〈研究ノート〉

HyperCardを使ったCAIスタックの作成に関する提案

岐阜職業能力開発短期大学校　　西尾和彦

How to Make a CAI Stack Using a HyperCard

Kazuhiko NISHIO

要約　CAI (Computer Assisted Instruction) 教材を作成する場合、その分野や目的に関わりなく、最も問題となるのは、制作者のコンピュータに対する理解や知識の度合と、その開発環境の選定にあると考えられる。逆の見方をすれば、作者が思い浮かべた教材のイメージを、できるだけ忠実に表現できる環境やツールが用意されており、それらの適用に関する知識があれば、コンピュータやプログラミング言語に関する知識が乏しい場合でも、教材作成は容易だといえる。

ここでは、まずHyperCardが生まれた歴史的背景と、その設計思想を取り上げ、利用者にやさしいプログラミング環境が、どのように具現化されているのか述べる。

次に、HyperCardをCAI教材の作成をするためのツールとして採用した場合に、教材の作者が最低限知っておかなければならないカードとスタックの概念および作成法、HyperTalkの言語仕様について解説する。

また、実際の教材作成の場面においては、処理の流れをある程度、定型化し、これを雛型とすることで開発効率を高めることも必要になると考えられる。そこで、頻繁に使われるであろう、雛形的なサンプルスタックを作成した。ここでは、次のような機能と特徴を持つ4つのサンプルスタックを取り上げ、作成上のポイントを解説した。

●何枚かのカードに記された問題に、学習者がYesまたはNoで答えるサンプル。すべての設問に対する解答が終わると、答え合わせと得点および評価が表示される。

●穴埋め形式の問題に、学習者が適切な語をキーボードから入力するタイプのサンプル。学習者がすべての解答を終えると、答え合わせと得点および評価が表示される。

●設問中の虫食い（角抜き）へ埋める形式の問題に対して、学習者は語群の中の一つを選択する（マウスでクリックする）タイプのサンプル。

●語群の中の一つを虫食い（角抜き）へドラッグ＆ドロップするサンプル。

1．はじめに

教育に携わるようになって、まだ日の浅い身ではあるが、“教える”という仕事の中で適切な答えを見いだせない問題が2つある。その一つは、講義の中で学生が理解の不足を感じた部分を、いつどのような形で教える側が補うかであり、もう一つは板書やOHP、またはプリントに代表される一般的な教育用メディアでは困難な、動的な事象や構造をどう表現するかである。

多分どちらの問題も、教育に関する技術や手法を研究しておられる多くの方たちが、様々な提言や提案をされているとは思うが、私も含め教育の現場では旧態依然の授業を未だ繰り返しているように感じる。

しかし、ここ数年、管理するためではなく、教えるためのツールとして教育現場へコンピュータを導入する事例が増えていることに着目したとき、このツールを活用することで先の問題に対処できるのではないかと考えた。つまり、教える側は、教えたいことを簡便にしかも多様に表現でき、学ぶ側は、それを自学自習の形で時間にとらわれることなく、簡単な操作で必要な部分だけ検索することができるシステムを、パーソナルコンピュータというツールによって実現できればと考えたのである。

そんな折、偶然出会ったHyperCardと、教育用スタックを収録した米国製のCD-ROMは、自分の考えを

論文受付日　1994. 7. 25

実現できるツールがあることと、それをすでに実践している教育者が多く存在するというカルチャーショックを与えてくれた。今でこそオーサリングソフトウェアには当たり前なマルチメディア性と、インタラクティブなソフトウェアの制作を可能にする能力を、HyperCardはすでに持っており、何よりもそれが、Macintoshに標準でバンドルされている点で安心感もあった。しかし、当時のハードウェアにはHyperCardを快適に動作させるCPU能力は備わっておらず、日本語が扱いにくかったり、カラー化ができないなどの欠点もあったため、出来の良い遊び道具位にしか認識されていなかった。また、Macintosh自体も高価で、普及率もパーソナルコンピュータ全体の数パーセントに過ぎず、HyperCardを教育用に使う機運はまだ生まれていなかった。

だが、これらの問題点が順に解決されていくと、同じ悩みを持つ教育者の多くがHyperCardに目を向け始め、専門誌に事例が紹介されたり、商用ネット上において活発な意見交換が行われているのを目にすることも多くなった。しかし、専門誌の記事や書店に並べられているHyperCard関連の書籍を見渡しても、パーソナルユース的な事例や個々のテクニックを解説したものが多く、HyperCardをCAIのツールとして使うための技術について解説したものは少数である。

この研究報告は、以上の現状をふまえ、まず、HyperCardを教材作成のツールとして利用するために最低限知っておかなければならない、HyperCardの概念、構造、および機能の一部を解説した。次に、先に示した問題を私なりに解決すべく試行した成果として、4つのサンプルスタックを紹介する。掲載したサンプルスタックは、スタックの操作性を決定し、ブラウザ（見る人）のスタックに対する興味を持続させる要である、ユーザーインターフェースの設計と実現手法の解説を目的としている。これは、HyperCardに不可欠なインタラクティブ（対話的）なプログラミングの理解に役立つと思われる。よって、HyperCardのマルチメディア的な要素については触れていないため、詳しくは論文の最後に記した参考文献等を参照していただきたい。

本論文では、まず、2章でHyperCardの歴史的位置づけについて述べ、コンピュータの利用者自身がソフトウェアを作る意義を論じる。3章ではHyperCardの設計思想を、利用者と扱う情報の観点から述べる。4章はスタックに納めることができる情報と、その形態について述べている。カードへ情報を格納し、スタックを作り上げ、それを制御するプログラムを用意するオーサリングの概要は、5章および6章で解説した。また、7章では、イベント（キーボードやマウスの操作）を検知し、スタックやカードの振る舞いを制御する手段である、HyperTalk言語について、メッセージとハンドラを中心に解説する。8章では、試作したサンプルスタックの目的、機能、カード構成、およびスクリプトの意味について解説する。

2 HyperCardの歴史的位置づけ

パーソナルコンピュータが製品として登場したのは1970年のことである。当時の製品、たとえばAppleII, Tandy TRS-80, CommodorePETは、いずれもBASICインタプリタを搭載していた。まだ流通ソフトが整っておらず、利用者がBASICを駆使してプログラムを作りそれを仕事に（多くは趣味に）使う、といった利用形態であった。利用者は、本来ならシステムアナリストやシステムエンジニア、プログラマが共同で行うべき作業を一人でこなさなければならなかった。初期のパーソナルコンピュータが一部のマニアにしか受け入れられなかったのは当然である。

ここに革命をもたらしたのがVisiCalcである。VisiCalcは、面倒な作表作業をパーソナルコンピュータで簡単にできるようにし、ビジネスマンに普及していった。つまり、マニア以外の人達がパーソナルコンピュータを使うようになったのである。ここから、プログラムを作る人と使う人の分離が始まったといえる。

作る人と使う人をより明確にしたのはMacintoshである。Macintoshは、ToolBoxという非常に強力な開発環境をプログラム開発者に提供することで、品質のよい流通ソフトの出現を促してきた。重要なのは、プログラム開発者の道具と、利用者の道具を分離したことである。

しかし、流通ソフトは、利用者に共通する要求だけを満たすものである。言い換えると、すべての利用者が少しずつ我慢せざるを得ない、という本質的な欠点を持っている。こうした流通ソフトの欠点を補うために登場したのがHyperCardだといえる。

HyperCardは、基本機能を情報管理に絞っている。コンピュータで行う仕事は、結局のところ情報管理だからである。また、個々の利用者が必要とする道具（スタックウェアと呼ばれる）を簡単に自作できるようにHyperTalkと呼ぶ言語を備えている。さらに文字だけでなく、絵や音などの情報も扱え、より直接的な表現ができる。したがって非常に汎用性がある。第一の波

をBASIC、第二の波をVisiCalcとすれば、第三の波はHyperCardである。HyperCardは従来の流通ソフトの欠点を補い、パーソナルコンピュータの応用分野を飛躍的に拡大することが可能である。

3　HyperCardの設計思想

HyperCardは、情報管理の道具である、情報の表現、情報の運用、情報の性質という三つの観点から、HyperCardの特徴と情報に関する考え方を眺めてみる。

3．1　情報の表現

我々はデータベースなどで情報を取り扱うとき、普通、文字で情報を表現していた。これはコンピュータ技術が十分進歩していないための制約といえる。しかし、現実の情報は、絵や音や臭いといった人間の五感で感じとるものの表現であり、直接的であればあるほど、より正確に表現することができる。

HyperCardは、文字と絵と音で情報を表現できる。また絵の場合は、静止画だけでなく、簡単な動画（アニメーション）も扱える。したがって、より正確に、より直接的な表現で情報を個々の利用者に提示できる。このことは、コンピュータの利用者層を拡大する上での重要なファクタである。

3．2　情報運用

HyperCardでは利用者を、1．情報を見る人(Browser)、2．情報を作る人（Author)、3．情報を扱う道具を用意する人（HyperTalkのプログラマ)、の三つに大別し、それぞれの利用者に適した道具を用意している。HyperCardの開発者であるビル・アトキンソン氏は、『情報を作る人がいるのならば、情報を見る人はその20倍はいる』という考えからHyperCardを設計したと語っている。

3．3　情報の性質

情報には、単体では殆ど意味をなさないという性質がある。たとえば、人の名前だけでは、その人のことは分からない。まず、その人に関する情報として住所、年齢、勤め先、家族構成を知り、次には勤め先や家族の概要、その次には上司や友人、というように情報の連鎖をどんどん手繰り寄せたくなる。HyperCardは、網の目のようにお互いに関連し合った情報を、利用者の好奇心のままにその関連を手繰れるような道具として設計されている。

3．4　HyperCardとデータベース管理システム

HyperCardは、情報を管理するという目的でデータベース管理システムとよくにている。違うのは、情報を管理する主体である。データベースでは、文字通り、情報の基地を作ってみんなで利用しようという考え方が基本である。そして、情報を表で表現し、表を表現するのに都合のよい、数学の関係代数を拝借してきてデータベースを管理しているのである。

これに対し、HyperCardでは、個人の頭にある記憶を拡張しようという考え方をとっている。情報を相互に関連しあった断片的な知識ととらえ、人の好奇心を利用して断片的な知識の連鎖を探すのである。

情報の共有という視野から情報管理を考えたときは、データベースが有利であろう。しかし、利用者の側から見れば、好奇心のまま情報を見つけてゆくHyperCardのほうが使いやすい。つまり、両者は互いに相反する長所と短所を持っているのである。

4　ブラウジングによる情報参照

HyperCardは、紙芝居のようなものである。1枚1枚の紙には絵や文字や音で表現した情報があり、利用者はこれを手繰りながら一連の情報を知っていく。これをブラウジング（Browsing）という。

HyperCardを起動してまず利用者が見るのはホームスタックである。これは、事務所でよく見かけるキャビネットを正面から見ていると考えればいい。キャビネットには縦横に並んだ引き出しがあるように、画面に表示されたアイコンは各分野の書類を格納した引き出しである。

HyperCardでは、この引き出しをスタックと呼んでいる。スタックは形式が同じ情報のカードの束である。利用者はスタックのアイコンをマウスで選択するだけで書類の引き出しを開けられる。

正確にいうと、ホームスタックもスタックの一種である。ここには、HyperCard内にどんなスタックがあるかを示すアイコンの一覧表（ホームカード）が入っている。

情報カードに書き込める情報の形式としては、フィールド、ピクチャ、ボタンがある。フィールドは文字列、ピクチャは絵を書き込む領域である（音を扱うには、HyperTalkでプログラミングする必要がある)。ボタンにはHyperCardのアクション（スクリプト）を記述し、利用者はボタンをマウスで選択することで、次の情報カードに進む、前の情報カードに戻る、別のスタ

ックの情報カードに移動する、といったことができる。先にHyperCardは情報の関連を手繰る道具だと説明したが、それをボタンで実現しているのである。

このようにボタンに触れるだけで、関連する情報を手当たり次第に手繰っていける。これは非常に重要なことである。人が情報を検索するとき、知りたい情報の範囲は、あらかじめ決まっていない。検索中に好奇心がなくなればそこで終わり、という”好奇心主導型の検索システム”が人間なのである。HyperCardでは、情報カードのボタンを見て、興味のある情報ならマウスでクリックして開く、という手順を繰り返す。したがって、HyperCardは、非常に人間に適した検索システムを提供しているといえる。

5 オーサリングでの情報の作成

観客が紙芝居を見るためには、物語を考え紙に絵を描いた制作者がいるはずである。HyperCardでは、紙芝居を作ることをオーサリング（Authoring）と呼んでおり、そのための道具も用意している。オーサリングの大まかな手順は次のようになる。

1) スタックを用意する
2) バックグラウンド（同一のスタック内の情報カードすべてに適用される共通フォーム）を作る
3) カード（情報カードそれぞれの情報）を作る

まず、スタックの用意は、ファイルメニューの”新規スタック”を選択するだけですむ。これで情報カードの引き出しであるスタックが用意される。次に引き出しに入れる情報カードを用意するわけである。これについては後述する。

5．1 バックグラウンドとカード

情報カードは、バックグラウンドとカードの二つの層で構成されている。これは、OHPでの講義を思い出せば分かりやすい。最初の1枚目のOHPシートに他のOHPシートと共通の図柄、たとえばタイルや目盛りを描いておき、その上には各月別のグラフなどを順次重ねていけば描くグラフすべてにタイルや目盛りを描き込まなくてもよいのである。この最初の一枚がバックグラウンドであり、その上に重ねるOHPシートがカードに相当する。

バックグラウンドにもカードにも、フィールドとピクチャとボタンを設定できる。注意しなければならないのは、バックグラウンドに設定したフィールドやピクチャやボタンが情報カードを手繰っても変わらない（移動しない）のに対して、カードに設定したそれらは情報カードを手繰るとカードと一緒にめくられてしまうことである。

再びOHPシートにたとえると、バックグラウンドのフィールドやピクチャやボタンは、初めの一枚目に直接描いた情報で、カードのそれらは一枚目の上に重ねるOHPシートに描いた情報である。そして、情報を手繰るということは、2枚目のOHPシートを取り替えながら、各月の売り上げグラフをみていくようなものである。2枚目のOHPシートをめくってしまえば、そこに描かれている情報もかわってしまうのである。

5．2 バックグラウンドを作る

バックグラウンドは、情報カードの共通フォームであり、基本的にはスタックごとに一つある。使いやすいスタックにするコツは、バックグラウンドで名刺箱とか手帳といった現実のありふれたものを模倣することである。一つは親しみやすいという利点があるが、それ以上に使う人がすでに操作方法になれているという点が重要である。スタックの利用者（Broser）は操作方法を勉強しなくても自然に使っていけるし、スタックを作る人（Author）は、操作マニュアルを用意しなくてもすむからである。バックグラウンドを作るには、まず編集メニューの”バックグラウンド”を選択し、画面をバックグラウンドの編集モードにする。その後、ピクチャ、フィールド、ボタンの順で設定していく。

ピクチャはMacPaintと同じ操作方法で描画できる。ツールメニューからブラシや消しゴムを選び、絵を描いていく。この絵を描くことで、”ありふれた物”を模倣するのである。

フィールドとボタンについては、まずオブジェクトメニューからそれぞれ”新規ボタン”と”新規フィールド”を選んで標準ボタンと標準フィールドを作りだす。そして、それぞれをダブルクリックするか、オブジェクトメニューの”ボタン情報”または”フィールド情報”を選んでダイアログを表示し、ボタンやフィールドの属性を選んでいく。

5．3 カードを作る

カードを作るには、まず編集メニューの”新規カード”を選択し、バックグラウンドに定義したフィールドにテキストを書き込む。普通は、これを繰り返していくだけである。しかし、バックグラウンドとは別に、カード固有の情報としてピクチャやボタンを設定することもできる。この場合の操作はバックグラウンドの

場合と全く同じであるが、前述のとおり、ここで作ったフィールドやピクチャやボタンは情報カードを手繰ると画面から消えてしまう。

オーサリングとは以上のように、情報の入れ物（スタック）と情報自身（情報カード）及び情報の関連を作る作業である。同様のことをデータベース管理システムでしようとすれば、第4世代言語（独立言語）を駆使してプログラミングするはめになる。

6　HyperTalk言語の概要

HyperTalkは手続き型のプログラミング言語である。これはデータベース管理システムに第4世代言語が用意されているのと似ている。第4世代言語がデータベース専用の表操作言語であるように、HyperTalkもHyperCard専用のスタック操作言語である。特徴は、

1）オブジェクト指向言語の特徴を少しばかり取り入れている
2）自然な英語の文法に似ている
3）インタプリタとしてHyperCardに組み込まれている

などである。

6．1　オブジェクトの属性

HyperTalkでのプログラミングは、一般的なCやPascalでの場合と大きく異なる。CやPascalでは、全体の制御をプログラムが行い、システムが用意した機能をサブルーチンとして使う。

HyperTalkではこの関係が逆転している。全体を制御しているのはシステム（HyperCard）であり、プログラムはシステムのサブルーチンとして働く。そして、プログラムは、ボタンやフィールド、カード、バックグラウンド、スタックの属性の一つとして定義される。つまり、プログラムは、ボタンやフィールドが押されたときHyperCardがとるべき行動の記述と考えれば良い。そのための小さなサブルーチンをたくさん作るのが、HyperCardのプログラミングの特徴といえる。

6．2　継承構造（inheritance）

これまでの説明から分かるように、HyperCardのスタックは、いくつかの要素で階層的に組み立てられている。そして、HyperTalkで書いたプログラムは、各階層を構成するオブジェクトの属性として定義されているのである。マウスボタンがクリックされると、HyperCardはその旨のメッセージを、マウスカーソルが指している階層から上位の階層に向かって流していく。そして、メッセージを受け取るべきHyperTalkプログラムに出会うと、メッセージの流れは止まり、出会ったプログラムに処理を依頼する。したがって、選択されたボタンにプログラムを設定しておく必要はなく、一番有利な階層を選んで、そこにプログラムを埋め込んでおけばいいのである。これは、オブジェクト指向言語の継承（inheritance）と呼ばれる概念を取り入れた機能である。

7　HyperTalkの言語仕様

HyperCardはオーサリング・ツールの一種である。オーサリング・ツールとは、CAIの分野でよく使われる言葉で”ユーザが少しの手間で情報処理の道具（大抵はプログラム）を作る環境を持ったシステム”という意味がある。たとえば、BASICなどのプログラミング言語もオーサリング・ツールである。HyperCardが従来のオーサリング・ツールと異なるのは、標準で添付されている小道具（ボタン、カード、スタック）を組み合わせて、目的の道具が作成できることである。

だだし、標準添付の小道具をそのまま利用するだけでは、できることは限られる。自分に適した形に道具を手直ししたり、新たに小道具を作成したりするにはHyperTalk言語を知っておく必要がある。

7．1　メッセージとハンドラ

HyperTalkを理解する上で最も重要となるキーワードがメッセージ、メッセージハンドラ、そしてオブジェクトである。HyperTalkというプログラミング言語は先に述べたようにオブジェクト指向が取り入れられているので、オブジェクトに対してメッセージを送り、オブジェクトがそれを受け取って処理をする。

HyperCardでは、ボタン、フィールド、カード、バックグラウンド、スタックの5つをオブジェクトと呼ぶ。

メッセージは、マウスボタンがクリックされたときやスタックが開いたとき、あるいはカードがめくられたときなどにオブジェクトに対して送られる。

オブジェクトは送られてきたメッセージを受け取って、自分自身に対してユーザがどのようなアクションを行ったのかということを知る。

たとえばボタンAをクリックすると、ボタンAに対して mouseUP というメッセージが送られ、ボタンBをクリックすると、ボタンBに対して mouseUp というメッセージが送られる。見方をかえれば、それぞれ

がクリックされたことを知るのである。

同じように、カードは openCard というメッセージを受けて自分が開かれたことを知り、closeCard というメッセージで閉じられたことを知る。

メッセージがオブジェクトに対して送られるケースは次の２つである。

- ユーザが何らかの操作を行ったことを知らせる（マウスボタンのクリックやキーボードの操作など）
- HyperCardの状況を知らせる（スタックが開いた／閉じた、カードが開いた／閉じたこと）

メッセージハンドラはメッセージを放送局からの電波に例えるなら、それを受信するチューナの機能を持つ。つまり、様々な電波（メッセージ）の中から、必要とするものだけを選択する機構といえる。

メッセージハンドラは次の構造を持っている。

```
on ＜メッセージ名＞ ［＜パラメータ・リスト＞］
      ＜スクリプト＞
end＜メッセージ名＞
```

メッセージがオブジェクトに送られると、同じメッセージ名のハンドラがそれを受け止め、ハンドラの内容が実行される。

たとえば、次のメッセージハンドラは mouseUp というメッセージをうけて実行される。このハンドラをボタンに書き込めばそのボタンをクリックすると次のカードへ移動する。

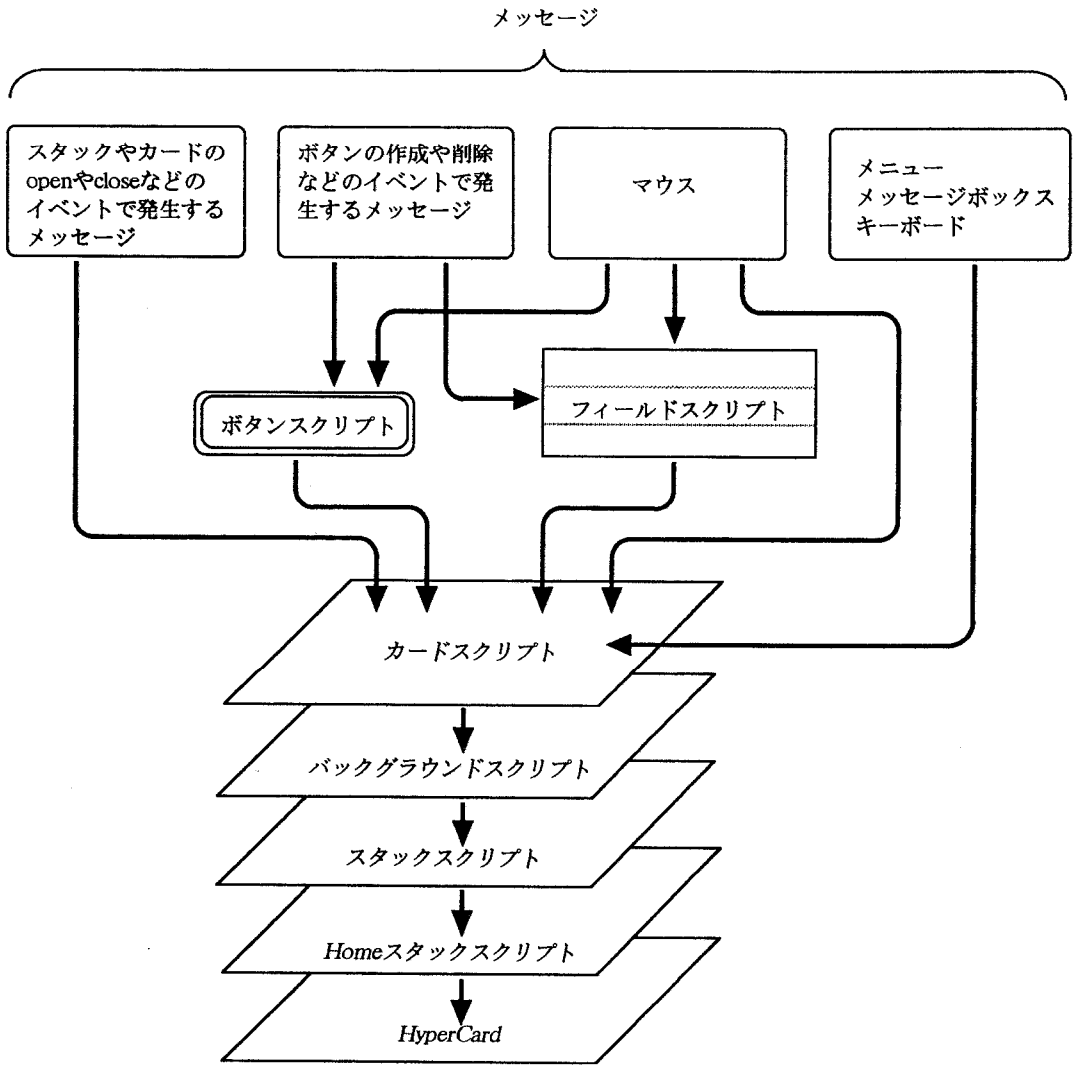


図7. 2-1　メッセージの継承

```
on mouseUp
    go to next card
end mouseUp
```

上のメッセージハンドラはクリックによって送られる mouseUp メッセージを受け止めて実行されるものなので、他の操作によって別のメッセージが送られてきても動作はしない。すなわち、ユーザの操作などを受け止めて何かのリアクションを行いたい場合には、その操作によって何というメッセージが送られるかを知ればよいことになる。

7．2　メッセージの継承

メッセージはオブジェクトに対して送られるが、そのオブジェクトに送られてきたメッセージを受け止めるメッセージハンドラがなかった場合、そのメッセージは次の階層のオブジェクトへと継承されるのである。オブジェクトの階層は決まっているので、メッセージの継承は次の順番で行われる。

1）ボタンまたはフィールドスクリプト
2）カードスクリプト
3）バックグラウンドスクリプト
4）スタックスクリプト
5）Homeスタックのスタックスクリプト
6）HyperCardアプリケーション

なお、ここで注意したい点が3つある。

一つ目は、すべてのメッセージがボタンまたはフィールドを継承の出発点にしないということである。メッセージの出発点はボタン、フィールド、カレントカードのいずれかである。たとえば、カードを開くと openCard というメッセージが出るが、これはボタンではなくカードに対して送られる。また、バックグラウンドが開いた場合には openBackground というメッセージが発生するが、このメッセージは直接バックグラウンドに送られるのではなく、カードに送られるのである。

2つ目はバックグラウンドボタンやバックグラウンドフィールドに送られたメッセージも、その次に継承されるオブジェクトはカレントカードであって、カレントバックグラウンドではないという点である。

3つ目は、ボタンやフィールドはお互いに重なり合って置かれることがあるが、それらが重なり合っていても、ボタンまたはフィールドの次にメッセージが継承されるオブジェクトはカレントカードである点である。画面で見えるオブジェクトの重なりはメッセージの継承とは関係ないのである。

7．3　オブジェクトの階層とメッセージハンドラ

オブジェクトの階層はメッセージハンドラの有効範囲と一致する。

メッセージハンドラをカードスクリプトに書き込むと、そのカードにメッセージが送られて着いたときにだけそのメッセージハンドラが動作するが、バックグラウンドスクリプトに置けば、そのバックグラウンドを共有するカードすべてにおいて有効なメッセージハンドラとして機能する。スタックスクリプトに置けば、そのスタックのすべてのカードで有効なメッセージハンドラとして機能するのである。

逆に、スタックスクリプトにあるメッセージハンドラと同じメッセージを受け止めるメッセージハンドラが特定のカードにあれば、そのカードのメッセージハンドラのほうが先にメッセージを受け取るので、そのカードだけは他のカードと違う機能を持たせるといったことが可能となる。

なお、継承されるのはメッセージだけではない。コマンドや関数もまたメッセージと同じように継承される。すなわち、ファンクションハンドラをどこに置くかはメッセージハンドラと同じ考え方になるのである。

以上を整理すると次のようになる。

- ●カードのオブジェクトに共通するスクリプトはそのカードスクリプトに置く
- ●バックグラウンドを共有するカードに共通するスクリプトはスタックスクリプトに置く
- ●すべてのカード、すべてのバックグラウンドに共通するスクリプトはスタックスクリプトに置く
- ●すべてのスタックに共通するスクリプトはHomeスタックのスタックスクリプトに置く

同名のハンドラが同じスクリプトエディタ内にある場合は、先に書いてあるハンドラがメッセージを受け止める。

8　サンプルスタック

HyperCardを使ったCAIスタックの作成例を紹介する。ここで取り上げるサンプルスタックは、「はじめに」の中でも述べたように、ユーザーインターフェースの設計と実現手法の解説を目的としている。

8．1のスタックは、カード上に書かれた設問に、Yes

またはNoのボタンをクリックすることで答える。設問のカードは5枚あり、これらすべてに対する解答が終わると、答え合わせと得点、および評価が表示される。このスタックではメッセージハンドラの基本的な使い方、カード間の情報授受と共有、処理の関数化とその適正な置場所に関して論じた。

8．2は文章中に虫食い（角抜き）が設けてあり、これに適切な語をキーボードから入力するタイプのスタックである。これも最後に答え合わせのカードを持つ。ここでは、利用者の利便を考慮し、カード上に複数のフィールドが存在する場合、マウスだけでなくキーボードからも書き込む対象となるフィールドを選択できるようにする手法を紹介する。

8．3のスタックは、文章中の虫食いに、語群から選んだ語をクリックし答えるタイプのスタックである。このスタックでは、オブジェクト間の情報転送と、現在選択されているオブジェクトがどれなのかを利用者に知らせ、ハンドリングする手法を中心に解説する。

8．4のスタックは、語群から選んだ一つを文章中の虫食いに、ドラッグ＆ドロップする方式を採用している。ここでは、オブジェクト自体を移動させる手法に注目してほしい。

8．1 設問にYesまたはNoのボタンをクリックして答える手法

オーソドックスな二者択一の解答形式をHyperCardを使って実現する場合、各設問に一枚のカードを割り当てることが多い。ここでポイントとなるのは、設問が複数ある場合に、何問目ではどちらが選ばれたかを、答えあわせのカードに反映させる手法にある。

紹介するスタックでは、グローバル変数を使って、カード間の情報の授受を実現した。

このスタックは7枚のカードで構成されている。図8．1－1に示す1枚目のカードには、このスタックの簡単な使い方が解説される。このカードの、”右矢印”ボタンをクリックすれば2枚目以降の設問カードへ移り、”家の形”のボタンをクリックすればホームカードへ戻ることができる。各ボタンには、それがクリックされたら次のカードへ移動する go to next card、およびホームカードへ戻る go home のスクリプトが書き込まれている。ホームカードへの移動には少々時間を要するため、set cursor to watch によって、マウスカーソルの形状を”watch”マークに変えている。

2枚目から6枚目の各カードには、図8．1－2に示すような設問と、”Yes”および”No”のボタンが配置されている。利用者は設問の内容が正しければ”Yes”、誤っていれば”No”のボタンをクリックしていく。各ボタンスクリプトには、それがクリックされたとき、スタックスクリプトに納められた関数を呼び出すよう指示されている。たとえば問題1（Q1）のカードの”Yes”ボタンには、

```
on mouseUp
  Q1Y
end mouseUp
```

と書かれており、スタックスクリプトの関数（Q1Y）を呼び出す。

このように、ボタンのハンドラからスタックスクリプトの関数を呼び出すことによって、同じ種類の処理を共通のオブジェクト内のスクリプトに置くことは、スタックの保守性を向上させる。

HyperCardは、オブジェクト指向的なプログラミングを可能にしているが、似たような処理が散在するこのような例では、すべてのオブジェクトから参照できる場所（オブジェクト）に、各処理内容を羅列したほうがメンテナンスや拡張が易しい場合もある。

関数（Q1Y）には、グローバル変数（q1）へ得点（20）をセットする put 20 into q1 の命令がある。また、”No”のボタンが押されたときに実行される、スタックスクリプトの関数（Q1N）では、put 0 into q1 の命令でグローバル変数（q1）へ0がセットされる。つまり、”1＋1は2か？”の設問に対して、正解である”Yes”のボタンがクリックされれば20点、不正解の”No”のボタンでは0点が変数（q1）に納められ、go to next card で次のカードへ移動する。グローバル変数（q1）は7枚目のカードにおける集計処理で参照している。

スタックスクリプトには、5枚の設問カードの”Yes”および”No”ボタンに連動して起動される関数（Q2Y, Q2N, Q3Y, ･･･, Q5N）が用意されており、Q.1と同様、Q.2であればq2、Q.3ならばq3のグローバル変数へ0または20が代入されることになる。

すべての設問に解答し終わると、図8-1.3に示す7枚目の答え合わせカードが表示され、”答え合わせ”のボタンをクリックすることで、利用者の答え、正解、得点、および評価が示される。

7枚目のカードのカードスクリプトは、hide cd fld ”フィールド名” によって、”あなたの答え”、”正解”、”評価”、”得点”の各フィールドを一時的に隠す。

“さあ！結果は”のボタンがクリックされると、これのボタンスクリプトからスタックスクリプトの関数

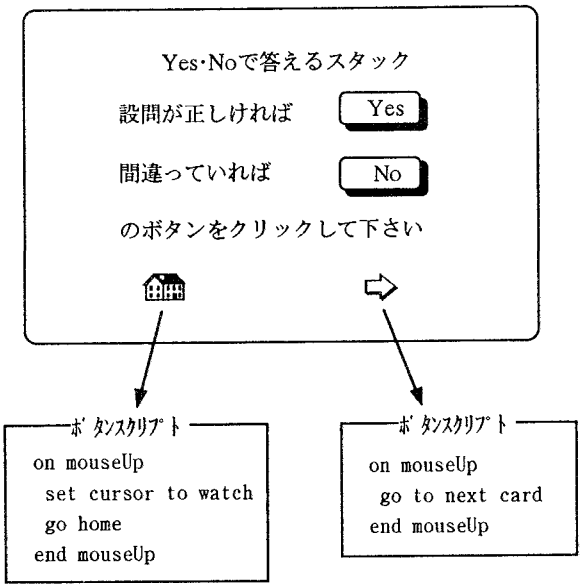


図8. 1-1 Yes・Noで答えるスタックの1枚目のカードとスクリプト

(ans) を呼び出した後、ハイドされていた（隠されていた）フィールドを見せる。スタックスクリプト内の関数 (ans) は、Q.1からQ.5までの利用者の解答と正解をフィールドに表示する関数（q1ans～q5ans）を呼び出す。各問いの利用者の得点は、グローバル変数（q1～q5）に記憶されているので put q1 + q2 + q3 + q4 + q5 into ans で集計し、その合計得点と評価をフィールドへ書き出す。

8．2 角抜きへ文字を入力する手法

カード上に書き込まれた文章の何カ所かが虫食いになっており、利用者がここへ適切な語句を入力する手法について考えてみる。

サンプルスタックの、一枚目のカード（虫食いへ文字を入力するカード）のレイアウトと、スクリプトを図8. 2-1に示す。

文章は、ツールパレットのペイントテキストツールを使ってカードへ書き込む。当然、虫食いの場所には、必要な文字数分の空白を置く必要がある。次に、虫食いにした場所へツールパレットのフィールドツールを使ってフィールドを置く。図8. 2-1に示すカードでは、5つの文字フィールド（フィールド名：fld1～fld5）が置かれている。ここで注意してほしいのは、フィールドの属性が図8. 2-2のように設定されていることである。

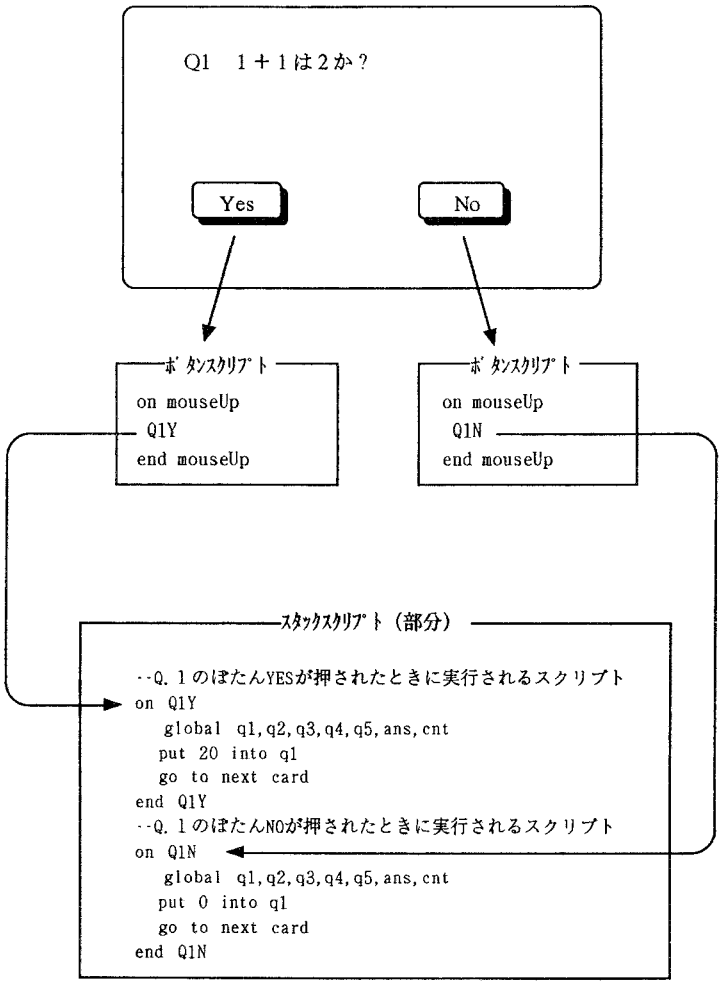


図8. 1-2 Yes・Noで答えるスタックの設問カードとスクリプト

一枚のカード内に複数のフィールドが存在する場合、それをどう選択するかが問題となる。通常、フィールドはマウスでそれをクリックし、選択された状態にしてから文字を入力する。しかし、いちいちマウスでクリックしていたのではキーボードから手を離さなければならない為、"tab" キーでフィールド間を移動することが多い。この"tab" キーと同じ働きを、"return" キーにも持たせるのが、フィールド情報内の" オートタブ" である。この項目をチェックしておくと、利用者が文字（または文字列）をフィールド内に入力した後、"return" キーを押すことで次のフィールドへ移動することが可能になる。

サンプルスタックは5つのフィールドに文字を入力したあと、" 答えあわせ" のボタンをクリックすることでスタックスクリプトのgetfield関数を呼び出す。関数内ではグローバル変数（f1, f2, ･･･, f5）へ各フィールドの内容を代入したあと、2枚目のカードへ移動する。

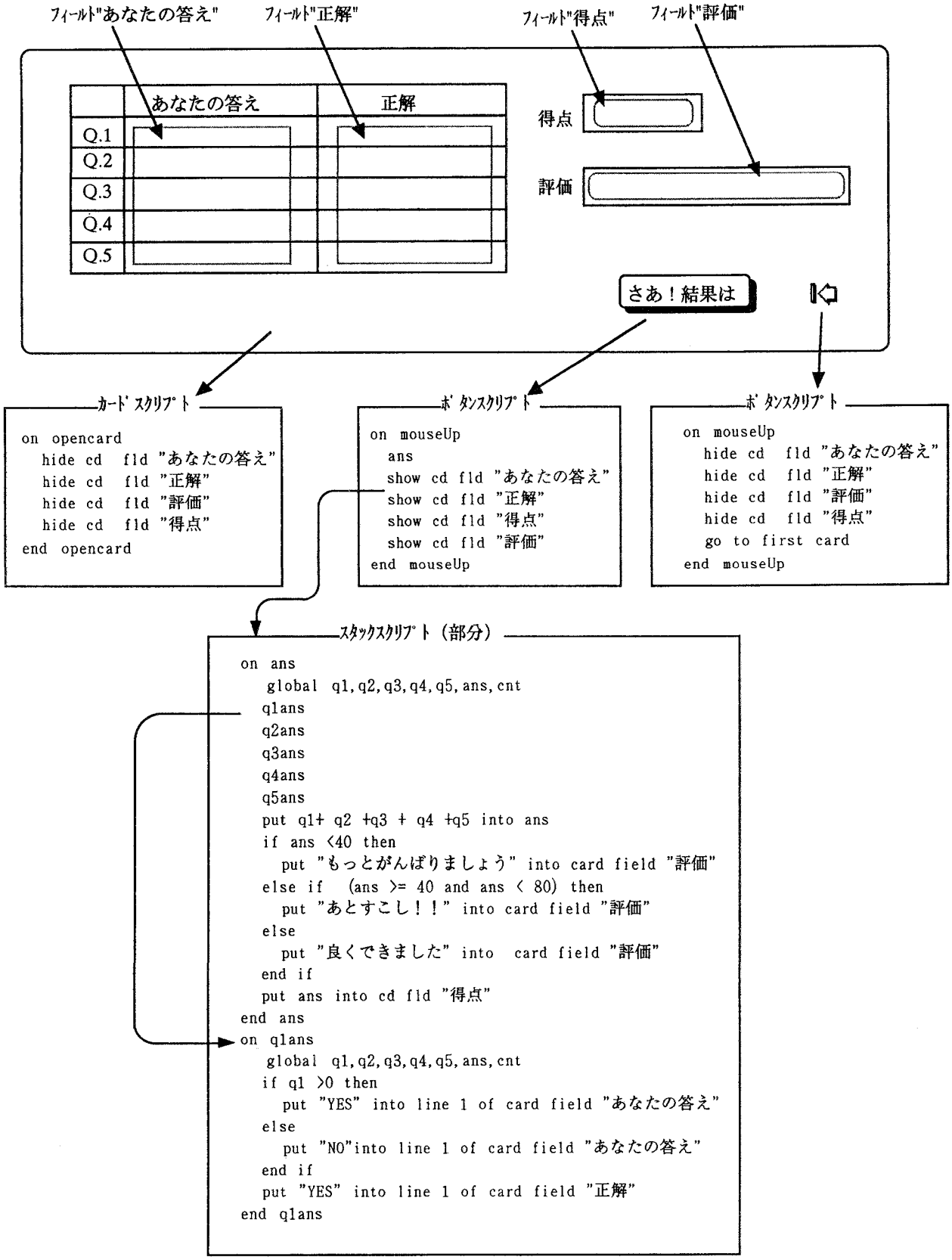


図8. 1-3 Yes・Noで答えるスタックの7枚目のカードとスクリプト

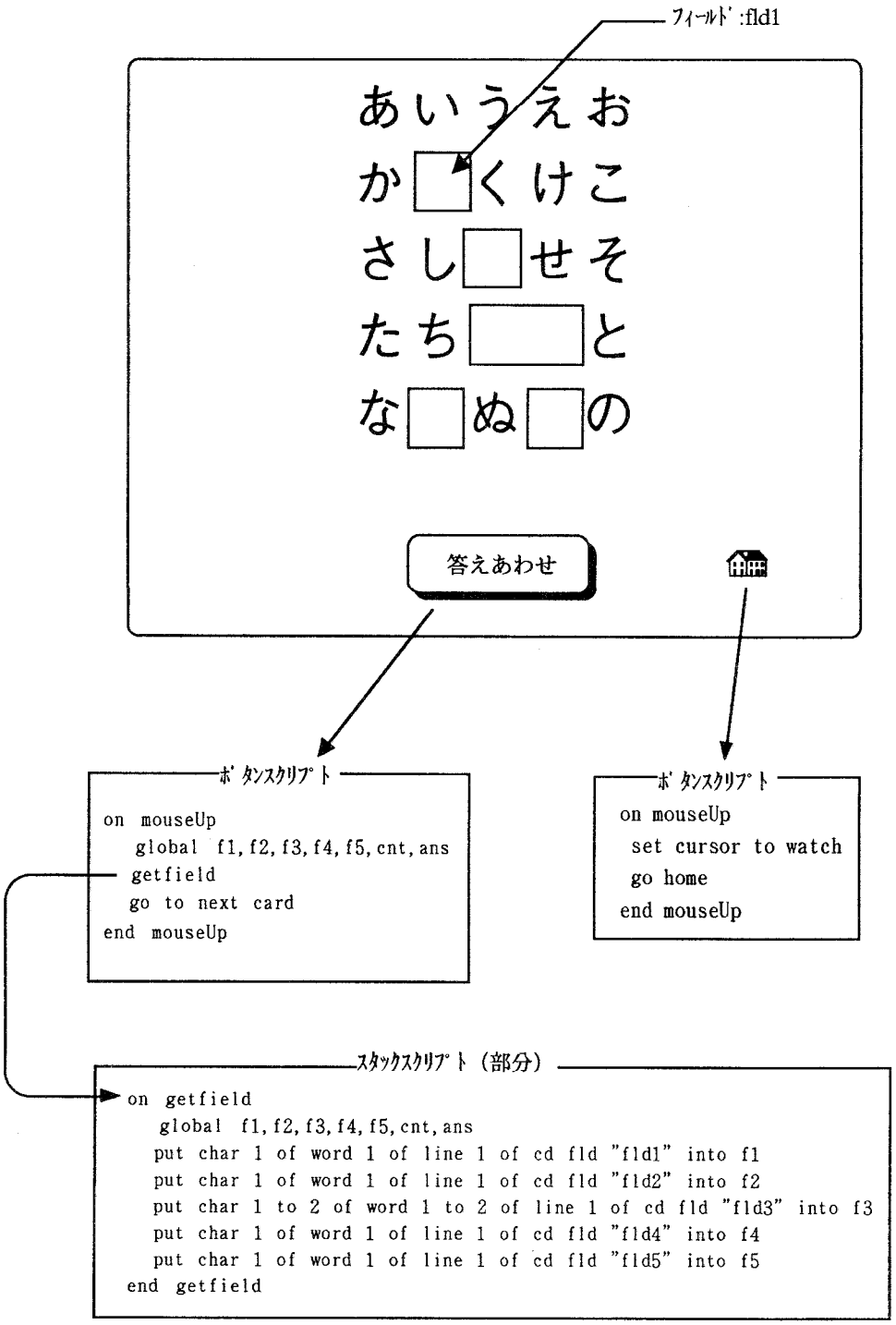


図8. 2-1　文章中の虫食いへ文字を入力するカード

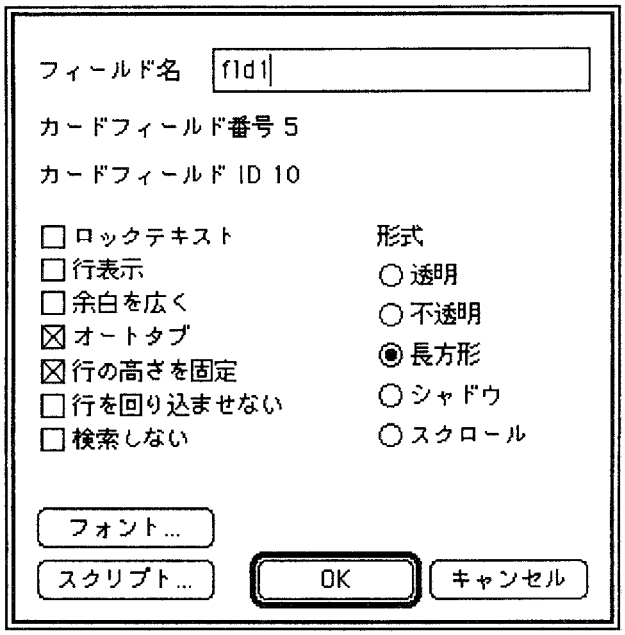


図8. 2-2 フィールドの属性

答えあわせをおこなう2枚目のカード（図8. 2 3）は、それが開かれた時点で、カードスクリプトが以下の処理を行う。

1） 利用者の答えを、”あなたの答え”フィールドへ書き出す。
2） 正解を、”正解”フィールドへ書き出す。
3） 利用者の答えと正解を照らし合わせ、得点を計算する。
4） メッセージウィンドウへ得点を書き出す。

8．3 語群をクリックして角抜きを埋める手法

ここでは、文章中の角抜きされた部分を、語群の一つをクリックして埋める手法を紹介する。

スタックの基本的な考え方は、角抜きを8. 2のスタックと同様、ロックテキストされていないフィールドで持ち、これへクリックされた語群フィールドの内容（文字列）をコピーすることにした。つまり、フィールド間の文字列コピーによって、選択された語が目的の角抜きへ複写される作用を利用している。

このタイプのスタックは、複数の角抜きがある場合、その中のどれが選択されているかを、どのように利用者に知らせ、スクリプトで認識するかが作成上のポイントになる。紹介するスタックでは、次のように考えた。

1） 語を埋めるフィールドは通常、四角で表示されるが、それがクリックされ選択状態になったとき、フィールド自体に影をつけることで利用者に認識させる。
2） スタックが開かれたとき、語を埋めるフィールドが選択されていなければならないことをメッセージボックスで利用者に知らせる。
3） フィールドが未選択の状態で、語群のいずれかがクリックされたときも、2）と同様のメッセージを利用者に伝える。
4） どのフィールドが選ばれているかを記憶するグローバル変数を用意し、語を埋めるフィールドが選ばれたときに、これへユニークな値をセットする。
5） 語群の中のいずれかがクリックされた場合は、4）のグローバル変数の値から現在選ばれている角抜きの場所を知り、このフィールドへ語を複写する。

カードのレイアウトとスクリプトを図8. 3-1に示す。図のとおり、サンプルスタックのカードには2つの角抜きフィールドと、4つの語群フィールドが用意されている。利用者は、まず何れかの角抜きフィールドをクリックして選択したあと、語群のどれかをクリックすることで角抜きを埋めることができる。

1） カードスクリプト

グローバル変数（a）は、どの角抜きが選択されているかを記憶する。変数が”1”ならばフィールドa、”2”ならb、”0”はどちらも選ばれていない状態である。カードが開かれた時点では0に初期化される。カードが開かれた時点では、どちらの角抜きフィールドも選ばれていないため、

set style of fld ”フィールド名” of cd id <カードID> to rectangle

の命令で、フィールドaおよびbを四角の枠線のみとする。

2） 角抜きのフィールドスクリプト

角抜きのフィールドに対するクリックをハンドルするため、そのフィールドスクリプトにはマウス検知のハンドラを設ける。フィールドaがクリックされた場合は、グローバル変数（a）は”1”にセットされる。また、フィールドaは

set style of fld ”フィールド名” of cd id <カードID> to shadow

によって影付きの枠表示となる。

フィールドbがクリックされたときは、グローバル変数（a）は”2”にセットされ、フィールドbが影付き枠表示となる。

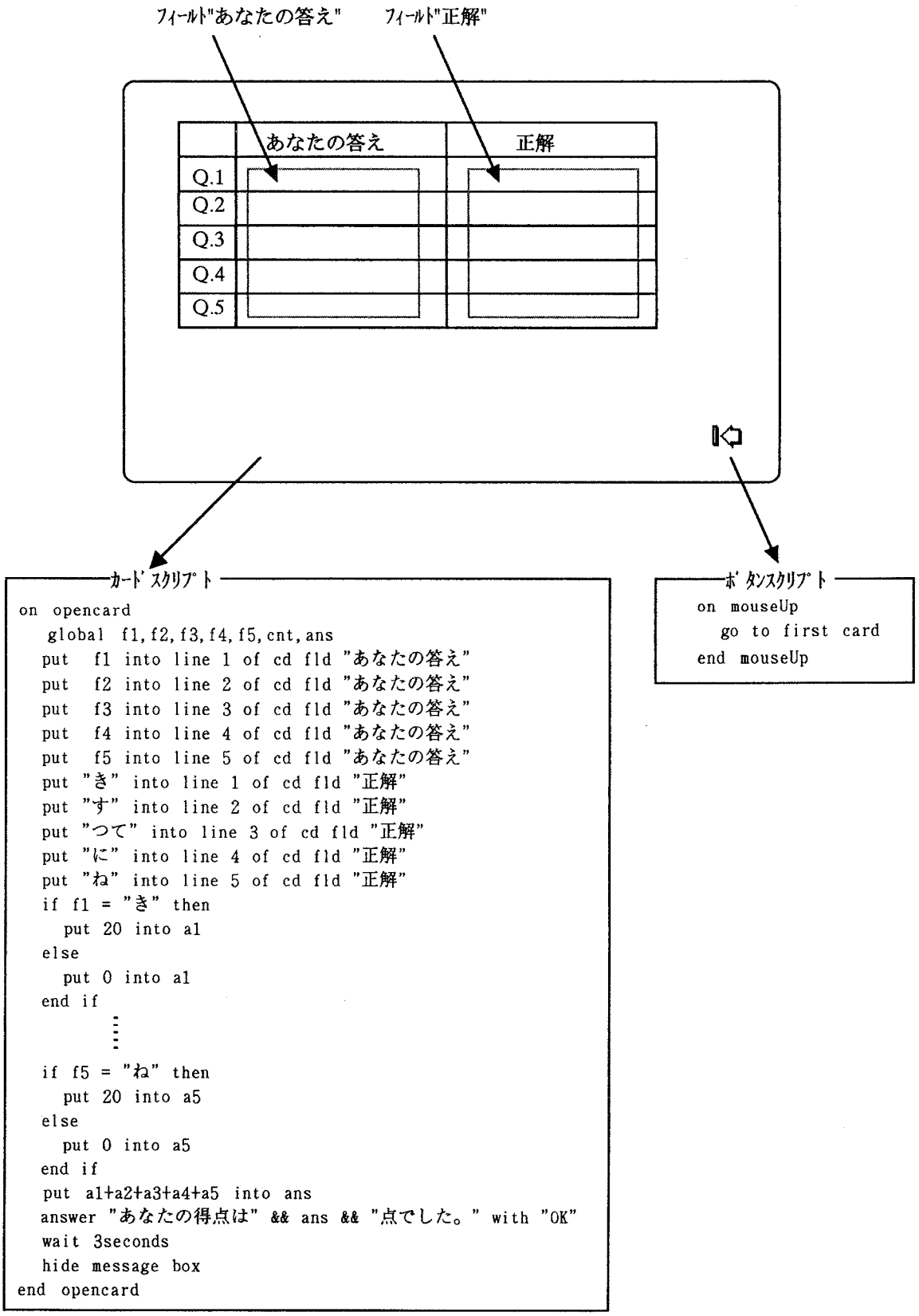
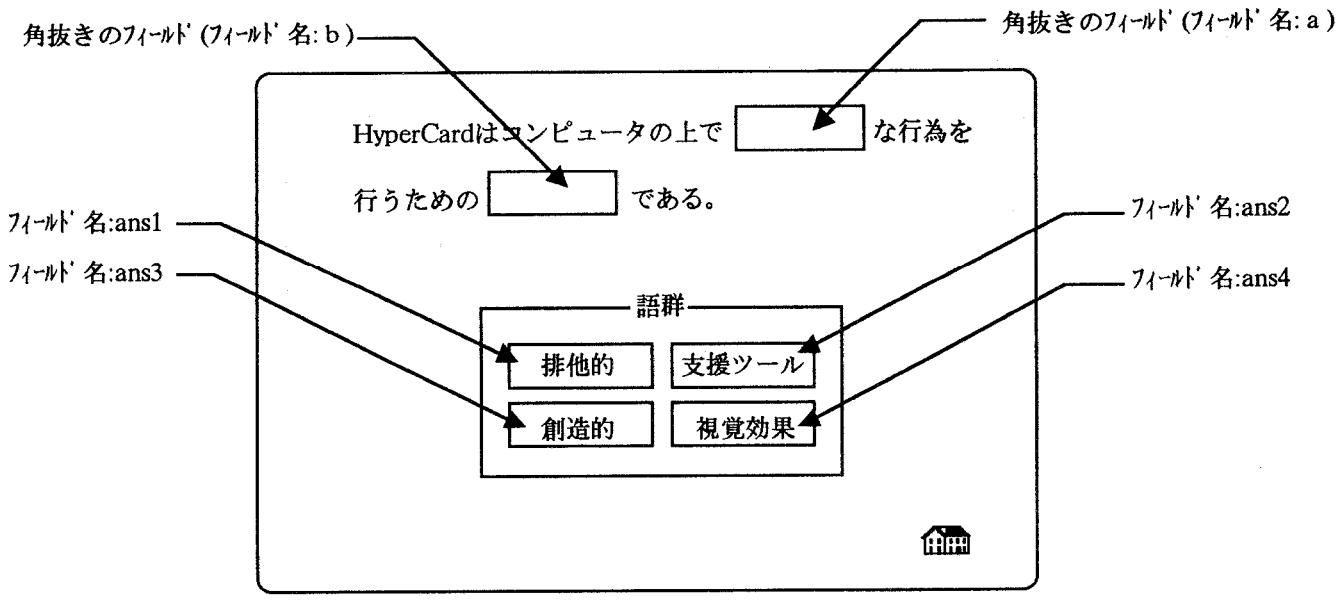


図8. 2-3　虫食いに入力するスタックの2枚目のカードとスクリプト



カードスクリプト

```
on opencard
  global a
  put 0 into a
  set style of cd fld "a" of cd id 6143 to rectangle
  set style of cd fld "b" of cd id 6143 to rectangle
  answer "フィールドをクリックしてから選んでください" with "Ok"
end opencard
```

角抜きのﾌｨｰﾙﾄﾞ(ﾌｨｰﾙﾄﾞ名:a)ｽｸﾘﾌﾟﾄ

```
on mouseup
  global a
  put 1 into a
  set style of cd fld "a" to shadow
  set style of cd fld "b" to rectangle
end mouseup
```

角抜きのﾌｨｰﾙﾄﾞ(ﾌｨｰﾙﾄﾞ名:b)ｽｸﾘﾌﾟﾄ

```
on mouseup
  global a
  put 2 into a
  set style of cd fld "a" to rectangle
  set style of cd fld "b" to shadow
end mouseup
```

語群(ﾌｨｰﾙﾄﾞ名:ans1)のｽｸﾘﾌﾟﾄ

```
on mouseup
  global a
  if a = 1 then
    put cd fld "ans1" into cd fld "a"
  else if a = 2 then
    put cd fld "ans1" into cd fld "b"
  else
    answer "フィールドをクリックしてから選んでください" with "Ok"
  end if
end mouseup
```

図8. 3-1 語群をクリックして角抜きを埋めるスタック

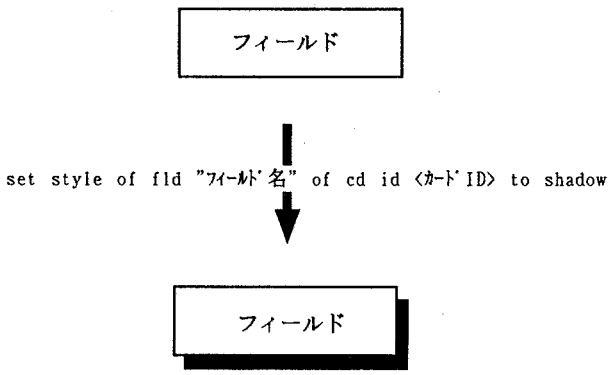


図8. 3-2　フィールドを影付きにする

3)　語群フィールドのスクリプト

語群のフィールド（ans1～ans4）のマウスクリックによるハンドラは、グローバル変数（a）の値を見て、そのフィールドの内容（文字列）をどこへ複写するか決定する。

図の例では、ans1のフィールドがクリックされたとき、変数aの値が"2"ならば、

put cd fld "ans1" into cd fld "b"

の命令で、角抜きのフィールドbへans1のフィールドの内容（"排他的"）が複写される。

8．4　語群からドラッグして角抜きへドロップする手法

語群から適切な語を選ぶ解答形式において、最も直感的なユーザーインターフェースは、語のドラッグ＆ドロップである。これは、選択した語をそのまま文章中の角抜きへ運んで落とす（埋める）ことを意味する。HyperTalkには、ボタン内でマウスが押されている間、ハンドルを可能にするメッセージが用意されている。ここで紹介するスタックは、このメッセージとマウスのボタンが離された時に発せられるメッセージのハンドラを組み合わせて、ドラッグ＆ドロップを実現した。

スタックのレイアウトを、図8. 4-1に示す。

図に示す4つのボタンは、カードが開かれた時点において、あらかじめ決めた場所へ位置づけられる。たとえば、"1960"のボタンはカードスクリプトの、opencardハンドラ内における以下の命令により、ボタンの左上がカード上の（X座標：64、Y座標：266）の位置へ移される。

set the topleft of btn "1960" to 64, 266

カードスクリプト内のグローバル変数（state）は、今現在どのボタンが角抜きの位置に置かれているのかを、0から4の数値で表現する。

次にボタン内のハンドラについて見てみよう。リスト7. 4-1のボタン"1960"のボタンスクリプトは、2つのハンドラで構成されている。

まず一つ目の on mouseUpのハンドラは、マウスがそのボタン内で押され、離された時点において、現在のマウス座標を読み、ボタンを角抜き位置か、語群の既定位置へ置くかを決めている。つまり、ドラッグされたボタンオブジェクトがドロップされた位置が、角抜きのエリア内であれば、ボタンを正確な角抜き位置へ置き、それ以外の場所でドロップされたのであれば、元の語群内の位置へ戻す制御を行っている。ハンドラ内から呼び出されるカードスクリプトの関数（setbtn）は、先に説明したグローバル変数（state）の値から、どのボタンが角抜きのエリア内に置かれたかを判断し、

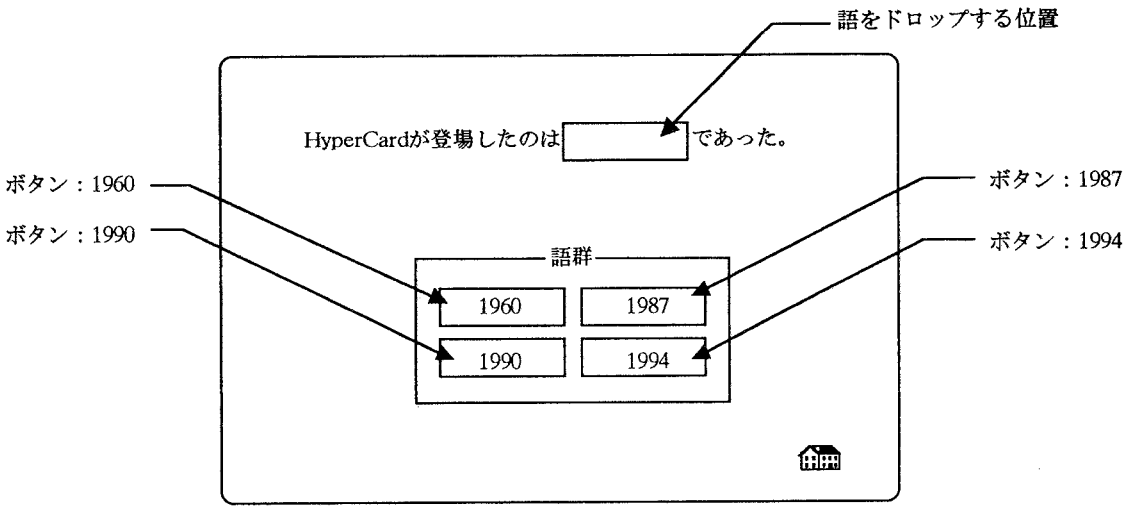


図8. 4-1　語群をドラッグ＆ドロップして角抜きを埋めるスタック

```
on mouseUp
  global status
  put mouseH() into x
  put mouseV() into y
  --角抜き内にマウスポインタが置かれているか?
  if (x >124 and y >127) and (x < 337 and y < 177) then
    --ボタンを角抜き位置へセット
    set the topleft of btn "1960" to 124,127
    put 1 into cnt
    --他のボタンを語群へ戻す関数(setbtn)
    setbtn
  --角抜き以外の場所でドロップされたか?
  else if x <> 171 and  y <> 290 then
    --ボタンを語群の中へ戻す
    set the topleft of btn "1960" to 64,266
    put 0 into cnt
  end if
end mouseUp

on mouseStillDown
  set the loc of me to the mouseLoc
end mouseStillDown
```

リスト8. 4-1 語群ボタンのハンドラ

```
on setbtn
  global state
  if state = 1 then
    --"1960"以外のボタンは語群へ戻す
    set the topleft of btn "1987" to 282,266
    set the topleft of btn "1990" to 64,316
    set the topleft of btn "1994" to 282,316
  else if state = 2 then
    --"1987"以外のボタンは語群へ戻す
    set the topleft of btn "1960" to 64,266
    set the topleft of btn "1990" to 64,316
    set the topleft of btn "1994" to 282,316
  else if state = 3 then
    --"1990"以外のボタンは語群へ戻す
    set the topleft of btn "1960" to 64,266
    set the topleft of btn "1987" to 282,266
    set the topleft of btn "1994" to 282,316
  else if state = 4 then
    --"1994"以外のボタンは語群へ戻す
    set the topleft of btn "1960" to 64,266
    set the topleft of btn "1987" to 282,266
    set the topleft of btn "1990" to 64,316
  end if
end setbtn
```

リスト8. 4-2 setbtn関数

それ以外のボタンを語群位置へ戻すことで、角抜きに複数のボタンが置かれるのを防いでいる。setbtn関数を、リスト8. 4-2に示す。

二つ目の、on mouseStillDownハンドラは、マウスがボタン、フィールド以外で押されている間中、メッセージを繰り返し受け取る。ハンドラ内の

set the loc of me to the mouseLoc

は、オブジェクト（ここではボタン）が、マウスポインタの置かれている位置に追従する（移動する）ようなメッセージを発している。

9 おわりに

PDSが納められた市販のCD-ROMには、必ずといってよいほど教育に関するスタックがいくつか納められている。それらの殆どは米国内で制作され、そのどれからも、制作者の”理解させよう”という熱意が伝わってくる。さすが、MacintoshとHyperCardを産みだした国の教育者たちは、この組み合わせを教材作成に巧く生かしているようで、そのセンスと技法には脱帽するばかりである。

では我国はどうかと見回してみると、国民機の罪なのか、CAIとは名ばかりの、そっけないお粗末なソフトが、”教育用”としてまかり通っているのが現状である。しかし、ここに来て、パーソナルコンピュータの勢力地図は大きく塗り替えられ、国民機の呪縛からのがれた多くの人達が、MacintoshとHyperCardに多くの可能性を見出しはじめているのは、喜ばしいことである。そんな一人である筆者も、通常の授業形態では表現や伝えることが困難なテーマを、スタックに表現し、学生に配布できる日を夢見て、HyperCardの世界へ足を踏み入れたのだが、敷居の低さに較べて、その奥行きは深く、未だ初心者の域を脱していない。

また、HyperCard自体もVersion2.2へと進化し、動画や音を特別なスクリプトを用意することなく扱えるようになったことで、オーサリングツールとしての完成度を高めている。これを利用すれば、実験風景をそのままカード内に映し出したり、それに肉声のコメントを付加したような自習教材の作成が可能になる。しかし、どのようなメディアやツールを使おうとも、利用者の興味を持続させる構成やアイディアは、制作者のセンスとツールに対する習熟度が相乗して生みだされることに変わりない。つまり、”理解させよう”という熱意を、多様な表現手法を駆使して具現化することが、良い教材を生む結果になると考える。

この論文は、初心者である筆者が、教材作成を何とかHpyerCard上で行おうと、四苦八苦している中で、書籍から得た知識や、サンプルとして作ったスタックの一部を紹介したものであり、完成度や完結度が低い

ことは重々承知している。しかし、これから『HyperCardで教材作成を』と考えておられる方たちの、何らかの指針になれば幸いである。論文に対するご意見等があれば、NiftyServe（ID：PXL05572）へメールを頂きたい。

10 参考文献

山口博幸, HyperTalkパワープログラミング　スタック自由自在, 翔泳社, 1993

大谷和利, HyperCardビギナーズガイド, ビジネスアスキー, 1990

掌田津耶乃, 入門HyperCard, アスペクト, 1992

掌田津耶乃, 応用HyperCard, アスペクト, 1991

掌田津耶乃, 実習HyperCard, アスペクト, 1991