ヒューマンインタフェース 34－2
（１９９１．１．１７）

プラガブルMVCに基づくUIMS:Talkie

桑原修二† 藤村茂‡ 富田昭司‡
横河電機株式会社研究開発４部†第２研究室 ‡第１研究室

Smalltalk-80 のアプリケーション作成のパラダイム MVC はプラガブル、依存性など UIMS としての基本的な性質を持っている。

グラフィカルなアプリケーションにおける見た目の情報やアプリケーション本体とのやりとりをするための情報を表すオブジェクトＧを、Ｖ・Ｃに対するＭと同様の位置付けで、MVC に組み入れる。ＭとＧをユーザの用途に応じて切り替えることにより、ユーザは、ユーザインタフェースの操作・編集についても、MVC の利点である対象物とＶ・Ｃとの分離、対話的な環境の恩恵を受けることができる。

本稿では、このＧを導入したMVC(MVCG)とそれに基づくUIMS、Talkieについて述べる。

Talkie
A UIMS Based on Pluggable MVC

Shuji Kuwahara† Shigeru Fujimura‡ Shoji Tomita‡
†Section 2 ‡Section 1 Corporate R&D Department IV
Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi, Tokyo, 180 Japan

MVC(Model-View-Controller), an application development paradigm in Smalltalk-80, has some basic UIMS features, such as 'pluggable' and 'dependency'.

G is an object that represents the look & feel of graphical application and the communication between application object and user-interface. We add G to MVC triad, in which G plays the similar role as M(model). According to purpose, user sees the alternative of M or G. Therefore, in user-interface editing, user takes the advantage of the interactive environment and the separation of the observing object from V-C pair.

This paper discusses MVC-G and Talkie that is a UIMS based on MVCG.

1 はじめに

Smalltalk-80[6][5](以下 Smalltalk と略す) におけるアプリケーション作成のパラダイムである MVC は、

- プラガブル MVC(pluggable MVC)
  いろいろなモデル (アプリケーション本体) に接続できるビュー・コントローラ (ユーザインタフェース) の仕組み。
- 依存性 (dependency)
  モデルとビューを分離しつつモデルの情報の変化をビューに伝える仕組み。

など、UIMS としての基本的な性質を持っている。そのため、Smalltalk 以外の言語・環境においても利用されることが多い。

本稿では、まず、MVC の現状について説明する。そして、ユーザインタフェース作成の観点から MVC について検討を行ない、問題点を述べる。その解決の一方法として MVC の拡張である MVCG を提案し、MVCG に基づく UIMS、Talkie について説明する。最後に、まとめを行なう。

2 MVC

MVC(Model-View-Controller) は、複数のウィンドウが集まってひとつの系をなすようなアプリケーションを作成するための Smalltalk における設計指針である。図 1 に MVC の概念図を示す。

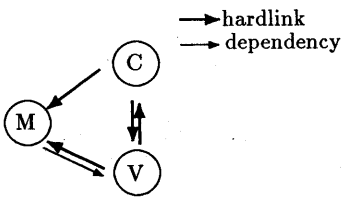

図 1 MVC

MVC 各々のオブジェクトは次のような役割を持つ。

- モデル (Model)
  ユーザの問題の対象となるもの。
- ビュー (View)
  ユーザにモデルを表示するもの。
- コントローラ (Controller)
  ユーザからビューへの入力を解釈するもの。

MVC は、どのようなモデルにも接続できるようなインタフェースの仕組み・一つのモデルを複数のビューから見ているようなアプリケーションの実現のための仕組みを導入したプラガブル MVC(pluggable MVC)[1][8] へと発展してきている。また、モデルの変化をビューに伝えるための仕組みとして依存性 (dependency) を利用している。

2.1 プラガブル MVC

「プラガブル (pluggable)」とは、容易にモデル (アプリケーション本体) に差し込めるようなビュー・コントローラ (ユーザインタフェース) の仕組みのことを示す言葉である。

プラガブル MVC[1] においては、ビューのインスタンスを生成する時に、引数としてモデルのメソッドのメッセージセレクタが渡され、ビューのインスタンス変数に設定される。このメッセージのことをアダプタメッセージ (adaptor message) という [2]。ビューはアダプタメッセージを介してモデルと情報をやりとりする。

例えば、Smalltalk のテキスト表示・編集用のクラスであるプラガブルなビュー TextView のインスタンス生成は次のようなメッセージ式により行なわれる。

```
aTextView := TextView
                on: anObject
                aspect: #text
                change: #acceptText:from:
                menu: #textMenu.
```

ここで、#text、#acceptText:from:、#textMenu がアダプタメッセージのセレクタであり、このクラスメソッド内で呼ばれるインスタンスメソッド on:aspect:change:menu: において、TextView のインスタンス変数 model、partMsg、acceptMsg、

[1] プラガブルとは、本来ビューを形容する言葉であるが、プラガブルなビューはコントローラとペアで使われ、モデルもそのビューにインタフェースを合わせることになるので、MVC を形容する言葉としても使われる。

menuMsg に anObject、#text、#acceptText:from:、#textMenu を代入している。各々のインスタンス変数の意味を詳しく述べると以下の通りである。

- model
  モデルとなるオブジェクト。
- aspectMsg
  モデルが変化したとき表示を変化させるべきかどうかを判断するための識別子(モデルにおいて変化メッセージ changed:の引数として渡される)。また、変化した情報を得るためのメッセージセレクタとしての役割も持つ。
- changeMsg
  ユーザが入力した情報をモデルに知らせるためのメッセージセレクタ。
- menuMsg
  メニューを得るためのメッセージセレクタ。

ビューでは、次のようなメッセージ式により、自身のインスタンス変数に入っているアダプタメッセージをモデルに送り、情報のやりとりを行なう。

```
model perform: partMsg.
```

変数を介してモデルと情報のやりとりをすることにより、モデルによらずビュー(とコントローラ)を部品として使用できる(プラグインできる)ようになっている。

2.2 依存性

依存性(dependency)は、モデルとビュー・コントローラを分離しつつ、モデルの変化をビューに伝えるための仕組みである。モデルがその情報の変化をビューに伝えたいときは、ビューを指定する必要はなく、次に示すように、単に、changed:anAspect というメッセージを自分(self)に送れば良い。

```
self changed: anAspect.
```

メソッド changed:は Object クラスにおいて、

```
changed: aParameter
    self dependents do:
        [:aDependent |
         aDependent update: aParameter]
```

のように定義されており、自分を見ているすべてのビューに対して update:anAspect というメッセージを送る。ビューは、引数として渡された anAspect(変化に関する付帯情報)を頼りにモデルの情報の取り出し、表示の更新を行なう。(クラス Object は、モデルをキーとして、それを見ているビューの集まりが値となるような依存性の辞書 DependentsFields をクラス変数として持っている。モデルとなったオブジェクトは Object のインスタンスメソッド dependents により自分と依存関係にあるビューを知ることができる。また、辞書 DependentsFields への登録は、生成されたビューにモデルを設定するときに行なわれる。)

3 MVC についての検討

3.1 UIMS と MVC

ユーザインタフェース管理システム(UIMS:User Interface Management System)は、ユーザインタフェースとアプリケーションの本体を分離し、ユーザインタフェースの部分を、アプリケーションとは別の専用のソフトウェアによって管理しようとするシステムである[12][7]。

UIMS の部品オブジェクトの構成のモデルとして、あるいは、アプリケーション本体とユーザインタフェースとを分離するためのモデルとして MVC(Model-View-Controller)を採用しているものが見られる。

例えば、鼎[13]では、編集対象(メディアと呼んでいる)の基本的な編集機能を提供する部分(エディタと呼んでいる)の構成のモデルとして、MVC に基づくものを採用している。ひとつのモデル(モデルはメディアを見ている)に複数の・異なったビューを設定することが容易になったと報告している。

また、InterfaceBuilder[3] や GUIDE[10] において、ユーザインタフェースの部品に対して、アプリケーション本体に送るメッセージのセレクタを設定することが行なわれる。これは、Smalltalk において、プラガブルなビューにアダプタメッセージのセレクタを設定するのと同様の考え方に基づくものといえる。

3.2 ユーザインタフェースの作成

MVC を用いることにより、Smalltalk ではモデル(アプリケーション本体)の操作・編集の環境としては、視覚的・対話的な環境を提供している。

しかし、MVC には InterfaceBuilder や GUIDE のように、視覚的・対話的にユーザインタフェースの作成を行なうための環境は用意されていない。また、アプリケーション使用時にユーザインタフェースをこのような環境で編集するための枠組もない。

3.3 アプリケーション本体の情報のユーザインタフェースへの反映

ユーザインタフェースとアプリケーション本体を分離しつつ、ユーザインタフェースにアプリケーションの構造を反映させるための手段として以下のものがある [7][14]。

- notification
- daemon
- active value

notification は直接(direct) と間接(indirect) に分けられる。例えば、MVC の依存性 (dependency) は間接の notification であり、InterfaceBuilder では (outlet リンクと呼ばれる) 直接の notification が用いられている。

notification の欠点として、アプリケーション本体にとって本質的でない変化メッセージを、明示的に書かなければならないことがある。とくに直接の notification の場合は、アプリケーション本体の中に直接ユーザインタフェースのメソッドを起動するメッセージを書くことになるので、ユーザインタフェースとアプリケーションの結び付きの度合は大きくなる。一方、間接の notification である Smalltalk の MVC の依存性では、変化メッセージは、アプリケーションは変化したことを自分に対していうだけで済むようになっているので、ユーザインタフェースとの結び付きは比較的小さくなっている。ただし、それに起因する、以下に示すような欠点も指摘されている [9]。

- 依存性の範囲が不明確
  モデルの変化メッセージに応答するビューの更新メッセージが、ビュー毎に分散・埋没してしまっている。
- 依存性の対象が不明確
  モデルは更新させたいビューを指定できず、更新メッセージを送られたビューが自分を更新させるべきか判断しなければならない。

daemon と active value は変化メッセージをプログラム中に明示的に書かなくて済む方法であり、村田らも上の欠点を解決する手段として、Encapsulator[11] に基づく daemon を Smalltalk に導入している。Encapsulator で関係のあるモデルとビューを包み、モデル/ビューに従来記述されていた変化/更新メッセージを after daemon として Encapsulator に記述することにより、上の欠点の改善を図っている。このような考えを進め、インスタンス間の関係の記述のために、ThingLab[4] のように制約(constraint) を導入する試みもある。

一方、依存性の問題点は問題点として認め、ユーザがアプリケーション作成の際感じる、その問題点に起因する負担を少しでも軽減しようとする考え方もある。例えば、Glazier[2] は、アダプタメッセージにより起動されるメソッドや変化メッセージの記述をアプリケーション本体の代わりにマニピュレータ (manipulator) というオブジェクトに担当させ、このマニピュレータを生成することにより、MVC について詳しくないユーザにもアプリケーションの作成を可能にすることを試みている。

4 MVCG と Talkie

4.1 MVCG

あるパラダイムが推奨されるプログラミング言語・環境においては、そのパラダイムを知り、活用することが肝要である。(その背景として、従来のプラガブル MVC で作られたクラス群の再利用をしたいという現実的な希望がある。)

このような考えに基づき、前章で述べたような、従来の MVC の枠組 (プラガブル MVC、依存性) に沿いつつ、対話的・視覚的にユーザインタフェースを作成するための方法を検討する。

4.1.1 MVCGの導入

あらかじめ用意されたユーザインタフェースの部品群(ライブラリ)を利用してアプリケーションのユーザインタフェースを作成するシステムでは、ユーザインタフェースの対話的な編集の対象となるのは、

- 見た目の情報
- アプリケーション本体と情報をやりとりするための情報

など簡単な属性である場合が多い。(ここでは、これらの情報を「グラフィカルな情報」と呼ぶことにする。)一方、表示の仕方・入力の受け付け方を変えるなどの比較的大きな変更は、例えば、サブクラッシングなどで、別の部品を実現することにより対処するのが、部品化の観点から好ましい。

MVCに照らし合わせると、ビュー・コントローラ自体を編集するのではなく、その一部分の情報を編集するということになる。グラフィカルなアプリケーションにおけるグラフィカルな情報の重要性とそれらの情報を編集可能な対象物としてとらえることの必要性とを考慮すると、この情報を一つのオブジェクトとして独立させ、モジュラリティを高めることが望まれる。

このような考えにより、我々は、グラフィカルな情報を表現するオブジェクトG(Graphical modelの略)をMVCに導入した。Gの持つ情報の主なものを以下に示す。

- 識別子
- 見た目
  画面でそのGの占める領域、境界の幅、色、内部の色、表示される絵、文字など。
- アプリケーションの情報
  ビューに設定するための、モデル(アプリケーション本体)のインスタンスを得るためのメッセージ式や、モデルの情報取り出し・設定のためにモデルに送れば良いメッセージ(アダプタメッセージ)のセレクタ。
- 自分の上に貼りつけられているG
- 自分の持つ情報を編集するためのメソッド群
- スクリプト
  ある状況で起動されるメッセージ式。

Gを組み入れたMVCであるMVCGを図2に示す。

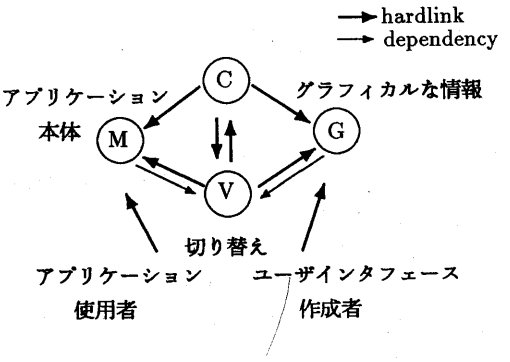

図2 MVCG

Gはビュー・コントローラに対して、モデルと同様の位置付けをする。すなわち、Gはビューとコントローラのインスタンス変数gModelに入れられ、Gの依存性の辞書にはビューが登録される。Gの生成は、ユーザがユーザインタフェースの部品を対話的な環境で作るときに行なわれる(ビュー・コントローラとともに生成される)。

ユーザは、ビュー・コントローラを通じてモデルおよびGを見る(操作・編集を行なう)。ただし、ユーザからモデルとGは同時には見えず、ユーザの用途に応じて、適切に切り替えられようにする。つまり、操作・編集の対象のオブジェクトとして、ユーザインタフェース作成時にはGを、アプリケーション使用時にはモデルをユーザに見せるようにする。

ユーザインタフェース作成時に、上に示したGの情報を変更すると、それは依存性によりビューに反映される。例えば、「アプリケーションの情報」に設定が行なわれると、ビューに対して新たにモデルが設定される。

MVCGにより、ユーザインタフェースの操作・編集についても、対象の明確化や対話的な環境といったMVCの利点を受けることが可能となる。

4.1.2 作成の支援

ものとして直接見えるものに対しては、直接的に操作できること(direct manipulatinであること)が

望ましい。ユーザインタフェースの作成を以下に示すような作業に分ける。

見た目の作成 ユーザインタフェースの見た目の作成(配置など)

プラグイン ユーザインタフェースのアプリケーション本体へのプラグイン

依存性の定義 アプリケーション本体とユーザインタフェース間の依存性の定義

そして、各々に対して、以下のような直接的な操作環境を提供する。

見た目の作成は、お絵書きツールの要領で行なえるようにする。内部的には、入力イベントの伝搬の順序などを決める部品の階層関係を自動的に作れるようにする。

プラグインは、アプリケーション本体を得るための情報と、アプリケーション本体と情報をやりとりするための情報を穴埋めの要領で行なえるようにする。内部的には、ユーザインタフェースがアプリケーション本体にプラグインされるようにする。

依存性の定義については、依存性/変化メッセージを視覚的に表現することを試みる。変化メッセージの発生・伝搬過程を追ってみると、

- ある部品に操作を加える。
- その操作がモデルへのメッセージとなり、モデルの中のメソッドを起動する。
- モデルが(特定はできないが)ある部品に変化メッセージを送る。

となる。これは、始点と終点だけを見れば、ある部品からある部品へのメッセージとみなせる。そこで、画面のある部品からある部品への矢印を引くことでこの依存性を表現できるようにする。

4.1.3 理解・設計の支援

UIMS の基礎となるパラダイムとして、オブジェクト指向を採用する場合が多い。オブジェクト指向言語を用いたプログラミングには、既存クラスの再利用・プロトタイピングが可能という性質から探索型のプログラミングスタイルが向いており、そのスタイルには次の 5 つの段階が互いに関わり合いながら存在している [16]。

- 理解・設計
  問題について、分析・既存クラス群から利用できるクラスを探す段階。そして、何をオブジェクトとして表現するか・クラス階層をどう構築するかのモデリングの段階。
- 入力・実行・デバッグ
  実際にプログラミング言語を用いて、問題のモデルを表現・実行する段階。

例えば、Smalltalk においては、ワークスペース、システムブラウザ、インスペクタ、デバッガなどのツールが用意されており、探索型のプログラミングを支援している。

我々は、プラガブル MVC のプログラミングにおいて、とくに理解・設計の段階において、既存のクラスおよびオブジェクトについてユーザが知りたい情報としては、以下のものがあると考えている。

- MVC はそれぞれどのクラスか?
- ビューはモデルに対してどんなメッセージを送るか?
- 依存関係は?

従来のブラウザ、インスペクタ、デバッガなどによりこれらの情報を覗き見ることはできる。しかし、これらのツールを使用して、画面上に直接見えるものに対して情報を知りたい場合、ユーザには、どちらかといえば感覚的でなく論理的な思考・操作が求められる。しかし、既存の UIMS には、従来プログラミング言語を用いて行なう必要のあった入力・実行の段階の作業をプログラミングなしで行なうことを可能にすることを目的とするものはある。しかし、理解・設計の段階に対して有効な方法を提供するシステムは見られない。

よって、ユーザインタフェースの作成のための直接的な操作感だけでなく、アプリケーションを作成するにあたっての

理解・設計

の段階において、直接的な操作感を実現するようにする。

4.2 Talkie

Talkie[2]は前節で述べたMVCGに基づくSmalltalk上のユーザインタフェースの作成のためのシステムである。また、UIMSとしての機能だけではなく、情報の媒体、検索、提示のためのシステムとして注目されているハイパーメディア的な機能(ユーザレベル、スクリプト、情報単位であるカードとカードのリンク等の機能)を導入している。

4.2.1 Talkieによるユーザインタフェース作成

Talkieのユーザインタフェースの部品には、カード、ダイアログボックス、ボタン、スイッチ、ラジオボタン、チェックボックス、テキスト、コード、リスト、グラフ、ゲージなどがある。各々は、前述したGおよびビュー・コントローラにより構成される。

Talkieで支援するアプリケーション作成の作業は、前節であげた以下のものである。

見た目の作成

プラグイン

依存性の定義

理解・設計

なお、ユーザの用途に応じてユーザの操作・編集の対象(MとG)を変えるユーザレベルを導入している。ユーザレベルには、オーサリング(authoring)とブラウジング(browsing)の2つがあり、各々の状態でユーザはGとMを見ることになる。上の作業は主にオーサリングで行なう。(一部、ブラウジングでも可能である。)また、アプリケーションを使用するときはブラウジングにする。(ユーザレベルに関わらずアプリケーションとして動作させることも可能である。)

以下では、具体例として、電卓を模したアプリケーションの作成をあげ、上の各項目とその他の機能について説明する。

見た目の作成

この作業はお絵書きツールの要領で行える。具体的には、カード(ひとまとまりアプリケーションを実現するためのユーザインタフェースの部品)上にカード以外の部品を適当に配置したり、見た目の属性の変更をしたりすることにより行なう。

図3において、上部の「Talkieです」というラベルのビューはTalkieのツールを集めたカードである。このカードから、左下のTalkieの部品を収めた部品カードを開く。このカードから、ユーザは部品をドラッグしてカードの上に貼りつける。ここでは、右下のカードにテキストを貼りつけようとしている。

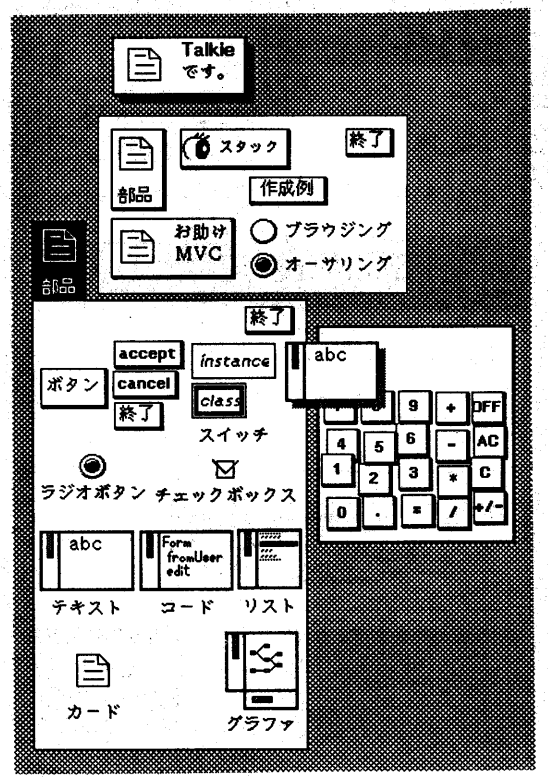

図3 ユーザインタフェースの見た目の作成

プラグイン

この作業は、穴埋め(フィルインザブランク)の要領で行える。具体的には、ユーザインタフェースの部品の上でプラグインをするためのダイアログボックス(ユーザとの対話用のカード)を開き、アプリケーション本体と接続するための情報の設定を行なう。このような操作環境を提供しつつ、内部的には、ユーザインタフェースとアプリケーション本体は従来のSmalltalkの依存性とプラガブルMVCに従う形態となっている。

[2]Talkieという名前には、「Smalltalk(Talk-)のパラダイムに沿いつつ、親しみやすい(-ie)インタフェースを目指す。さらに音声などのメディアも扱おう。」という意味が込められている。

図４において、上の「お助けMVC」というラベルのカードは、プラグインや依存性の定義を行なうためのツールを収めたカードである。ここでは、プラグインツールを起動して、下のダイアログボックスで、電卓の「５」というボタンに関する設定を行なっているところである。「インスタンスを得るメッセージ式」という欄のdefaultとは、スーパビューと同じインスタンスを使用することを表している。

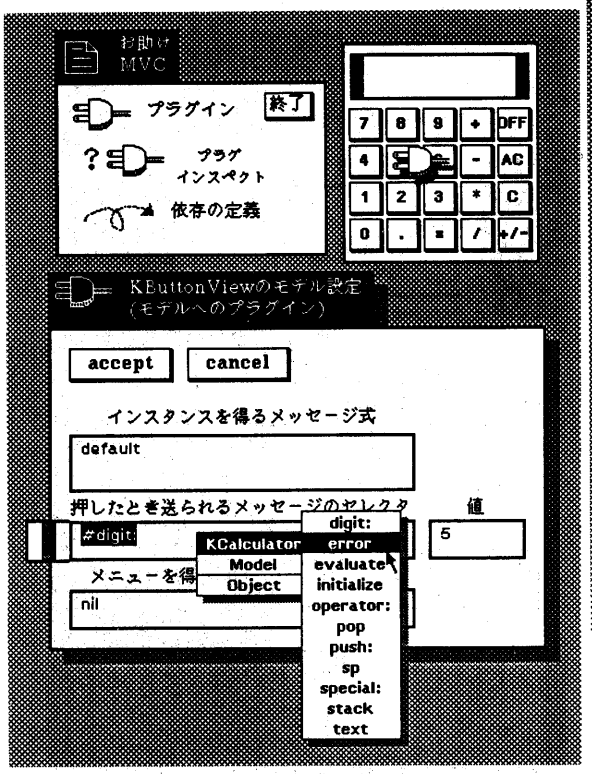

図４ ユーザインタフェースをアプリケーション本体へプラグイン

依存性の定義

この作業は、依存関係を持たせたい画面上の部品から部品へ矢印を引くことにより行なう。

図５は「お助けMVC」カードの依存定義ツールを起動して、電卓の「５」というボタンとテキストの依存性を定義しているところである。画面下のダイアログボックスには、矢印の始点のボタンに(プラグインツールで)設定された、「アプリケーション本体への情報入力のためのメッセージセレクタ」に対応するアプリケーション本体のメソッドが表示されている。「changed追加」というボタンを押すことにより、適切な変化メッセージがメソッドに挿入される。なお、変化メッセージの引数は、矢印の終点の部品の「アプリケーション本体から情報を得るためのメッセージセレクタ」が使われる。

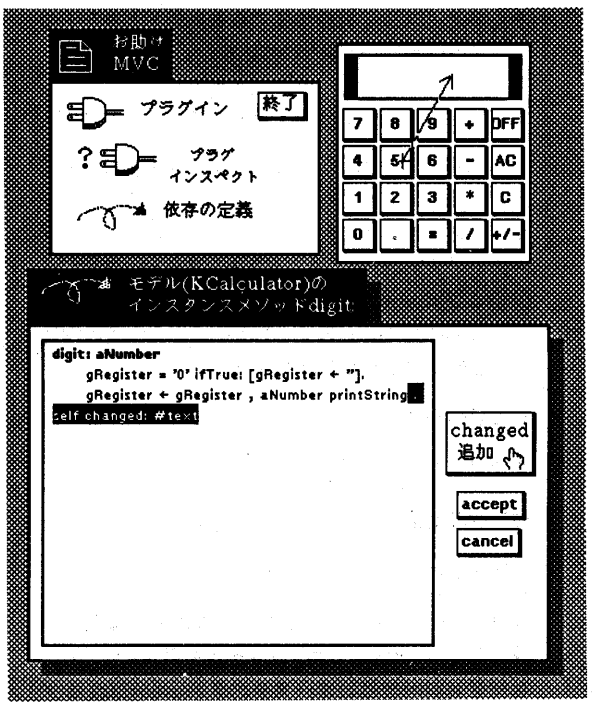

図５ 依存性の定義

理解・設計

この作業を支援するために、プラグインスペクタと呼ぶツールを用意している。

例えば、図６はプラグインスペクタにより、画面上のシステムブラウザを覗いている様子を示す

その他

スタックブラウザと呼ぶカード群を保持するスタックを操作するツールがある。カード間のリンクをグラフィカルに見せるなどの機能がある。

各ユーザインタフェース部品のGは自分を再生するSmalltalkのソースコードを出力することができる。そして、カード単位、またはスタック単位でのファイル入出力機能を持つ。好きな時点でファイ

ル入出力できるので、アプリケーション本体を完成させる前にユーザインタフェース部分だけを先に作っておくことも可能である。従来のプラガブルなビューを生成するメッセージ式と似た形式で出力されるので、従来の Smalltalk のプログラミング環境に親しんでいるユーザにも馴染みやすいと考えられる。

4.2.2 作成経験からの評価

ある既存のアプリケーション本体のユーザインタフェースを作成する場合、アダプタメッセージのセレクタさえわかれば、そのメッセージにより起動されるメソッドや、そのアプリケーション本体のクラスの詳細を知ることなしにユーザインタフェースの作成が可能である。例えば、Talkie の使い方とプラガブル MVC については理解しているが、Browser クラスの詳細は知らない筆者のうちの一人が、Browser クラスの調査を含み 30 分程度の時間で、Browser クラスをそのまま変更を加えず使用して、システムブラウザと機能的に同じものを作成可能であった。(図 6)

このように、Talkie の部品で表現できる機能からなる、プラガブル MVC を利用して作成されている Smalltalk のアプリケーションならば、そのモデルのクラスをそのまま再利用して、Smalltalk のプログラミングをすることなしに、同機能のユーザインタフェースを Talkie により実現できる。

一方、既存のクラスでなく新たにアプリケーション本体のクラスを作成する場合(例えば、前章の電卓など)も、ユーザインタフェースの見た目の作成は短時間で済む。

大規模な、アプリケーション例としては、知的システム AUK の開発環境である HyperAuk[15] や工程計画支援システムが筆者らにより作成されている。

いずれのアプリケーションを作成する場合も、ユーザインタフェースの部分については、Smalltalk のプログラミングはほとんどすることなしに済むが、アプリケーションのプラグイン・依存性の定義などの操作には、MVC の概念の理解が必要であった。反面、Talkie は MVC の学習にも役立つようである。

なお、Talkie の各ツールのユーザインタフェースも Talkie 自身を使用して編集が可能である。

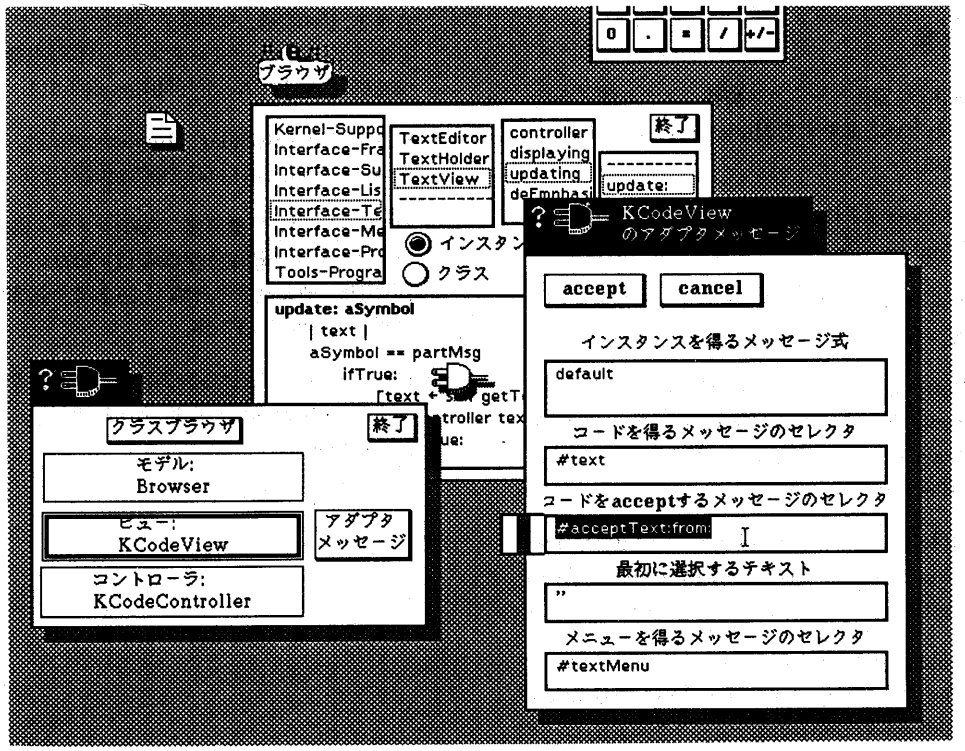

図 6 既存のアプリケーションの理解

5 おわりに

グラフィカルなアプリケーションにおける見た目の情報やアプリケーション本体とのやりとりをするための情報を表すオブジェクトGを、V・Cに対するMと同様の位置付けで、MVCに組み入れた。MとGをユーザの用途に応じて切り替えることにより、ユーザは、ユーザインタフェースの操作・編集についても、MVCの利点である対象物とV・Cとの分離、対話的な環境の恩恵を受けることができるようになった。本稿では、このGを導入したMVC(MVCG)とそれに基づくUIMS、Talkieについて述べた。ユーザインタフェースの作成時の支援だけでなく、理解・設計の支援を行なうことも検討した。

現状では、部品の再利用を前提としているため、ユーザインタフェースで編集の対象となる情報を比較的簡単な属性(グラフィカルなアプリケーションにおける見た目の情報やアプリケーション本体とのやりとりをするための情報)に限っている。新たな部品の作成のための本格的な支援が、今後の課題である。

参考文献

[1] 青木淳, 富士ゼロックス情報システム株式会社: 使わないと損をする Model-View-Controller, 1988.

[2] Alexander.J.H: Painless Panes for Smalltalk Windows, Proc.OOPSLA'87,pp.287-294.

[3] キャノン株式会社:NeXT ユーザーズガイドV1.0版, 1989.

[4] Borning,A.,Duisberg,R.: Constraint-Based Tools for Building User Interfaces, ACM Transactions on Graphics,Vol.5,No.4 (1986),pp.345-374.

[5] Goldberg,A.: Smalltalk-80, The Interactive Programming Environment,Addison Wesley,1984.

[6] Goldberg,A. and D.Robson: Smalltalk-80, The Language and its Implementation, Addison-Wesley,1983.

[7] 萩谷昌己: グラフィカルなユーザインタフェースとその開発環境について, bit,Vol21, No.6(1989),pp.51-62.

[8] Lalonde,W.R.,Pugh,R.P.: Inside Smalltalk Volume II,Prentice-Hall inc.,1991.

[9] 村田真, 楠本浩二, 上林憲行:Smalltalk-80 へのフィールドとデーモンの導入とその有効性,WOOC'87,1987.

[10] 大谷浩司, 角野宏司, 児島彰, 萩谷昌己, 服部隆志, 劉樹苓: GMW ウィンドウ・システム上のアプリケーション構築について, コンピュータソフトウェア, Vol7,No.1(1990),pp.45-60.

[11] Pascoe,G.A.:Encapsulators: A New Software Paradigm in Smalltalk-80, Proc. OOPSLA'86,pp.341-346.

[12] Pffaf,G.E.: User Interface Management Systems, Springer-Verlag,1985.

[13] 暦本純一, 垂水浩幸, 菅井勝, 山崎剛, 猪狩錦光, 森岳志, 杉山高弘, 内山厚子, 秋口忠三:エディタを部品としたユーザインタフェース構築基盤:鼎, 情報処理, Vol31,No.5(1990),pp.602-611.

[14] Szekely,P.A.:Modular Implementation of Presentation, CHI+GI 1987,pp.271-278.

[15] 富田昭司, 藤村茂, 桑原修二:知的システム構築用シェル AUK の構築支援環境 (2) 知的システム構築環境, 情報処理学会第 41 回全国大会講演論文集 (2),1990,pp.17-18.

[16] 横手靖彦:オブジェクト指向言語のプログラミング環境, 情報処理,Vol.30, No.4(1989), pp.334-346.