アイコンを用いた作業空間の実現

山口真悟*・西村世志人**・田中稔**

Realization of Iconic Workspace

Shingo YAMAGUCHI, Yoshihito NISHIMURA and Minoru TANAKA

Abstract

Iconic programming systems provide end-users effective application environments. However, most of them don't support constructing environment themselves.This paper discusses design principles and implementational issues of an object-oriented iconic system with emphasis on maintainability and extensibility. The system is defined by a set of icons, that users can easily extend. Each icon consists of resources, which are objects of smaller granularity such as string, bitmap, and so on. Based on this framework, we have been developing an interactive environment for construction of application iconic systems.

1．はじめに

コンピュータの普及に伴いエンドユーザプログラミングの考え方が支持されるようになり，ユーザフレンドリなプログラミングシステムへの需要が高まっている．これを得る一つのアプローチとして，マンマシーンインターフェイスにおいて視覚的情報の有用性を利用した視覚的プログラミングシステムが研究されている［1］.

我々はアイコニックプログラミングシステム(iconic programming system，以下IPSと略す)［2］［3］の実行環境と構築環境の開発を行なっている．IPSは，各アイコンが保持する絵シンボルとポインティングデバイスを用いたアイコンの空間的操作により，容易にプログラミングを行なえる．本研究で提案するIPS「アイコンシステム」は，アイコンを実世界のメタファ(metaphor)としてとらえ，それらの重ね合わせ操作を実世界におけるユーザの作業に対応づけることにより，操作性に優れたユーザインターフェイスを提供している［4］.

また，これまで提案されたIPSの多くは，エンドユーザに有効なプログラミング環境を提供するが，異なるアプリケーションを実現するための有効な支援環境を伴っていない．この問題を解決するため我々は，保守管理性及び拡張性を重視したシステム構成を研究している．アイコンシステムの機能は与えられたアイコンの集合によって定まり，したがってシステムの拡張はアイコンの追加定義によって実現される．拡張性を高めるためにアイコンを構成する粒度の小さいオブジェクトであるリソース（resource）を導入した.

以下，2．ではIPS「アイコンシステム」の実行環境について述べ，本システムにより実現されたアプリケーション例を示す．3．ではシステムの構築支援モデルと，その機構を実現するリソースについて述べる．最後に4．で結論と今後の課題について述べる.

*知能情報システム工学専攻
**知能情報システム工学科

2．アイコンシステム実行環境

2.1 アイコンの定義

本研究で定義するアイコンは，ユーザの直観的な理解を促す絵シンボルと，複数の属性及び機能をカプセル化したオブジェクトであり，それらは実世界に存在する“もの”のメタファとして設計される．

実在の“もの”を普遍的なオペレータあるいはオペランドに区別することができないのと同様に，アイコンもその両方の性質を備えている．それぞれのアイコンの振舞いは複数のものが可能であり，その中から適切なものが実行時に，他のアイコンと重ね合わせた時に動的に決定される．図1にアイコンの振舞いの例を示す．

ここでは紙のメタファである“ペーパ”アイコンを異なるアイコンと組み合わせることにより，異なる機能が実行される例を示す．“ペーパ”アイコンと，“ルーペ”アイコン，“ポスト”アイコン，“ペン”アイコン，“ペーパ”アイコンを重ね合わせた時，それぞれ見る(view)，送る (send)，書く (write)，併合する (merge) 機能が起動される．

このようなアイコンの振舞いを実現するために，各アイコンは交信可能なメッセージのリストを保持する．メッセージリストにはactiveとpassiveの二つがあり，passiveメッセージはそのアイコンがもつ機能に対応し，activeメッセージは他のアイコンの機能実行のトリガとなる．

ユーザが二つのアイコンを重ね合わせると，各々のアイコンがもつactiveメッセージを交換し，自身のpassiveメッセージと比較する．ここでマッチしたメッセージが二つのアイコン間で実行可能な機能であり，それが実行される．ここで二つ以上のメッセージがマッチした場合には実行可能な機能が表示され，ユーザがそのうちの一つを選択することにより所望の機能を実行できる．このような機構はオブジェクト指向技法の多相性 (polymorphism) に基づいており，全てのアイコンはメッセージに対して統一されたインターフェイスを有する．

図2に“ペーパ”アイコンと“ペン”アイコンを重ね合わせた場合の例を示す．ここでは“write”が実行可能なメッセージとして選択されている．

またユーザは適時にアイコンを作成 (create) あるいは消去 (remove) することができる．ユーザは，システムにより与えられた抽象クラスからデフォルト値をもつアイコンを導出し，それをカスタマイズすることにより新しいアイコンを作成する．抽象クラスについては3.2で詳述する．ユーザは本システムが提供する編集機能を利用して，必要なスロットを埋めることにより所望のアイコンを容易に作成できる．

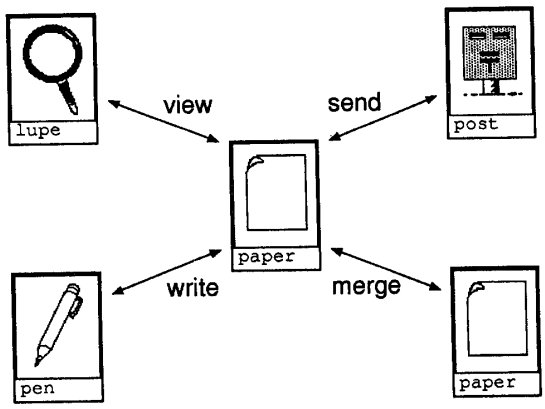

図1：アイコンの振舞いの例

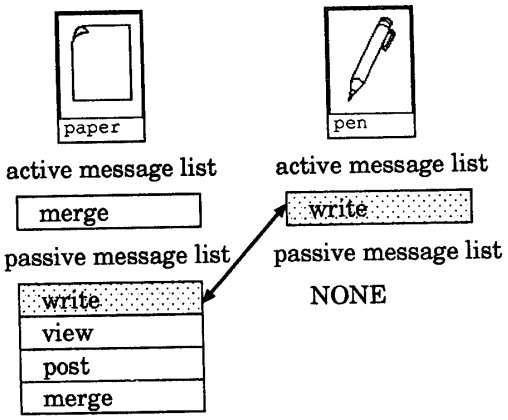

図2：メッセージパッシングによる機能選択

2.2 アイコンシステム

アイコンシステムは2.1で定義したアイコンを用いて，ユーザの作業空間を模擬するプログラミングシステムである．本システムでは，ユーザが試行錯誤しながらアイコンを対話的に重ね合わせることにより，実世界の作業の「手順」を視覚的に遂行することをプログラミングと定義する．

アイコンシステムにおける作業の基本単位は式(1)で定義され，これは「icon Aとicon Bを用いて作業した結果icon Cがつくられる」ということを意味する．

$$\text{icon A} \circ \text{icon B} \rightarrow \text{icon C} \qquad (1)$$

ここで○は実行される機能を示す．式(1)中のiconCは作業結果を保持し，そのアイコンは次の作業に使用できる．この極めて簡単な規則を理解すれば，ユーザは複雑なプログラミング知識を必要とせず，処理している問題の解決に集中できる．

以上のようなアプローチを実現するアイコンシステムの実行画面を図3に示す．アイコンは**IconTableウインドウ**（図3左）に配置されている．そこからユーザは作業空間である**WorkBenchウインドウ**(図3右)にアイコンをドラッグし，他のアイコンと重ね合わせることによって作業を行なう．図3では，**WorkBenchウインドウ**にある“ペン”アイコンに“ノート”アイコン(アイコン名“Abstract”）を重ね合わせ，ノートに記入する作業を行なっている様子を示す．ここで“ノート”アイコンの属性値を編集するためのウインドウ（図3右下）は，アイコンの機能により作成される．ユーザが必要な記入を終え**exitボタン**を押すと，作業の結果を保持する新しいアイコンが作成される．

このような処理を操り返すことによって一連の処理を実行する．

2.3　アプリケーションの動作例

ここでは，アイコンシステム上に構築されたアプリケーションの動作例を示す．「表を作成し，その表をグラフに作図する」一連の処理を実行する．

この作業では**IconTableウインドウ**に表示されている図4に示すアイコンを使用する．

1．“表”の作成

新たに表を作成する場合には，“表”抽象クラスからデフォルトの値を有する“表”アイコンを作成し，図5に示す編集ウインドウを用いてカスタマイズを行なう．行，列の名前や数値を記入した後，編集ウインドウ中の**Generateボタン**をクリックする．この結果，記入した値を保持した“表”アイコンを得ることができる．

既に“表”アイコンが存在する場合には，既存の“表”アイコンに“ペン”アイコンを重ね合わせ，表中の数値を修正する．この結果，修正された値をもつ“表”アイコンが得られる（図6参照）．

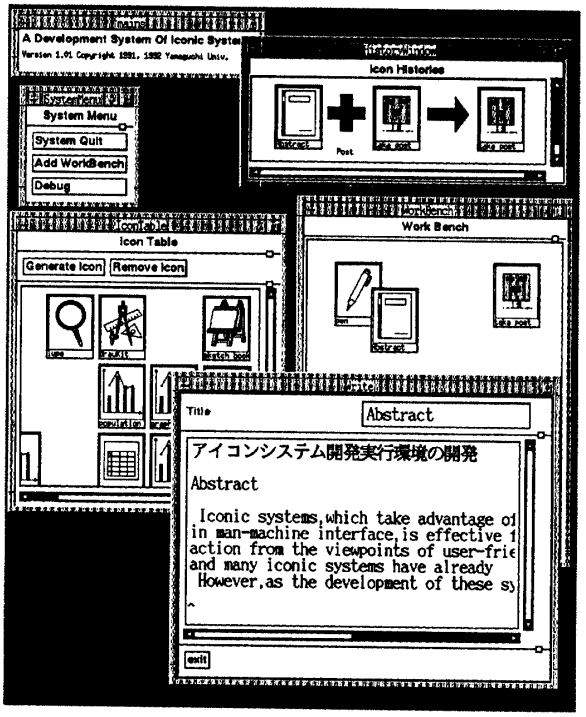

図3：アイコンシステム実行画面

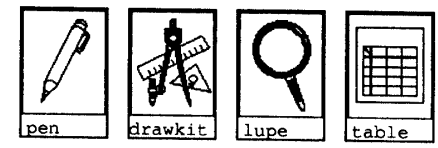

図4：アプリケーションで用いられるアイコン群

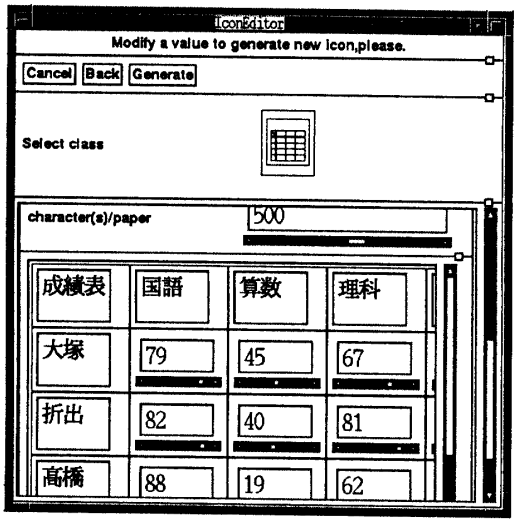

図5：“表”アイコンの編集画面

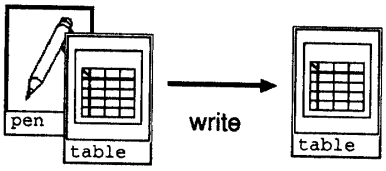

図6：“表”アイコンの作成

2．“グラフ”の作成

“表”アイコンと“ドローキット”アイコンを重ねて“graph”メッセージを選択することにより，グラフを保持する“グラフ”アイコンを作成する（図7参照）．

3．“グラフ”の表示

“グラフ”アイコンと“ルーペ”アイコンを重ねて“view”メッセージを選択することにより，“グラフ”アイコンが保持するデータ，すなわちグラフを参照できる（図8，図9参照）．

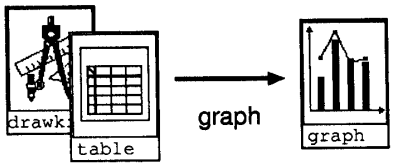

図7：“グラフ”アイコンの作成

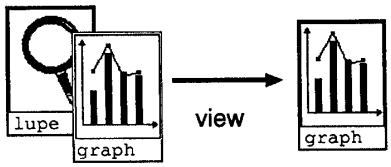

図8：“グラフ”アイコンの表示

3．アイコンシステム構築環境

3.1 構築環境

アイコンシステム構築環境の構成階層を図10に示す．アイコンシステムは任意の個数のアイコンと有限個の抽象クラス，そしてそれらを制御するカーネルからなる．本システムはエンドユーザとシステムデザイナの二つのユーザレベルを支援の対象としている．

エンドユーザは2．で述べたアイコンシステムを利用するユーザである．エンドユーザはシステムデザイナにより提供される抽象クラスからアイコンを作成し，処理を行なう．

しかしエンドユーザがアイコンシステムのコンセプトと異なる処理を要求した場合，その処理はアイコンシステムに拒否されるか，またはユーザの意図と異なった処理が行なわれる可能性がある．あるアイコンに対してシステムデザイナが定義した意味と，エンドユーザがそのアイコンから受け取る意味が相違する問題の解決は重要であり，今後の課題として残されている．

一方，本システムにおけるシステムの拡張は，システムデザイナによる抽象クラスの追加定義により行なわれる．

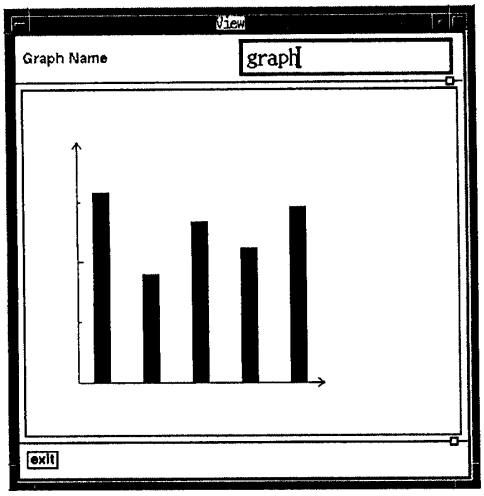

図9：グラフ表示画面

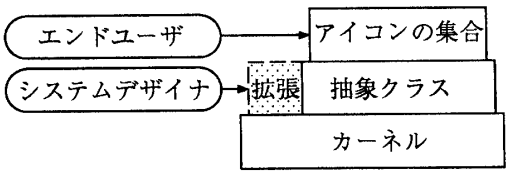

図10：アイコンシステム構築環境の構成階層

3.2 抽象クラス

抽象クラスはアイコンの共通な仕様を保持する型枠であり，抽象クラスの集合はアイコンシステムのコンセプトを定義する．抽象クラスはカーネルから完全に分離されており，インスタンスであるアイコンの作用はメッセージパッシングにより実現されるため，システムデザイナは単に抽象クラスを追加定義することによりアイコンシステムを容易に拡張することができる．

複数の抽象クラスの実装/設計に対して大きな二つのスタイル――一方は複数の独立したクラスを集める「森林的(forest)」アプローチであり，他方はSmalltalkのような完全に階層化されたアプローチ[6]である――のいずれかを選択する必要がある．我々は[4]で考察を行なった結果，唯一の親をもつフラットな構造を採用している．これはアイコンが実世界の“もの”のメタファという大きな粒度であり，実世界の“もの”に対して普遍的なクラス階層を決定することの難しさと，システムデザイナの捉え方によってクラス階層が無数に

存在しうることを考慮した結果である．

クラス階層導入の目的には実装効率の向上と保守管理の向上の二つを挙げることができる．以前我々は実装効率がオブジェクトの類似性を判断する基準と考え，そこから設計したクラス構造が同時に容易な保守管理と拡張を提供できると仮定した．しかし実装効率上の継承関係は必ずしもユーザの直観に一致するとは限らない．したがって先の目的を達成するために，各々異なるアプローチを用いる必要があろう．本研究では実装効率のために，継承ではなく必要な（場合によっては複数の）クラスの部分集合を利用するアプローチを採用した．このアプローチについては3.3で詳述する．一方保守管理に対するアプローチは今後の課題として残されているが，エンドユーザの要求に迅速に対応できるように，抽象クラスの“意味的な集まり”を定義する必要があろう［4］．

3.3　リソース

我々はアイコンをより小さな粒度のリソースから成る複合オブジェクトとして実現する方法を提案している［4］．

アイコンが実世界の“もの”のメタファとしてユーザの直観的な理解を目的としているのに対し，リソースはアイコンの設計及び実装の支援を主な目的としている．あらかじめ定義された（任意のアイコンあるいはカーネルとの）メッセージパッシングによるアイコンの動作を保障するために，抽象クラスには一定のインターフェイスが要求される．システムデザイナが，これらの機構を含む適切なクラスを設計することは容易ではなく，またシステムデザイナの興味はそのクラスで実現する属性と機能にある．したがってこのようなインターフェイスの記述はシステム内部に隠蔽されるべきである．

また先に述べたように本システムの抽象クラスは非階層構造であるため，継承を用いたクラスの実装は考えていない．しかし既に実現されたクラスの属性とそれに付随する機能の再利用は当然要求されるであろう．

このような問題を解決するために，インターフェイス部分を内部に隠蔽し，またそれ自身が再利用可能な汎用オブジェクトであるリソースを定義する．システムデザイナは，リソースを利用することにより構文的な整合性の問題から開放され，定義するクラスの意味的な実装に集中することができる．

リソースは次のように定義される．

リソース：＝

　リソース名：{**データ**，**メソッド**}

リソースはデータそしてメソッドをカプセル化したオブジェクトである．リソース名はその抽象クラス内で有効な，局所的な[1]タグであり，クラス定義時に与えられる．したがってシステムデザイナは，同一のリソースクラスから異なる意味を持つリソースを作成できる（図11参照）．

ここでは文字列型リソースから異なる意味をもつ“題目”と“文章”のリソースを導出し，“論文”抽象クラスを構築している．リソースクラスの集合はシステムによりあらかじめ提供されており，システムデザイナはそれらを適切に組み合せて，アプリケーションで必要とされる新しい抽象クラスを定義する．またリソース名はエンドユーザへのプロンプトとしても用いられる．これについては3.6で述べる．

3.4　複合オブジェクトとしてのアイコン

3.3で定義したリソースを用いて抽象クラスを定義する．

抽象クラス：＝ {
　active メッセージリスト，
　passive メッセージリスト，
　アイコンスクリプト，
　レイアウト，
　リソース*
}

アイコンスクリプト：＝
　{(**passive メッセージ**：**スクリプト**;)*}

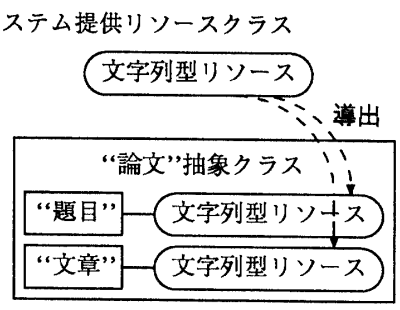

図11：リソースによる抽象クラスの実装

[1]リソース名を局所的なスコープとする根拠は，リソース名がクラスの静的な実装に際してデザイナの主観的な意味付けにより定義される属性であり，外部から見たそれらの動的な意味属性は異なるであろうという直観に基づく．これはリソースがもつ意味属性の粒度が情況(context)に大きく依存するためである．したがって我々はオブジェクトの外部に存在するリソースのリソース名は有効でないと考えている．しかしリソースの意味属性が有する，情況に応じた構造については検討の余地がある．

スクリプト：＝

{(リソースへのメッセージ｜制御構造)⁺}

レイアウト：＝

{(リソース名：レイアウトパラメータ；)*}

ここで*は0個以上の操り返し，⁺は1個以上の繰り返しを表す．

アイコンスクリプトは，そのアイコンがメッセージを受けた際に実行する機能の記述である．アイコンは先に述べたように複数のリソースから構成される．アイコン間で交換されるメッセージはエンドユーザの直観的な理解を目的としているため，アイコンを構成するリソースはそれらのメッセージを直接理解することができない．したがって各リソースへメッセージを分解する必要があり，それを記述したものがアイコンスクリプトである．

アイコンスクリプトはpassiveメッセージの個数だけのエントリをもち，各エントリはpassiveメッセージのタイプをもつフィールドと，そのメッセージを受けた際に実行されるスクリプトから構成される．スクリプトはリソースへのメッセージと制御構造によって表わされる．

一般に制御は順次，分岐，繰り返しの三つの構造により記述されるが，本システムではアイコンの機能の粒度を小さく定義することにより，そのような制御構造は，エンドユーザの複数のアイコンによる対話的な操作により構成されると考えている．これは実世界のオブジェクトが，その内部に制御構造をもたないという直観に基づいている．しかしある機能を並列に処理することは，しばしば起こりうる．このような処理を実現するために，リソースへのメッセージを並列に処理する機構が要求される．スクリプトは並列処理の同期のために，いくつかのシーン (scene) と呼ばれる処理群に分割される．同じシーンに属するリソースの機能は並列に処理される．ユーザは必要な処理を終える時に，システムにシグナルを与えることにより，次のシーンへ進むことができる．

レイアウトは機能実行時に用いられる対話ウインドウの設計に関する情報である．本システムでは対話ウインドウをフェイス(face)と呼ぶ．アイコンのフェイスは，さらにリソースのフェイスから構成される．図12にフェイスの概念図を示す．

システムデザイナはリソースフェイスをアイコンフェイス上で自由に配置することにより，高いユーザインターフェイスを得ることができる．

全てのリソースはフェイスのサイズを属性として保持し，その値はクラス定義時にシステムデザイナにより与えられる．詳細は3.6で述べる．またリソースフェイスのサイズは，リソース値に対して量的な制約[2]を与える．

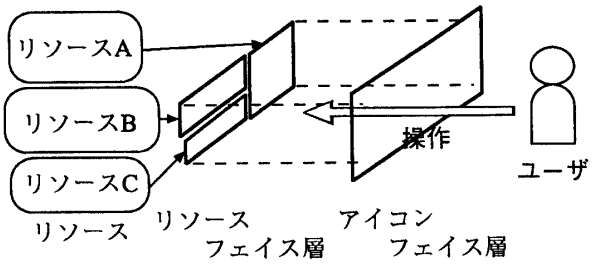

図12：フェイスの階層構造

アイコンはレイアウトパラメータを用いて，リソースフェイスをそのアイコンフェイス上に配置する．ここでレイアウトパラメータは，リソース名とアイコンフェイス上の絶対座標の組で与えられる．リソースフェイスのサイズとレイアウトパラメータの管理の分割は，柔軟なフェイス構成を目的としている．

3.5 リソースによるアイコンの機能実行

アイコンの機能実行（式(1)参照）の概要を図13に示す．

2.1で述べた方法にしたがってユーザが任意のメッセージを決定すると，カーネル及びアイコンは以下の手順にしたがってアイコンの機能を実行する．以下メッセージ送信側のアイコンをactiveアイコン，受信側をpassiveアイコンと呼ぶ．

Ⅰ．カーネルはactiveアイコンからリソースの複製を取得し，それらを引数としてpassiveアイコンにマッチしたメッセージを送る．

Ⅱ．メッセージに対応する機能実行の結果，作成されるアイコンの型枠を作成する．この型枠は，そのクラス固有のデフォルトリソースを保持するインスタンスである．

Ⅲ．アイコンスクリプトを参照して，メッセージを各リソースが理解できる粒度まで分解する．

Ⅳ．分解されたメッセージを受けたpassiveアイコンのリソースは，自身がもつ属性とactiveあるいはpassiveアイコンのリソース群をもとに処理を行い，処理結果をもつ新しいリソースを作成する．

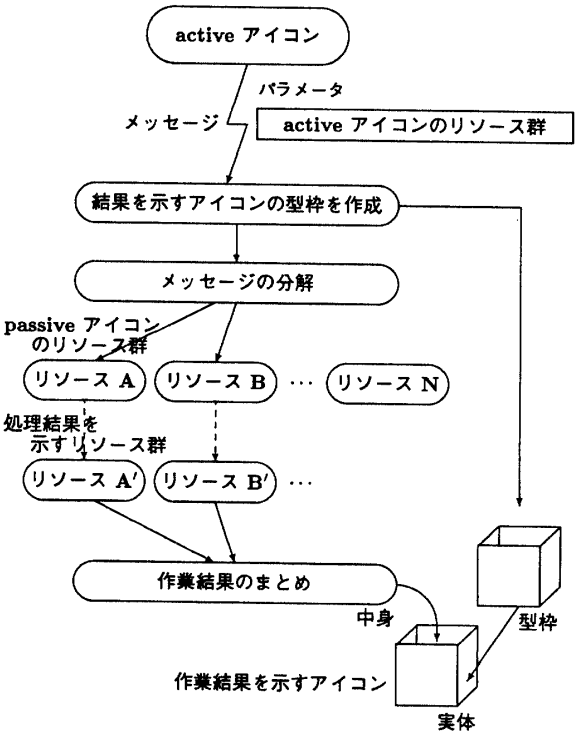

図13：リソースによるアイコンの機能実行の流れ

V．作成された新しいリソースの集合はアイコンの機能実行の結果であり，これらをII．で作成した型枠に代入することによって作業結果を保持するアイコンが得られる．

3.6　クラスの作成

システムデザイナは3.4で述べた機構を有する抽象クラスを，クラスエディタを用いて対話的に作成できる．

クラスエディタの機能はスクリプト編集とアイコンフェイス編集からなる．スクリプト編集は，active，passiveメッセージリストの定義とpassiveメッセージに対応したスクリプトを作成することにより，アイコンの機能を定義する．一方，アイコンフェイス編集はリソースの編集とレイアウトの編集により，アイコンの属性を定義する．現在アイコンフェイス編集についてはプロトタイプのインプリメントを行っているが，スクリプト編集は設計段階である．以下アイコンフェイス編集について述べる．

[2]例えば240×240pixelのフェイスサイズをもつ文字列リソースは，10×5文字しか保持することができない．しかしこのような制約の有効性は検討の必要がある．

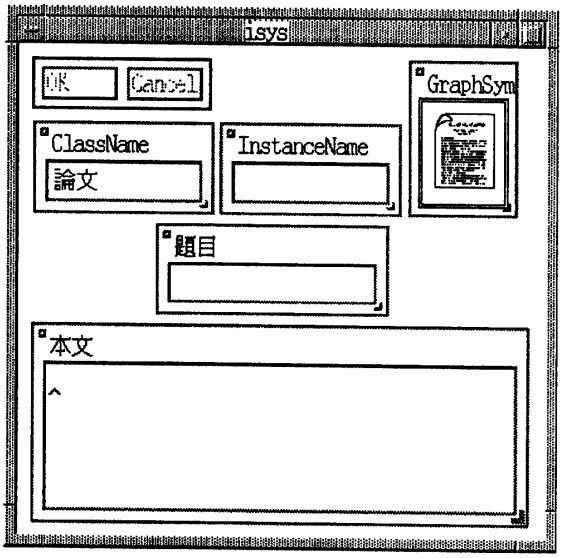

図14：アイコンフェイス編集画面

アイコンフェイス編集では，システム定義のリソース型から，所望の抽象クラスに必要なリソースを作成する．まずシステムデザイナはリソース名を決定し，抽象クラスにリソースを追加する．次にアイコンフェイス上におけるリソースフェイスのレイアウトパラメータを，ドラッグとリサイズにより定める．最後に必要に応じてリソース値を変更して，そのアイコンにおけるデフォルト値を与える．このような操作を繰り返すことによりリソースとアイコンフェイスを編集する．図14にアイコンフェイス編集画面の例を示す．

図14中のアイコンフェイス上には，五つのリソースフェイスがシステムデザイナの設計により配置されている．各リソースフェイスはリソース名とリソース値を含む矩形領域で示される．ここでリソース名“ClassName”，“InstanceName”，“GraphSymbol”の各リソースは，全ての抽象クラスに実装される特別なリソースである．“ClassName”リソースと“GraphSymbol”リソースにはクラス固有のデフォルト値が設定されている．また同じ文字列型リソースから導出された，ユーザ定義の“題目”リソースと“文章”リソースが，保持する文字数を制限するよう適当なサイズが与えられている．

4．まとめ

本稿ではアイコンシステムの実行・構築支援環境に対するアプローチについて報告した．本システム上に構築されるアプリケーションアイコンシステムは，エ

ンドユーザに対して使い易いユーザインターフェイスを有するプログラミング環境を提供すると考えている.

また構築支援環境において,我々が提案したリソースを用いたアイコン定義に基づき,システムデザイナに視覚的かつ対話的な抽象クラスの作成,すなわち柔軟なシステムの拡張を実現する.またアイコンはリソースの複合オブジェクトとして実現されるため,構造及びインターフェイスが統一されており,再利用及び保守管理性に優れている.

今後の課題として粒度の大きいメッセージを粒度の小さいメッセージへ分解する組織的な方法,さらにリソース及び抽象クラスの知的管理方法の探求が挙げられる.また提案した手法のインプリメントを通したさらに詳細な検証を行いたい.

本研究の一部は科研費(重点領域(1),課題番号04219105)及び(一般C,課題番号036800 32)の援助によった.

参考文献

1) Nan C. Shu著,西川博昭訳,「ビジュアルプログラミング」,日経BP社,1991.

2) M. Hirakawa, M. Tanaka, and T. Ichikawa, "An Iconic Programming System, HI-VISUAL", *IEEE Trans. on Software Engineering*, Vol.16, No.10, pp.1178-1184, 1990.

3) Tscheligi, M., Penz, F., and Manhartsberger, M., "N/JOY-The World of Objects", Proc., *IEEE Worksshop on Visual Languages*, pp.126-131, 1991.

4) 山口,栁木,西村,田中,"アイコンシステム構築環境",人工知能学会第16回ヒューマンインターフェイスと認知モデル研究会,pp.27-36,1992.

5) 山口,西村,田中,"アイコンシステムのためのオブジェクト作成ツールの開発",情報処理学会第10回プログラミング研究会,pp.81-88,1993.

6) Doug Lea, User's Guide to GNU C++ Library, Free Software Foundation. Inc., pp.17-18, 1992.

(平成5年4月15日受理)