構造化オブジェクトの視覚的プログラミング

二木 誠司　松田 芳樹　藤原 ひろみ

(株) 日立製作所　中央研究所
国分寺市東恋ヶ窪 1-280

LIVE はオブジェクトの内部状態を視覚化し，ユーザがこれを直接操作した履歴からプログラムを生成する視覚的なプログラミング支援システムである．従来の LIVE では OA (Office Automation) 応用に重点を置き，電子秘書などの高レベルのオブジェクトの操作を対象としていた．本報告ではさらに低位のプログラミングにおける有効性を評価する目的で組み込んだ，配列やリスト，構造体といった構造化オブジェクトとその操作について述べる．またこれらを用いていくつかの例題を記述し，その記述性を調べた．その結果リストの操作や並列同期処理を含むメソッドを具体例に基づいて視覚的に記述できることを確認した．

Visual Programming for Structured Object

Seiji Futatsugi　Yoshiki Matsuda　Hiromi Fijihara
Central Research Laboratory, Hitachi, Ltd.
1-280, Higashi-koigakubo, Kokubunji-shi, Tokyo, 185, Japan

LIVE is a visual programming support system, which visualizes inner states of objects, and generates programs from the history of users' direct manipulations. The former implementation of LIVE focused on OA (Office Automation) applications, and had high level objects such as electric secretaries. The target of this paper is the visualization and direct manipulations of lower level objects, for the evaluation of the effectiveness of LIVE methods. We call these objects as structured objects that include arrays, lists, and structures. We describe some example programs using these objects, and evaluate the ability of description. As the result, we confirm that the system visually supports users to describe problems, such as list manipulations and parallel processing with synchronization.

1. まえがき

高度な知的作業であるプログラミングの支援のため，視覚情報を利用した視覚言語システムが注目を集め，広く研究されている．LIVE (Language for Intelligent and Visual Environment) [1][2] は下に示す特徴を持つ，プログラミングを専門としないエンドユーザを対象とした視覚言語システムである．

LIVE の特徴

(1) オブジェクト指向言語

オブジェクトは内部状態と，それを操作するメソッドから構成される．処理の対象となるインスタンス・オブジェクトはクラス・オブジェクトにメソッド "Create" を指示して作成する (今後インスタンス・オブジェクトのことを単にオブジェクトと呼ぶ)．

(2) 視覚的プログラミング

オブジェクトをディスプレイ上に具体的な内部状態の値を伴って表示する．ユーザは表示対象をマウスで選択してメソッドの実行を指示することができる (今後この操作を直接操作と呼ぶ)．

(3) 例示によるプログラミング

メソッド実行の履歴を記録し，これを新しいメソッドとすることができる．また複数の履歴を参照することにより，条件分岐を含んだメソッドとすることもできる．

(4) マルチメディア環境

3次元コンピュータ・グラフィックスや音声，ビデオといったメディアをオブジェクトの表示，直接操作に使用できる．

LIVE のユーザは，例とするオブジェクトをディスプレイ上に配置しておき，これに直接操作を加えることによって処理を進める．直接操作の結果はすぐさま表示や音声によって確認できるため，途中間違いを起こしたとしてもこれを簡単に発見できる．所望の結果が得られた場合，それまでに記録された履歴を新しいメソッドとして再利用できる．このように，LIVEにおけるプログラミングは，ディスプレイ上の表示オブジェクトとの対話作業である．

これまで LIVE では，ハノイの塔を記述した原理確認システム[1]と，電子秘書オブジェクトが単純な繰り返し作業を代行する OA (Office Automation) システム[2]を実現してきた．これらのシステムで個々の問題は記述できたが，新しい問題を記述する度に，組み込みオブジェクトを用意しなければならないという問題があった．

そのため我々はより低レベルでのプログラミングにおける LIVE の有効性を確認するための，組み込みオブジェクト (今後構造化オブジェクトと呼ぶ) の設計を行ないこれを実現した．2章では構造化オブジェクトを視覚化し，直接操作するための仕様について述べる．

また別の課題として，電子秘書に複数の仕事を同時に頼みたい場合などの，並列プログラム記述の必要性もあった．そこでLIVE では GHC[3] に代表される並列論理型言語のセマンティクスを実現している．3章では GHC で例題とされる問題を LIVE で構造化オブジェクトを用いて記述することによって，その記述能力を確認する．

2. 構造化オブジェクトの仕様

2.1 論理的仕様

一般的な構造を記述する構造化オブジェクトは，内部状態として任意のオブジェクトを持つものとする．また記述する問題によって適した内部状態の形式を用意する．数値計算の記述のため整数インデックスでアクセスできる配列を，ストリームの記述のため先頭 (Car) 要素と残りの (Cdr) 要素でアクセスできるリストを用意した．またより一般的な問題の記述のため，名前でアクセスできる構造体も用意した．

内部状態オブジェクトは，構造化オブジェクトへの "UnifyState" メソッドの指示によってアクセスする．"UnifyState" の引数として内部状態名と既定義のオブジェクトを指定する．構造の要素が未束縛の(論理)変数オブジェクトで，これに値をもった引数オブジェクトを指定して "UnifyState" を実行した場合には構造への書き込みとなり，その反対の場合には読みだしとなる．

2.2 表示方式

LIVE では作成されたオブジェクトはその視覚表現を持つ．図1 に基本的なオブジェクトの表示例を示す．視覚表現は左側のオブジェクトの名前を表示する部分と，右側の状態を表示する部分に分れる．ここに示すように整数と浮動小数とは小数点の有無によって，また文字列オブジェクトは両肩のマークによって他のオブジェクトと区別される．変数オブジェクトは，未束縛の場合は図に示す特殊な表記で

あり，束縛されれば束縛先のオブジェクトの視覚表現と同じに変更される．内部状態の値の表示部はOSF / Motif のボタン・ウィジットを用いて視覚化する．

図2 に，構造化オブジェクトの表示例を示す．基本的なオブジェクトの場合と同じように，オブジェクトの名前は左(上)に，状態は右(下)に表示している．構造化オブジェクトでは各要素オブジェクトの状態を知ることが重要であり，各要素の状態を表示することが必要となる．ただし常に要素表示することは，限られたディスプレイ領域を無駄にすることにもなり，好ましくない．そこで図に示すように，簡略表現と，(要素オブジェクトの視覚表現を内部に持つ)詳細表現の両方を持つようにし，ユーザの指示によってこれを切り替えられるようにする．切り替えは，各オブジェクトへメソッド "Expand" または "Iconify" を指示することによって行なわれる．

詳細表現時には OSF/Motif のスクロール・ウィジットを用い，内部に要素オブジェクトの視覚表現を配置する．スクロール・ウィンドウの大きさは右下のサイズ変更ボタンの操作によって変更する．また図に示すように表示の属性を変更するためのメニューも用意する．"Show Name" 項目を選択することで，内部状態の名前が表示される．"Fit Object" 項目を選択することで，スクロール・ウィンドウが要素を表示するのに適切なサイズに変更される．"Strict View" はリスト・オブジェクトにだけ用意されており，その機能については後述する．視覚表現の切り替えは，各ウィジットを動的に破壊/生成することによって行なう．

リスト・オブジェクトには特別に複数の詳細表現を持たせて，その構造を理解しやすくする．これを図3 に示した整数 3, 4, 5 を並べたリスト "lst" の表示例を用いて示す．図の上部に示したように，このリストの構造はセルをポインタでつないだものであり．リストの最後の要素は空のオブジェクト "Nil" である．したがって意味的には (b) に示す表示がリストの構造を正確に表現している．しかし多くの場合リスト構造は線形であるため，(a) に示すように各セルの先頭要素だけをたどる表現をサポートする．双方の表示の切り替えは表示属性メニューの "Strict View" を選択することによって行なう．また (c) に示すように双方の表現を組み合わせることもユーザの指示によって可能である．

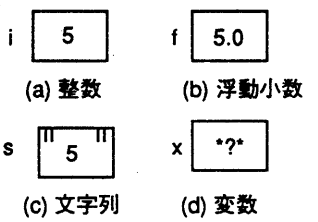


図1. 基本的オブジェクトの表示例

2.3 直接操作方式

オブジェクトへのメソッド実行指示は下に示す文法規則によって行われる．

LIVE 文法規則

<メソッド実行指示> ::=
  <Receiver> {<Argument>}* <Method> <Mode>
<Receiver> ::= <オブジェクト名>
<Argument> ::=
  <オブジェクト名> または <リテラル>
<Mode> ::= '.' または '?'

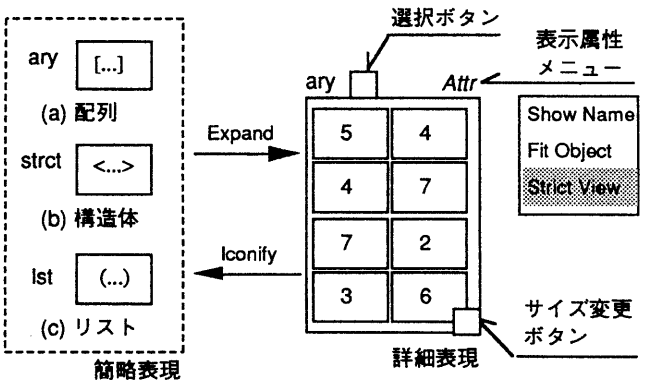
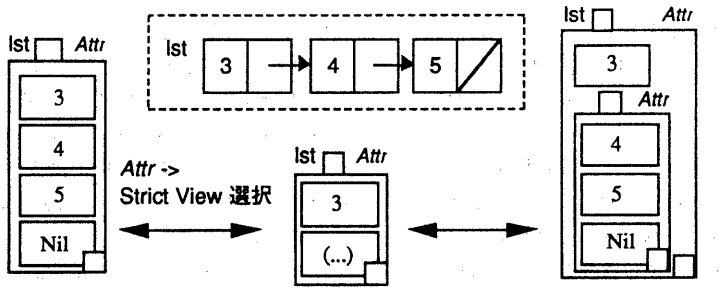


図2. 構造化オブジェクトの表示例



(a) 先頭要素をたどって表示　(b) 単一セルを表示　(c) 組み合わせ表示

図3. リストオブジェクトの表示例

ここで<Receiver> は指示を受けるオブジェクトであり，<Argument> は引数オブジェクト，<Method> はメソッドである．直接操作の際には関係ないが，履歴化されメソッドとなったとき対応するメソッドは，<Mode>が '.' の場合には並列に，また'?" の場合には逐次に実行される．GHC との対応で示せば，'.' 指定がボディ記述部にあたり，'?' 指定がガード記述部にあたる．

基本的にこれらの指示は，LIVE 実行環境で準備するコマンド入力領域にユーザがキーボード入力する．しかし次に示す入力はマウス操作によっても行うことができる．これによってより直感的な直接操作感覚を実現できる．

(a) オブジェクト名

オブジェクトの名前は，ボタン・ウィジットによって状態を表示している部分をマウスで左クリックすることによって入力できる．構造化オブジェクトの詳細表現時には，図2に示す選択ボタンを左クリックする．

構造化オブジェクトの要素となっているオブジェクトの状態表示部をダブル・クリックした場合は，その要素へのアクセスメソッドの実行を指示する．構造が複数のオブジェクトの入れ子になっている場合には複数のアクセスメソッドを実行する．

(b) メソッド名

オブジェクトの状態を表示している部分でマウスの右ボタンを押し続けることによって，メソッド一覧がメニュー表示される．メニューにはそのオブジェクトで実行可能なメソッドだけが表示される．これを選択することによってメソッド名を入力することができる．

3. メソッド記述例

本節では先の仕様を実現した構造化オブジェクトを使用したメソッドの作成例および実行例を示す．また作成したプログラムを GHCのものと比較し，その記述性について考察する．

3.1 リストの接続

図4 に2つのリストを接続する "Append" メソッドの作成例を示す．このメソッドの仕様は，指示オブジェクトと第1引数オブジェクトを接続して，第2引数の変数オブジェクトに束縛することとする．

(a) は作成が開始された状態である．左のパネル上に配置されたリスト "a" と "b"，変数オブジェクト "x" に対してメソッドの実行を指示する．これに応じてシステムはこのメソッドが新しいメソッドであることを判断して，右側の記述用のパネルを作成する．記述用のパネルの左上段には指示オブジェクトと引数オブジェクトが表示される．また左中段には指示オブジェクトの内部状態(Car : 先頭要素と，Cdr : 残りの要素．Size 状態は現在のところ常に2である) が表示される．内部状態のオブジェクトには直接操作を許していないため，通常のオブジェクトの表示とは異なる．その他のパネル内の構成要素については図中に示す．

(b) は記述に必要な一時的オブジェクトを作成した場面である．指示オブジェクトの先頭要素 (Car状態) と残りの要素 (Cdr 状態) をそれぞれ変数オブジェクトに束縛してアクセスする．この操作は，内部状態の表現をマウスでダブルクリックするだけで実行できる．また変数 "rest" をあらかじめ作成している．

(c) は結果を出力する変数オブジェクトを，先頭が指示オブジェクトの"Car" 状態，残りが "rest" オブ

ジェクトであるリストオブジェクトに束縛する指示を行った場面である．束縛の結果は即座に確認できる．

(d) は "a" の残りの要素に対して，第1引数と，"x" の残りの要素を表す"rest" を指定して同じメソッドを再帰的に呼び出した場面である．処理は "a" からたどれる最後の要素まで進み，そこで残りの要素が空のオブジェクト "Nil" となる．

"Nil" オブジェクトにとって，"Append" メソッドは新規メソッドである．LIVE ではオブジェクトに定義されていないメソッドの実行を指示されるか，メソッドの実行に失敗した時点で，そのメソッドを記述するためのパネルが作成される．(e) は "Nil" オブジェクトの "Append" メソッドを記述するためにシステムが作成したパネルである．

(f) はユーザが要求に応えて記述した場面である．リストの前半はすでに作成されているので，後半部を引数リストオブジェクトに束縛すればよいことが簡単に理解できる．メソッドの記述を終了するには，右下に表示された "Success" ボタンをマウスでクリックする．

(g) のパネル内で目的の仕様を満たしたことを確認し，"Success" ボタンをクリックする．これで "Append" メソッドの記述が終了した．(h) では作成されたメソッドを別のオブジェクトに対して指示し，正しく動作することを確認している．

下にこれまでの操作によって作成された "Append" メソッドのソースコードを示す．このうち下線を引いた部分がキーボードからの入力であり，残りはマウスによる操作で入力した．

3.2 クイック・ソート

図5 にクイックソート・アルゴリズムを実現したメソッドの実行例を示す．ここでは値が定まった配列要素を入力とし，未束縛の変数オブジェクトを要素とする配列オブジェクトを出力として用いている．

クイックソート・アルゴリズムでは，全体の問題が部分に分割されて複数のメソッドが並行して実行される．そのため，処理が並行して行なわれ，値が定まった部分から表示に現われていることに注意されたい．ここで示すように，LIVE ではアルゴリズムの実行をアニメーションによって観察することもできる．

指示オブジェクトがリストの場合	Var "Car" Create? a "Car" Car UnifyState? Var "Cdr" Create? a "Cdr" Cdr UnifyState? Var "rest" Create? rest Car x Cons. Cdr b rest Self. SuccessEnd.
指示オブジェクトが空の場合	rest b =. SuccessEnd.

作成された "Append" ソースコード

3.3 素数の生成

図6 に素数を生成する処理の実行例を示す．ここではストリームを使用した並列同期処理を実現している．

メソッド "Gen" は指示オブジェクト以上，引数オブジェクト以下の整数リストを生成する．またメソッド "Shift" は順に並んだ整数リストのうち，素数だけを選択してリスト化する．2つのメソッドの間では，整数リストをストリームとして共有している．"Gen" がストリームのはじめから値を具体化していき，"Shift" はこれを使用して素数リストを生成している．図に示すように，ストリームの状態を具体的に観察できる．

3.4 GHC との比較

ここで示した例によって，LIVE が簡単なリスト技法と，問題の分割による並列計算，ストリーム並列計算を記述できることを示した．この他にも，差分リストを用いたクイックソートの記述や，ストリームのマージ，構造体を用いたデータベースなどを記述した．これらによって LIVE が広範囲の問題を記述できることが確認できた．

下に記述の簡便さを比較するため，同じアルゴリズムを用いて GHC でリストの接続プログラムを記述した例を示す．

GHC によるリスト接続プログラム

```
append ( [ A ¦ X ], Y, Z ) :- true ¦
        Z = [ A ¦ Z1 ], append ( X, Y, Z1 ).
append ( [], Y, Z ) :- true ¦ Z = Y.
```

先に示した LIVE のソースコードと比較すると GHC の方がコード量は少ないが，実際にキーボードから入力する量は LIVE の方が少ない．またGHC ではリスト構造を特別な記法でアクセスしているが，LIVE では要素のダブル・クリックによってアクセ

ス・メソッドを指示可能である．またGHCでは要素を持つリストと空リストの場合をはじめから記述しなければならない．LIVEでは要素を持ったリストについて記述をはじめ，のちに空リストの記述が必要になった時点でこれを記述している．

4. おわりに

ここで組み込んだ構造化オブジェクトと，電子秘書オブジェクトやメールシステム等を組み合わせて，スケジューリングやメールの分類といった実際的なメソッドも作成した．例えば，電子秘書がメール配達人・オブジェクトとストリームを共有し，メール配達人がストリームの先頭にメールを束縛する度に，電子秘書がこれを解釈するメソッドを実行するというものである．電子秘書はストリームに未処理のメールがない場合には別のメソッドを実行可能である．LIVEに少し慣れたユーザなら，こういった並列プログラムを簡単に記述することができる．

並列論理型言語に関する視覚化システムとしては，Pictorial Janus[4] やプロセス指向デバッガ[5] などが発表されている．これらは複雑な制御構造やソースコードの構造を視覚化することで，プログラミングを簡単化しようというものである．ただし簡単化されてはいても，記述する内容はテキストによるものと同等であり，ユーザには抽象的な思考が要求されている．

一方 LIVE ではオブジェクトの状態を視覚化し，これをユーザに直接操作させている．ユーザは具体例に対して作業を進めるだけでメソッドが生成される．この点が，LIVEの最も大きな特徴であると考えている．

参考文献

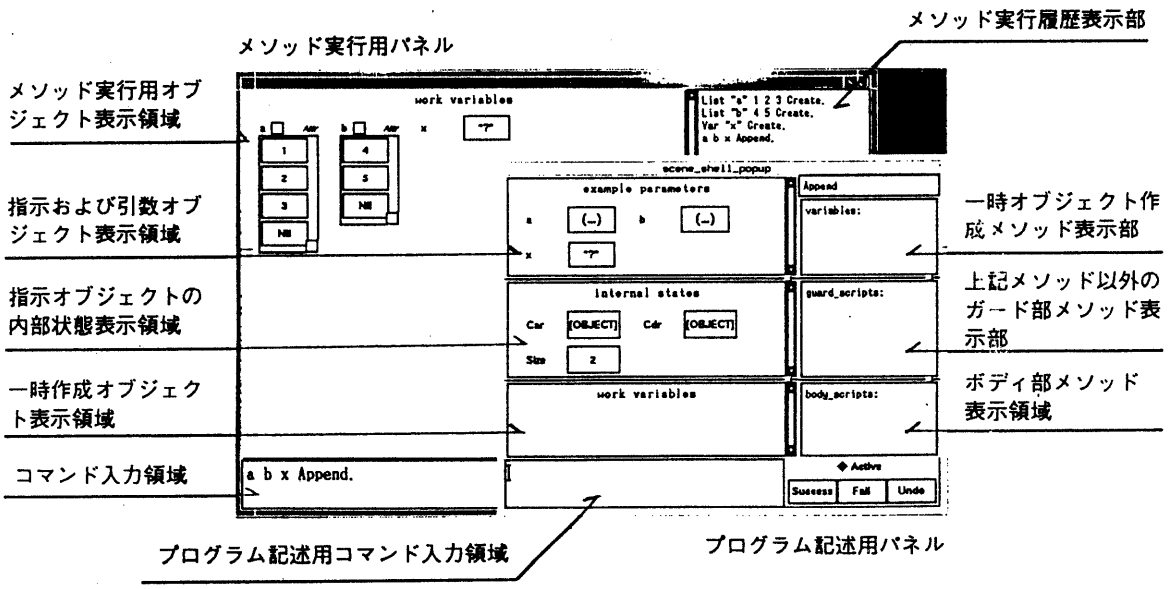
[1] Kojima, K., Matsuda, Y., Futatsugi, S. : LIVE - Integrating Visual and Textual Programming Paradigms -, Proceedings of IEEE Workshop on Visual Language, pp.80-85, October, 1989

[2] 二木, 藤原, 松田 : 視覚的OAプログラミング環境 : ソフトウェア科学会ソフトウェア研究会予稿集, 1991

[3] 淵 一博 監修, 古川 康一・溝口 文雄 共著 : 並列論理型言語 GHC とその応用 : 共立出版株式会社, 1987

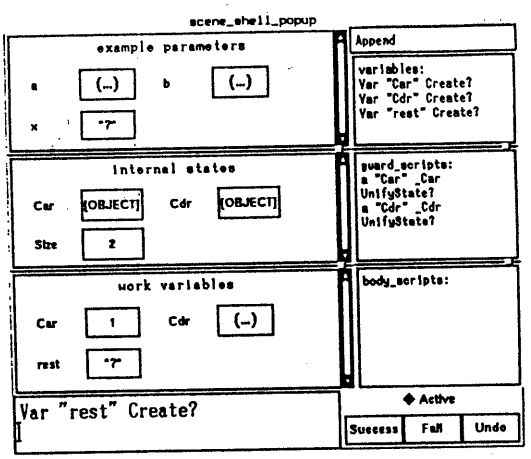
[4] Kahn, K.M. : Concurrent Constraint Programs to Parse and Animate Pictures of Concurrent Constraint Programs, Proceedings of FGCS'92, pp.943-950, ICOT, July, 1992.

[5] Maeda, M. : Implementing a Process Oriented Debugger with Reflection and Program Transformation, Proceedings of FGCS'92, pp.961-968, ICOT, July, 1992.

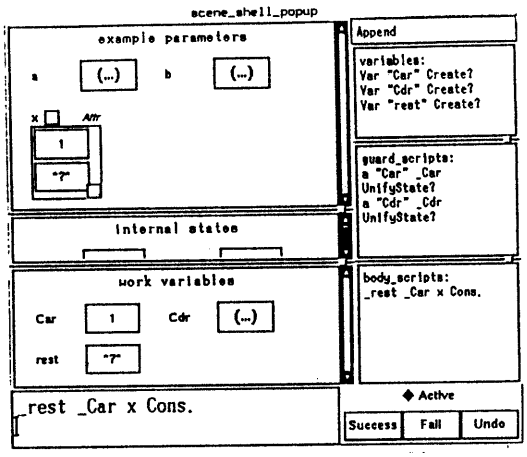


(a) リストオブジェクト"a" と"b"、変数オブジェクト"x" を例として "Append" の実行を指示.

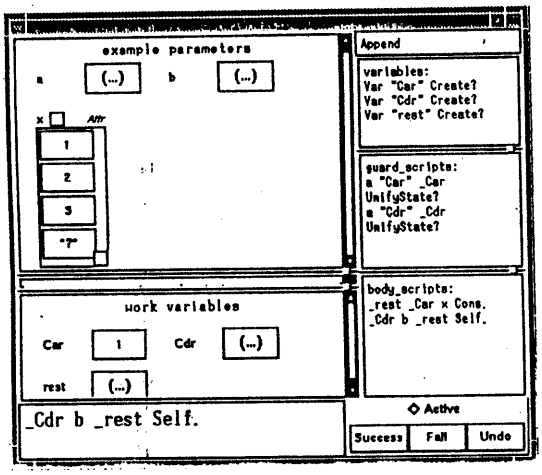
図4. リスト接続メソッドの作成例



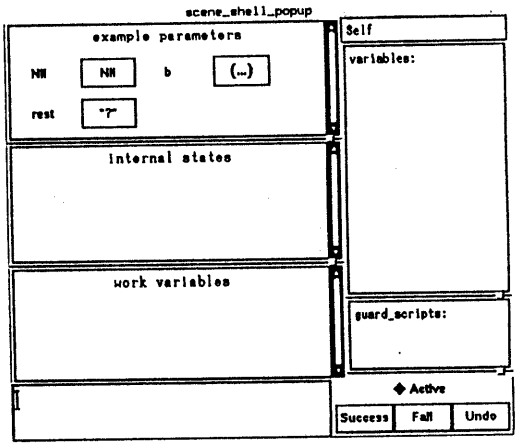
(b) リストの"Car", "Cdr" 内部状態を一時作成した変数オブジェクトに束縛．変数 "rest" の作成．



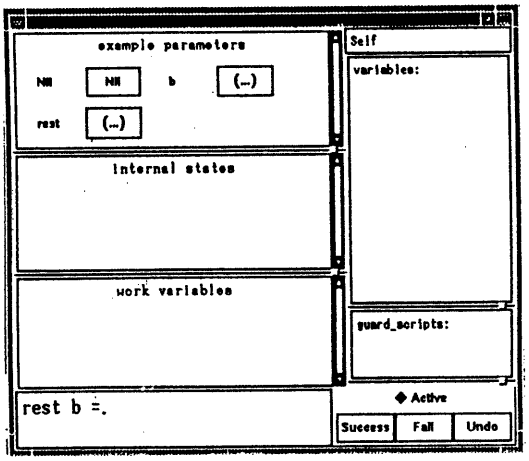
(c) 出力オブジェクト "x" の先頭要素を，指示オブジェクトの先頭要素とする．



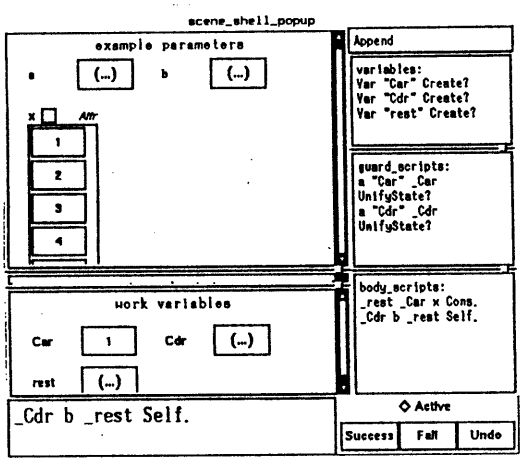
(d) 残りの要素に対して再帰的実行を指示．処理は "a" からたどれる最後の要素まで進む．



(e) システムが空オブジェクト "Nil" の "Append" メソッド記述パネルを作成する．

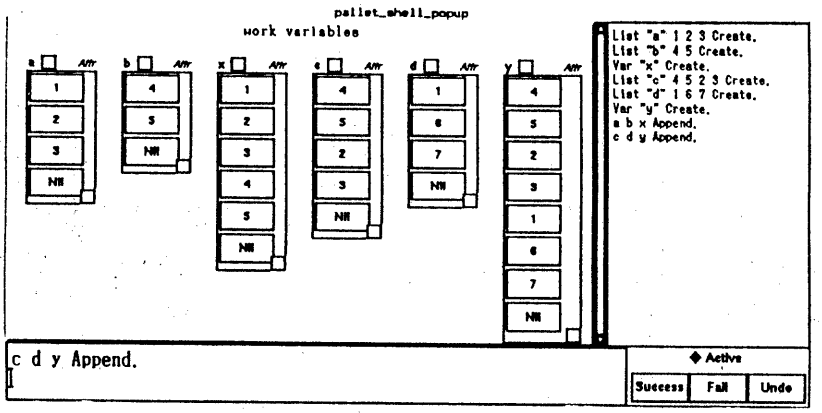


(f) 出力オブジェクトを第1引数に束縛する．



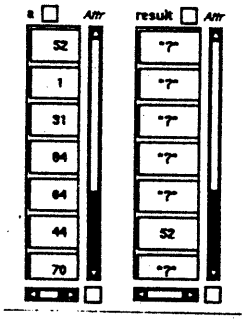
(g) (f) によって出力オブジェクトの仕様が満たされた．

図4. リスト接続メソッドの作成例 (つづき)

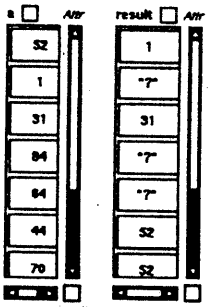


(h) 作成された "Append" プログラムの再実行

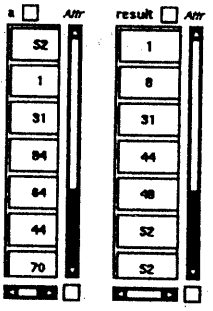
図4. リスト接続プログラムの作成例 (つづき)



(a) "a" を並べ替えた結果を "result" に束縛する

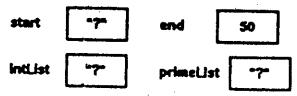


(b) 実行のスナップショット



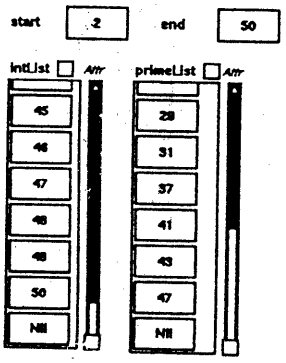
(b) 実行結果

図5. クイックソートの実行

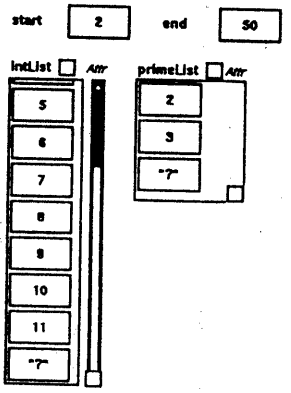


```
start end intList Gen.
intList primeList Shift.
```

(a) "Gen" と"Shift" メソッドの実行指示

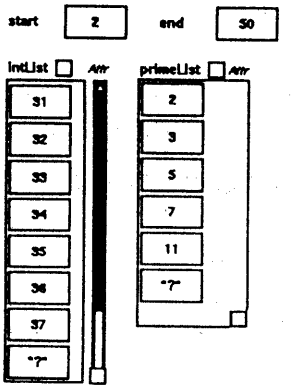


(d) 実行結果



```
start 2 =.
```

(b) "start" に整数 2 を束縛．休眠していた"Gen" と "Shift" メソッドが実行を開始



```
start 2 =.
```

(c) 実行のスナップショット

図6. 素数生成処理の実行