令”）の形で与える．操作のうち，WRITE 操作と write 操作は書き込むべき内容を同時に指定する必要があるが，コマンド引数の中に直接置くこともできるし，標準入力から取ることもできる．READ 操作と read 操作のときには，読み出された内容が rvm から標準出力として返される．

典型的には，UNIX のシェルスクリプトは次のように書ける．すでに，二つの rv が起動されているものとし，それぞれのニックネームを，名前１，名前２とする．

rvm -n “名前１” -c “指令１” | “フィルタプログラム” |rvm -n “名前２” -c “指令２”

ただし，“|” は，UNIX シェルが提供するパイプ機構である．前段の標準出力を後段の標準入力へつなぐ働きを持つ．

ここで，指令１は，rv に LOCK 操作で鍵をかけたのち，WRITE 操作と SEARCH 操作で探索対象のウィンドウを確定する．次に，GET 操作と read 操作でウィンドウ制御データをウィンドウから読み出して標準出力へ流したのち，UNLOCK 命令で鍵を外す．フィルタプログラムは，標準入力から流れてきたウィンドウ制御データを調べ，思うように変更して，標準出力へ流す．指令２は，rv に LOCK 操作で鍵をかけたのち，WRITE 操作と SEARCH 操作で探索対象のウィンドウを確定する．次に，write 操作と PUT 操作で標準入力から読み出したウィンドウ制御データをウィンドウに書き出したのち，UNLOCK 命令で鍵を外す．総体として，あるウィンドウから別のウィンドウへ，ウィンドウ制御データに必要な修正を施しながら，コピーできたことになる．このほか，ウィンドウ制御データの内容を基にウィンドウを探索したり，逆に，探索結果の内容を基にウィンドウ制御データを修正したりもできる．

rvm を利用できるのは，UNIX のシェルスクリプトに限らない．プロセスを起動し，標準入出力を扱えるものなら rvm を介して rv を利用できる．例えば，Emacs エディタでは，Emacs Lisp プログラムを書いて，rvm を起動して交信するコマンドを作成できるので，Emacs エディタのコマンドからウィンドウを操作することが可能となる．

4.3 実施例：グループウェアのウィンドウの集中管理

rv と rvm を利用して，グループウェアのウィンドウを集中管理する実施例を示す．GrIPS はグループ発想作業のためのグループウェアである[5]．GrIPS は，複数のグループウェアと複数の個人向けアプリケーションの集合体として作られており，多数のウィンドウが開かれる．グループ発想作業の作業は，アイデアを膨らませる発散段階とアイデアをまとめる収束段階の二段階からなり，これを繰り返す．その際，各段階で使用するアプリケーションの組み合わせ方が違う．両段階共通に使われるアプリケーション，発散段階でのみ使われるアプリケーション，収束段階でのみ使われるアプリケーションの区別がある．発想作業の段階の移行に応じて，使われるアプリケーションのウィンドウはまとめて開き，使われないアプリケーションのウィンドウはまとめてアイコン化したい．

管理すべきウィンドウは複数のウィンドウシステムに渡り，しかも，ウィンドウを生成したプロセスも多様である．しかし，rv を用いれば，この種のウィンド

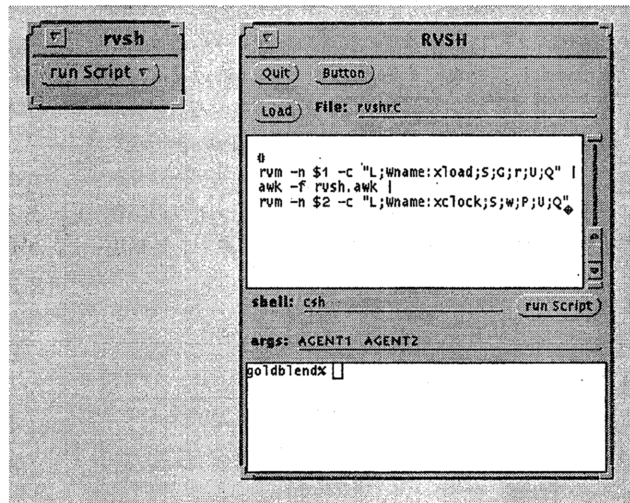

図 3 rvsh プログラムの二様態：ボタン形態（左）とパネル形態（右）
Fig. 3 The two shapes of the rvsh program: the button shape (left) and the panel shape (right).

ウ管理は容易にできる．なぜなら，rv は，ウィンドウを生成したアプリケーションや管理者であるウィンドウシステムとは無関係に，ウィンドウをウィンドウとして扱うからである．rv の能力を使えば，UNIX のシェルスクリプトからウィンドウの状態を切り替えるのはたやすい．ウィンドウが現在開いていればアイコン化を指定し，ウィンドウがアイコンになっていれば開くことを指定すればよい．GrIPS のウィンドウ管理は，以上の操作を，該当するすべてのアプリケーションにわたって行えばよい．

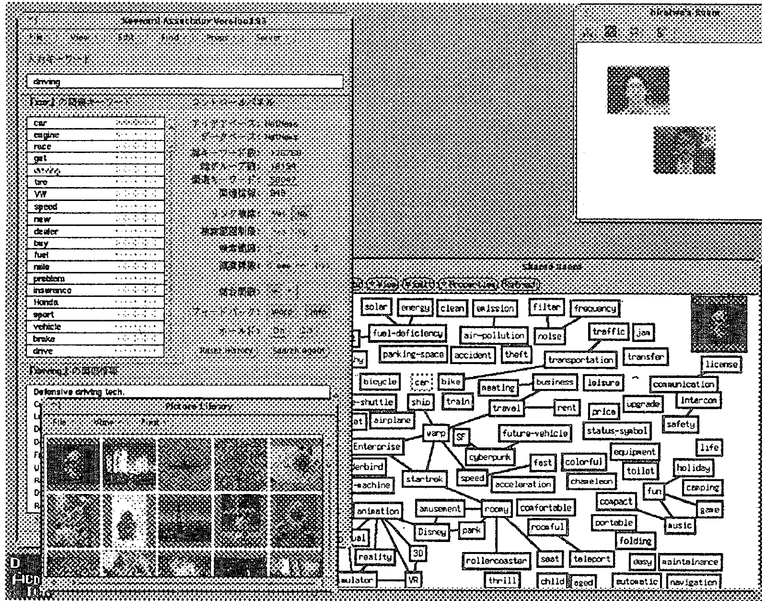

図 4 GrIPS における発散段階
Fig. 4 GrIPS: the divergence phase.

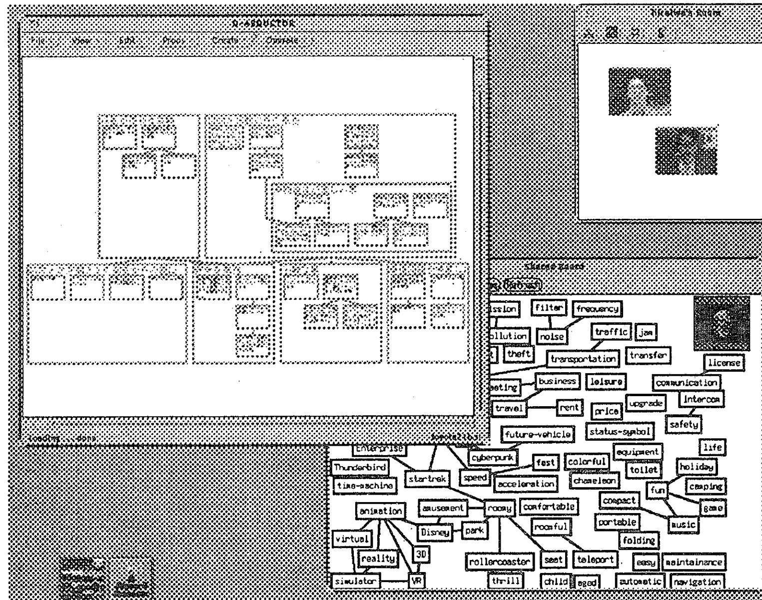

図 5 GrIPS における収束段階
Fig. 5 GrIPS: the convergence phase.

ところで，UNIX のシェルスクリプトがマウス操作一つで簡単に起動できれば便利である．そこで，rvsh と呼ぶプログラムを作成した（**図 3**）．rvsh は起動時にシェルスクリプトを読み込む．rvsh は二つの形態を持ち，二つの形態はマウス操作一つで容易に切り替えることができる．ボタン形態ではボタンを一つ持つだけとなり，マウスでボタンをクリックするだけで，読み込んでおいたシェルスクリプトを実行できる．パネル形態では，シェルスクリプトを編集できるほか，シェルやシェルスクリプトに与える引数をキーボードで変更できる．

rvsh のボタンを押すたびに，発想作業の段階が交互に切り替わる．発想作業が発散段階（**図 4**）にあるときに，rvsh のボタンを押すと，収束段階（**図 5**）に必要なツール群に切り替わる．発想作業が収束段階（図 5）にあるときに，rvsh のボタンを押すと，発散段階（図 4）に必要なツール群に切り替わる．

5. まとめ

ウィンドウをアプリケーションプロセスから安全に操作する仕組みとして，リフレクティブウィンドウ操作機構を提案した．ウィンドウの状態を安全に操作する仕組みの基礎に，手続き型リフレクションを利用している．ウィンドウの状態を変更するには，最初，ウィンドウの制御データの複製をとって，修正がウィンドウの実際の状態に直結しないように切り離しておいてから，複製されたウィンドウの制御データを変更する．最後に，元のウィンドウに書き戻して，修正の効果を実際のウィンドウの状態へ反映させる．この変更の仕組みに加えて，操作の対象とするウィンドウを，いつでも何回でも切り替えることができる．切り替えるウィンドウは，ネットワークでつながった他のウィンドウシステムのウィンドウでもよい．さまざまな探索条件を与えて，望むウィンドウを探し出せる．これらの機能を総合して用いて，グループウェアのウィンドウ管理が容易に行えることを示した．

参考文献

1) 神田陽治：Retrovirus：新しいウィンドウ操作メカニズム，情報処理学会論文誌，Vol. 34, No. 8, pp. 1780-1791 (1993).
2) イブ・K・ニコルス著，高木俊治訳：遺伝子治療とはなにか，ブルーバックス B 923，講談社 (1992).
3) Rao, R.: Implementational Reflection in Silica, *LNCS*, No. 512, Springer-Verlag, pp. 251-267 (1991).
4) Gettys, J. and Scheifler, R. W.: *Xlib—C Language X Interface*, MIT X Consortium Standard (1991).
5) 神田陽治，渡部　勇，三末和男，平岩真一，増井誠生：グループ発想支援システム：GrIPS，人工知能学会誌，Vol. 8, No. 5, pp. 65-74 (1993).
6) Smith, B. C.: Reflection and Semantics in a Procedural Language, *MIT LCS TR-272* (1982).
7) Maes, P.: Issues in Reflection, *Meta-Level Architectures and Reflections*, North Holland (1988).

付録A　手続き型リフレクション

手続き型リフレクション[6]は「動的自己改変を安全に行う」ための基本メカニズムである．自己改変システムでは，システムの動作を決定する情報（以下ではシステム制御データと呼ぶ）が，システムの動作の過程で読まれたり書き換えられたりする．手続き型リフレクションが組み込まれていない自己改変システムでは，システム制御データはシステム自身によって読み書きできる所に置かれている（**図A.1**）．動的自己改変の過程では，システムの動作に悪影響を与えないように，書き換えには細心の注意が必要とされる．

図 A.2 に，手続き型リフレクションが組み込まれた自己改変システムを図解する．自己改変システムを，改変される部分と，改変を行う部分に明確に分離する．前者をオブジェクトレベルシステム，後者をメ

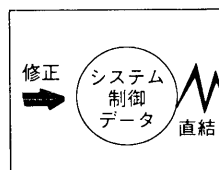

図 A.1　安全ではない動的自己改変システム
Fig. A.1　Unsafe dynamic self-modifiable system.

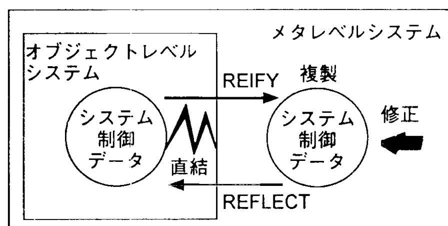

図 A.2　手続き型リフレクションによる安全な動的自己改変システム
Fig. A.2　Safe dynamic self-modifiable system with procedual reflection.

タレベルシステムと呼ぶ．システム制御データはオブジェクトレベルシステムの内部にあり，オブジェクトレベルシステムの実行を制御する．自己改変中は，システム制御データの複数が，メタレベルシステムの内部に置かれ，メタレベルシステムによって操作される．

手続き型リフレクションに基づく自己改変は，次の三段階で実行される．第一段は，REIFY 操作である．オブジェクトレベルシステムの内部のシステム制御データが，メタレベルシステムの内部に複製される．第二段は，複製されたシステム制御データの修正である．システム制御データが複製されてしまえば，もはやオブジェクトレベルシステムとは関係がなくなるので，メタレベルシステムは安全にシステム制御データを修正できる．第三段は，REFLECT 操作である．修正後のシステム制御データを，オブジェクトレベルシステムの内部に書き戻す．書き戻されたシステム制御データは，オブジェクトレベルシステムの今後のふるまいを決定する．

メタレベルシステムが複製に施した修正結果によっては，自己改変後にオブジェクトレベルシステムの実行に支障が出ることがありうる．これは，自己改変が正しく行われたかどうかの検証の問題であり，リフレクションの責任ではない．リフレクションが保証するのは，自己改変中の中間状態がオブジェクトレベルシステムには悪影響を与えない保証である．これは，修正途上のシステム制御データが，オブジェクトレベルシステムとは切り離されていることで実現されている．

手続き型リフレクションにおいて大切なポイントは以下の二点である．

リフレクティブ操作の不可分性 (indivisible)

REIFY 操作と REFLECT 操作は，それぞれ不可分な基本操作として実現されねばならない．これは各操作前後での，システム制御データの一貫性を保つために必要である．REIFY操作では不可分に複製が取られなければならない．REFLECT 操作では不可分に書き戻されねばならない．

リフレクティブ操作における暗黙のデータ表現変換

システム制御データの，オブジェクトレベルシステムに最適なデータ表現と，メタレベルシステムに最適なデータ表現は一般には異なる．REIFY 操作では，オブジェクトレベルシステムのデータ表現から，メタレベルシステムのデータ表現への変換を行う．REFLECT 操作では，メタレベルシステムのデータ表現から，オブジェクトレベルシステムのデータ表現への変換を行う．

手続き型リフレクションは，これまでプログラミング言語に組み込まれてきた．このプログラミング言語で問題を記述するときには，問題を実際に解くプログラム部分と，そのプログラム部分を監視するプログラム部分に分けて記述することになる．問題解法のプログラム部分の実行がオブジェクトレベルシステムにあたり，問題解法を監視するプログラム部分の実行がメタレベルシステムにあたる．逐次型言語に組み込んだ場合は，オブジェクトレベルシステムもメタレベルシステムも一つずつで，メタレベルシステムが活動しているときには，オブジェクトレベルシステムは停止している．並列/並行型言語に組み込んだ場合には，オブジェクトレベルシステムあるいはメタレベルシステムは，同時に複数存在しえる．とくに並列型言語の場合には，レベルを問わず複数のシステムが同時に活動できる．

付録B リフレクティブアーキテクチャ

リフレクティブアーキテクチャ[7]もまた，「動的自己改変を安全に行う」ための基本メカニズムである．リフレクティブアーキテクチャは，オブジェクトレベルシステムの実行系を安全に改変するインタフェースを提供する．実行系はオブジェクトレベルシステムの実行を司るから，実行系の再構築は，オブジェクトレベルシステムの自己改変を引き起こす．

安全に実行系の改変を許す一つの方法は，オブジェクトレベルシステムの実行を一瞬止め，実行系の一部を入れ換える仕組みを提供することである．オブジェクトレベルシステムの実行系は，システム制御データを読み込み，処理し，更新するという一連の処理の繰り返しであり，繰り返しの切れ目でなら，実行系を新しいものに安全に交換することができる．リフレクティブアーキテクチャの有効性は，実行系をどのように見せ，どの部分を置き換え可能にするかで決まる．実行系全体を置き換えるのは，もっとも汎用でどんな変更も可能だが，置き換えの戦略を示してくれないので有効とは言えない．実行系を階層的に構成したり，論理的なかたまりをオブジェクト化したりして，機能単位のまとまった置き換えを可能とすることで，有効性を向上できる．

手続き型リフレクションとリフレクティブアーキテ

クチャの関係は相互補完的と言える．オブジェクトレベルシステムの活動とは，システム制御データを実行系が処理することにほかならない．手続き型リフレクションは，システム制御データを安全に修正する仕組みを提供する．リフレクティブアーキテクチャは，実行系を安全に再構築する仕組みを提供する．この二つの違いから，手続き型リフレクションは局所的な動的自己改変に向いていると言え，リフレクティブアーキテクチャは大規模な動的自己改変に向いていると言える．

(平成4年5月22日受付)
(平成6年4月21日採録)

神田　陽治（正会員）

1981年東京大学理学部情報科学科卒業．1986年同大学工学系研究科情報工学専門課程修了．工学博士．同年より富士通(株)入社．現在，(株)富士通研究所情報社会科学研究所に勤務．並列処理，ユーザインタフェース，グループウェア等の研究に従事．電子情報通信学会，日本ソフトウェア科学会会員．