吉原 泰介

株式会社ミガロ

RAD事業部 技術支援課

フォーム継承による効率向上開発手法

フォーム継承は
開発効率やメンテナンス性をよくするためのプログラム側の手法である。
それによって、アプリケーションの動作がよくなるわけではない。
フォーム継承をどのように行えば効率のいい開発を行えるのか、
それを考察する。

略歴
1978 年 3 月 26 日 生
2001 年龍谷大学法学部卒
2005 年 7 月株式会社ミガロ入社
2005 年 7 月システム事業部配属
2007 年 4 月 RAD 事業部配属

現在の仕事内容
Delphi/400 や JACi400 の製品試験、および月 100 件にのぼる問い合わせのサポートやセミナー講師などを担当している。

- 開発効率、メンテナンス性がよい開発手法
- フォーム継承のポイント
- アプリケーション内の継承構成
- フォーム継承によるメリット

1 開発効率、メンテナンス性がよい開発手法

システム構築から運用までを行う場合、開発者はプロジェクトに費やす工数において、以下の 2 点を考慮する必要がある。

●開発効率
どれだけ工数をかけずに、開発ができるか
●メンテナンス性
どれだけ工数をかけずに、変更や修正が容易にできるか

では、プログラムの開発手法にはどういった形があるだろうか。

1 つの画面の単純なアプリケーションを作成するとき、ほとんどの開発者は 1 つのフォームにプログラムを組み込んでアプリケーションを作成する。これを「開発手法 A」とする。

それに対して、1 度作成した画面を流用して開発できるように、フォーム継承を行ってアプリケーションを作成するやり方を「開発手法 B」とする。

【開発方法】
A：プログラムが全て 1 つのフォームの中に集約されたもの
B：プログラムがフォーム継承を行って作成されたもの

この 2 つの手法でそれぞれ開発をした場合、どういった違いがあるのかを考察してみよう。

【照会プログラムを 1 画面開発した場合】
●開発効率
A：継承を考慮しないで単純に開発できるので、開発工数がかからない。
B：継承を考えながら開発するぶん、A より開発工数がかかる。
●メンテナンス性
A：メンテナンス対象は 1 つのフォームだけなので、変更が容易。
B：継承を考慮しての変更となるため、A よりスキルが必要。

ここで重要なのが、A と B の手法で作成された 2 つのアプリケーションに、動作上の違いはないということである。フォーム継承は、開発効率やメンテナンス性をよくするためのプログラム側の手法であり、それによってアプリケーションの動作がよくなるわけではない。

また、この結果を見ると、フォーム継承を行わないほうが優れているようにように見える。

では、次のケースはどうだろうか。

【同じような照会プログラムを 10 画面開発した場合】
●開発効率
A：単純に 1 画面 × 10 の開発工数がかかる。
B：フォームが継承された部分は再利用できるため、各画面の固有部分だけの開発工数で済む。【図 1】
●メンテナンス性
A：共通の変更が発生すると、10 画面全てに同じ対応を行う必要がある。
B：共通の変更が発生しても、継承元

図1

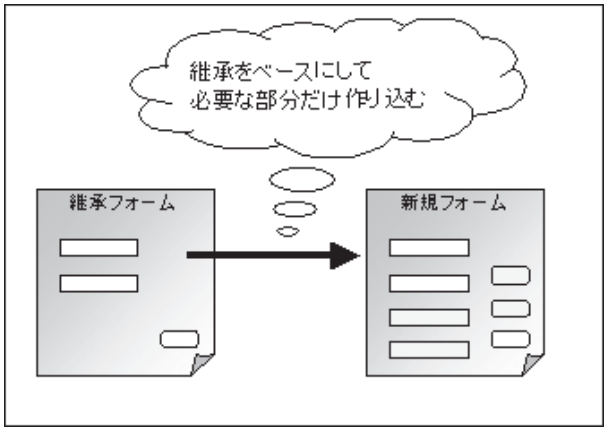

継承をベースにして
必要な部分だけ作り込む
継承フォーム
新規フォーム

図2

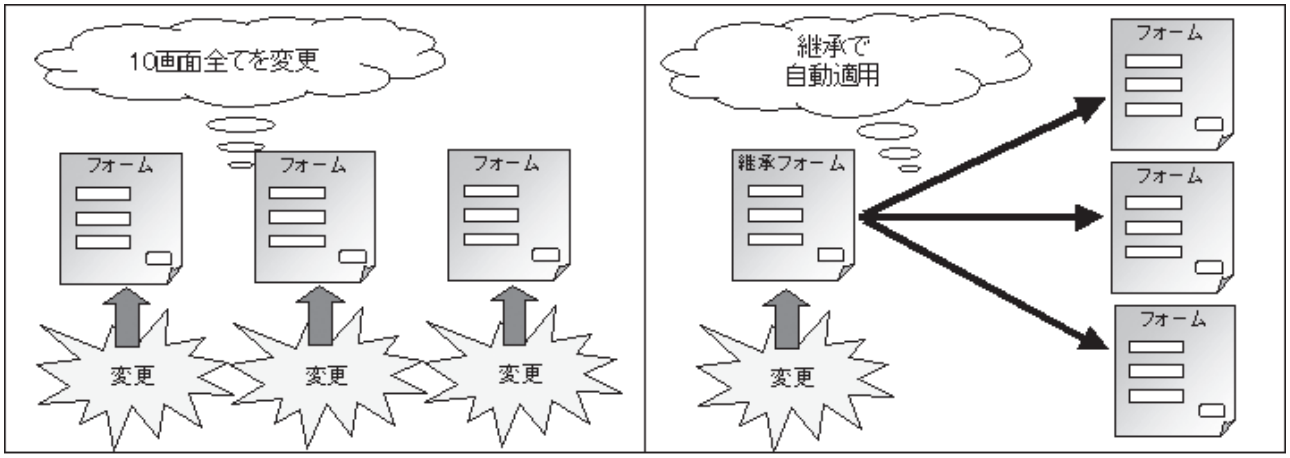
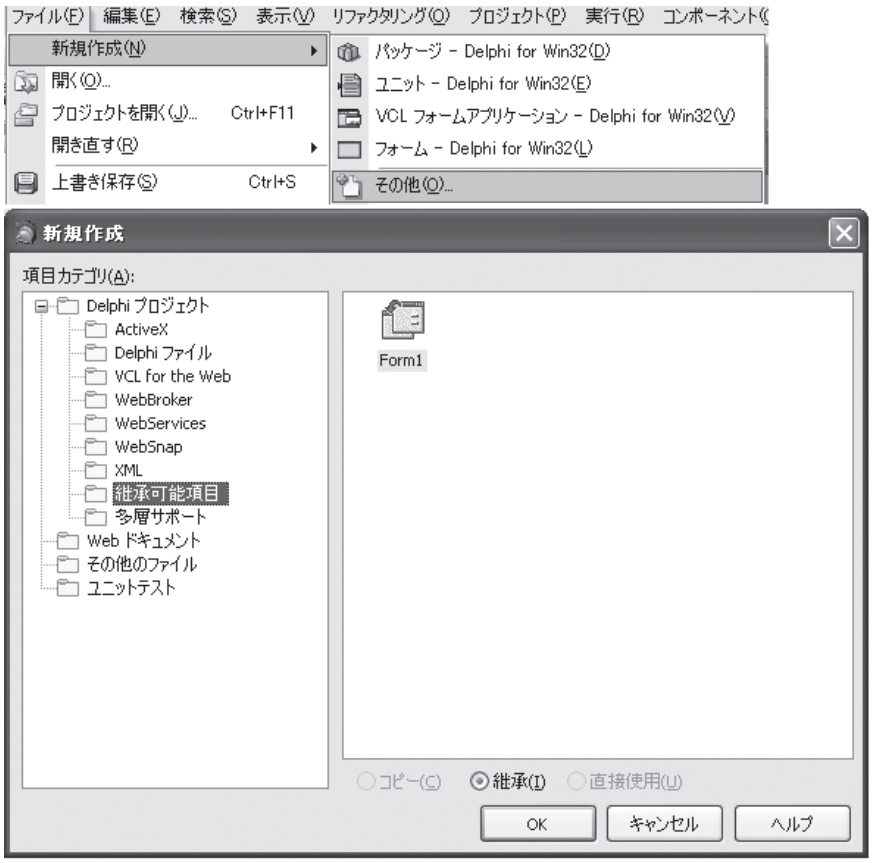

10画面全てを変更
フォーム
フォーム
フォーム
変更
変更
変更
継承で
自動適用
継承フォーム
フォーム
フォーム
フォーム
変更

図3


ファイル(F) 編集(E) 検索(S) 表示(V) リファクタリング(O) プロジェクト(P) 実行(R) コンポーネント(
新規作成(N)
開く(O)...
プロジェクトを開く(J)... Ctrl+F11
開き直す(R)
上書き保存(S) Ctrl+S
パッケージ - Delphi for Win32(D)
ユニット - Delphi for Win32(E)
VCL フォームアプリケーション - Delphi for Win32(V)
フォーム - Delphi for Win32(L)
その他(O)...
新規作成
項目カテゴリ(A):
Delphi プロジェクト
ActiveX
Delphi ファイル
VCL for the Web
WebBroker
WebServices
WebSnap
XML
継承可能項目
多層サポート
Web ドキュメント
その他のファイル
ユニットテスト
Form1
コピー(C)
継承(I)
直接使用(U)
OK
キャンセル
ヘルプ

フォームを変更すれば10画面全てに全て適用される。【図2】

Aは工数がかかるだけでなく、プログラムの開発/変更量が多い。ということは、それだけ人為的なミスが発生する可能性も高く、プログラム試験の工数もそれに伴って増大する。

それに対して、Bはプログラムの開発/変更量が少なく、また、継承化された部分は基本的に全て同じ動作となる。そのため、試験の確認パターンも必要最低限で済む、という違いがある。

ただし、上記2例からわかるように、継承化は必ずしも有効な開発手法ではあるとは限らない。継承の理解が誤っているために、逆に開発効率やメンテナンス性が悪くなってしまったという話もよく耳にする。

では、フォーム継承はどのように行えば効率がよい開発が行えるのか、これを順を追ってみていこう。

2 フォーム継承のポイント

フォーム継承がどういった特性を持つのかを把握しておかないと、煩雑なプログラムになり、後々のメンテナンスで逆に手間がかかってしまう。

では、フォーム継承を作成していくポイントをまとめてみる。

2-1. フォーム継承とは

継承とは、オブジェクト指向プログラミングにおいて、すでに定義されているオブジェクトをもとに、拡張や変更を加えた新しいオブジェクトを定義することである。

例えば、Excelで帳票を作成する場合、同じ帳票を印刷するたびに新規のExcelから作り込むことはあまりしないだろう。必要と思われる帳票フォーマットに対して、定型となるExcelをあらかじめ用意しておき、そのExcelに必要な内容だけ入力して印刷すれば手間もかからず、効率よく帳票が作成できる。

これと同じように、フォームの定型があらかじめ用意されていれば、そのフォームに対して必要な箇所だけ手を加えればよいので、便利ではないだろうか。これが、フォーム継承である。

継承フォームの新規作成についても押さえておこう。

Delphiのメニューから［ファイル］→［新規作成］→［その他］から、同じプロジェクト内のフォームを選択して、新規フォームを作成する。これにより、選択したフォームを継承した新しいフォームを作成することができる。【図3】

2-2. コンポーネントの配置

では、継承元フォームを作成するうえで、定型となるフォームにはどういったコンポーネントを配置するのが望ましいかを考えてみる。

仮に、継承元フォームに、照会画面で使う全ての画面項目を配置したとする。そうすると、全く同じ照会画面を作成する場合にはほとんど継承だけで動作することになり、一見非常に便利に思える。

しかし、少し違う照会画面になった場合、必ず不要になるコンポーネントがあることに気づくだろう。【図4】

・不要なコンポーネント

この不要なコンポーネントが多いと、その不要なコンポーネントを非表示にしたり、動作しないように制御したりすることで、逆に開発に手間がかかってしまう。また、不要なコンポーネントが存在すると、その画面の動作はそのぶんパフォーマンスが悪くなる。フォームの継承を行ううえで失敗するのは、このケースであることが非常に多い。

継承は便利である一方、継承で配置したコンポーネントは、継承先フォームでは削除することはできない。継承先フォームへの制約となることも意識する必要がある。

つまり、継承元フォームに配置するものは、その継承を利用して作成する画面に共通して必ず存在するコンポーネントだけにすることが望ましい。

例えば、検索を行う照会画面で項目の要/不要を選別してみる。【図5】

●**検索項目**：画面によって異なるので、不要になる可能性がある。
●**検索ボタン**：検索画面には必ず存在するので、必要である。

継承元フォームを作成するうえで最も重要なポイントは、継承画面に本当に必要な項目だけを過不足なく洗い出すことである。これは、上流工程の画面設計で、共通の画面レイアウトを考える際に意識しておくと、継承化の時にも格段に楽になる。

2-3. コンポーネントとコーディング

では、そうしたことを概念として理解しながらも、なぜ余計な画面項目を継承元フォームに配置して、失敗するケースが多いのだろうか。

その理由の多くは、処理プログラムを書くためである。

例えば、［クリア］ボタンを押すと、画面項目のEditコンポーネントの表示内容をクリアする機能を実装したとする。

その場合、開発者は、ソース1のようにコーディングしたい。【ソース1】

しかし、画面上にEdit1コンポーネントがないと、ソース1のようなコードは記述できないために、しかたなく継承元フォームに不要なEdit1・・・を配置してしまう。

では、検索項目をクリアする処理は、個別の画面フォームでコーディングするべきだろうか？ しかし、同じ処理を個別の画面にコーディングするのであれば、継承を行う意味が薄れてしまう。

・動的なコーディング

このような場合は、画面固有のコンポーネント名に依存しない、動的なコーディングを行う必要がある。

例えば、継承元フォームで、ソース2のようにコーディングする。画面のコンポーネント名がわからなくても、全てのEditコンポーネントを動的にクリアすることができる。【ソース2】

なお、これを応用すれば、Formの代わりにPanelコンポーネントのようなコンテナコンポーネントをあてはめて、コンテナコンポーネント上だけの制御も行える。

このように継承元フォームでは、できるだけ画面固有のコンポーネント名に依存しないコーディングが有効である。これによって、無駄なコンポーネントを配置する必要がなくなり、継承元フォーム

図4

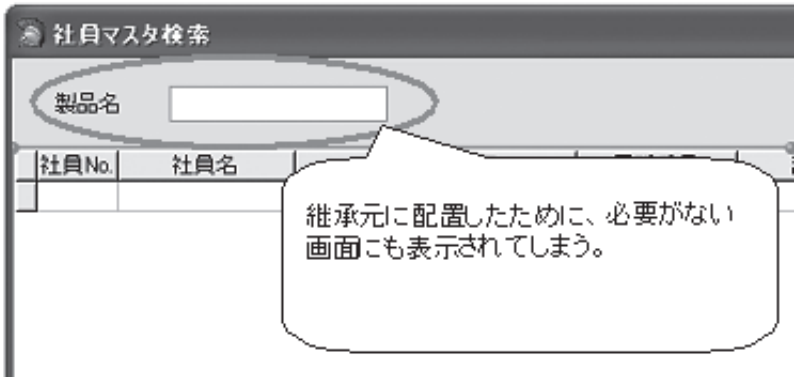

社員マスタ検索
製品名
社員No.
社員名
継承元に配置したために、必要がない
画面にも表示されてしまう。

図5

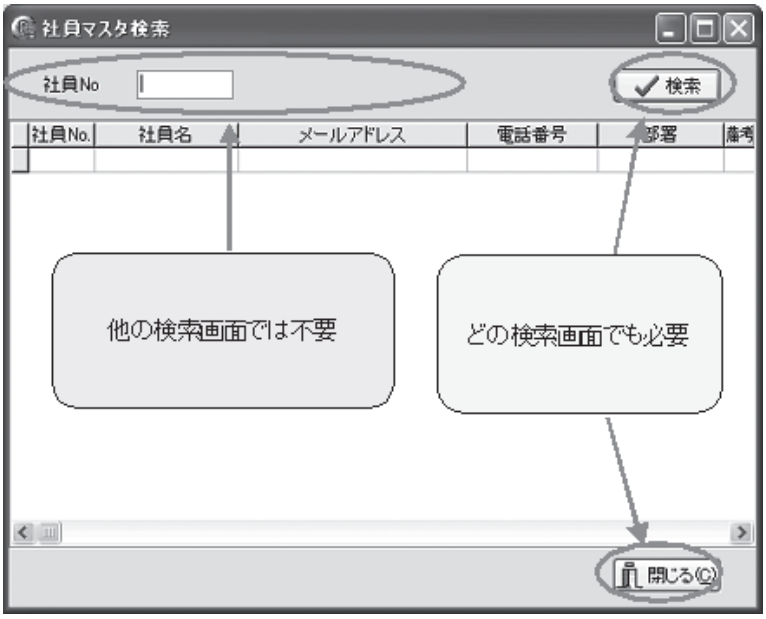
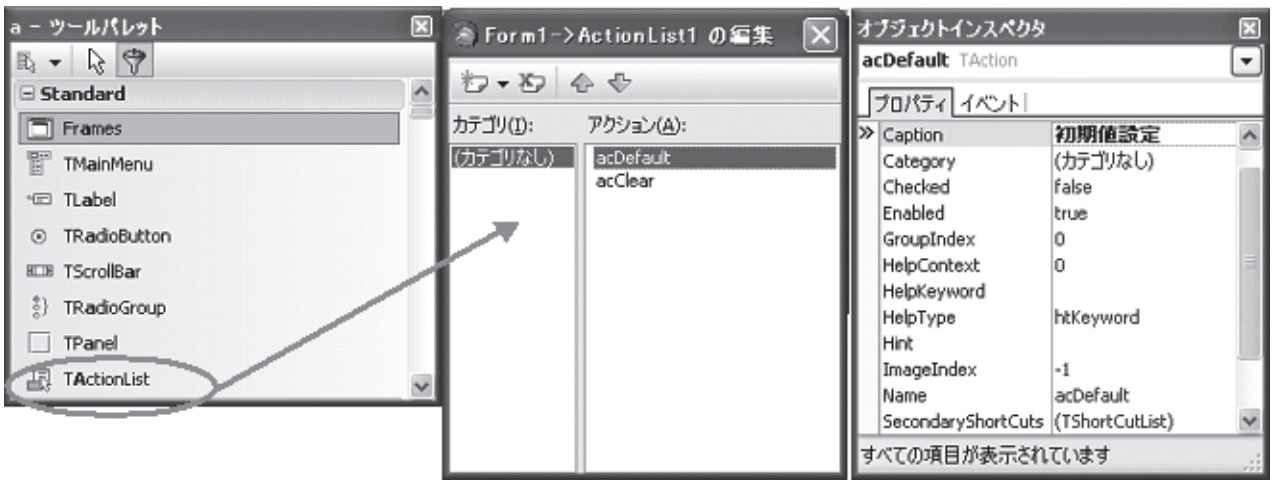

社員マスタ検索
社員No
検索
社員No.
社員名
メールアドレス
電話番号
部署
備考
他の検索画面では不要
どの検索画面でも必要
閉じる(C)

図6


a - ツールパレット
Standard
Frames
TMainMenu
TLabel
TRadioButton
TScrollBar
TRadioGroup
TPanel
TActionList
Form1->ActionList1 の編集
カテゴリ(I):
アクション(A):
(カテゴリなし)
acDefault
acClear
オブジェクトインスペクタ
acDefault TAction
プロパティ
イベント
Caption
初期値設定
Category
(カテゴリなし)
Checked
false
Enabled
true
GroupIndex
0
HelpContext
0
HelpKeyword
HelpType
htKeyword
Hint
ImageIndex
-1
Name
acDefault
SecondaryShortCuts
(TShortCutList)
すべての項目が表示されています

での活用範囲をさらに広げることができる。

2-4. イベント制御

継承元フォームに配置するコンポーネント同様、どのイベントをどこまで継承元で制御するかも重要なポイントである。

例えば、［閉じる］ボタンを押下したときにフォームを閉じるといったイベントは、継承元フォームで記述するということは想像しやすいだろう。また、検索や更新などの画面固有の処理は、継承先フォームに処理を記述すればよい。これらの切り分けは比較的簡単に行える。

では、画面起動時の処理を行うイベントは、継承元フォームと継承先フォームのどちらで処理を記述するべきだろうか。画面起動時に行う処理を考えてみよう。

【画面起動時の処理】
①画面項目をクリアする。
②画面項目に初期値をセットする。

①は、画面全体に対しての処理なので、前述の 2-3 のように継承元フォームで処理できそうである。一方、②の初期値は、画面ごとに内容やセットする項目が異なるため、継承先フォームで処理することになるだろう。

なお、こういった場合、イベントの処理内容はどちらかにまとめて記述する必要がある、というようなことはない。

①のイベント処理は、継承元フォームに記述しておくことができる。
②の内容は、継承先フォームで同イベントに記述する。

ここで気をつけなければならないのは、同じイベントが、それぞれどのタイミングで実行されるかということである。【ソース 3】【ソース 4】

同じイベントで処理を記述した場合、継承元フォームのイベント内容は inherited の箇所で実行される。

これは、デバッグ実行時における F7 キーでのトレースや ShowMessage を組み込んで、実行される実際の順番を確認すると理解しやすい。

つまり、こういったイベント内の内容を切り分けることによっても、継承先フォームでの開発負担を軽減することができる。

2-5 アクションコンポーネントの利用

別の視点からイベントを見てみようか。前述の①と②の処理順は決まっているので、この処理順を、継承元フォームで制御することはできないだろうか。

そのような場合に便利に使えるのが、TActionList というコンポーネントである。TActionList では、TAction というアクションコンポーネントを持つことができる。この TAction コンポーネントであらかじめ想定される動作を用意しておけば、継承元フォームで、継承先のイベントの制御に利用することができる。【図 6】

例えば、②の初期値設定処理を想定して、アクションとして acDefault という TAction コンポーネントを継承元フォームに配置しておく。

このアクションを利用して、前述の画面起動時の処理を変えてみよう。【ソース 5】【ソース 6】

一見、同じようなプログラムに見えるが、処理①②の順番を制御しているのは、継承元フォームである。

これにより、フォーム継承を利用して個別画面（継承先フォーム）を作成する開発者は、inherited などの継承を考慮する必要がなくなる。acDefaultExecute イベントに初期値設定の処理を記述しておけば、継承元フォームが自動的に適切な箇所でその処理を呼び出してくれる。

例えば、acDefault は初期値設定処理、acSearch は検索処理など、アクション名を決定して継承元フォームに配置しておく。そうすれば、各開発者はそのアクションのロジック作成のみに専念することができ、また、システム全体としても動作の統一を行うことが容易になる。

3 アプリケーション内の継承構成

2 で、継承元フォームの作成のポイントをまとめた。では、この継承元フォームは一体どの程度用意しておけばよいのだろうか。

フォームの継承は何階層でも多重継承が可能なので、汎用性を考えると細分化された継承フォームを用意する必要がある。

例えば、メニュー画面、検索画面、更新画面から構成されるアプリケーションの構成を考えてみよう。

まず単純に、3 つの画面フォームのパターンが考えられる。【図 7】

このうち、アプリケーション共通の画面デザイン（タイトルラベルやセッション表示など）を別々に作成する必要はないので、画面デザインだけの共通フォームを用意する。【図 8】

また、同じように検索処理画面においても、一覧検索画面や詳細情報検索画面などレイアウトが異なる可能性があるので、検索ボタンを配置した検索の共通画面を用意する。同様に、更新画面も画面のパターンを用意する。【図 9】

このように、継承フォームを細分化すれば、より多くの画面に対応することができる。その際、構築するシステムの規模や複雑さ、画面パターンなどによって、その細分化のレベルは異なる。

また、新しいパターンの画面が追加された場合は、それに対応する継承フォームを追加していけばよい。そのため、後から継承フォームが追加できるように細分化を考えておく必要がある。

ただし、継承フォームの細分化によって汎用的で便利になる一方、継承フォーム自体の管理が手間になってしまっては意味がない。その見極めには十分に配慮する必要がある。

4 フォーム継承によるメリット

フォーム継承を利用するときには、さまざまな開発ポイントがある。

システム構築当初に、これらをしっかり検討してフォームの継承を利用すれば、システム構築では、以下のような利点が得られる。

●開発効率
① 1 画面あたりの開発工数が少なくなる。
同系統の画面作成に、同じ工数をかける必要がなくなる。
②開発者に求められるプログラムの敷居を下げることができる。
難易度の高い共通ロジックを継承元で

図7


メニュー画面
検索画面
更新画面

図8

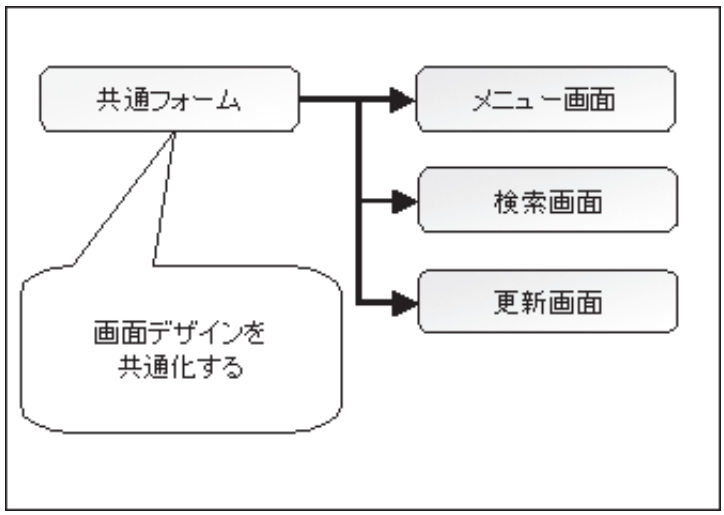
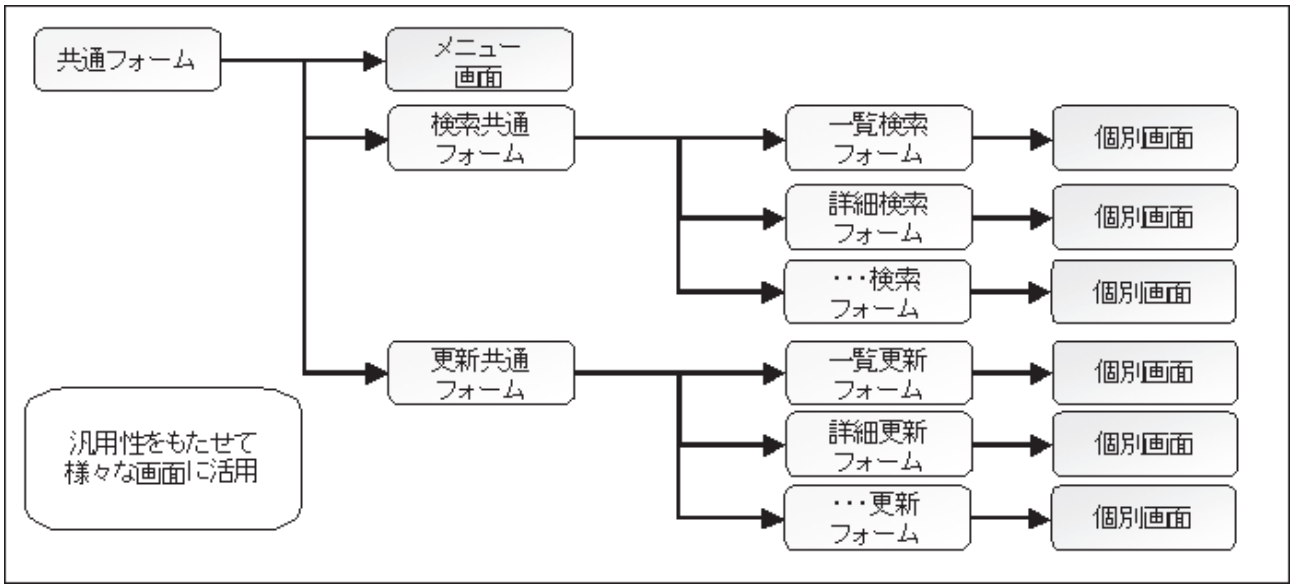

共通フォーム
メニュー画面
検索画面
更新画面
画面デザインを
共通化する

図9


共通フォーム
メニュー
画面
検索共通
フォーム
一覧検索
フォーム
個別画面
詳細検索
フォーム
個別画面
・・・検索
フォーム
個別画面
更新共通
フォーム
一覧更新
フォーム
個別画面
詳細更新
フォーム
個別画面
・・・更新
フォーム
個別画面
汎用性をもたせて
様々な画面に活用

行うことで、各画面開発で必要とされるプログラムの難易度が下がる。

③画面動作の統一を開発者依存ではなく、継承上で管理できる。
開発者によるプログラムのばらつきを継承元で統一する。

●メンテナンス性

④システムが統一された作りになることで、管理が容易になる。
システムの画面や動作が統一されることで、画面ごとの動作やレイアウトの違いなどの煩雑な問題を解消できる。

⑤画面共通の変更が継承元の変更工数だけで対応できる。
画面共通の変更が継承元一箇所で行えるため、迅速かつ安全なメンテナンスが行える。

これらは、システムの規模や開発人員が多いほど､効果をあげることができる。また、洗練された継承の仕組みは、新たに開発する別のシステムにも流用して生かしていくことができるだろう。

M

ソース1

```
// 項目のクリア
Edit1.Clear；
Edit2.Clear；
・・・
```

ソース2 クリア処理

```
// フォーム全てのコンポーネントを処理
for i := 0 to Form1.ControlCount - 1 do
begin
  //Edit であれば Edit として扱う
  if (Form1.Controls[i] is TEdit) then
    TEdit(Form1.Controls[i]).Clear;
end;
```

ソース3 継承元フォーム

```
// 継承元フォームの FormCreate イベント
procedure TFormBase.FormCreate(Sender: TObject);
begin
  // 処理①
end;
```

ソース4 継承先フォーム

```
// 継承先フォーム画面の FormCreate イベント
procedure TForm1.FormCreate(Sender: TObject);
begin
  inherited; // ここでイベントが継承されて①が実行される
  // 処理②
end;
```

ソース5 継承元フォーム

```
// 継承元フォームの FormCreate イベント
procedure TFormBase.FormCreate(Sender: TObject);
begin
  // 処理①
  acDefault.Execute; // ここで処理②の内容が実行される。
end;
```

ソース6 継承先フォーム

```
// 継承先フォーム画面の acDefault の Execute イベント
procedure TForm1.acDefaultExecute(Sender: TObject);
begin
  inherited;
  // 処理②
end;
```