

オブジェクト指向言語における 部品階層の記述の問題点とその改良について

市川武彦 魚井宏高 萩原兼一 首藤勝

大阪大学基礎工学部

Smalltalk, C++等のオブジェクト指向言語では、静的に定まる部品階層(part-of関係の階層)も全て動的に構築するように記述している。それゆえ、そのような部品階層を成すオブジェクト間のメッセージの配送を全てプログラマが記述する必要がある等の問題点がある。そこで本稿では、静的に定まる部品階層を記述するための枠組を提案し、この枠組を用いることで部品階層を成すオブジェクト間で交換されるメッセージの量と部品階層の変更に伴うプログラムの変更を減らすことができることなどを述べる。

Programming Part Hierarchy in Object-Oriented Languages.

Takehiko ICHIKAWA Hirotaka UOI Ken-ichi HAGIHARA Masaru SUDO

Faculty of Engineering Science, Osaka University

1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

Part-of relationship is the relationship among an object and some objects generated by the former. We call the hierarchy of this relationship "part-hierarchy" in this paper. In typical object-oriented languages such as Smalltalk, C++, and Objective-C, we describe part-hierarchy as it is made dynamically. Therefore it is necessary that programmers describe message-delivery among objects in part-hierarchy explicitly.

In this paper, we propose a framework called **domain** for describing static part-hierarchy; besides we point out that the number of messages exchanged in execution and the number of changes in source program when part-hierarchy is changed are decreased by using domain.

1 まえがき

オブジェクト指向で記述されたアプリケーションにおけるオブジェクト間の関係には、主にis-a関係、part-of関係がある。

オブジェクト指向言語において重要なis-a関係は、クラス-サブクラス間の関係である。オブジェクト指向プログラミングにおいて、is-a関係は、クラスを設計する際に継承を利用するために用いられる。この関係を記述する枠組は、Smalltalk[10]やObjective-C[2], C++[3]など、オブジェクトをクラスによって定義するオブジェクト指向言語において、クラス継承という言語の機能として備えられている。一方、本稿において、part-of関係とはあるオブジェクト（全体オブジェクトと呼ぶ）とそのオブジェクトによって生成される複数のオブジェクト（部品オブジェクトと呼ぶ）との関係を言い、この関係は一般的に階層を成す。例として図1に示すウィンドウアプリケーションを考える。

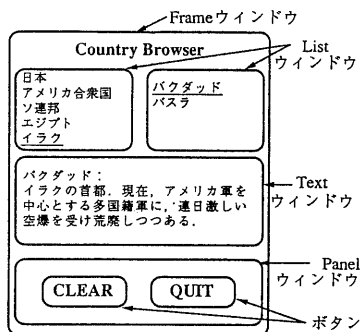


図1 ウィンドウアプリケーションの例。

X-toolkit[5]やXView[9], ET++[4]などのウィンドウアプリケーションを構築するためのツールでは、ウィンドウを構成する各部品（ウィンドウ、ボタン等）をそれぞれオブジェクトとして実現している。そして、それらのオブジェクトのpart-of関係を記述することによりアプリケーションを構築する。この例の場合、図2に示すようにFrameウィンドウであるオブジェクト“aCountryBrowser”は、部品オブジェクトとして2個のListウィンドウ“countryList”, “cityList”とTextウィンドウ“cityData”, そしてPanel“aPanel”を持つ。また、aPanelは2個のボタン“clearButton”, “quitButton”を部品オブジェクトとして持つ。本稿ではこのようなpart-of関係の階層を部品階層（Part Hierarchy[1]）と呼ぶ。

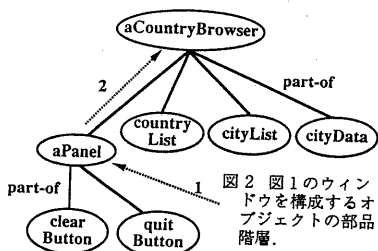


図2 図1のウィンドウを構成するオブジェクトの部品階層。

部品階層は、いつも全てプログラミング時に決定できる、すなわち静的に定まるわけではない。しかし、ウィンドウアプリケーションのようにウィンドウを構成する部品（ボタン、ウィンドウ等）が静的に定まるものも多く、また部品階層全てが静的に定まらなくとも、その一部分が静的に定められる場合もある。しかし、SmalltalkやObjective-C, C++等の汎用のオブジェクト指向言語¹では、一般的には部品階層を全てプログラム実行時に、すなわち動的に構築するように記述する。このために生じる問題点として次の2点が挙げられる。

（1）静的に定まる部品階層も動的にしか構築できないために、部品階層を成すオブジェクト間のメッセージの配送を全てプログラマが記述する必要があること。

（2）一つの部品階層の記述が、複数のクラスの定義中に分散してしまうこと。

そこで本稿ではこのような問題点を解決するために、静的に定まる部品階層を記述するためのドメインという枠組を提案する。ドメインは、従来全て動的にしか決定できないかのように部品階層を記述していたため利用できなかった、“静的に定まる”ことで得られる利点を利用することを目指す。

2 部品階層の記述の問題点とその改良について

本節では、従来のオブジェクト指向言語において、部品階層を記述する際の問題点について述べ、それを解決するために必要と考えられる機能について述べる。

2.1 例題アプリケーション

Country Browserについて

本稿では以降、例題として図1に示すアプリケーションCountry Browserを用いる。本節ではこのCountry Browserについて説明する。

Country Browserは、マウスによって選択された国に存在する都市についての情報をウィンドウ上に表示したり、それをウィンドウ上で編集したり、ファイルに保存したりする機能を持つウィンドウアプリケーションである。

Country Browserのウィンドウは、国名のリストを表示するListウィンドウ（Country Listと呼ぶ）と、都市名のリストを表示するListウィンドウ（City Listと呼ぶ）、選択された都市についての情報の表示、編集を行なうTextウィンドウ（City Dataと呼ぶ）を持つ。また、CLEARボタンをクリックするとCity Dataの表示内容が消去され、QUITボタンをクリックするとアプリケーションが終了するものとする。図1中のCountry Listにおいて、ユーザはマウスのクリックでどれか一つの国を選択できる。図1では、イラクが選択されている。City Listには、Country List上で選択された国の都市名のリストが表示され、同様にユーザはマウスのクリックによりどれか一つの都市を選択できる。図1では、イラクの都市のリストが表示され、バグダッドが選択されている。City Dataには、City List上で選択した都市に関する情報が表示され、ユーザがそれを編集したり、ファイルへ保存したりすることができる。図1では、バグダッドに関する情報が表示されている。

リスト1にオブジェクトaCountryBrowserのクラスCountryBrowserの定義をObjective-C風言語で記述した場合の一部分を示す。

リスト1. クラスCountryBrowserの定義の一部

（Objective-C風言語で記述した例）。

```
=CountryBrowser : Frame
{cityList,countryList,cityData,panel,path,fname} //インスタンス変数
//メッセージnewによって起動されるメソッドnew
+new { countryList = [ListWindow new];
//クラスListWindowにnewを送り、生成されたインスタンスをインスタンス変数leftListWindowに代入する
cityList = [ListWindow new];
cityData = [TextWindow new];
pane = [Panel new];
return([super new]); }
//最後に自分自身(Frameクラスのインスタンス)を生成して返す
```

¹以降この様なオブジェクト指向言語を単にOOLと表記する。

```

..... (中略) .....
//メソッドcloseの定義
-close {
    [aList close];
    [aText close];
    [aPanel close]; return self; }

-rightClick{ ..... (略) ..... }
-leftClick{ ..... (略) ..... }
..... (以下略) .....

```

2. 2 部品階層の記述の問題点 (1)

従来のOOLでは、オブジェクト間のpart-of関係は全て動的に確立されるようにクラスの定義中に記述している。つまり、プログラマは必要な部品オブジェクトの生成、それらのpart-of関係を示すポイントの設定(オブジェクトへのポイントをインスタンス変数に代入)を行うようなメソッドを定義しなければならない。このように、静的に定まる部品階層も、動的にしか構築できないため部品階層を成すオブジェクト間のメッセージの配送を全てプログラマが記述する必要がある。

クラス継承階層には、あるオブジェクトの属するクラスにおいて理解できない、すなわち対応するメソッドが定義されていないメッセージは、見かけ上、そのスーパークラスに処理を依頼する継承機能がある。一方、従来のOOLで部品階層を記述した場合、単にpart-of関係にあるオブジェクトへのポイントをインスタンス変数に持つだけで、part-of関係にあるようなオブジェクト間のメッセージの配送を自動的に行なう機構は存在しない。それゆえ、このような機能が必要ならばプログラマがアプリケーション毎に記述する必要がある。例えば、図1に示したウィンドウ中のPanelの領域内でマウスを左クリックしたときを考える。このとき、オブジェクトaPanelに左クリックされたことを伝えるメッセージleftClickが送られるとする(図2の破線矢印1)。このとき、aPanelがこのメッセージを理解できるならば、すなわち、クラスPanelかそのスーパークラスに対応するメソッドが定義されているならば、そのメソッドが起動され、メッセージleftClickに対するPanel独自の動作が行なわれる。しかし、leftClickに対する動作をaPanelを部品オブジェクトとするオブジェクトaCountryBrowserで行いたい場合には、aPanelが受け取ったleftClickを部品階層をさかのぼってaCountryBrowserに転送しなければならない(図2の破線矢印2)。これまでの従来のOOLで部品階層を記述した場合、このようなメッセージの配送を行いたいときには、leftClickを受信したとき、それをaCountryBrowserに転送するように、PanelのメソッドleftClickを定義しなければならない。以下に、メソッドleftClickの定義を示す。

リスト2. クラスPanelの定義の一部

```

=Panel:Window[whole contents]
+new { clearButton = [Button new];
      quitButton = [Button new];
      return([super new]); }
..... (中略) .....
//クラスPanelのメソッドleftClickの定義
-leftClick { [whole leftClick]; return self;}
//wholeは全体オブジェクトを指すインスタンス変数
..... (以下略) .....

```

このように従来のOOLで部品階層を記述した場合、全体オブジェクトにメッセージを転送してその処理を依頼するようなメッセージの上昇伝播機能を実現する場合、その途中にあるオブジェクトのクラスには、そのメッセージを部品階層において親にあたるオブジェクトに転送するだけのメソッドを定義しておく必要がある。

る。つまり、最終的にどの部品オブジェクトがそのメッセージを処理するかを、プログラマがプログラム記述時に意識しなくてはならない。X-toolkitやXViewなどのツールが対象としているXウィンドウシステムでは、このような機能をオブジェクト生成時に指定した親ウィンドウ(オブジェクト)へのイベントの上昇伝播(Propagate)と呼んで、実際に備えている[5]。しかし、従来のOOLは、そのような機構を言語の機能としては備えていない。

また、図1に示すようなウィンドウを閉じるとき、オブジェクトaCountryBrowserにメッセージcloseを送るとする。オブジェクトaCountryBrowserは、さらに自分の部品オブジェクトとなっているオブジェクト(countryList, cityListなど)にウィンドウを閉じることを指示するためcloseを送る。従来のOOLでこのようなことを記述すると、リスト1に示すクラスCountryBrowserのメソッドcloseの定義のように、部品階層を成す各オブジェクトのクラスに、その全部品オブジェクトへメッセージを配送するメソッドを定義しなければならない。ウィンドウを開くメッセージや、移動、サイズ変更などのメッセージも、このメッセージcloseと同様の定義を部品階層を成す各オブジェクトのクラスにする必要がある。このため、closeのように部品オブジェクト全てに配送したいメッセージが増えたときや、部品オブジェクトの数や名前が異なるアプリケーションを構築するときは、複数のクラスの定義の修正が必要となる。

次に、CountryListにおいてマウスの左クリックである国が選択されたときを考える。このとき、それに合わせてCityListの内容を選択された国の都市名のリストに書き換え、CityDataの表示内容を(都市名がまだ選択されていないので)消去する。このようなオブジェクト間の同期をSmalltalkやET++のMVCモデルで採用されている更新伝播(Change Propagation)[4][6]と呼ばれる機構を用いて行うとする。更新伝播によるオブジェクト間の同期は次のように行われる。あらかじめ決められている更新の指示を行うメッセージchangedの引数に、更新の対象となるウィンドウのオブジェクト名を渡し、全てのオブジェクトにそれを送る。そのメッセージを受け取ったオブジェクトのうち、引数と自分の名前が一致したオブジェクトのみが表示内容を更新する。このように更新伝播は、更新の対象となるオブジェクトを動的に探索する機構である。しかし、CountryBrowserのようなウィンドウアプリケーションのように部品階層が静的に定まっていて、更新の対象となるオブジェクト(この場合cityList, cityData)も静的に定まっている場合、このような方法は実行時にやり取りされるメッセージの量や実行時間の効率の点から見て好ましくないと考える。そこで、X-toolkitやXViewのように直接更新の対象となるオブジェクトcityList, cityDataに国名が選択されたことを知らせるメッセージselectedCountry:(選択された国名)を送信する方法を採用することを考える。この場合、オブジェクトcountryListのクラスCountryListのリストの項目がクリックで選択されたときに送られてくるメッセージselected:に対応するメソッドを、以下のようにオブジェクトcityList, cityDataにメッセージselectedCountry:を送るように定義しておく必要がある。

リスト3. クラスCountryListの定義の一部。

```

=CountryList:List[cityList,cityData]
..... (中略) .....
//メソッドselectedの定義。引数は選択された国名
-selected:country {
    [cityList selectedCountry:country];
    [cityData selectedCountry:country]; }
..... (以下略) .....

```

この方法を採用した場合、プログラム実行時に更新伝播の機構のように無駄なメッセージがやり取りされることはない。しかし、

この方法の欠点として次の2点が挙げられる。

- ・メッセージselectedCountry:を直接cityListやcityDataに送るためには、あらかじめcountryListがそれらへのポイントを得ておく必要がある。
- ・更新を指示するオブジェクトが増える(減る)などの機能の変更があった場合、countryListのクラスの定義を書き換える必要があり拡張性に欠ける。

従来の研究の問題点

以上のような問題点を解決するため、部品階層を成すオブジェクト間のメッセージの配送を自動的に行う機構を言語が備えることが考えられる。例えば、静的に定まるクラス間の関係を定義する文法を備えているオブジェクト指向言語DSM (Data Structure Manager) [11]では、クラス間に定義された関係を通してオブジェクトのコピーと消去および保存を行なうメッセージを全てのオブジェクトに伝播させる機能を持っている。しかし、これ以外の任意のメッセージを自動的に関係のあるオブジェクトに配送するような機能は備えていない。

また、文献[1]ではPart-Wholeジレンマ²を解決するために、Smalltalkを次のように拡張している。

- 部品オブジェクトを特別なインスタンス変数に格納し、部品オブジェクトにはそのインスタンス変数の名前前でアクセスできるようにする。

これを図2に示すウィンドウの部品階層の例を用いて説明する。この部品階層の根であるaCountryBrowserの部品オブジェクトであるcountryList、cityListがそれぞれインスタンス変数countryList、cityListに格納されているとする。このとき、オブジェクトcountryListへのポイントは、以下に示すようにオブジェクトaCountryBrowserにメッセージcountryListを送れば得られる。

aCountryBrowser countryList.

- 部品オブジェクトにアクセスするための特別なメッセージ(compoundメッセージ)を導入する。

Smalltalkにおける通常のメッセージをsimpleメッセージとするとcompoundメッセージは、<path>.<simpleメッセージ>の形式で書かれる。ここでpathは部品オブジェクトを格納したインスタンス変数の名前を、この部品階層の根の方から"."で区切って列挙したものである。例えば、図2においてオブジェクトclearButtonにメッセージisPushedを送るときには、

aCountryBrowser aPanel.clearButton.isPushed

↑ ↑
path simpleメッセージ

とする。また、このpathは部品階層に現れる途中の部品オブジェクト(を格納したインスタンス変数)の名前を列挙する必要はない。上で示したcompoundメッセージは、

² オブジェクトの情報隠蔽の原則からすれば、オブジェクトを構成する部品オブジェクトは内部状態(インスタンス変数)として外部には見せないようにして、部品オブジェクトへのアクセスは全体オブジェクトが仲介すべきである。しかし、このとき全体オブジェクトは、部品オブジェクトが理解できる(そのクラスに対応するメソッドが定義されている)メッセージをすべて理解する必要がある。例えば、図2に示すウィンドウの部品階層では、オブジェクトaCountryBrowserは部品オブジェクトcountryList、cityList、cityData、aPanel、clearButton、quitButtonが理解できるメッセージを全て理解することが必要がある。一方、部品オブジェクトを全体オブジェクトの内部状態としなければ、外部から部品オブジェクトに直接メッセージを送って何かを要求したり部品オブジェクトを修正したりできる。これは全体オブジェクトの完全性、すなわちオブジェクトの情報隠蔽の原則を破っていることになる。これを文献[1]ではPart-Wholeジレンマと呼んでいる。

aCountryBrowser clearButton isPushed.

と書けばisPushedは部品オブジェクトclearButtonまで配送される。このような機能を文献[1]ではメッセージのフォワーディング(forwarding)と呼んでいる。全体オブジェクトのクラスにインスタンスを生成するメッセージnewを送ったとき、メッセージのフォワーディングを用いて全ての部品オブジェクトの属するクラスにnewを送り、部品階層を構築することができる。メッセージフォワーディングは、path上の途中の部品オブジェクト(を格納したインスタンス変数)の名前は省略できるが、最終的な送り先の部品オブジェクト名を略すことはできない。そのため、全ての部品オブジェクトに同じメッセージを送る場合でも、部品オブジェクトが理解できないメッセージを全体オブジェクトへ自動的に転送する場合でも、最終的にどの部品オブジェクトがそのメッセージを処理するかを、プログラマがプログラム記述時に意識しなくてはならないことに変わりはない。それゆえ、本節で述べたような問題点を解決できているとは言えない。

2.3 ドメインオブジェクト (Domain Object)

2.2節で述べたような問題点を解決するために、オブジェクト間のpart-of関係を確立し保持する仕組みを導入して、各オブジェクトがインスタンス変数にpart-of関係にあるオブジェクトへのポイントを持たないようにする。そして、この仕組みに以下の2つの機能を持つオブジェクト間のメッセージの配送を自動的に行う機構を持たせることを考える。一つ目の機能は、部品オブジェクトが理解できないメッセージを全体オブジェクトへ自動的に転送するメッセージの上昇伝播機能であり、二つ目は、メッセージを受信したオブジェクトの部品オブジェクト全てに同じメッセージを配送するメッセージの放送機能である。

この仕組みを本稿ではドメインオブジェクトと呼ぶ。図3に、図2に示すウィンドウの部品階層をドメインオブジェクトによって確立、保持するときのドメインオブジェクトとオブジェクトの間の関係の概略を示す。各部品オブジェクトには、対応するドメインオブジェクトが必ず1個存在する。このとき、部品オブジェクトは、それに対応するドメインオブジェクトに属すると言う。例えば、図3においてオブジェクトaCountryBrowserに対応するドメインオブジェクトは、aCB_Domainであり、このときaCountryBrowserは、aCB_Domainに属する。

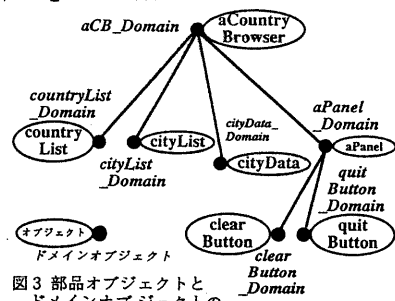


図3 部品オブジェクトとドメインオブジェクトの間の関係の概略。

1個のドメインオブジェクトは、部品階層の1段を保持する。例えば、図3においてaCountryBrowserが属するaCB_Domainは、aCountryBrowserを頂点とし、オブジェクトcountryList、cityList、cityData、aPanelを葉とする部分木を保持している。また、同様に、aPanelが属するドメインオブジェクトaPanel_Domainは、aPanelを頂点とし、オブジェクトclearButton、quitButtonを葉とする部分木を保持している。属するオブジェクトは、自分を部品オブジェクトとしている全体オブジェクトや部品オブジェクトが何かということ

は知らない（それらへのポインタをインスタンス変数に持たない）。一方、個々のドメインオブジェクトは、自分に属するオブジェクト、及び階層上親と子に当るドメインオブジェクトを知っている。このように部品階層をドメインオブジェクトの階層で実現する。この階層において、あるドメインオブジェクトの親に当るドメインオブジェクトをスーパードメインオブジェクト、子に当るドメインオブジェクトをサブドメインオブジェクトと呼ぶ。例えば、図3においてドメインオブジェクトaPanel_Domainは、clearButton_Domainのスーパードメインオブジェクトであり、かつaCB_Domainのサブドメインオブジェクトである。

2. 3. 1 ドメインオブジェクトのメッセージ配送機構

2. 2節で述べた問題点を解決するためにドメインオブジェクトに部品階層を成すオブジェクト間のメッセージの配送を自動的に行う機構を持たせる。この機構を本稿ではドメインオブジェクトのメッセージ配送機構と呼ぶ。本節では、このメッセージ配送機構に必要と考えられる機能について述べる。

メッセージ配送機構の機能

メッセージを直接送りたいオブジェクトに送るのではなく、それが属するドメインオブジェクトに送ることで、ドメインオブジェクトのメッセージ配送機構を用いたメッセージの自動配送を行う。そのためにドメインオブジェクトのメッセージの配送には、次の3通りを用意する（図4）。

- [1] 属するオブジェクトへの配送 (delivery)
- [2] スーパードメインオブジェクトへの転送 (forwarding)
- [3] サブドメインオブジェクトへの配送

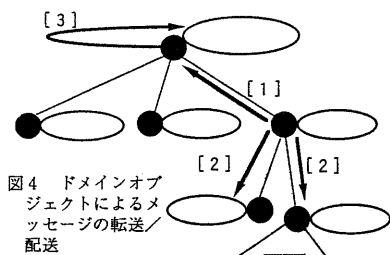


図4 ドメインオブジェクトによるメッセージの転送/配送

2. 2節で述べた問題点を解決するために、メッセージ配送機構に次のA. B.の機能を持たせる。

A. メッセージの上昇伝播機能。

オブジェクトがメッセージを受け取った場合、それに対応するメソッドがオブジェクトのクラスに見つからなければ、スーパークラスのメソッドを探すために、メッセージは見かけ上クラス継承階層をさかのぼっていく。この機能は、これと同様の機構を部品階層において実現する機能である。この機能に全体オブジェクトへのメッセージの転送を自動的に行なうので、プログラマがメッセージを転送するためだけのメソッドを定義する必要がなくなる。具体的なメッセージの配送は、次のように行われる。

- (1) ドメインオブジェクトは、受信した、あるいは転送されてきたメッセージをまず属するオブジェクトに配送する。属するオブジェクトがこのメッセージを理解できなければスーパードメインオブジェクトに転送する。
- (2) 転送は、部品階層の根に当るドメインオブジェクトまで行う。それまでに配送されたどのオブジェクトも理解できないメッセージは、エラーとなる。

B. メッセージの放送機能。

メッセージを受信したドメインオブジェクトとそのサブドメイ

ンオブジェクトに属する全てのオブジェクトにメッセージを配送する機能である。この機能を利用したいときには、メッセージに何等かのオプションを付けさせ、他の（普通の）メッセージを区別することとする。例えばリスト1に示したクラスCountryBrowserのメソッドcloseの定義はリスト4のようになる。この定義は、メッセージcloseはドメインオブジェクトaCB_Domainとそのサブドメインオブジェクト（cityList_Domain, country_Domain, clearButton_Domain, quitButton_Domain等）に属するオブジェクト全てにメッセージcloseを送ることを意味している。

リスト4. クラスCountryBrowserの定義の一部（変更後）。

```
=CountryBrowser : Frame
(cityList,cityData,cityData,panel,path,fname) //インスタンス変数
..... (中略) .....

//メソッドcloseの定義
-close{ [aCB_Domain close ALL]; }
..... (以下略) .....
```

この例では、“ALL”が放送機能を使ってメッセージを配送することを要求するオプションである。

メッセージの転送/配送の方式

メッセージの転送/配送は、通常のメッセージ送信として行う。すなわち、メッセージに対応するメソッドの実行に必要な情報（インスタンス変数など）は、メッセージを配送されたオブジェクトのものを使う。これをデリゲーション (delegation[7]) で行う⁵ 場合、次のような問題が生じる。クラス継承の場合、スーパークラスにデリゲートしたメッセージに対応するメソッドの実行に必要な情報は、そのメッセージを最初に受信したオブジェクトが継承して持っている。しかし、部品階層を成すオブジェクトは、一般に異なるクラスのインスタンスである。それゆえ、部品階層において上位に位置するオブジェクトのメソッドの実行に必要な情報を、そのメッセージを最初に受信した下位のオブジェクトが持っているとは限らない。また、例えば偶然にも同名のインスタンス変数を持っていたとしても、その意味や型が異なる場合がある。このような理由から、メッセージの転送/配送をデリゲーションで行う場合、必ずしもメソッドの正しい実行が行われるとは言えない。

2. 3. 2 プログラム実行時のメッセージ配送

プログラム実行時に部品階層を確立し、保持するドメインオブジェクトがメッセージ配送機構を持つことにより、そのような機能をプログラマがアプリケーション毎に記述する必要を無くすることができる。しかし、ドメインオブジェクトのメッセージ配送機構は、意味的には動的なメソッド探索である。ドメインオブジェクトは、2. 3. 1節で述べた機能を用いて、部品階層を成しているオブジェクトの中から、ドメインオブジェクト宛に送られてきたメッセージを理解できる、すなわちそのクラスがメッセージに対応するメソッドを定義している、あるいは継承しているオブジェクトを探す。これは、クラス継承階層において、あるオブジェクトTOが受け取ったメッセージに対応するメソッドを、TOのクラスCの定義において発見できなければ、クラス継承階層をさかのぼってCのスーパークラスを探索することに相当する。しかし、このようなメソッド探索を、動的には起こらないようにできる。なぜならドメインオブジェクトの階層で確立している部品階層は、静的に定まるものであり、かつメッセージ配送機構の機能は上昇伝播機能と放送機能の2種類だけであり、それぞれメッセージの配送の仕方が決っている。それゆえドメインオブジェクトは、自

⁵メッセージに対応するメソッドの実行に必要な情報（インスタンス変数など）は、最初にそのメッセージを受信したオブジェクトのものを使う。

分宛に送られてきたメッセージが最終的に配送されるべきオブジェクトを、それぞれのクラス定義を調べることでコンパイル時に知ることができる。以上のことから（普通の）オブジェクトからドメインオブジェクトへのメッセージ送信は、コンパイル時に全て、メッセージ配送機構によって最終的に配送されるオブジェクトへのメッセージ送信に置き換えることができるからである。このようなことを行うためにはコンパイル時に、表1のようなディスパッチテーブルをドメインオブジェクト毎に作成しておけば良い。ディスパッチテーブルには、上昇伝播機能、放送機能によってメッセージが配送され得るオブジェクトが理解できるメッセージ毎に、それを理解するオブジェクトで、かつ

1. 上昇伝播機能によって配送されるオブジェクト（A）及び
2. 放送機能によって配送されるオブジェクトのリスト（B）

を記載しておく。上昇伝播機能の場合は、列（A）記載されたオブジェクトに直接配送すれば良く、また放送機能の場合は、列（B）のリストにある全てのオブジェクトに送信すれば良い。また、この表に載っていないメッセージがドメインオブジェクトに送られてきたときは、不当なメッセージとして直ちにエラーとできる。

表1. ドメインオブジェクトのディスパッチテーブル

メッセージ	(A)	(B)
msg 1	obj11	obj11,obj14,....
msg 2	obj25	obj22,obj24,....
.....		
msg n	objn1	objn5,objn6,....

以上のことから、ドメインオブジェクトのメッセージ配送機構の上昇伝播機能や放送機能は、意味的には動的なメソッド探索であるが、実際にはメッセージを理解できるオブジェクトにのみ直接配送するので、無駄なメッセージのやり取りは行われなと言え。

2. 3. 3 メッセージ配送機構を用いた メッセージパッシング

プログラム実行時に、他のオブジェクトに直接メッセージを送りたいときには、2. 2節でも述べたようにあらかじめ送信先のオブジェクトへのポインタを何等かの方法で得ておく必要がある。そこで、メッセージパッシングも全てメッセージ配送機構を用いて行うことを考える。この場合、オブジェクトは他のドメインオブジェクトに属するオブジェクトへ送るメッセージも自分が属するドメインオブジェクトに送る。このメッセージは、自分では理解できないからドメインオブジェクトのメッセージ配送機構によって他のドメインオブジェクトやオブジェクトに配送してもらうことになる。

例えば、図5においてオブジェクトaPanelがオブジェクトaCountryBrowserにメッセージleftClickを送りたいとする。このとき、aPanelは、leftClickを自分が属するドメインオブジェクトaPanel_Domainに送る（図5の実線矢印）。aPanel_Domainは、自分に属するオブジェクトaPanelにそのメッセージを配送する（図5の破線矢印1）。しかし、クラスPanelおよびそのスーパークラスにはleftClickに対応するメソッドが定義されていない（定義されていればクラスPanelにおいてleftClickが処理されてしまう）から、aPanelDomainによってスーパードメインオブジェクトaCB_Domainに転送される（図5の破線矢印2）。そして、aCB_DomainによってleftClickは、それに属するオブジェクトaCountryBrowserに配送される（図5の破線矢印3）。また、2. 3. 2節で述べたように、

実際には図5の破線矢印4で示すように配送される。

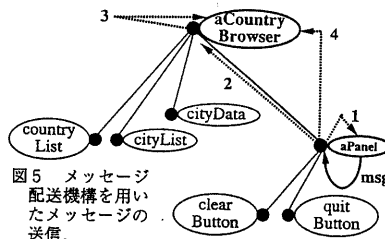


図5. メッセージ
配送機構を用い
たメッセージの
送信。

メッセージ配送機構によるメッセージの配送は上昇伝播機能、放送機能の2通りなので、メッセージの送り先は制限される。しかし、普通のメッセージパッシングもメッセージ配送機構を用いて行うようにすれば、次のような利点が得られる。

- (1) メッセージのやり取りが減少する。

2. 2節で述べた動的に表示内容の更新の対象を探索する更新伝播機構を用いた場合より、ドメインオブジェクトのメッセージの配送機構を用いる方がオブジェクト間でやり取りされるメッセージの量が減少するので実行時間の効率もあがると考えられる。例えばメッセージの配送機構を用いた、更新の他のオブジェクトへの通知は次のように行われる。Country Listにおいてユーザがマウスのクリックによってある国名を選択した場合、部品階層の根に相当するドメインオブジェクトaCB_DomainにメッセージselectedCountry:(選択された国名)を放送機能を用いて配送してくれるようにオプションを付けて送る。よって、リスト3に示したクラスCountryListのメソッドselected:の定義は次のようになる。

リスト5. クラスCountryListの定義の一部（変更後）。

```
=CountryList:List{cityList,cityData}
..... (中略) .....
//メソッドselectedの定義。引数は選択された国名
-selected:country {
    [/ selectedCountry:country ALL];
    //"/は、部品階層の根に当るドメインオブジェクトを示す。"ALL"は放送機
    //能を用いて配送することを指示するオプション
    ..... (以下略) .....
```

ドメインオブジェクトのメッセージ配送機構によって、メッセージselectedCountry:countryは、ドメインオブジェクトaCB_Domainとそのサブドメインオブジェクトに属する全てのオブジェクトに配送される。そこでメソッドselectedCountry:をオブジェクトcityList及びcityDataのクラスに定義しておけば、配送されたメッセージselectedCountry:countryに対応して各ウィンドウの表示内容の更新が行える。また2. 3. 2節で述べたように、このようなメッセージの配送は、プログラム実行時には、リスト3のselected:の定義のようにcityList, cityDataに直接メッセージselectedCountry:を送るようにできるので、メッセージのやり取りの量は、更新伝播機構を用いるよりも改善されている。

- (2) 機能の拡張が容易である。

ドメインオブジェクトのメッセージの配送機構を用いてメッセージパッシングを行うようにすれば、各オブジェクトのクラスの定義には、特定のオブジェクトやドメインオブジェクトにメッセージを送信する記述が無くなる。ゆえに他のウィンドウのオブジェクトの名前や、更新等の指示をどのオブジェクトに通知しなければならないか等に依存することなくクラスを設計できるので、新しいウィンドウを追加するなどの機能の追加が容易になる。例えば、次に述べるような機能を持つMapウィンドウをCountry Browserに追加することを考える。

Mapウィンドウ

世界地図を表示するウィンドウである。Country Listで国名が選択されたとき、該当する国の位置をブリンクで表示する。また、さらにCity Listで都市名が選択されたときはブリンクしていた国の拡大図を表示して、該当する都市の位置を×印で示す。

このとき、特定のオブジェクトやドメインオブジェクトにメッセージを送信する記述をクラスの定義に記述していた場合、オブジェクトcountryListやcityListのクラスの定義も変更して、Country ListやCity Listで国名や都市名が選択されたとき、MapウィンドウのオブジェクトaMapにもそれを通知するようにならなければならない。しかし、クラスCountryBrowserにの定義において、メッセージの送信を全てメッセージ配送機構を用いて行うようにしていれば、aMapのクラスを変更、あるいはサブクラスを設計して、selectedCountry:やCity Listで都市名が選択されたことを通知するメッセージselectedCity:(選択された都市名)に対応するメソッドを定義するだけで、他のcountryList、cityLane、cityDataのクラスの定義を変更する必要はない。すなわち機能を通知するときは、その機能を実現するクラスを設計して部品階層に組み込むだけで良く、既存の機能(クラス)を変更する必要はない。

2. 4 部品階層の記述の問題点 (2)

従来のオブジェクト指向言語では、あるオブジェクトの部品オブジェクトの種類(クラス)や数を、そのクラスの定義中に記述していた。例えば、図2の部品階層を従来の記述方法で記述する場合、リスト1に示すように、クラスCountryBrowserには部品オブジェクトを格納するためのインスタンス変数countryList、cityList、cityData、panelなどを定義しておく必要がある。また、メッセージnewに対応するメソッドnewは、CountryBrowserのインスタンスaCountryBrowserを生成するときに部品オブジェクトも全て生成するよう記述しなければならない。さらに、オブジェクトaPanelも部品オブジェクトとして2個のクラスButtonのインスタンスを持つので、それを格納するためのインスタンス変数clearButton、quitButtonを宣言しておく。また、クラスPanelのメソッドnewも、リスト2に示すように、インスタンス生成時に部品オブジェクトであるButtonのインスタンス2個を生成するように記述しておく。

同様に、各部品オブジェクトが更にいくつかの部品を持つ場合、そのクラスに部品オブジェクトの数だけのインスタンス変数を宣言し、それらの部品オブジェクトを全て動的に生成するようなメソッドを定義しなければならない。このように、従来のOOLで部品階層を記述した場合、一つの部品階層の記述が複数のクラスの定義中に分散してしまう。このため、プログラムが実行時に構築する部品階層を知るためには、複数のクラスの定義中に分散している部品階層を構築する記述から読み取らねばならない。

従来の研究の問題点

この様な問題点を解決するため、静的に定まる部品階層をクラスの定義とは独立に、まとめて記述する文法を言語が備えることが考えられる。

OMT (Object Modeling Technique) [8]は、クラスの設計のための手法である。OMTダイアグラムを用いれば、メソッドやインスタンス変数の名前だけでなく、クラス間の関係もあわせて記述できる。2. 2 節でも触れたDSMは、CAE/CADシステム開発のために設計されたC言語を拡張したオブジェクト指向言語であり、オブジェクト間の関係としてOMTの持つクラス間の関係を記述する枠組を備えている。この枠組はモジュール (module) と呼ばれる。文献 [11]においてモジュールは関係のあるオブジェクトのバッケー

ジングの単位と定義されている。その記述は、クラスの定義と、それらの間の関係の定義の記述 (is-a, part-of, ユーザ定義の関係) から成る。この様に、DSMは静的に定まるクラス間の関係を定義する文法は備えているが、部品階層の記述が複数のモジュールに分散する点で、部品階層の記述が複数のクラス定義に分散する問題を解決できているとは言えない。

2. 5 ドメイン (Domain)

部品階層の記述が複数のクラス定義に分散する問題を解決するため、静的に定まる部品階層は一箇所にまとめて記述できるようにする。“静的に定まる部品階層を記述する”ことは、ここでは2. 3 節で述べた“ドメインオブジェクトの定義を記述する”ことである。ドメインオブジェクトの定義を本稿ではドメイン記述と呼ぶ。このときドメインは部品階層において、クラス階層におけるクラスに相当し、ドメインオブジェクトはそのインスタンスに相当する。ドメイン記述

ドメイン記述は、図6に示すとおり0個以上の属するオブジェクトと0個以上のサブドメインオブジェクトの宣言から成る。サブドメインオブジェクトのドメイン (これをサブドメイン⁴と呼ぶ)の指定方法には、図6に示すように(a)既存のドメインの定義を再利用する方法と、(b)新たにドメインを定義する方法と、(c)それらを組み合わせた方法がある。この様にサブドメインオブジェクトを再帰的に定義することで部品階層を記述する。初期化メッセージ列には、生成された属するオブジェクトに最初に送るべきメッセージの列 (例えば、ウィンドウのオブジェクトならばサイズの指定を行うメッセージなど) を記述しておく。クラスの定義で言えば、ドメイン名はクラス名、属するオブジェクトは、インスタンス変数やメソッドの定義に相当する。また、クラスを定義するときは、既存のクラスの定義を継承して再利用するために、クラス継承階層において親となるクラスをスーパークラスとして指定する。一方、ドメイン記述においては、サブドメインとして既存のドメインを指定することで、ドメイン記述を再利用する。

あるドメインに属するオブジェクトとそのサブドメインに属するオブジェクト間には、part-of関係が成立する。そして、サブクラスを再帰的に定義することでクラス継承階層を記述するように、サブドメインを再帰的に定義することによって静的に定まる部品階層を記述する。

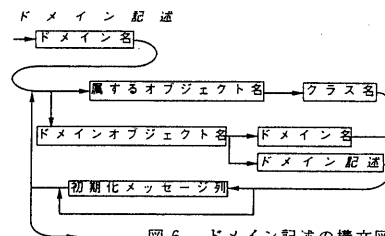


図6 ドメイン記述の構文図

ドメインオブジェクト・オブジェクトの名前の表記方法

ドメイン記述やクラス定義の中で、ドメインオブジェクトやオブジェクトの名前の名前の指定は相対表記できるようにする。例えば次の様な記述において

../countryList

..は、スーパードメインオブジェクトを意味する。また、../cityListは、スーパードメインオブジェクトに属するオブジェクトあるいは

⁴ 同様にスーパードメインオブジェクトのドメインのことをスーパードメインと呼ぶ。

はサブドメインオブジェクトであるcityListを意味する。このように、ドメインオブジェクトやオブジェクトの名前の相対表記は、UNIXにおけるディレクトリを指定する記法にならった記法を用いる。

2. 5. 1 アプリケーション"Country Browser"の部品階層のドメイン記述

Country Browserの部品階層(図2)をドメインを用いて記述したプログラムをリスト6に示す。(1),(2)等はそれぞれブロックの番号であり、以下の説明の中で用いる。オブジェクトのクラスの指定は、CLASSの後に書く。生成したインスタンスに送る初期化メッセージの指定は、INITの後に(msg1; msg2;)の形式で記述する。また、ドメイン名はDOMAINの後に書く。

リスト6. 図2に示す部品階層のドメインを用いた記述。

DOMAIN CB_Domain //ドメイン名 (1)	
aCountryBrowser = //属するオブジェクト名	
{CLASS CountryBrowser; //クラス名 (2)	
INIT {label("Country Browser");	
open;}; //初期化メッセージ列	
};	
countryList_Domain = // (サブ) ドメインオブジェクト名	
{DOMAIN CountryList_Domain //ドメイン名 (4)	
countryList = {CLASS CountryList}	
INIT {.....初期化メッセージ列.....};	
(3);	
cityList_Domain =	
{DOMAIN CityList_Domain	
countryList = {CLASS CountryList;	
INIT {.....初期化メッセージ列.....};	
};	
cityData_Domain =	
{DOMAIN CityData_Domain	
cityData = {CLASS TextPane;	
INIT {.....初期化メッセージ列.....};	
};	
aPanel_Domain =	
{DOMAIN Panel_Domain	
aPanel = {CLASS Panel};	
quitButton_Domain =	
{DOMAIN QuitButton_Domain	
quitButton = {CLASS QuitButton};	
};	
clearButton_Domain =	
{DOMAIN ClearButton_Domain	
clearButton = {CLASS ClearButton};	
};	
(1);	

ブロック(1)

アプリケーション全体に対応するドメインCB_Domainを定義している。このドメインには、CountryBrowserクラスのインスタンスaCountryBrowserと、サブドメインオブジェクトとしてドメインCountryList_DomainのドメインオブジェクトcountryList_Domain、ドメインCityList_DomainのドメインオブジェクトcityList_DomainおよびドメインCityData_DomainのドメインオブジェクトcityData_Domainが属している。

このドメイン記述が記述している部品階層は、ドメイン

CB_Domainのドメインオブジェクトを以下の様に宣言することによって、プログラム実行時にCB_Domainのドメインオブジェクトを生成することで確立される。

CB_Domain aCB_Domain;

ドメインオブジェクト及びオブジェクトは、ドメイン記述で宣言された順に生成される。この例の場合、まずドメインオブジェクトaCB_Domainが生成された後、(最初に宣言されている)それに属するオブジェクトaCountryBrowserが生成される。続いてサブドメインオブジェクトcountryList_Domainが生成され、そのドメイン記述で(最初に宣言されている)それに属するオブジェクトcountryListが生成される。ドメインCountryList_Domainにさらにサブドメインオブジェクトが宣言されていれば、同じようにそれに属するオブジェクト及びサブドメインオブジェクトが生成される。その後、サブドメインオブジェクトcityList_Domain、cityList_Domainの順に生成される。また、指定された初期化メッセージは、次節で述べるinitializeのように他のオブジェクトのポイントが必要なメッセージもあるので、オブジェクトの生成が全て終了後、オブジェクトの生成順と同じ順序でそれぞれに送られるとする。

ブロック(2)

ドメインCB_Domainに属するオブジェクトを宣言している。オブジェクトaCountryBrowserはクラスCountryBrowserのインスタンスである。

ブロック(3)

ドメインCountryBrowserのサブドメインオブジェクトcountryList_Domainを宣言している。ブロック(4)においてそのドメインCountryList_Domainを定義している。

この場合のクラスCountryBrowserの定義の一部をリスト7に示す。Country Browserの部品階層をドメインに記述したのでリスト1にあった部品オブジェクトを生成するメソッドnewは無くなっている。また、部品オブジェクトへのポイントを格納するインスタンス変数countryList, cityList, cityData, panelも不要になった。

リスト7. クラスCountryBrowserの定義の一部(変更後)

```
=CountryBrowser : Frame {path,filename} //インスタンス変数
//メソッドcloseの定義
-close{ [aCB_Domain close ALL]; }

-rightClick{ ..... (略) ..... }
-leftClick{ ..... (略) ..... }
..... (以下略) .....
```

2. 5. 2 メッセージ配送機構で行えないメッセージパッシング

この例では、全てのメッセージパッシングをドメインオブジェクトのメッセージ配送機構の上昇伝播機能、放送機能を用いて行うことができた。しかし、配送機構による配送だけでは適切なオブジェクトにメッセージを送信できない場合もあり得る。そのようなときは、従来のようにメッセージを送りたいオブジェクトやドメインオブジェクトへのポイントをあらかじめ得ておいて、直接メッセージを送るように記述する必要がある。そのために、次のようなメッセージ(ここでは、このメッセージの名前をinitializeとする)を考える。initializeは、その引数にオブジェクトがプログラムの実行時に直接メッセージをやり取りするなど、そのポイントを必要とするドメインオブジェクトやオブジェクトの名前を記述する。initializeの引数に記述されたドメインオブジェクトあるいはオブジェクトの名前は、コンパイル/リンク時にそれらへのポ

インタに変換される。そこで、引数に適当なドメインオブジェクトやオブジェクトの名前を指定したinitializeを、2.5節で述べた属するオブジェクトへの初期化メッセージとして次のように記述しておく。

リスト8. メッセージinitializeの使用例。

```
DOMAIN CB_Domain //ドメイン名
aCountryBrowser = //属するオブジェクト名
{CLASS CountryBrowser; //クラス名
INIT (label("Country Browser"); open;
      initialize(..countryList,..cityList,..cityData);
};
```

..... (以下略)

そしてそのオブジェクトのクラスの定義を行う際に、メソッドinitializeの定義中で引数をインスタンス変数に格納するようにしておけば、そこにドメインオブジェクトあるいはオブジェクトへのポインタが代入される。この様に、メソッドの記述中にはドメインオブジェクトとオブジェクトを区別することなくメッセージを送るように書くことができる。例えば、オブジェクトaCountryBrowserがオブジェクトcountryList, cityList, cityDataに直接メッセージを送るようにしたいときは、クラスCountryBrowserのメソッドinitializeを以下のように定義しておけばよい。

```
-initialize(leftList,rightList,dataText){
    countryList = leftList;
    cityList     = rightList;
    cityData     = dataText;      (以下略)}
```

//countryList, cityList, cityDataはインスタンス変数

もし、メッセージの送り先がドメインオブジェクトであった場合には、ドメインオブジェクトによってそのドメインオブジェクトに属するオブジェクトにメッセージが配送される。このようなメッセージを導入することの利点は、クラス定義の記述とドメイン記述の独立性が高くなることである。これは、部品階層の記述(ドメイン記述)が変更されても、このメッセージの引数に変更後の適切なドメインオブジェクトあるいはオブジェクトの名前を記述すればクラスの定義は全く変更する必要がないからである。また逆に、ドメイン記述はあくまで実行時のオブジェクトの部品階層の記述であり、initializeの引数に渡したドメインオブジェクトやオブジェクトをクラスの定義中でどの様に使用しているかは全く無関係になるからである。

3 考察

2.3.3節で述べたように、ドメインオブジェクトのメッセージ配送機構をメッセージパッシングに用いた場合、メッセージのやり取りの減少や機能の拡張が容易になるといった利点がある。ところで、ドメイン記述に記述するのは、静的に定まる部品階層である。それゆえ部品オブジェクトが成す構造が階層構造でない場合は、それを階層になるように変更する必要がある。この際に部品階層をどのように構築するか、すなわち各オブジェクトをそれぞれどのオブジェクトの部品オブジェクトとするか、という問題が生じる。メッセージの配送機構を用いることによる利点を生かしたいと考え、配送機構の上昇伝播機能、放送機能によってメッセージパッシングが行えるように部品階層を構築する必要がある。しかし、実行時のメッセージの配送に合わせて部品階層を構築した場合、直観的・視覚的なpart-of関係とは異なるpart-of関係が作られることもあり得る。クラス継承階層においても、新しいクラスを定義するとき、それをどのクラスのサブクラスとすべきかというこれと同様な問題が存在する。

2.5節で述べたように、本稿で提案する枠組は、静的に定ま

る部品階層をドメイン記述として一箇所にまとめて記述できる方法を提供する。このように部品階層の記述をクラスの定義と分けて行い、かつ2.3.3節で述べたように全てのメッセージパッシングをドメインオブジェクトのメッセージ配送機構を用いて行うならば、部品オブジェクトの種類(クラス)や数が異なるアプリケーションにもそのクラスがそのまま再利用できる。例えば、図2においてFrameウィンドウを表すオブジェクトaCountryBrowserは、そのクラスCountryBrowserに、部品オブジェクトがListウィンドウを表すオブジェクト2個、Textウィンドウを表すオブジェクトおよびPanelを表すオブジェクトを各々1個であることを定義している。このような部品オブジェクトに関する記述をクラスの定義とは別にドメイン記述として記述しておけば、例えばTextウィンドウを2個持つようなFrameウィンドウが必要なアプリケーションや、FrameウィンドウがTextウィンドウではなくCanvasウィンドウを持つアプリケーションやListウィンドウを1個しか持たないアプリケーションのように、部品階層が異なるだけのアプリケーションにそのまま再利用できる。つまり、本稿で提案する枠組においては、オブジェクト間のpart-of関係は個々のアプリケーションに依存すると考え、それをクラス間の関係として定義するのではなくインスタンス間の関係としてアプリケーション毎にドメイン記述として記述する。

ドメイン記述は複数のオブジェクトから成る部品階層を記述したモジュールと言える。また、実行時には、異なるドメインオブジェクトに属するオブジェクトへのメッセージの送信はメッセージ配送機構によって行えるので、そのドメインオブジェクト及びサブドメインオブジェクトに属する個々のオブジェクトに直接アクセスするのではなく、それらが属するドメインオブジェクトを介してアクセスするようにできる。それゆえドメインは、アプリケーションの(一般に複数のオブジェクトから成る)部品毎の情報隠蔽を実現することができる。すなわち、複数のオブジェクトをまとめたものに部品としての独立性、再利用性を持たせることができる。例えば、図1に示すウィンドウ(w1)とそれとは異なる部品階層を成すウィンドウ(図7のw2)から成るウィンドウを持つようなウィンドウアプリケーション(図7のw3)を考える。

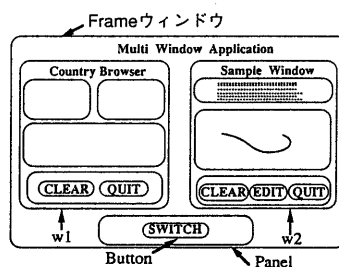


図7 ウィンドウアプリケーションの例(w3).

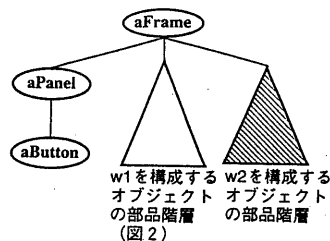


図8 w3(図7)を構成するオブジェクトの部品階層。

このとき、w3のFrameウィンドウは、w1とw2を部品として持つ。ゆえに、w3の部品階層は、w3のFrameウィンドウを表すオブジェクトを根とし、w1及びw2の部品階層を部分木とするような階層になる(図8)。

よってこの2枚のウィンドウの部品階層を記述したドメインを、w3の部品階層を実現するドメインのサブドメインとして再利用できる。w1の部品階層を記述したドメイン記述をリスト6に示したCB_Domain、w2の部品階層を記述したドメインをSW_Domainとすると、w3の部品階層を記述したドメインMWA_Domainの記述はリスト9のようになる。

リスト9. ドメインMWA_Domainの定義。

```
DOMAIN MWA_Domain {
    aFrame = { CLASS Frame };
    aCountryBrowser = { DOMAIN CB_Domain };
    aSampleWindow = { DOMAIN SW_Domain };
    aPanel_Domain =
    { DOMAIN Panel_Domain
        aPanel = { CLASS Panel };
        aButton_Domain =
        { DOMAIN Button_Domain
            switchButton = { CLASS SwitchButton }
        }
    }
}
```

リスト9において、下線部分が既存のドメインCB_Domain、SW_Domainの定義を用いてサブドメインオブジェクトを宣言している部分である。このように、ドメインは複数のオブジェクトから成るモジュールとして再利用することができる。

今後の課題としては、実用上メッセージの配送が2通りで充分か否かを検討することがある。もし、配送の種類が2通りでは実用上不十分であるならば、

- ドメイン記述時にドメイン毎に配送方法を指定できるようにする、
- プログラム実行中にもドメイン毎に配送方法を変更できるようにする、
- メッセージごとに配送方法を指定できるような記法を導入するなどの改善策が考えられる。

また、部品階層が完全には静的に定まらず、その一部が動的に変化するような場合をドメインでどのように実現するかということを検討する必要もある。

一部分が動的に変わ得る部品階層をドメインで実現する場合や、メッセージの配送方法を上のb.で述べたように実行中にも変更できるようにした場合、メッセージ配送機構が効率良く働くように実現すること考慮する必要がある。メッセージ配送機構は、具体的には2.3.2節で述べたようなメッセージの配送先を記述したディスパッチテーブルを、コンパイル時にドメインオブジェクトごとにつくることによって実現できる。実行時に部品階層やメッセージの配送方法を変更するためには、このディスパッチテーブルを動的に書き換える必要がある。このテーブルの書き換えを効率良く行わなければ、アプリケーションプログラム全体の実行時間の効率が低下する。また、c.で述べたようにメッセージごとに配送方法を指定できるようにすると、同じメッセージでも配送方法が異なる場合があるので、2.3.2節で述べたような方法では直接最終的に配送されるべきオブジェクトにメッセージを送るようにすることはできない。

4 あとがき

本稿では、まず従来のオブジェクト指向言語が、静的に定まる

部品階層を記述するための枠組を言語の機能として備えていないことによる問題点を指摘した。そこで本稿では、そのような枠組としてドメインを提案した。今後はドメインを、従来のオブジェクト指向言語に対して実装を行ない、より多くの実用例でドメインの問題点、および有用性の検証を行いたい。

謝辞

11頃ご討論頂く首藤研究室の皆様には感謝致します。本稿は一部文部省科学研究費補助金・奨励研究(A)(03780256)、一般研究(C)(03680030)、重点領域研究1(X)(02249102)、総合研究(A)(02302047)、マツダ研究助成金、及び稲盛財団研究助成金の補助を受けている。

参考文献

- [1]Blake,E. and Cook,S.: "On Including Part Hierarchies in Object-Oriented Language, with an Implementation in Smalltalk", ECOOP'87, Lecture Notes in Computer Science, Springer-Verlag, New York, pp.41-50 (1987).
- [2]Cox,B.J.: "Object Oriented Programming: An Evolutionary Approach", Addison-Wesley Publishing Company, Reading, Massachusetts (1986).
- [3]Dewhurst,S.C. and Stark,K.T.: "C++入門", ドキュメントシステム訳, 小山裕司 監訳, アスキー (1990).
- [4]Gamma,E. and Weinand,A.: "ET++ - A Portable C++ Class Library For a UNIX Environment", Tutorial 6, OOPSLA'90 / ECOOP'90 (1990).
- [5]Heller,D.: "X Toolkit Intrinsics Programming Manual", The X Window System Volume 4, O'Reilly & Associates, Inc, Sebastopol, CA (1988).
- [6]小嶋隆一: "Smalltalkプログラミングマニュアル", JICC出版局 (1989).
- [7]Leberman,H.: "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems", Proceedings of OOPSLA '86, Special Issue of SIGPLAN Notices VOL.22, Addison-Wesley Publishing Company, Reading, Massachusetts, pp.214-223 (1986).
- [8]Loomis,M.E.S., Shah,A.V. and Rumbaugh,J.E.: "An Object Modeling Technique For Conceptual Design", ECOOP'87, Lecture Notes in Computer Science, Springer-Verlag, New York, pp.192-202 (1987).
- [9]森下,安積: "XViewプログラミング入門", スベック (1990).
- [10]ビンソン,L.J. and ウィーナー,R.S.: "Smalltalk: オブジェクト指向プログラミング", 富士ゼロックス情報システム 訳, 羽生田栄一 監訳, トッパン (1990).
- [11]Shah,A.V., Rumbaugh,J.E., Hamel,J.H. and Borsari,R.A.: "DSM: An Object Relationship Modelling Language", Proceedings of OOPSLA '89, Special Issue of SIGPLAN Notices VOL.24, Addison-Wesley Publishing Company, Reading, Massachusetts, pp.191-202 (1989).
- [12]Shigeru Watari, Shinji Kono, Ei-ichi Osawa, Rik Smoody, and Mario Tokoro: "Extending Object-Oriented Systems to Support Dialectic Worldviews", Proceedings of Advanced Database System Symposium '89, pp.161-168 (1989).