グラフィクスとCAD　110−15
（２００３．２．１４）

図形エディタの操作履歴と図形依存関係を図式的に扱うマクロシステム

笛木 規雄　中山 健　西田 簿　小林 良岳　前川 守

地図などのように多数のオブジェクトを図形エディタで扱う場合、繰り返し操作が多いためマクロ化が有用である。しかし、線形構造である文字列を対象とするテキストエディタのマクロに比べ、明確な構造のない図形を対象とし、任意の部分を直接操作できる図形エディタではマクロの定義や修正が難しい。そこで操作履歴を、その操作に関係した図形オブジェクト間の依存関係とともに時間軸上に図示し、グラフィカルにマクロ定義や選択的 Undo が行なえるシステムを作成した。マクロに含めるべき操作とオブジェクトの把握や、マクロ化に際しての入力値の変数化・定数化などの指定が履歴図上で容易にできる。

A Visual Environment for Scripting Drawing Task with Example

Norio Fueki, Ken Nakayama, Susuki Nishida, Yoshitake Kobayashi, Mamoru Maekawa

Hebogram is a new visual scripting language and environment for scripting tasks for a drawing editor. *Hebogram* system watches operations in a drawing editor, diagrammatically presents the history, and allows the user script in a programing-*w*ith-example manner on the diagram. The history diagram depicts dependencies among argument objects, as well as sequence of operations. To specify a script, the user (1) selects ordered set of operations, then (2) assign one of three types of the parameterization/constant type for each object involved in the selected operations.

1　はじめに

コンピュータで行なう操作の多くは反復を伴う。この反復の手間を軽減するために作業を自動化するスクリプトが用いられる。このスクリプトはマクロとも呼ばれ、テキストエディタや表計算ソフトなど多くのアプリケーションで利用できる。

スクリプトの作成には、スクリプトをプログラムとして書く方法と、ユーザが行なった操作を記録する方法がある。前者は、自由度が高く様々なスクリプトを定義できる反面プログラミング能力が必要であり、非プログラマである一般ユーザには向かない。後者は、操作をそのまま記録するため操作さえできれば非プログラマでも手軽に利用できる。ただし単なるスクリプト記録では、内容を変更できないため定型的な操作しかできない。そこで一般ユーザが直接スクリプトを書くことなく操作履歴をもとにしてスクリプト定義、さらに定義スクリプトを組み合わせて機能拡張を行なえる *Hebogram* システムとその操作体系を構築する。

電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications. 1-5-1 Chofugaoka, Chofu-shi, Tokyo 182, Japan

Hebogram システムは、スクリプト定義したい対象システムと組み合わせるサブシステムとして動作する (図 1)。スクリプト定義は、直接プログラミングせず、対象システムの操作履歴をもとに行なう。操作履歴は、関数および関数の入出力の依存関係を伴う履歴図として *Hebogram* システムに描く。スクリプトは、履歴図上でスクリプト定義する部分を直接操作で選択指定する。スクリプトに含める関数の引数は、変数化や定数、参照指定できる。作成したスクリプトは、対象システムに新機能として組み込み利用できる。またスクリプトを編集したり、さらに組み合わせて新しいスクリプトにしたりできる。

2　操作例

オブジェクト e_1 と e_2 のオブジェクト間の距離を求める操作を考える。このとき e_1 と e_2 を定数指定したスクリプトを定義すると e_1 と e_2 の位置の値を含むスクリプトを定義する。このスクリプトは図 2(a) の`mfix( )`のように引数をとらず、スクリプト定義時の e_1 と e_2 の位置から距離を計算する。e_1 を定数、e_2 を参照指定すると e_1 の位置の値と e_2 の名前を含むスクリプトを定義する。このスクリプトは図 2(a) の`mref( )`のように引数を取らないが、

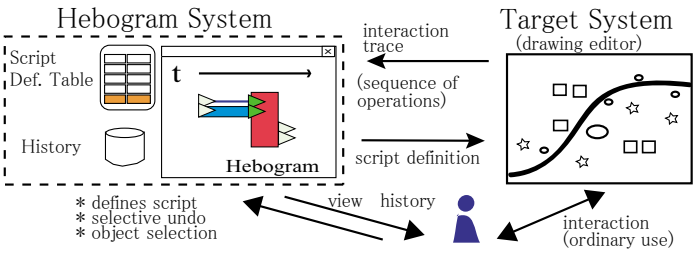
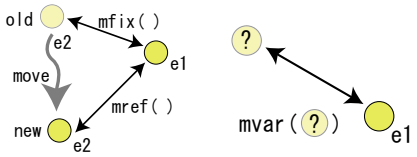


図 1: システムの全体構成

表 1: 操作と履歴

処理	操作	履歴の内部表現
h_1	駅 e_1 の中心点オブジェクト e_2 作成	GetCPoint(<e_1,e_{1Data}>) → (<e_2,e_{2Data}>)
h_2	e_2 から半径 1km の円 e_3 作成	MakeCircle(<e_2,e_{2Data}>,1000) → (<e_3,e_{3Data}>)
h_3	あるコンビニ e_4 の中心点 オブジェクト e_5 作成	GetCPoint(<e_4,e_{4Data}>) → (<e_5,e_{5Data}>)
h_4	e_5 が領域 e_3 内にあるか判定	Inside(<e_3,e_{3Data}>,<e_5,e_{5Data}>) → (<e_6,e_{6Data}>)
h_5	真偽によって返り値を設定	If(<e_6,e_{6Data}>,<e_5,e_{5Data}>,'( )) → (<e_7,e_{7Data}>)
h_6	e_4 から e_2 までの距離を計算 Mac1 を定義	CalcLength(<e_7,e_{7Data}>,<e_2,e_{2Data}>) → (<e_8,e_{8Data}>)
h_7	Mac1 をコンビニ集合 e_9 に適用	Map(Mac1,<e_9,[···]>) → (<e_{10},[···]>)

e_2 が移動した場合、スクリプト適用時の e_2 の位置から距離計算を行なう。e_1 を定数、e_2 を変数指定したスクリプトを定義すると e_1 の位置の値と e_2 を示す仮引数を含むスクリプトを定義する。このスクリプトは図 2(b) のmvar(?) のように変数指定したオブジェクトを引数に取り、引数として与えたオブジェクトと e_1 の位置の距離を計算をする。スクリプ



(a) 定数・参照例　　(b) 変数例

図 2: *Hebogram* 引数設定例

トは、まず最初に 1 つのオブジェクトに対して 1 度操作を行ない、その操作をスクリプトとして定義する。複数のオブジェクトに操作を行なうときは、定義スクリプトを複数のオブジェクトに適用する。例として、駅 e_1 と地図内の全コンビニ集合 $e_9 = [\cdots]$ を与えて e_1 から 1km 以内のものだけ選択状態にする操作を考える。まずコンビニ e_4 を変数入力し、e_4 が駅 e_1 の中心点 e_2 から 1km 以内にあるとき選択するスクリプトMac1 を定義し、次にMac1 を e_9 の要素すべてに適用する (表 1)。GetCPoint は、与えられたオブジェクト e_1 の中心位置に新らたに点オブジェクト e_2 を作成する。MakeCircle は、点と半径を入力し円領域を作成する。Inside は、領域内にオブジェクトがあるかどうか判定し、論理値を返す。If は 3 引数を取り、1 つ目に論理値を指定し論理値が真のとき 2 つ目の引数を、偽のときに 3 つ目の引数を関数の出力にする。'( ) は返り値なしを示し、この値が返り値になったときは、それ以降の処理は行なわない。CalcLength は、2 点を引数に取りその間の距離を計算する関数である。

Hebogram システムではオブジェクトを名前 e_x とそのデータ e_{xData} の組で表現する。ここで e_4 は任意のコンビニ,e_6 は論理値をとる。このとき図 3 のように *Hebogram* と呼ばれる履歴図が描かれる。*Hebogram* は操作の順に右に伸びる。この

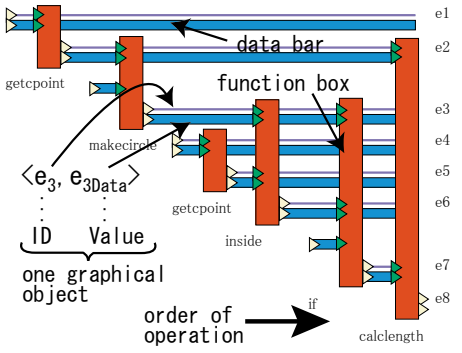


図 3: 履歴画面

履歴図をもとに、駅から半径 1km 以内のコンビニを選択する処理を再現するスクリプトを定義するときは *Hebogram* を図 4 のように選択する。この時の内部表現は表 2 のようになる。 ここで

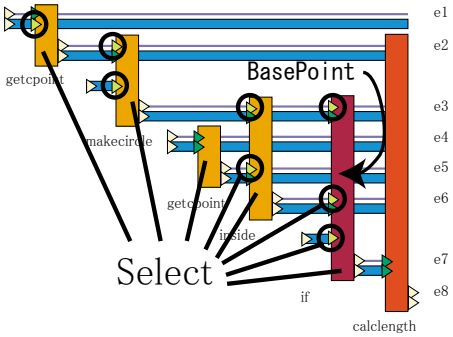
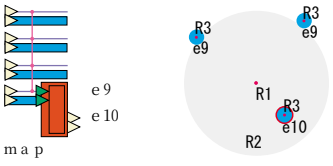


図 4: *Hebogram* 選択例 1

表 2: 選択時の内部表現

処理	選択指定	内部表現
h_1	<e_1,e_{1Data}>:定数化指定	<c_1,e_{1data}>
	<e_2, e_{2Data}>:参照化指定	R_1
h_2	1000:定数化指定	<c_2, 1000>
	<e_3, e_{3Data}>:参照化指定	R_2
h_3	<e_4,e_{4Data}>:変数化指定	<X_1,X_{1Data}>
	<e_5,e_{5data}>:参照化指定	R_3
h_4	<e_6,e_{6data}>:参照化指定	R_4
h_5	'( ):定数化指定	<c_3,'( )>

c_1,c_2,c_3,R_1,R_2,R_3,R_4,X_1,X_{1Data} はスクリプト定義のときに *Hebogram* システムがつけた一意な ID で定数や参照、変数形式を示す。この選択で定義できるスクリプトは、コンビニを変数入力して、定数化した駅から半径 1km 以内にあるコンビニだけ選択する。スクリプトに`Mac1` という名前をつけ定義すると、対象システムに定義スクリプトの名前と引数の型と数が送られ、対象システムで定義スクリプトが使えるようになる。スクリプト`Mac1` を全てのコンビニ e_9 に適用し、条件にマッチするコンビニ e_{10} だけを選択するには、関数とリストを引数に取る`Map` を使う。e_9 と e_{10} はリストデータであり、e_{9Data} は、リストの要素となるコンビニの ID とデータの組を記録する。`Map` は、リストの各要素それぞれに引数として与えた関数を適用する。このときの履歴図は図 5(a) となり、対象システムでは図 5(b) のような画面が表示される。この例では 3 つのコンビニからなるリストを入力し、その結果 1 つのコンビニだけが条件にマッチし選択されている。以上のように関数の引数を設定したスクリプトを定義し、再利用する。このようなスクリプトを繰り返し定義することで一般ユーザでも機能拡張や任意のスクリプトを定義できる。



(a) *Hebogram*　(b) 適用画面

図 5: スクリプト適用時の *Hebogram*

3 *Hebogram* システム

Hebogram システムは、対象システムのサブシステムとして動作する。*Hebogram* システムは対象システムの操作履歴から履歴図を描くウインド、定義スクリプトを保存するスクリプト定義表、履歴保存部分からなる (図 1)。適用できるシステムは、操作ごとに履歴を *Hebogram* システムに送る機構と *Hebogram* システムからの関数実行を受け付ける機構を持ち、関数の入出力が明らかでなければならない。

3.1 *Hebogram* システムで扱うデータ

Hebogram システム固有のデータ型は、Int 型、論理型、オブジェクト型、ペア型、Int 型のリスト型、論理型のリスト型、オブジェクト型のリスト型である。リストの要素は、すべて同じ型だが、ペアは型に関係なく、２つのデータを対にしたものである。オブジェクト型やオブジェクト型のリスト型は、対象システム固有のデータを *Hebogram* システムで取り扱うための型であり、この型のデータの処理自体は全て対象システムが行なう。

Hebogram システムには、算術演算、論理演算、集合演算、条件、Map、リスト操作などの関数がある。Map は、関数とリストを引数に取り、リストの要素それぞれを関数に適用し、適用結果をリストとして返す。リスト操作は、リストの生成、リストからの要素の取り出し、リストの要素の並べ替え等である。

3.2 履歴

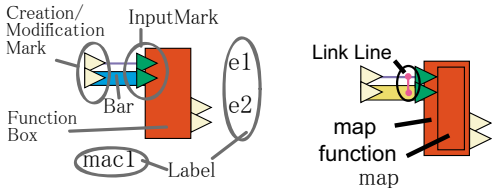
履歴は、対象システム上の関数適用を入出力オブジェクトと共に通し番号 $h_1, h_2, \cdots$ をつけて順に記録したものである。マウスの動きやボタンの押下など低レベルな操作は記録しない。

履歴の内部表現　この操作履歴は`Function(Inputs)`→`(Outputs)` という形式をとる。ここで、`Function` は関数名、`Inputs` は入力、`Outputs` は出力の列である。`Inputs`,`Outputs` には、オブジェクトのとき ID とデータ本体である Data のペア $(ID, Data)$ を、定数のときは Data だけを記録する。履歴にデータだけを保存すると、同じ場所に同じデータを持つ円が２つあるときに ID なしでは判別できない。逆に ID だけを保

存すると、時々刻々データが変わるオブジェクトは、ID だけではどのデータを指すのか分からない。例えばオブジェクト e_1 の時刻 t_1 におけるデータ $e_{1Data}^{(1)}$ と時刻 t_2 におけるデータ $e_{1Data}^{(2)}$ があるとき ID の e_1 だけではどちらのデータを指すのか分からない。そこでオブジェクトを関数の入力にするときは、オブジェクトの ID とデータをペアにして保存する。

Hebogram 履歴図は、対象システムの操作履歴を依存関係をもとに描いた図である。オブジェクト数が増えると履歴図は縦に伸びる。図上では関数を関数ボックスと呼ばれる縦長の長方形で、オブジェクトの ID や状態などの値をデータバーと呼ばれる横線で表す。オブジェクトは細線と太線で、関数の定数引数は太線で表す。関数ボックスの左側に関数の入力を描き、右側に関数の出力を示す線を描く。

オブジェクトの生成や更新、定数の指定が行なわれたときは、それぞれの線上に白三角形を描く。オブジェクトや定数を関数ボックスに入力するときは、細線や太線上に三角形を描く。細線や太線は、対象システム上からオブジェクトが消えない限り横に伸びる。ただし定数の太線は、関数ボックスに入力した時点で消える。関数は関数ボックスの下に、ID はバーの右端にラベルを描く (図 6(a))。



(a) *Hebogram* (b) Map 関数

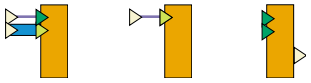
図 6: *Hebogram* システム

関数を引数に取る関数は、引数の関数を呼び出し関数内に描く (図 6(b))。リストは細線と太線からなり、オブジェクトのリストはリストの細線からオブジェクトの細線にリンク線を描く (図 5(a))。定数リストは、リストの細線と定数の太線をリンク線で結ぶ (図 6(b))。

3.3 スクリプト

スクリプトは、関数とその引数を定数や参照、変数形式で定義し、引数と関数の列を処理順に保存したものである。

引数渡しの機構および定数 参照、変数はスクリプトを定義するための引数渡しの形式である。定数は、スクリプトに関数の引数を固定値として含める。参照は、オブジェクトの ID だけをスクリプトに記録しておき、スクリプト適用時のオブジェクトの値が使われる。なお、オブジェクト以外は参照形式にできない。変数入力は、スクリプト適用時に ID と値を入力するため、仮引数をスクリプトに記録する。*Hebogram* ではそれぞれの形式を図 7 のように描画する。



(a) 定数 (b) 参照 (c) 変数

図 7: スクリプトの入力の種類

スクリプト定義表 スクリプト定義表は、定義スクリプトを次の形式で記録する。

(MacroName,(Inputs) → (Outputs),
(Reference),(Constant),(Body))

Inputs は、変数の仮引数の列を、Outputs はスクリプトの出力の列を表す。Reference は参照形式、Constant は定数形式で定義するオブジェクトを記録する。Inputs,Outputs は、履歴と同様にオブジェクトの ID とオブジェクトのデータ本体Data のペア $(ID, Data)$ を記録する。Body は、スクリプトの本体を記録する部分でスクリプトに含める関数を $(F_1, F_2, \cdots, F_n)$ のように実行順に記録する。図 4 で選択した履歴は、表 3 のような内部表現を持つ。これをスクリプト定義すると表 4 のMac1のようになる。 次に、図 8 のように履歴を選択する。この

表 3: 選択時の内部表現

履歴	内部表現
h_1	GetCPoint(c_1) → (R_1)
h_2	MakeCircle(R_1,c_2) → (R_2)
h_3	(GetCPoint(X_1) → (R_3)
h_4	Inside(R_2,R_3) → (R_4)
h_5	If(R_4,X_1,c_3) → (X_2)

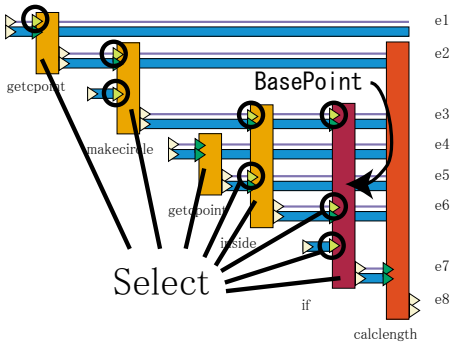


図 8: *Hebogram* 選択例 2

とき定義するスクリプトは、表 4 の駅 e_1 を定数から参照に変えたものであり、表 4 のMac2 のような

表 4: スクリプトの定義

スクリプト書式Mac1	スクリプト書式Mac2
(Mac1,((X1,X1Data)),((X2,X2Data)), (R1,R2,R3,R4),((c1,e1Data),(c2,1000),(c3,'( ))), ((GetCPoint(c1) → (R1)), (MakeCircle(R1,c2) → (R2)), (GetCPoint(X1) → (R3)), (Inside(R2,R3) → (R4)), (If(R4,X1,c3) → X2)))	(Mac2,((X1,X1Data)),((X2,X2Data)), (e1,R1,R2,R3,R4),((c1,1000),(c2,'( ))), ((GetCPoint(e1) → (R1)), (MakeCircle(R1,c2) → (R2)), (GetCPoint(X1) → (R3)), (Inside(R2,R3) → (R4)), (If(R4,X1,c2) → X2))))

書式になる。Mac1 とMac2 のスクリプトでは駅を定数にするか参照にするかの違いがあり、Mac1 は駅の位置を固定とするがMac2 では駅の位置はスクリプト適用時の現在位置から処理を行なう。

スクリプトダイアグラム　定義したスクリプトは、図 7 のように参照指定した部分は細線で、変数指定した部分は関数の入力を示す三角形だけ描く。その他は履歴と同じである (図 9)。

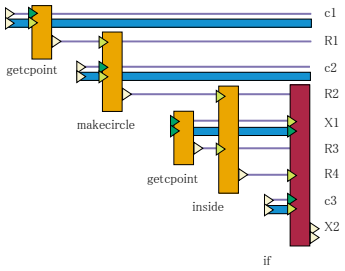


図 9: 定義スクリプト

スクリプト定義ルール　スクリプト定義表の定数Constant や変数入力Inputs, 出力Output に記録する ID は、一意なものをつける。ここで、Body の F_1 が $f_1(I_1)\rightarrow(O_1, O_{1D})$, F_2 が $f_2(I_2)\rightarrow(O_2, O_{2D})$, F_n が $f_n(I_n)\rightarrow(O_n, O_{nD})$ という関数名と入出力を持つとする。スクリプトの入力Inputs の変数 ID を X_i、Data 変数を X_{iD}、スクリプトの出力Outputs の変数 ID を X_o、Data 変数名を X_{oD} とする。また、スクリプトの定数Constan の ID を c、Data を c_D、参照定義オブジェクトReference の ID を R, その Data を R_D とする。

このとき、Body で使用する ID の I_1, I_2,$\cdots$,I_n は、X_i,X_o,c,R のどれかと等しい。X_i,X_o,c,R の IDはBody で複数回使われることもある。システムは同じ ID には同じオブジェクトを入力すると解釈する。たとえば関数 F_1 の出力 O_1 が、F_2 の入力 I_2 になるとき O_1 と I_2 の ID は等しくなる。$Body$ の最後の関数 F_n の出力 (O_n, O_{nD}) は、必ずスクリプトの出力になる。スクリプトの入力や出力は、複数個ある場合もある。

3.4　スクリプトの定義方法

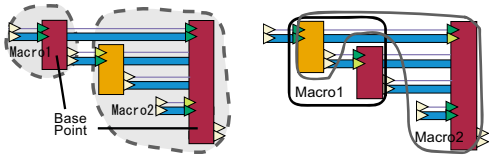
スクリプトは、$Hebogram$ 上で関数ボックスと三角形を直接選択して定義する。最初に選択した関数ボックスがスクリプト定義の基点となる。基点はスクリプト選択の開始点であり、基点の出力はスクリプトの出力になる。選択できる関数ボックスは基点の入力側に連なる関数ボックスとその入出力だけで、細線上の三角形を選択すると参照形式、太線上の三角形を選択すると定数形式, 両方とも選択しないと変数入力になる。太線と細線の三角形は排他選択になっており、同時に選択できない。定数形式は太線を生成した関数ボックスをスクリプトに含められないが、参照形式は、それ以前の関数ボックスも選択できる。関数ボックス A を選択し、その入力を参照形式で選択したときは、入力オブジェクトを参照する (図 10(a))。関数ボックス A の出力が関数ボックス B の入力となっているときに関数ボックス A、B を選択し、ボックス間の三角形を選択すると、関数ボックス A の出力を参照する (図 10(b))。選択範囲に隣接しない関数やその入出力を選択する

(a) オブジェクト参照　(b) 関数出力参照

図 10: 参照先の例

ことはできないが、基点を追加定義することで離れた位置の関数ボックスも選択できる。図 11(a) のように 2 つの基点を設定した場合、2 つ以上の出力を持つスクリプトが定義される。1 つ目の出力は選択範囲の最初から基点まで、2 つ目は 1 つ目の基点の出力から 2 つ目の基点までをスクリプト定義する。また図 11(b) のように履歴図を選択した場合、それぞれの基点から選択範囲を独立した範囲として、2 つの出力を併せ持つスクリプトとして定義する。

スクリプト定義アルゴリズム　$Hebogram$ の関数ボックスとその入出力バーは、履歴と一対一に対応す



(a) 2つの基点例　　(b) 2つの基点例

図 11: 2つの基点を持つスクリプト

る。そこで、*Hebogram* の選択部分だけ取り出し、関数ボックスと入出力バーをスクリプト定義アルゴリズムに与えると表 4 のようなスクリプトを定義する。スクリプト定義アルゴリズムは、関数ボックスのうち、時間的に最も早い位置のものから順に ID を置換する。オブジェクトの参照にはオブジェクトの ID を、関数の出力参照と定数形式には一意な ID を、スクリプトの入力と出力には ID と Data を格納する仮引数を割り当てる。割り当てた ID は選択形式によって`Input`,`Output`,`Reference`,`Constant` に記録する。そして`Body` に一意な ID を振りなおした履歴を時間順に記録する。

スクリプト実行アルゴリズム　対象システムで定義スクリプトに引数を与えて実行すると、対象システムからスクリプトの問い合わせが行なわれる。スクリプトの問い合わせが行なわれると、*Hebogram* システムはスクリプト定義表の`Body` の最初の関数 F_1 を取り出す。そして F_1 の引数 ID_1 に対応した Data をスクリプトや対象システムから取り出し、F_1 に入力して対象システムに送る。定数形式の引数はスクリプトの`Constant` から、参照形式の引数は対象システムに ID に対応した Data を問い合わせる。変数入力の引数は、スクリプトが実行された段階でスクリプトの引数を`Inputs` の ID と Data に記録するため、`Inputs` から ID に対応した Data を取り出す。1つの関数の処理が終わると`Body` から次の関数を取り出し同様の処理を行なう。関数の出力が`Output` の仮引数として定義されているときは、実行結果を`Output` に格納する。`Body` の全ての関数の処理が終わった段階で、スクリプト定義表の`Outputs` に記録している ID と Data のペア全てを対象システムに送る。

4　関連研究

関連研究として、グラフィカルアプリケーションの例示や図形操作からスクリプトを作成するシステム [1, 2, 3]、例示の履歴からマクロを定義するシステム [4, 5]、ブラウザやプログラムなどの実行コマンドの履歴をスクリプトとして記録し、再利用するシステム [6] などがある。

5　議論

操作履歴を図にすることで操作の依存関係を明示でき、これまでのスクリプト定義システムより直感的操作が可能である。またスクリプトを手入力することなく、履歴図からスクリプトを定義するため、一般ユーザも使うことができる。スクリプトの引数を変数だけでなく定数や参照形式としてスクリプトに含めることで時間変化のあるオブジェクトを扱うシステムにも対象可能である。Map 機能より複数のオブジェクトに同じ処理を行なうこともできる。

しかし実行履歴とオブジェクトの増加に伴い、履歴画面が分かりにくくなる点や履歴図からスクリプト定義をするのに慣れが必要という問題点もある。また対象システムから *Hebogram* システムに履歴を送る機能、スクリプトの定義と対象時には *Hebogram* システムから送られてくる内容を実行して結果を *Hebogram* システムに返す機能などが必要となる。さらに対象システム上で扱う関数とその引数の個数と型も予め *Hebogram* システムに定義しておく必要がある。

参考文献

[1] Myers, B. A.: Scripting Graphical Applications by Demonstration, in *CHI*, pp. 534–541 (1998).

[2] Witten, I.: PBD systems: when will they ever learn (1995).

[3] DiGisno, C. and Eisenberg, M.: Self-disclosing design tools: A gentle introduction to end-user programming (1995).

[4] Kosbie, D. S. and Myers, B. A.: Extending Programming by Demonstration with Hierarchical Event Histories, in *EWHCI*, pp. 128–139 (1994).

[5] Feinder, S.: A History-Based Macro By Example System (1992).

[6] Miller, R. C. and Myers, B. A.: Integrating a Command Shell into a Web Browser, pp. 171–182 (2000).