

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

山口 真悟[†] 西村 世志人[‡] 田中 稔[‡]

[†]山口大学大学院工学研究科 [‡]山口大学工学部

視覚的プログラミングシステムは、エンドユーザに有効なアプリケーション環境を提供するが、それらの多くは構築支援環境を伴っていない。我々は視覚的プログラミングシステムの一つとして保守管理性と拡張性に注目したオブジェクト指向アイコンシステムを研究開発している。本研究におけ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、システムの機能の拡張はアイコンの追加定義によって行われる。そこではアイコンをより小さな粒度のリソースの複合オブジェクトとして実現している。本稿ではリソースの有用性と、それを用いた構築支援環境について報告する。

Development of Object Generation Tools in Iconic System

Shingo Yamaguchi[†] Yoshihito Nishimura[‡] Minoru Tanaka[‡]

[†]Graduate School of Engineering, Yamaguchi University

[‡]Faculty of Engineering, Yamaguchi University

Visual programming systems provide end-users effective application environments. However most of them don't support constructing environment themselves. Our work aims to achieve 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 some 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

ソフトウェア開発におけるノンプログラマの増加に伴い、ユーザフレンドリなプログラミングシステムに対する需要が高まっている。これを得る一つの方法として視覚的情報の有用性とオブジェクト指向技法を組み合わせた視覚的プログラミングシステムが、研究開発され成果を挙げている [1][2]。

多くの視覚的プログラミングシステムはエンドユーザに対して、有効なプログラミング環境を提供するが、そのシステム開発にはさらに多くのコストが要求されており、システム開発を支援するツールが要求されている。

このような要求に応えるために本研究では、保守管理性 (maintainability) 及び拡張性 (extensibility) に注目したシステムの研究開発を行なっている。本システムの機能は、与えられたアイコンの集合によって定まる。システムの拡張はアイコンの追加によって実現されるが、さらに拡張性を高めるためにアイコンより粒度の小さいリソースを用いた構築支援論理モデルを導入した。

以下、2.では我々が開発したアイコンを用いた視覚的プログラミングシステムの実行環境について述べる。3.ではユーザによるシステムの対話的な構築環境と、このような機構を実現するリソースについて詳述する。最後に 4.で結論と今後の課題について述べる。

2. アイコンシステム実行環境

我々が開発したアイコンシステムの概要について述べる。

2.1 アイコン

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視覚的情報と、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 [3]。

実在のオブジェクト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ように、全てのアイコンはその両方の性質を同時に備えている。また一つのアイコンの振舞いも唯一に固定されておらず、他のアイコンと組み合わせる時に動的に決定される。

このような機構を実現するために、各アイコンは受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に対応している。

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、ユーザはそれらを視覚的に選択することにより対応する機能を実行できる。図 1 に表アイコンとドローイングキットアイコンを重ね合わせた場合の例を示す。

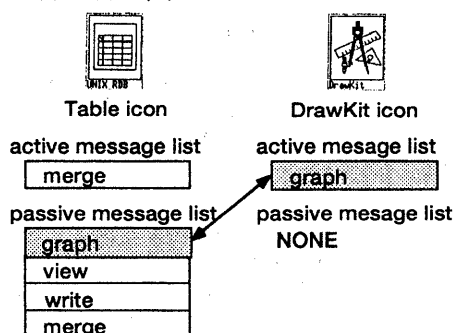


図 1: 双方向メッセージパッシングによる機能選択

またユーザは必要に応じてアイコンを作成あるいは破壊することができる。新しいアイコンの作成は、与えられた抽象クラスをカスタマイズすることにより行なわれる。抽象クラスについては 3.2 で詳述する。本システムは対話を用いた属性値の

編集機能を提供しており、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。

2.2 アイコンシステム

アイコンシステムは2.1で述べたように、アイコンを重ね合わせることにより、実世界の「もの」Aと「もの」Bを用いて作業した結果「もの」Cが作られる。」という作業を模擬する。これらの振舞いは式1で定義される。

$$\text{icon}A \circ \text{icon}B \longrightarrow \text{icon}C \quad (1)$$

ここで $\circ$ は実行される機能を示す。

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2に示す。

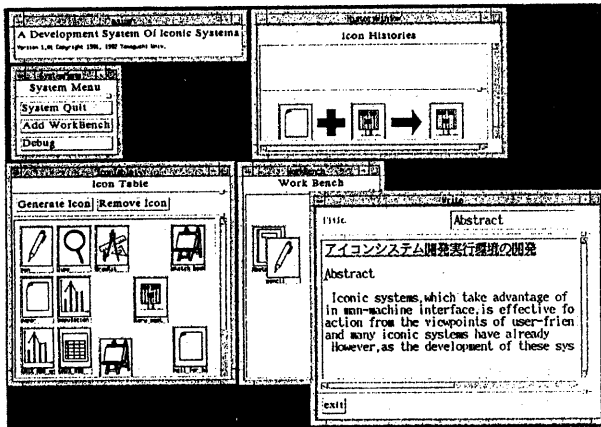


図 2: アイコンシステム実行画面

アイコンは初めに IconTable ウィンドウ (図 2 左下) に配置される。そこからユーザは作業空間である WorkBench ウィンドウ (図 2 中央下) 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 2 では、WorkBench ウィ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、ノートに文字を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ィンドウ (図 2 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。

このような処理を繰り返すことによって一連の処理を実行する。

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を図 3 に示す。

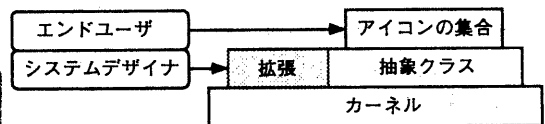


図 3: アイコンシステム構築環境の構成階層

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ーの二つのユーザレベルを支援対象としている。

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ー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ーが定義した意味と、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

システムデザイナーによる抽象クラスの追加定義により行なわれる。

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、作成される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、システムデザイナー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。

全ての抽象クラスは多相性 (polymorphism) をもち、メッセージに対して統一されたインタフェイスを有する。ただしメッセージを実行するか否かは、各々の passive メッセージリストを参照することにより決定される。

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 small talk のような完全に階層化されたアプローチ [5] 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、システムデザイナー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と保守管理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を、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。

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 (resource) の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、抽象クラスは一定のインタフェイスが要求される。システムデザイナーが、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ー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。システムデザイナーは、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。

我々はリソースを次のように定義する。

リソース :=

{ リソース名, データ, メソッド }

リソースはリソース名、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識別子であり、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もつリソースを作成できる。リソー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わせて、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。

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 ;) * }

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 個以上の繰り返し、+ は 1 個以上の

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状況 (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、状況に応じた構造については検討の余地がある。

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る、その記述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。

しかしある機能をモーダレス (modeless) 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。ユーザは同じシーンに属するリソースの機能をモーダレスに処理し、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。図 4 にスクリプトの構造を示す。ここでは各リソースのメッセージのパラメータについては省略した。

レイアウトは機能実行時に用いられる対話ウィンドウの設計に関する情報であり、本研究では MVC モデルのビュー (view) とコントローラ (controller) を組み合わせた対話ウィンドウをフェイス (face) と呼ぶ。アイコンのフェイスは、さらにリソースの

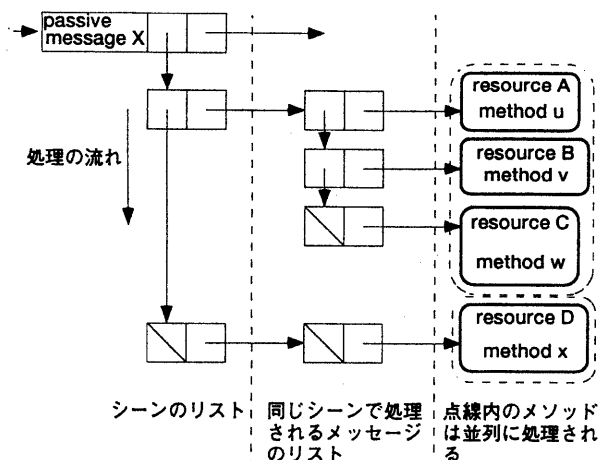


図 4: スクリプトの構造

フェイスから構成される。図 5 にフェイスの概念図を示す。

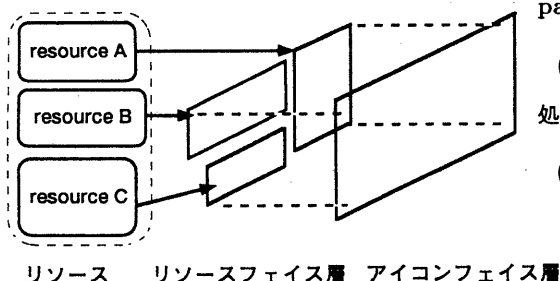


図 5: フェイスの階層構造

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ーにより与えられる。詳細は 3.5 で述べる。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。

アイコンはレイアウトパラメータを用いて、リ

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。

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。

リソースを用いたアイコンの機能実行の概要を図 6 に示す。

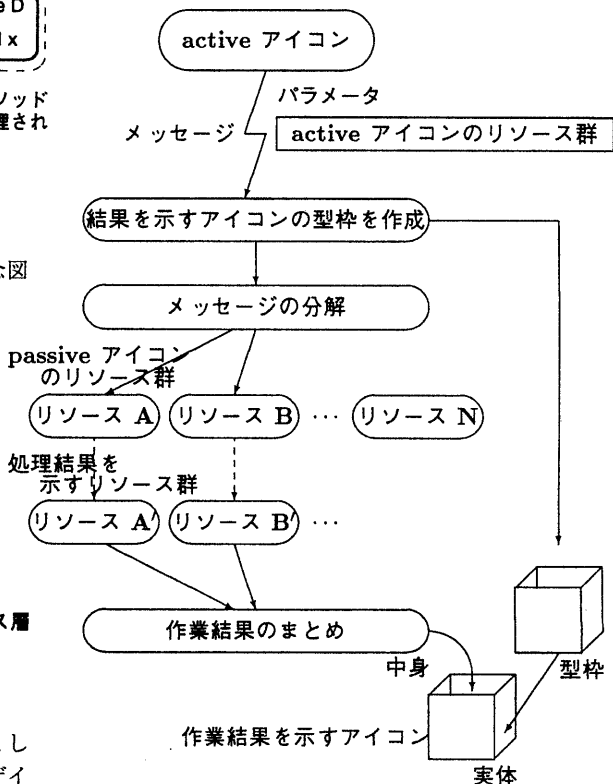


図 6: リソースによるアイコンの機能実行の流れ

2.1 で述べた方法にしたがってユーザー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 アイコンにマッチングメッセージを送る。
- 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。この型枠はオブジェクトであり、そのクラス固有のデフォルトリソースを保持する。
- 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。
- 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。

3.5 クラスの対話的作成

システムデザイナーは 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。

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ー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 7 にアイコンフェイス編集画面を示す。

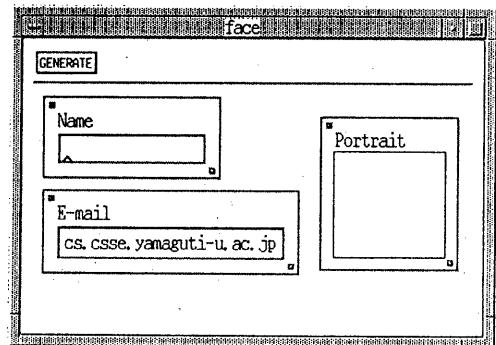


図 7: アイコンフェイス編集画面

図 7 中のアイコンフェイス上には、三つのリソースフェイスがシステムデザイナー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。

作成中の抽象クラスは“プロフィール”抽象クラスであり、二つの文字列リソース (リソース名 “Name”, “E-mail”) とビットマップリソース (同 “Portrait”) からなる。リソース “E-mail” にはデフォルト値が与えられている。

このような視覚的かつ対話的なアイコンの属性定義は、システムデザイナーの設計を直接具現することを可能にし、また試行錯誤的に抽象クラスを定義する構築環境を提供する。またノンプログラマでも、実世界で既に獲得されているオブジェク

ト作成のノウハウを利用することにより、価値の高いオブジェクト定義が可能であると考えられる。

4. おわりに

本稿ではアイコンシステムの構築支援環境に対する我々のアプローチについて報告した。我々が提案したリソースを用いたアイコンの定義に基づくアイコンクラスの作成は、システムデザイナー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、構造及びインタフェー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 03680032)の援助によった。

参考文献

- [1] 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.
- [2] Tscheligi, M., Penz, F., and Manhartsberger, M., "N/JOY-The World of Objects", Proc., *IEEE Workshop on Visual Languages*, pp.126-131, 1991.
- [3] 山口, 栩木, 西村, 田中, "アイコンシステムの開発", 情報処理学会 第 44 回全国大会, 6K-1, 1992.
- [4] 山口, 栩木, 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] Doug Lea, *User's Guide to GNU C++ Library*, Free Software Foundation. Inc., pp.17-18, 1992.