最優秀賞

自社用開発フレームワークの構築
ーなぜフレームワークが必要か 高効率な内製実現のために

駒田 純也 様
ユサコ株式会社
システムグループ マネージャー

ユサコ株式会社
http://www.usaco.co.jp/top.html

海外の学術雑誌、書籍の輸入販売を中心に事業を展開している。特に医学、薬学等の自然科学分野に強みを持つ。近年、学術情報媒体の多様化に伴い、電子ブック、データベース、各種ソフトウェアの取り扱いを強化。学術情報を通じ、知的情報の創造、蓄積、共有による社会貢献を目指している。

Delphi/400導入の経緯

ユサコは、海外の医学系学術誌やデータベースの輸入販売などを行っている会社であり、ユーザー企業の立場で、System/38 の時代から IBM のメインフレームを使い続けている。

当時から基幹システムとして販売管理システムを自社開発しており、現行の基幹システムも開発開始から 16 年、運用開始から 14 年の間機能拡張を行いながら使い続けてきた。

とはいえ、現場で扱わなければならない情報量が格段に増え、エミュレーター画面の表示能力にも限界を強く感じ、インターフェースを再構築するためのツールを探していた。

他にも CASE ツール等も検討していたが、開発やユーザー操作の自由度が低く、Delphi/400 であればどんな要望にもフレキシブルに応えられると考えた。

なぜフレームワークが必要か

Delphi/400 での開発は自由度が高く、CASE ツールのように煩雑な事前作業は必要ないが、その反面、開発者ごとにデータベースやコーディング上の命名基準、画面構成や配色、エラー処理といったものがバラバラになってしまう恐れがある。

この問題を回避するには、モデリングやコーディング規則だけでなく、ユーザーに同じ見た目で、同じ操作基準を持った、一貫性のあるシステムを提供する「開発フレームワーク」が必要である。しかし、製品化されているものがなく、内製するに至った。

加えて、フレームワークを使用することにより仕様変更に強く、開発やテスト工数を削減し、バグも減らせるという利点も重要であった。

フレームワーク概要

フレームワークは現行システムに不足している管理用テーブルや、標準コンポーネントを拡張したオリジナルコンポーネント、キャッシュ用エンティティクラス等の共通クラス、入力値チェックや値選択機能、アプリケーション自動配布の仕組みで構成している。

開発に当たっては、コーディング量を減らし、開発効率と保守性を高めるため、オブジェクト指向でいうカプセル化(※)やロジックの汎用化 (汎化) を意識した。【図 1】

また、オブジェクト指向言語でクライアント / サーバー系システムの開発をフレームワーク部分から行うのは初めてであったため、開発支援を受け試行錯誤しながらの構築であった。

※カプセル化とは

オブジェクト内のデータを直接操作できないように、その構造を外部から隠蔽すること。オブジェクト内の仕様変更が

図1　フレームワーク概要

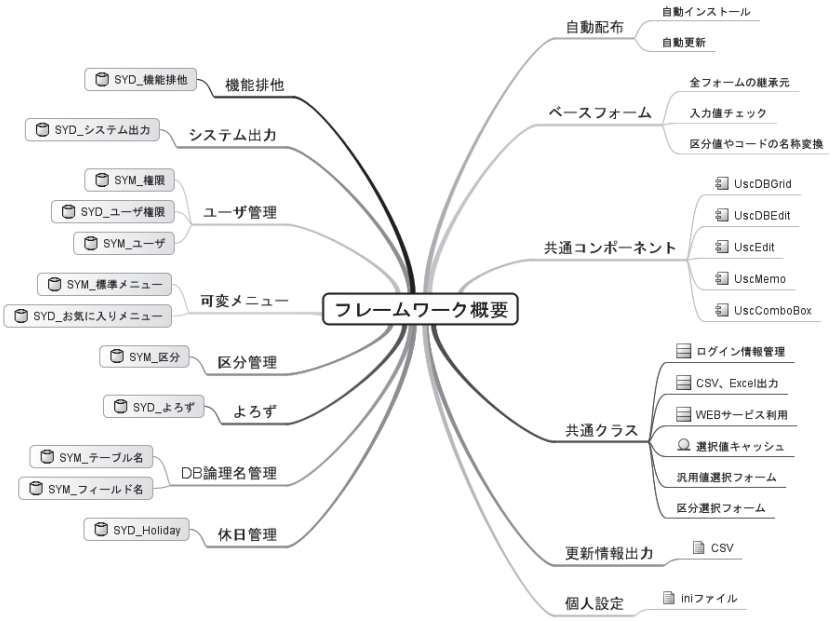
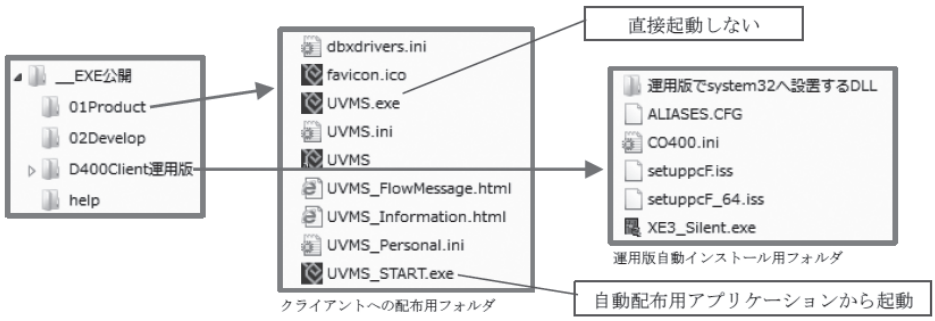


図2　アプリケーション自動配布



ソース1　サンプルコード

```
procedure xxx.Form_OnCreate(Sender: TObject);
var
  i: Integer;
begin
  //DBGrid の列タイトルを設定する
  dbgMain.fSetLogicalName(cdsLgcFld,'VCCIZREP');

  //入力項目のラベルを設定する
  for i := 0 to ComponentCount - 1 do begin
    if Components[i] is TLabel then begin
      if cdsLgcFld.Locate('TblPhyName;FldPhyName',
        VarArrayOf(['VCCIZREP',TLabel(Components[i]).Caption]),[loCaseInsensitive]) then
      begin
        TLabel(Components[i]).Caption:=cdsLgcFld.FieldByName('FldLgcName').AsString;
      end;
    end;
  end;
```

対象テーブルのフィールド論理名を
格納した ClientDataSet

```
SELECT TblPhyName, FldPhyName, FldLgcName
FROM LIB.SYMFldName
WHERE TblPhyName = 'VCCIZREP'
```

```
{* 表題：DBGrid の列タイトルを設定する
* 引数 1[i]：CDSLgcName：対象テーブルのフィールド論理名を格納した ClientDataSet
* 引数 2[i]：TableName：対象テーブル名}
procedure TUscDBGrid.fSetLogicalName(CDSLgcName: TClientDataSet; TableName: string);
var
  i: Integer;
  dbgSelf: TUscDBGrid;
begin
  dbgSelf:=self;
  for i := 0 to dbgSelf.Columns.Count - 1 do begin
    if CDSLgcName.Locate('TblPhyName;FldPhyName',
      VarArrayOf([TableName,dbgSelf.Columns[i].FieldName]),[loCaseInsensitive]) then
    begin
      dbgSelf.Columns[i].Title.Caption:=CDSLgcName.FieldByName('FldLgcName').AsString;
  end;
end;
```

他のプログラムに影響せず、他からも影響されないため、プログラムの開発効率と保守性が高まり、再利用しやすくなる。

開発効率と保守性を高める工夫

社内 SE の仕事は開発以外にも多岐にわたるため、内製を維持するには開発効率が非常に重要である。また、現場の要望に迅速に応え、競争力を上げるために、保守性も同様に重要である。

そのため、いかに同じコードを記述せずコード数を減らすか、誰が書いても同様の記述になるかを意識しており、そのいくつかを紹介する。

(1) アプリケーションの自動配布

EXE ファイルと Delphi/400 をクライアント PC へ事前にインストールせず、ユーザーの初回起動時に自動でインストールする仕組みを構築した。初回起動時以降は、EXE ファイルや Delphi/400 のバージョンアップを自動で行うようにしており、インストールと更新作業の手間をなくした。

これにより、ひんぱんにアプリケーションの仕様変更や Delphi/400 のバージョンアップがあったとしても、配布の手間を気にせずクライアントを最新の状態にすることができる。

各クライアントの設定を保管する ini ファイルに関しては、初回起動時にはファイルをコピーしているが、それ以降は ini ファイルの内容をチェックして追加されたセクションおよびキーのみ追加し、設定情報を消さないようにしている。64bit 版 OS のクライアントも対応済である。【図 2】

(2) フィールド論理名管理

DBGrid の列名や入力項目（DBEdit）のラベル Caption をデザイン画面で設定するのではなく、データベースに格納したフィールド論理名を画面表示時にコードから設定することにした。これにより、類似した記述の重複をなくし、変更箇所を一元化することで開発効率を上げている。

DBGrid の列名設定のコードは、DBGrid を拡張したオリジナルコンポーネントに記述している。入力項目のラベル Caption 設定箇所は、使用しないフォームもあるため汎化していない。【ソース 1】

(3) 可変メニュー

メニュー内容は、データベース化したものをツリービューを使って表示する仕組みとした。フォームの変更が不要であるためビルド回数を減らせるとともに、各メニューの有効無効を切り替えるといった対応も、データベースの変更のみで可能とした。

メニューは、開発側で用意した標準メニューと、各自がオリジナルメニューを作成できるお気に入りメニューの 2 つを用意し、各自で工夫できるようにした。【図 3】

メニュークリック時に開くフォームは、データベースに文字列として格納することになるため、他の情報と合わせ record 型として TTreeNode の Data プロパティへ格納し、メニュークリック時に読み出している。

フォームクラスは、FindClass 関数で探せるようにフォームの Create イベント等で事前に RegisterClass 手続きを使い、登録しておく必要がある。【ソース 2】

(4) 入力値チェック

オリジナルコンポーネントのプロパティに設定した値を使用し、全角 / 半角 / 半角大文字 / 数値型に合致する文字列かどうか、文字型フィールドでは EBCDIC 換算した文字数に収まっているかどうか、数値型フィールドでは数値フォーマットに合致するかどうかの妥当性チェックを行っている。

オリジナルコンポーネントへのプロパティとイベント設定は、コード上で行っている。ただし、手作業でのコーディングは行わず、Excel 化したテーブル仕様書（※）をもとに入力項目仕様書（図 4）を作成し、Excel 関数で作成したコードを貼り付けているだけである。【図 4】【ソース 3】【ソース 4】

また、妥当性チェックのコードも共通クラスに記述してあるため、ほぼノーコーディングで行っている。【ソース 5】

※ DSPFFD コマンドで FILE 化し、ODBC 経由で Excel に取り込む

(5) 値選択とローカルキャッシュ

区分やマスター値については通常、CD や ID としてフィールドに格納されているが、それを名称や他のフィールド値から検索して選択するための入力補助機能を備えている。【図 5】【図 6】

選択画面の Grid に表示される内容は事前にキャッシュしておき、高速な表示や検索を実現した。選択画面表示時にリフレッシュも可能である。

区分値選択のキャッシュは各フォームでそれぞれ行い、マスター値選択のキャッシュはキャッシュ用のクラスで行っている。【ソース 6】

(6) 値変換とエンティティクラス

各レコードの詳細を表示する欄では、DataSource の DataChange イベントを使いフォーム表示時やデータ変更時に CD や ID を名称に変換し、CD や ID とは別にわかりやすく表示している。【図 7】【ソース 7】

SQL 文に ID や CD に対応した区分やマスターテーブルを結合して表示せず、事前にエンティティクラスに読み込んだデータを使用している。そのため高速で、CD や ID を直接入力した際にもサーバーと通信することなく、表示内容を更新できる。【ソース 8】

また、手入力した CD や ID がデータベース上に存在するかどうかも ClientDataSet の Validate イベントを使い、その場でチェック可能としている。【図 8】

今後の予定・計画

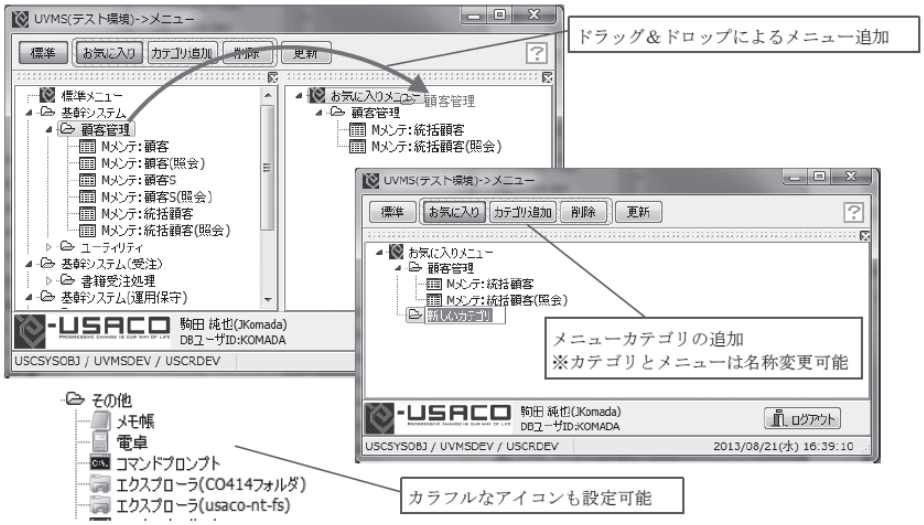
(1) 業務ポータルとしてのグループウェア

Delphi/400 とは別の統合開発環境で構築しているグループウェアの Web サービス連携機能について、これを Delphi/400 に移植し、更新情報やアラートをグループウェアに最新情報として書き込めるようにする。

(2) 印刷方法の拡張

現在の印刷方法は、帳票サーバーへ CSV データとしてデータを渡して印刷する方法しかなく、その場でプレビューすることができない。そのため、QuickReport や Excel を使った出力手段を実装したい。

図3　可変メニューの仕組み



ソース2　レコード型定義のサンプルコード

レコード型定義のサンプルコード

```
//各メニューノードの情報格納用レコード型
TMenuData = record
  FuncID:      string;  //機能 ID
  ReqAuthID: string;  //必要権限 ID
  FormName:   string;  //フォーム名
  StartPath: string;  //起動 Path
  DispMode:   string;  //表示モード
  FormMutex: string;  //フォーム排他
  ImageIdx:   Integer; //イメージ Index
end;
PMenuData = ^TMenuData; //TMenuData のポインタ型
```

フォームクラス事前登録のサンプルコード

```
//findClass で利用するための準備
RegisterClass(TfrmMntUser);
RegisterClass(TfrmMntUsrAuth);
RegisterClass(TfrmMntCust);
```

レコード型からデータを取り出すサンプルコード

```
var
  frmClass: TfrmUscBaseClass; //フォームは全て UscBase を継承している
  frm: TForm;
  exeText, paramText: PChar;
begin
  { 機能の起動処理 }
  //フォーム名が設定されているかどうかで処理を分ける
  strFormName:=TMenuData(TTreeView(Sender).Selected.Data^).FormName;
  if strFormName <> '' then begin
    strFrmDispMode:=TMenuData(TTreeView(Sender).Selected.Data^).DispMode; //Read:読取(照会)
    //フォーム起動
    frmClass:=TfrmUscBaseClass(FindClass(strFormName));
    frm:=frmClass.Create(Self,strFrmDispMode);
    frm.Show;
  end else begin
    //外部プログラムを起動
    exeText:=PChar(TMenuData(TTreeView(Sender).Selected.Data^).StartPath);
    paramText:=cfGetParamStr(exeText, pgmPath);
    ShellExecute(Handle, nil, PChar(pgmPath), paramText, nil, SW_NORMAL);
  end;
```

図4　入力項目仕様書

入力項目仕様書

英字名	表記	任意	初期値	非表示	マスタ	区分	IME開	数値	半角	半UC	JP	型	長	桁	小
CLSTFORMAT	免税顧客区分		0			○	○	○				P		1	0
CCEFFICIENT5	係数 書籍固定値A	○	0.00				○	○				P		5	2
CCEFFICIENT6	係数 備考	○									○	O	60		
YOBIDATE1	日付 予備1	○	0	○								P		8	0
YOBIDATE2	日付 予備3	○	0	○								P		8	0
YOBIDATE3	日付 予備2	○	0	○								P		8	0
YOBIFLD1	FLD予備1	○		○								A	10		
YOBIFLD2	FLD予備2	○		○								A	10		
YOBIFLD3	FLD予備3	○		○								A	10		
OTHERKBN1	その他の区分1	○		○								A	1		
OTHERKBN2	その他の区分2	○		○								A	1		
OTHERKBN3	AgentRank	○				○	○		○			A	1		
OTHERKBN4	支払日休日時	○				○	○		○			A	1		

必須	任意	IME開	数値	半角	半UC	JP	初期値
dbeCUS		dbgMain.	if fldNam				
	cdsCus	dbgMain.	if fldNam				cdsCust\
	cdsCus					if fldNa	cdsCust\
	cdsCus						cdsCust\
	cdsCus						cdsCust\
							cdsCust\
							cdsCust\
							cdsCust\
							cdsCust\
							cdsCust\
	cdsCus	dbgMain.		if fldNar			cdsCust\
	cdsCus	dbgMain.		if fldNar			cdsCust\

Excel 関数によるコード作成

(3) フィールド指定検索の拡張

対象テーブルの指定フィールドでレコードを検索する機能があるが、CD や ID でしか検索できない。これを、関連テーブルの名称（例えば、社員マスターにある社員名）を使って検索できるように拡張する。（※）【図 9】

※「含む」「=」等のオペランドリストは、フィールド型に対応した内容に自動変更。

(4) バッチ処理の実行管理

現在は IBM i 上でバッチ処理を実行しているが、Delphi/400 への置き換えも検討している。その場合は Windows サーバー上で動くことになるが、Windows のタスクスケジューラーを使うのではなく、独自の実行管理アプリケーションの構築を考えている。

フレームワーク構築で感じたこと

当社のように少人数で自社システムの開発運用を行うためには、いかに冗長コードやデータを減らし、再利用性を高めるかが重要であると考えられる。しかし、情報システム部門にはインフラ管理やユーザーサポートをはじめ、開発以外の仕事についても多くの時間を要し、開発自体も次々に相談や依頼が舞い込んでくる。

後々のことを考えれば、開発効率を上げるためのリファクタリングが重要であることはわかっている。とはいえ、ユーザーにとって目に見えるものではなく、その効果を捉えにくいものであるため、時間を割り当てにくいというジレンマを感じた。

今後も拡張やリファクタリングを続けていくことになるが、自社用にゼロから構築するのは情報量も少なくなかなか骨の折れる作業であり、テクニカルレポートのようなユーザー同士の情報共有がさらに活発に実践されていけば、IBM i と Delphi という強力な組み合わせがもっと広がるのではないかと感じた。

ユーザー企業が集まり、勉強会という形でアイデアを出し合ったり相談をする機会があってもよいとも思う。

M

ソース3　Excel関数で作成したコードサンプル

```
//任意入力フィールド
cdsUnifyCust.FieldByName('UNIFYCUSTNAMEENG').Required:=False;
//'Numeric'数値フィールド制限
fSetDispObjPrpNum(dbeUNIFYCUSTNO,ekNumeric,ctNumeric,ttNone,5,2,False);
```

ソース4　オリジナルコンポーネントのプロパティ設定サンプルコード

```
//表題：画面表示用オブジェクト(TUscDBEdit)へ、プロパティをセットする ※Numeric 専用
procedure TfrmUscBase.fSetDispObjPrpNum(dbeSrc: TUscDBEdit; EditKind: TEditKind; ChkType:
TChkType; TrnsType: TTrnsType; IntSize, FracSize: Integer; AllowMinus: Boolean);
begin
  dbeSrc.prpEditKind:=EditKind; //種別
  dbeSrc.prpChkType:=ChkType; //チェックタイプ
  dbeSrc.prpTrnsType:=TrnsType; //変換タイプ
  dbeSrc.prpIntSize:=IntSize; //NUMERIC 型のフィールド長 ※NUMERIC(X,Y)の X 部分
  dbeSrc.prpFracSize:=FracSize; //NUMERIC 型の小数部長さ ※NUMERIC(X,Y)の Y 部分
  dbeSrc.prpAllowMinus:=AllowMinus; //マイナスの入力を許可するかどうか
  dbeSrc.ImeMode:=imClose;
  dbeSrc.OnKeyDown:=dbeOnKeyDownNum;
  dbeSrc.OnKeyPress:=dbeOnKeyPressNum;
end;
```

イベントもコード上で設定

ソース5　入力チェック時のサンプルコード

```
//表題：NUMERIC 型に対応した DBEdit の汎用イベント
procedure TfrmUscBase.dbeOnKeyPressNum(Sender: TObject; var Key: Char);

  //仮想文字列の作成
  dbeTemp:=TUscDBEdit(Sender);
  strTemp:=fGetVirtualStringNum(dbeTemp.Text, dbeTemp.SelStart, dbeTemp.SelLength, Key,
dbeTemp.prpAllowMinus);

  //仮想文字列が入力規則に合致しているか
  //※intSize フィールドサイズ fracSize 小数部の桁数 ※整数部＝フィールドサイズ - 小数部桁数
  if not cfChkNumberFmt(strTemp,dbeTemp.prpIntSize,dbeTemp.prpFracSize,dbeTemp.prpAllowMinus)
then Abort;
end;

//表題：対象値が NUMERIC フィールド定義に合っているかチェック
function cfChkNumberFmt(strTemp: string; intSize, fracSize: Integer; allowMns: Boolean): Boolean;

  if (Pos('-',strTemp) > 0) and (not allowMns) then Abort; //マイナスが許可されていないのに入っていないか
  if Pos('-',strTemp) > 0 then strTemp:=StringReplace(strTemp,'-','',[]);
  //小数点があるか
  if Pos('.',strTemp) > 0 then begin
    iIntPart:=Length(Copy(strTemp,0,Pos('.',strTemp)))-1; //整数部の桁数
    iFracPart:=Length(Copy(strTemp,Pos('.',strTemp)+1,MaxInt)); //小数部の桁数
  end else begin
    iIntPart:=Length(strTemp);
    iFracPart:=0;
  end;
  //桁数がフォーマット内か
  iIntMax:=intSize - fracSize; //整数部の最大桁数
  iFracMax:=fracSize; //小数部の最大桁数
  if (iIntPart > iIntMax) or (iFracPart > iFracMax) then Result:=False
  else Result:=True;
end;
```

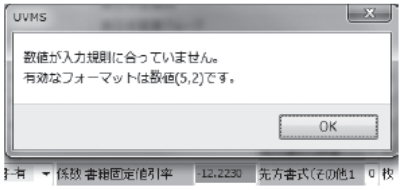


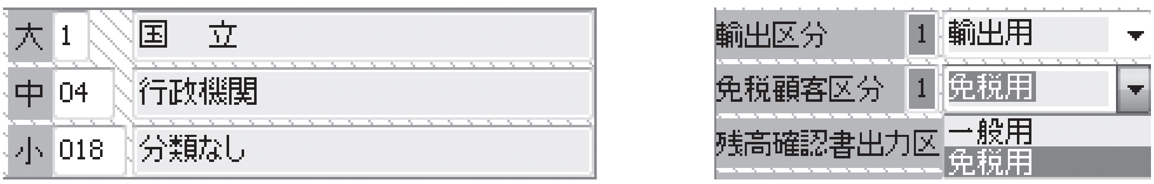
図5　マスター値選択:社員コード

図6　区分コード

ソース6　区分選択画面呼び出しのサンプルコード

```
fldName:=TUscDBEdit(Sender).Field.FieldName;
trnsType:=TUscDBEdit(Sender).prpTrnsType;

//区分の場合
   frmKubunSelect:=TfrmKubunSelect.Create(self);
   strKubunID:=TUscComboBox(FindComponent('cbx'+fldName)).prpKubunID;
   //区分選択画面内の ClientDataSet(cdsSelect)へデータ追加

   while not cdsKubun.Eof do begin
      if cdsKubun.FieldByName('KubunID').AsString = strKubunID then begin
         frmKubunSelect.cdsSelect.AppendRecord(['',cdsKubun.FieldByName('KubunCD').AsString,
            cdsKubun.FieldByName('KubunCDName').AsString]);
      end;
      cdsKubun.Next;
   end;

   //選択フォーム表示
   try
      if frmKubunSelect.ShowModal = mrOk then begin
         //区分値のセット
         if not cdsLclValue.Modified then cdsLclValue.Edit;
         cdsLclValue.FieldByName(fldName).AsString:=
            frmKubunSelect.cdsSelect.FieldByName('KubunCD').AsString;
      end;

   end;
```

図7　コードから名称に変更して表示



ソース7　DataSourceのDataChangeイベントでの呼び出し例

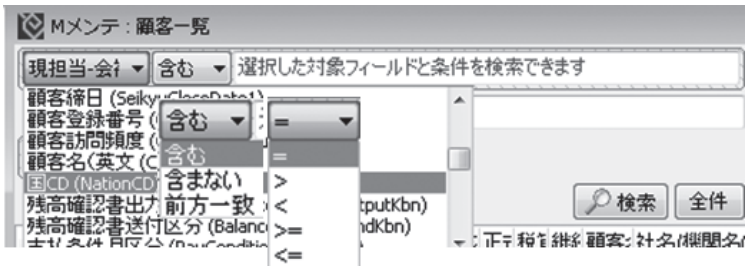
```
//レコード移動時には全項目を、フィールド値変更時には該当項目だけを処理
notAssigned:=not Assigned(Field);
if Assigned(Field) then fldName:=Field.FieldName;

//マスターデータエンティティから参照
if notAssigned or (fldName = 'CLASSIFIED') then fSetNameTrnsFields('Edit', 'CLASSIFIED',
fdmEntity.cdsClass, edtClassName);

//区分データから参照
if notAssigned or (fldName = 'EXPORT') then fSetNameTrnsFields('KubunCbx', 'EXPORT', cdsKubun,
cbxExportKbn);
```

ソース8　名称表示用オブジェクトに値をセットするサンプルコード

```
//引数 1[i]：TrnsType：変換タイプ、 引数 2[i]：FldName：対象のフィールド名
//引数 3[i]：cdsTrns：値変換用の CDS、引数 4[i]：DspObj：画面に名称を表示するコンポーネント
procedure TfrmUscBase.fSetNameTrnsFields(TrnsType, FldName: string; cdsTrns: TClientDataSet; DspObj:
TObject);

begin
  if TrnsType = 'Edit' then begin //マスターデータエンティティから参照
    edtTemp:=TUscEdit(DspObj);
    blnDetect:=cdsTrns.Locate(edtTemp.prpCDFldName,cdsLclValue.FieldByName(FldName).AsInteger,[]);
    if blnDetect then begin
      edtTemp.Text:=cdsTrns.FieldByName(edtTemp.prpTgtFldName).AsString;
    end else begin
      edtTemp.Text:='該当なし';
    end;
  end;

  if TrnsType = 'KubunCbx' then begin //区分データから参照
    cbxTemp:=TUscComboBox(DspObj);
    if cdsTrns.Locate('KubunID;KubunCD',
      VarArrayOf([cbxTemp.prpKubunID,cdsLclValue.FieldByName(FldName).AsString]),[]) then begin
      cbxTemp.ItemIndex:=cbxTemp.Items.IndexOf(cdsTrns.FieldByName('KubunCDName').AsString);
    end else begin
      cbxTemp.ItemIndex:=-1;
    end;
  end;
end;
```

図8　存在チェック結果

図9　検索画面の拡張



MEMO