ソフトウェア工学 102-14
（1995．1.26）

クラスツリー間の接続に基づく視覚的プログラミング手法

北村 操代　杉本 明
三菱電機株式会社 中央研究所

ソフトウエア開発を容易にするため、視覚的プログラミングが注目されている。また一方、近年オブジェクト指向技術の浸透とともに再利用部品としてのクラスライブラリの整備が進んでいる。本稿ではまず、これら既存の複数のクラスライブラリを組み合わせてソフトウエアを構築するため、クラスライブラリ間の連携動作をモジュール性良く定義する手法として、クラス間接続と呼ぶ方式を提案する。また本稿では、クラスおよびインスタンスを立体図形として3次元空間中に投影し、その中でクラス間接続を実現することを通じて、プログラミングを行う手法を提案する。さらに、配水管網シミュレーションシステムへの適用例も述べる。

Visual Programming with Interclass Connection

Misayo Kitamura and Akira Sugimoto
Mitsubishi Electric Corporation　Central Research Laboratory

The number of class libraries (reusable software components) are increasing in these days. For constructing software, it is requierd to put these class libraries together. We aim to use these libraries easily using visual programming environment. This paper describes a interclass connection method for defining cooperative actions between class libraries with good modularity. This paper also describes a visual programing environment in three-dimensional space. A pipenetworks analysis example is also described.

1 はじめに

ソフトウエア開発において、プログラミング、デバッグ、プログラム理解などの点から、ソフトウエアの視覚化とその環境下でのプログラミングが望まれている。これを実現するため、オブジェクト指向プログラムを視覚的に構築する環境として、[1]、[2]、[3] などの研究がなされている。

一方、近年オブジェクト指向技術の浸透とともに再利用部品としてのクラスライブラリの整備が進んでいる。今後のシステム開発では、用途別に作られた様々な種類のクラスライブラリを組合せ、全体の機能を実現していくことになると思われる。この時、容易に組合せられるかが、生産性向上の鍵となる。

本稿では、既存のクラスライブラリを組み合わせることによってソフトウエアを構築するための、ソフトウエアの視覚的構築の一手法について述べる。まず、複数のクラスライブラリ間の連携動作をモジュール性良く定義する手法として、クラス間接続と呼ぶ方式を提案する。また、クラス間接続によって構成されるソフトウエアを視覚的に作成するため、クラスおよびインスタンスを立体図形として3次元空間中に投影し、その中でクラス間接続を実現することを通じて、プログラミングを行う手法を提案する。

以下本稿では、まず複数のクラスツリーの組み合わせの必要性について論じたのち、一般的なクラス間接続について説明する。次に、用いる組み合わせの種類を限定した上で、ソフトウエアを3次元空間中で視覚的に構築する環境について述べる。また、配水管網シミュレーションシステムへの適用例も述べる。

2 複数のクラスツリーの組合せ

オブジェクト指向ライブラリでは、上位クラスからの継承によりオブジェクト部品のクラスが定義されている。継承関係によって密接に関連付けられたクラス群をクラスツリーと呼ぶものとする。ライブラリを利用する場合、クラスツリー上の特定のクラスを基に、アプリケーションに応じたサブクラスを作成する。ライブラリ内に複数のクラスツリーが格納されていても、全体があるフレームワークによって構成されている場合には、別々のクラスツリー上から生成したオブジェクト間の連携動作が、それらのオブジェクトの上位クラス間で規定されている場合が多い。

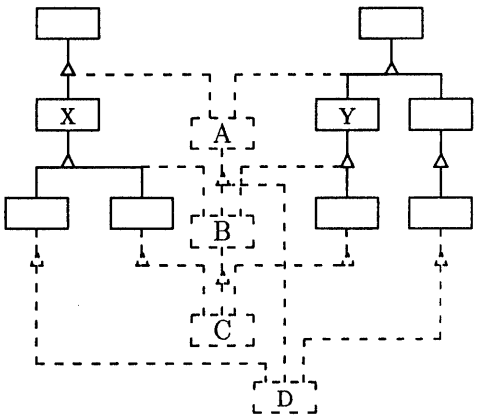

図 1: 多重継承例

ところが、個別目的、個別分野に対応して作成されたフレームワークライブラリを組み合わせ、全体のシステムを開発する場合、個々のクラスツリー上のクラス、あるいはそのクラスから独自に作成したサブクラス等から生成したオブジェクト間の連携動作は、組合せ時に実現する必要がある。

2.1 インスタンス間の組合せ

まず、複数のクラスツリーから別個にオブジェクトを生成し、それらのインスタンス間の関係を設定することで、連携動作を実現する手法がある。

開発するソフトウェアで用いるインスタンスが固定数の場合は、ディスプレイ上に視覚化したインスタンス間の関係設定による視覚的プログラミングが考えられる。筆者らはインスタンス間の関係をリンクやグルーと呼ぶオブジェクトとして定義し [4]、インスタンス間の関係をマウスを用いた対話的操作により動的に変更することで、ソフトウェアをカスタマイズする環境を試作した [5]。

ところが、設備管理システムやシミュレータ等では、ソフトウェア動作中にインスタンスが動的に生成、削除される。このような場合には、連携動作をクラス間に設定する必要がある。

2.2 多重継承による組合せ

クラス間の関係を設定する手法としては、多重継承を用いる方法が考えられる。図1に多重継承

を用いた組合せの例を示す。図で実線は既存のクラスツリーを示し、破線は組合せのために新規に作成するクラスを示す。図でクラスAは、2つのクラスツリー間を連携させるための基本的な枠組みを記述した抽象クラスである。クラスBには、クラスX、クラスYのサブクラス間の特殊な組合せ方を記述している。図のクラスC，Dでは、具体的にオブジェクトを生成するクラスを定義している。そのため、AやBのクラスを継承すると共に、それぞれのクラスツリーから特殊化されたクラスを継承している。

それぞれのクラスツリーが、ある特定の視点からのモデル化を記述していると考えた場合、現実のシステム開発に必要な多様な側面を持つオブジェクトを、図のような多重継承により定義することは自然である。しかし、オブジェクト指向言語に従来から備わっている多重継承機構では、クラス間の結び付きが強すぎ、多重継承したクラスの記述が複雑になりやすい。

例えば、それぞれのクラスツリーにおいて、別個の機能、目的を持つメソッドがたまたま同一名で定義されている場合には、それから生じるコンフリクトを解決しなければならない。また、同一の目的を持つメソッドに対しても、図1のクラスCのように多くの継承パスが存在することにより、メソッド選択の優先順位を決定しなければならない。さらに、継承関係が複雑になることにより、ソフトウェアが理解しにくいものとなる。

3 クラス間接続と接続メソッド

本章では、クラスツリー間の組合せ時の連携動作を記述するための方式として、クラス間接続と呼ぶ機構を導入する。クラス間接続は、次のような接続メソッドから構成される。

3.1 接続メソッド

接続メソッドは、あるクラス(起点クラスと呼ぶ)のインスタンスへのメッセージをトリガとして、連携する他のクラス(終端クラスと呼ぶ)のインスタンスへ作用するメソッドである。次の2種類がある。

1. インスタンス生成メソッド
   起点クラスか起点クラスのサブクラスがインスタンスを生成した場合に、終端クラスのインスタンスを生成する。2つのインスタンスには、本機構により接続関係が設定され、どちらかのインスタンスが削除されると、一方のインスタンスも削除される。

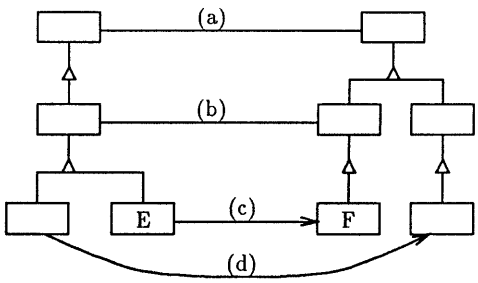

図 2: クラス間接続例

2. 連携メソッド
   インスタンス生成メソッドにより、起点クラスと終端クラスのインスタンス間に接続関係が設定された時、起点クラスのインスタンスへのメッセージをトリガとして実行される。実行形態には次の3種類がある。
   - replace: 連携メソッドをインスタンスのメソッドの代わりに実行する。
   - before: 対象インスタンスのメソッドを実行する前に、連携メソッドを実行する。
   - after : 対象インスタンスのメソッドを実行した後に、連携メソッドを実行する。

3.2 クラスツリー間の接続

起点クラスと終端クラスの組合せが共通な接続メソッドを、一つにまとめたものがクラス間接続である。例えば、クラスαとクラスβ間の接続の中に、αを起点、βを終端とする接続メソッドと、βを起点、αを終端とする接続メソッドが共に含まれる。図2に図1と同様な組合せをクラスツリー間の接続で示した。クラス間接続の中にインスタンス生成メソッドが含まれる場合、その起点クラスから終端クラスへの矢印を表示している。

3.3 継承関係

クラス間接続は下位の接続に継承される。図2でcの接続はa, bの連携メソッドを継承し、dの接続はaの接続のみを継承する。多重継承と違って、図2のクラスEやFのメソッド自体がa

やbの接続により影響を受けることはない。接続は接続で、クラスツリーはクラスツリーで各々独自の継承がなされた後、接続メソッド内のトリガにより組み合わされる。

4 視覚的開発環境とその実現

本章では、クラス間接続により構成されるソフトウエアを開発するための、視覚的プログラミング環境を実現する手法について述べる。まず、実現方針について論じ、次にソフトウエアの可視化方式とプログラミング環境を説明する。

4.1 実現方針

2と3では、クラス間接続の一般的な枠組について述べてきた。この枠組によれば、個々の接続メソッドを定義する度にプログラムを記述することにより、種々のクラスツリーを柔軟に組み合わせてソフトウエアを構築することが可能となる。

しかし、この機構によるプログラミングをすべて視覚的開発環境下で提供しようとすると、プログラミング時に接続メソッドのコーディングをその環境中で行うことになる。これでは、プログラミング時の視覚的操作が煩雑になり、視覚的開発環境本来の操作性の容易さを殺してしまうことになる。

そこで、クラス間接続の連携メソッドを以下の2種類に限定する。

1. 属性一致連携メソッド
   クラスPの属性pとクラスQの属性qの値を一致させる連携メソッド。一方の属性値が変更された際に、afterトリガでもう一方の属性値を変更する。
2. 関数実行連携メソッド
   当該クラスのある関数を実行した際に、連携先のクラスの関数(引数なし)を実行する連携メソッド。

これらを用いるだけでも、クラス間の組み合わせによって行うプログラミングのうちの相当部分をカバーできると思われる。そこで、プログラミング操作で用いる接続メソッドとしては、インスタンス生成メソッドと、この2つの連携メソッドだけを提供する。

本稿の実現では、クラスライブラリとして筆者らが開発したGhostHouse[5]を利用している。

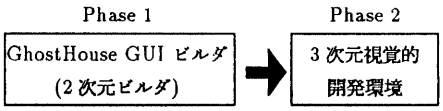

図3: ソフトウエア作成の流れ

GhostHouseでは属性設定のフレームワークが一様に定まっており、属性一致連携メソッドや関数実行連携メソッドを対話的に設定することができる。以下の例では、GhostHouse上に用途別に作られた複数のクラスツリーを扱う。

本稿の実現によるソフトウエアの視覚的な作成の流れを図3に示す。まず、GhostHouseに備えられたGUIビルダ機能を用いて、画面のレイアウトや画面展開等を視覚的に構築する。この時メイン画面が完成され、その中にはオブジェクトが動的に生成、削除されるシートが用意される。次に、3次元視覚的開発環境下でクラス間接続を作成する。この作業により、シート上にプログラム実行時に動的に現れるオブジェクトの連携動作を定義する。この移行の際に、3次元表示を行う際のクラスツリーの最上位クラスと、その中で画面上に表示を行う機能を持つクラスを指定する。

4.2 3次元空間への可視化

クラス間の関係定義操作や実行時の挙動の把握のため、クラスやインスタンスを何らかの図形として画面上に表示する必要が生じる。しかし、クラス数やインスタンス数が多くなり、クラス間の関係が入り乱れるため、平面上でクラスやインスタンスを操作することは難しい。そこで、クラス、インスタンス、およびそれらの関係を3次元空間を用いて可視化する。[1]では3次元空間にクラスとそれに属するメソッドを視覚化しているが、筆者らはクラス、インスタンス、クラス間接続を視覚化している。

本方式でプログラミング対象として用いられるクラス、および、すべてのインスタンスは3次元空間中の立体図形に投影される(図4)。本稿で取り扱うソフトウエアは実行時の対話用のGUI画面を伴うものとする。3次元空間は、2次元のGUI画面と、GUI画面の裏側のインスタンス領域、その上部のクラスツリー領域からなる。この3次元空間を以下ソフトウエア空間と呼ぶ。

2次元のGUI画面には、プログラム理解やデ

バッグを容易とするため、通常のソフトウエア稼働時の表示がそのまま示される。

クラスツリー領域には、クラスを可視化した立方体が配置される。1つのクラスツリーは図4のように画面と若干の角度を持つ平面上にある。クラスツリー平面どうしは平行に配置される。クラスの継承関係は、OMT[6]を3次元化し、線分と四角錐によって表示される。クラス(クラスを可視化した立方体を以下単にクラスと呼ぶ)にはクラス名が表裏2面に表示される。クラスツリー内のレイアウトは、根のクラスの位置から自動的に計算される。

クラス間接続はクラスツリー間のクラスとクラスを結ぶ線分として可視化される。画面表示を行う機能を持つクラスツリー平面は、ソフトウエア空間中最も画面に近い位置に置かれる。

インスタンス領域には、インスタンスを可視化した球が配置される(この球を以下単にインスタンスと呼ぶ)。配置される位置は、そのクラスを通る、画面に平行な平面上である。GUI画面への表示機能を備えたクラスのインスタンスは、そのGUI画面上の表示位置と同じx,y座標の位置に可視化される。それ以外のインスタンスは、原則としてインスタンス間の接続関係を持つインスタンスと同じ位置(x,y座標)に配置される。

インスタンス間の接続関係、および、クラスとインスタンスの関係は、通常表示されない。ユーザの指定時などに線分で結んで表示される。

4.3 プログラミング

プログラミング機能としては、接続メソッドの作成(接続の新規作成を含む)、接続クラスの変更、接続内の接続メソッドの変更、接続の削除、が提供される。

接続メソッドの作成と接続クラスの変更では、マウスで対象クラスを指定する。新規メソッドの作成時には、接続メソッドの種類を指定した後にマウスで2つのクラスを指定する。クラスを指定したときに、属性一致メソッド、関数実行連携メソッドの場合には、それぞれクラス中で接続可能な属性一覧、関数一覧が画面上に表示され、その中から選択することによって、接続の詳細を指定する。一方、接続クラスの変更時には、マウスで接続の端点をドラッグしながら新しいクラスを指定する。

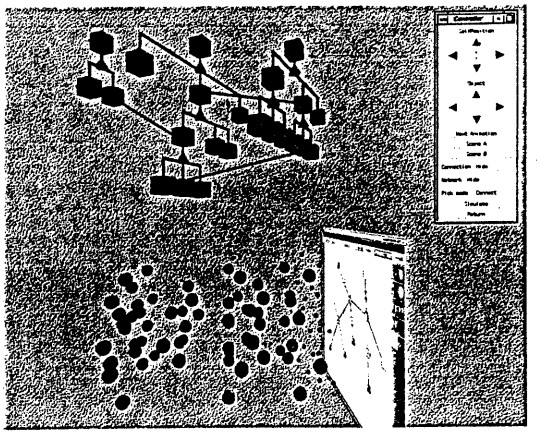

図4: 可視化されたソフトウエア空間

プログラミングによってインスタンス生成メソッドの作成や変更が行われた場合、新しい接続に基づいて起点側から終点側のインスタンスが生成される。またその前に接続されていた場合には、それに基づいて生成されていたインスタンスが削除される。

なお、これらのプログラミング操作には現在マウスを用いているが、3次元空間中の任意の位置への移動はマウスでは難しいため、3次元ディスプレイと3次元ポインティングデバイスを利用して入力するようにする予定である。

4.4 挙動の追跡とデバッグ

本方式のソフトウエア空間では単にプログラムを行うのみでなく、ソフトウエアの挙動を見ることもできる。これにより、メソッドの流れを視覚的に捉えることができ、ソフトウエアの理解、デバッグが容易となる。

インスタンスの挙動追跡のため、現在メソッド実行中のインスタンスを色変え表示し光らせる。このとき、同時にインスタンスの属するクラスと実行中のメソッドが定義されているクラスも色変え表示される。また、クラス間接続中の連携メソッドが発行された場合には、そのクラス間接続が色変え表示されるとともに、対応するインスタンス間接続が表示される。

本環境下では全てのインスタンスを常に表示するわけではなく、画面からはみ出すものを表示しない、特定のクラス(とそのサブクラス)のインスタンスのみを表示する、という制限を設けることができる。

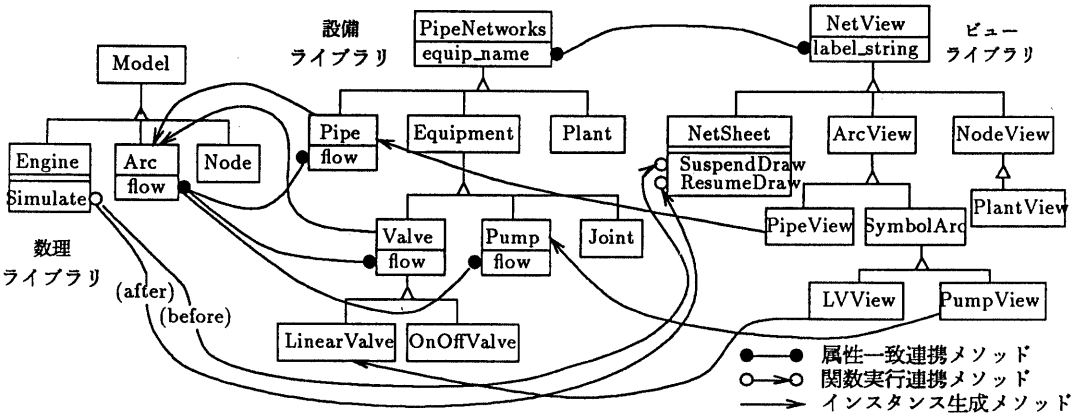

図 5: 管網解析アプリケーションで用いるクラスとクラス間の接続メソッド

5 適用例

筆者らは本方式を管網解析アプリケーションに適用している。管網は管、ポンプ、バルブ、配水場などから構成されるネットワークであり、管網解析は、管網の個々の設備を流れる既知の水量や水圧差から、未知の水量や水圧差の値を求めることである。

管網解析アプリケーションは、管網の設備データを持つクラス群(設備ライブラリ)と、管網から抽出される非線形連立方程式を解くクラス群(数理ライブラリ)と、画面表示のための部品クラス群(ビューライブラリ)を組み合わせることによって構成できる。これらはGhostHouseクラスライブラリ上に実現されている。

数理ライブラリは、電気回路や管網などの流量と圧力を持つ系を解析するための汎用ライブラリである。一方、ビューライブラリは、CASEツールの開発にも利用できるネットワークエディタ機能を持つ。

これらライブラリのクラス構成(一部略)を図5に示す。また図には、管網解析アプリケーションとしてクラスツリーを組み合わせるための接続メソッドの一部も示している。

6 おわりに

本稿では、まずクラス間接続によるクラスライブラリの組合せ手法について説明し、次にこれによって構成されるソフトウエアの視覚的なプログラミング環境について述べた。

本稿では特にクラスライブラリGhostHouseに属する複数のクラスツリーを対象とし、連携メソッドとして、属性一致、引数なしの関数実行の2つに限定して実現する場合について説明した。クラスツリー間の相互作用は、クラスツリー内のオブジェクト間の相互作用と比較して疎であると期待される。しかし、実際にアプリケーションプログラムを作成するためには、上記の2つの連携メソッドだけでは不十分であると思われる。接続操作の容易性を保ちながらの視覚的な接続機能の拡充が今後の課題である。また、接続の詳細の表示方法についても課題が残されている。

参考文献

[1] 小林, 小池: "オブジェクト指向言語C++のクラスライブラリ視覚化に関する研究", インタラクティブシステムとソフトウエア I: 日本ソフトウエア科学会 WISS '93 (竹内彰一(編)), 近代科学社, 1994.

[2] 三ツ井, 中村: "オブジェクト指向プログラム理解のための視覚化技法", 情報処理学会ソフトウエア工学研究会, SE-99-17, 1994.

[3] 阿部, 吉田, 中川: "オブジェクト指向プロトタイピングのための視覚的支援環境", 情報処理学会ソフトウエア工学研究会, SE-99-18, 1994.

[4] 北村、杉本: "生成・カスタマイズ手法によるソフトウエア開発方式", 第48回情処全大 pp.5-165 — 166 (1994)

[5] 北村, 杉本: "GUI生成・編集機能を持つクラスライブラリ GhostHouse", 情報処理学会ソフトウエア工学研究会, SE-86-5, 1992.

[6] J.Rumbaugh et al.: "Object-Oriented Modeling And Design", Prentice-Hall, pp. 38 - 43 (1991)