情報処理学会論文誌 Vol.54 No.4 1573–1580 (Apr. 2013)

テクニカルノート

手続き型言語におけるデータバインディング機構の提案と構造化設計への適用

荻原 剛志[1,a)]

受付日 2012年11月30日, 採録日 2013年1月11日

概要： オブジェクト指向によるソフトウェア開発では，プロパティ間のバインディングを利用してオブジェクトを連携させる手法が広まってきている．オブジェクト指向を用いない手続き型言語のプログラミングにおいても，機能を提供するコード自体を変更せずに，モジュール間の結合を変更しやすくする方法があれば，独立性，再利用性を損なわずにソフトウェアの変更や拡張を容易に行うことができる．本稿ではこのために coval と呼ぶ仕組みを提案する．coval は複数のモジュール間で共有される変数を利用して制御を受け渡す仕組みであり，C 言語のライブラリとして実装した．coval を使って C 言語のモジュールを関連づけることにより，既存のコードへの変更を抑えつつ，機能を追加，変更できることを確認した．本稿ではさらに，小規模な組込みソフトウェアへの適用例を示し，構造化モデリングをベースとしたシステム開発における coval の有効性について論じる．

キーワード： データバインディング，手続き型言語，組込みシステム，構造化モデリング，ソフトウェアコンポーネント

A Data Binding Mechanism for Procedural Languages And Its Application to Structured Design

Takeshi Ogihara[1,a)]

Received: November 30, 2012, Accepted: January 11, 2013

Abstract: Recently, in the software development by object-orientation, data-binding techniques become popular, which make objects cooperate by binding their properties. Even using procedural programming languages which do not have object-oriented features, if some mechanism that can easily change the binding among software modules is provided, it is possible to smoothly modify and extend the software, preserving the independency and the reusability of modules. This paper proposes a data-binding mechanism named coval, which is implemented as a library in C. Covals transfer control via a variable shared among related modules. With covals, appending and replacing features become easier, suppressing modification of the existing code. This paper also shows small embedded programs with covals, and discusses the usage of covals in the structured design of systems.

Keywords: data binding, procedural languages, embedded systems, structured modeling, software components

1. はじめに

オブジェクト指向ソフトウェア開発では，オブジェクトのプロパティを相互に結合（バインド）させることによって，ソースコードを書き換えることなく，ソフトウェアの構成を変更する手法が用いられるようになっている．この手法はデータバインディングと呼ばれる．

データバインディングは，値の間に依存関係のある複数のプロパティに注目し，オブジェクトを結合させるものである．一方が値を更新すると，他方はそれに応じたアクションを自動的に行う．プロパティ間のバインディングの

[1] 京都産業大学コンピュータ理工学部
Faculty of Computer Science and Engineering, Kyoto Sangyo University, Kyoto 603–8555, Japan
[a)] ogihara@cse.kyoto-su.ac.jp

© 2013 Information Processing Society of Japan

指示は，そのアプリケーションの部品の組み合わせ方の記述であり，クラスやモジュールの機能からは独立している．そのため，XML 形式のデータなどとしてソースコードの外に記述することも多い．

データバインディング自体はオブジェクト指向の概念と直接関係しているわけではないため，C 言語をはじめとする手続き型言語（以下，本稿では非オブジェクト指向の言語という意味で用いる）にも適用可能であると考えられる．C 言語は現在も組込みシステムの開発などで広く利用されているが，C 言語に適用可能なバインド機構の提案はこれまでにない．

本稿では，手続き型言語の複数のモジュール間でデータバインディングを実現するための機構として coval [1], [2] を提案し，システムの設計と実装に適用する方法について論じる．

coval [*1] は，モジュール間で共有される値を保持し，値の更新を契機としてモジュールの持つ手続きを呼び出すことができる．coval を相互に関連づけ，複数のモジュールが連携できるようにする操作をバインドと呼ぶ．バインドを利用してモジュール間の関係を変更したり，新たなモジュールを追加したりする際，既存のモジュールのコードを書き直す必要はない．このため，手続き型言語においても，システムの変更や機能追加を容易に行うことができる．

coval は C 言語のライブラリとして実装され，必要なメモリ量，オーバヘッドはわずかである．OS の存在しない組込み環境であっても問題なく動作することを確認している．

以下では，オブジェクト指向開発におけるデータバインディングの例を示し，その概念について論じる．次に coval の仕組みと機能について説明し，簡単なプログラム例を示す．さらに構造化モデリングにおいて coval を利用する方法について述べ，C 言語によるシステム開発においてもデータバインディングの考え方が有用であることを示す．

2. バインド機構

2.1 データバインディングの概要

ここでは，Microsoft 社の .Net Framework の開発基盤の 1 つである Windows Presentation Foundation（WPF）の解説 [3] を引用し，データバインディングの概要を説明する．

図 1 において，右側は共有される値を管理するソースプロパティで，左側はソースプロパティの変化に従うターゲットプロパティを表す．典型的な例としては，ソースプロパティがモデル，ターゲットプロパティがユーザインタフェース（UI）部品である場合を想定できる．OneWay は，ソースプロパティに対する更新によってターゲットプロパティが自動的に更新される場合である．一方，ターゲットプロパティの更新はソースプロパティに反映されない．例として，現在時刻を表示する場合を考えることができる．TwoWay はソースプロパティ，ターゲットプロパティのどちらの更新も，他方を自動的に更新させる場合であり，例として，アラームを鳴らす時刻を表示，再設定する場合を考えることができる．OneWayToSource は OneWay の逆で，ターゲットプロパティの更新がソースプロパティを更新する場合である．また，図には記入されていないが，ソースプロパティの値でターゲットプロパティを初期化し，それ以降は変更を反映させない OneTime というタイプもあげられている．

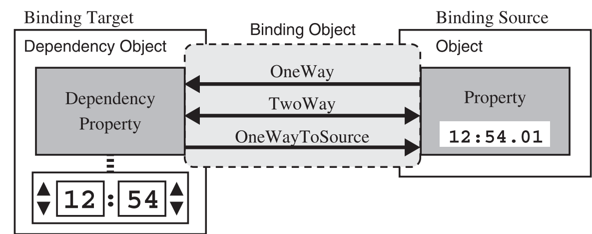

図 1　WPF データバインディングの概念—文献 [3] の図に加筆
Fig. 1　The Concept of WPF Data Binding [3].

2.2 データバインディングのいくつかの例

前節で示した WPF データバインディング [3] では，プロパティを相互に結び付ける役割のオブジェクトが必要であり，Binding オブジェクトと呼ばれる．ターゲットプロパティは複数個存在していてもよい．プロパティ間のバインドに関する情報を，XML ファイルに記述しておくこともできる．開発言語としては C# や Visual Basic などが利用できる．

JavaFX は，いわゆるリッチインターネットアプリケーション（RIA）の構築のためのプラットフォームと位置づけられており，XML の記述によって UI が作成できる．JavaFX のデータバインディング [4] でも，XML 記述から一部の機能を利用でき，プロパティ間のバインディングのためにはオブジェクトを用いる．単に値を共有，同期するだけではなく，更新した値に演算を適用して相手のプロパティに伝えるなどの機能も提供している．

Cocoa バインディング [5] は Apple 社の Mac OS X のアプリケーション開発に用いられる．プロパティ間の関連付けは開発環境のビジュアルエディタで行えるため，ソースコードに記述する必要はない．Cocoa バインディングは Objective-C のランタイムシステムの機能を使って実現されている．

Adobe 社の RIA 開発環境である Adobe Flex でもバインディングが可能である [6]．また，JavaScript にも Backbone.js [7] をはじめ，データバインディングを可能にするフレームワークがいくつか存在する．

[*1] coval の名は化学の共有結合（covalent bond）から借りている．

© 2013 Information Processing Society of Japan

2.3　データソースとのバインディングについて

XML や JSON などの形式のデータをソースコードとマッピングするための技術もデータバインディングと呼ばれている．特に，XML スキーマからソースコードを生成し，プログラムとデータのインタフェースとして利用する手法を XML バインディングと呼び，Java や C 言語をマッピングの対象とした提案がいくつか存在する [8], [9].

ただし，これらはリレーショナルデータベースに対する O/R マッピング [10] と同様，データソースのスキーマをオブジェクトのインタフェースで表すことが主な目的である．本稿では 2.1 節で示したように，プロパティどうしを動的に関連づけることによってモジュール間を連携させる仕組みを扱う．

2.4　本稿におけるバインド機構

以上のように，既存のバインディング手法はオブジェクト指向環境で利用するために提案されている．

一方，手続き型言語においても，モジュール，または動的にメモリ領域を獲得した構造体に属性値を持たせ，その値によってプログラムの動作を変えたり，処理結果を値に反映させたりすることができる．このような属性値は，オブジェクト指向言語におけるプロパティと同じ役割を持つものと見なせる．これらを相互に連携させることによって，手続き型言語を使ったシステムの構築においてもデータバインディングと同じ効果を得ることが可能である．

本稿では，このような機能を提供する仕組みとして coval を提案する．coval は一種の変数として値の取得と設定を行うことができ，さらに値の取得および設定の際に自動的に起動する関数を設定しておくこともできる．coval は相互にバインドして値を共有でき，1 つの coval で値を更新することによって他の coval の関数が自動的に起動されるようにできる．この仕組みにより，2.1 節で示したプロパティの振舞いが実現でき，オブジェクト指向環境におけるデータバインディング手法と同様なシステム構築が可能になる．

次章で coval の機能と実装について，より詳しく説明する．

3.　coval の概要

3.1　coval 構造体

coval は，モジュール間で値を共有する仕組みを備えた一種の変数であり，実体は構造体として実装される．coval には値を格納，参照することができ，さらに同じ型のデータを保持する他の coval と値を共有するようにできる．この操作をバインドと呼ぶ．バインドされたすべての coval は共通の変数を参照しているため，いずれかの coval で値を更新すると他のすべての coval の値も変化したように見える．

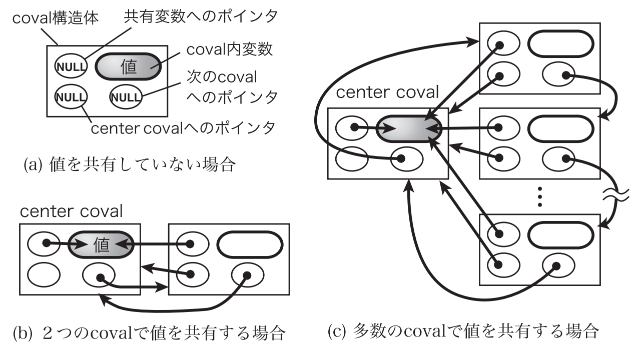

図 2　coval 構造体とバインドの実現
Fig. 2　Structure of coval and the implementation of binding.

表 1　coval に関する主なマクロ
Table 1　Macros applied to covals.

マクロ名	機　能
CovalTypeOf(T)	T 型の値を持つ coval 構造体型を表す
CovalBind(cp1, cp2)	2 つの coval をバインドする
CovalUnbind(cp)	バインドを解消する
CovalSetCallback(cp, f, vp)	コールバック関数を設定する
CovalValue(cp)	coval の値を参照する
CovalAssign(cp, s)	coval の値を更新する

注) cp：coval 構造体へのポインタ，vp：void *型のポインタ，
f：関数ポインタ，s：共有するデータ型の値．

coval にはコールバック関数を対応させておくことができ，バインド中に共有変数に対して更新の操作が行われると自動的に呼び出される（3.3 節，3.4 節を参照）．更新の前後で値が変化しなくても呼び出しは発生する．関数を指定しなければ，値が更新されても，その coval については何も起きない．バインドされた coval どうしがコールバック関数を指定しているかどうかで，OneWay や TwoWay などの動作を実現できる．

coval 構造体の概念を**図 2** (a) に示す．構造体はバインドに必要な情報のほか，図では省略したが，起動される関数へのポインタなども含む．coval は，任意の型の値を扱えるように，マクロとライブラリを組み合わせて実装されている．型 T のデータを共有変数として内部に含む coval 構造体の型は，CovalTypeOf(T) と表される（**表 1**）．

3.2　バインドとアンバインド

バインドはマクロ CovalBind（表 1）を用いて，プログラムの実行中に動的に行う．マクロの第 1 引数の coval はすでにバインドされていてもよいが，第 2 引数にはまだバインドされていない coval を指定する．バインド後に共有される値は，第 1 引数が保持，または共有している値になる．図 2 (b) は 2 つの coval 構造体が値を共有する例である．

バインドされた coval の集合を coval グループと呼ぶ．coval グループ内には，共有値を保持し，動作を管理するための coval が 1 つ存在し，center coval と呼ぶ．図 2 (c) は

© 2013 Information Processing Society of Japan

複数の coval が center coval の保持する変数を共有する概念を示す．ただし，coval へのアクセスにはすべてマクロを用いるため，coval 構造体の具体的な内容や center coval の存在を意識してプログラミングする必要はない．center coval も含め，任意の coval はバインドを解消して，共有関係から取り除く（アンバインドする）ことができる．

3.3 コールバック関数の指定

coval にコールバック関数を設定するには，マクロ CovalSetCallback を使う（表 1）．コールバック関数は次の形式で定義しておく．第 1 引数は，この関数を指定した coval 変数へのポインタ，第 2 引数は任意のポインタである．

```
int 関数名 ( coval へのポインタ , void * )
```

coval 変数 cov にコールバック関数 fn を設定するには次のようにする．引数 ptr は任意のポインタで，関数 fn が起動されるときに第 2 引数として渡される．これによって，複数の coval に対して同じコールバック関数を指定しても，引数によって動作を変えることができる．

```
CovalSetCallback(&cov, fn, ptr);
```

コールバック関数は coval をバインドする前に設定しておけるため，ローカルな関数を使って，実装をモジュールの内部に隠蔽することができる（4.2 節の例を参照）．

3.4 値の参照と更新

共有変数の値の参照，更新はマクロ（表 1）を介して行う．個々の coval は 3.1 節で示した型を持ち，共有変数に対する操作に関して，コンパイル時に型のチェックが行われる．

値を更新するマクロは CovalAssign で，共有変数の値を更新した後，coval グループ内にコールバック関数があれば順番に起動する．ただし，引数の coval 自体に設定されている関数は呼び出さない（4.2 節の例を参照）．coval がバインドされていない場合，その coval の持つ値が更新されるだけである．

値を参照するマクロ CovalValue は，coval がバインドされていれば共有変数の値を返し，バインドされていなければ，その coval の持つ値を返す．

3.5 実装とオーバヘッド

coval の機能は C 言語のライブラリとして実装した．標準ライブラリも含めて，他のライブラリにはいっさい依存していない．このため，標準的なライブラリが整っていない組込み機器での利用にも問題はない．実行に必要なメモリは，コード部分以外には，各 coval 構造体に割り当てられる分だけである．32 ビットモデルの Intel チップ向けのコードを gcc コンパイラで生成した場合，int 型を保持する coval 構造体は 44 バイトで済む．コンパイル後のコード（テキスト）は約 2.1 キロバイトである（比較のためオブザーバパターンの例題 [11] を調べると，サーバの部分が約 2.8 キロバイトであった）．

coval 構造体が値を保持する部分は，バインドされている間は使われない．また，center coval でしか利用しない領域も多い．動的にメモリを割り当てることでこういった領域の無駄を省く実装も考えられるが，現状では，次節で述べるような組込み環境での利用も考慮し，動的にメモリ獲得を行わない実装としている．

実行時には，共有変数にアクセスする際のポインタの参照と，coval グループ内を走査して関数を順番に呼び出す部分がオーバヘッドとなる．

4. coval の使用例

4.1 H8 マイコン基板とプログラム例

市販されている実験用マイコン基板 [13] のプログラムに coval を含めて，動作することを確認した．実験環境は，CPU（H8SX/1655，48 MHz）基板にアナログ入出力などを備えた拡張基板，タッチセンサを備えたカラー LCD（320×240 ドット）基板を増設したものである．メモリは ROM が 512 kB，RAM が 40 kB のみで，OS はないが，スタートアップルーチンやタッチパネルの制御用ルーチンは提供されている．

タッチパネルの制御用ルーチン（サンプル）では，10 ms 間隔で割込みを繰り返してタッチの有無を判定し，タッチ位置の座標などの情報を特定の変数に格納する．これを利用するアプリケーションでは，イベント取得ループの中で変数の値を調べ，タッチがあれば対応する動作を行うというプログラムを記述すればよい．

図 3 は簡単なプログラム例で，スライダの役割をする部品を操作して，そこから得られた値を 1 辺の長さとする正方形を描く．関数 GC_Slider はスライダ領域内にタッチがあれば，スライダを描き直してから PEN_DOWN という値を返す．スライダのノブの位置が構造体のメンバ knobx

```
sFrame sFMEM;    // struct for the LCD
sliderBody sld;  // struct for the slider
/* 中略 */
while(1) {
  if (GC_Slider(&sld, &sFMEM) == PEN_DOWN) {
    drawRect(sld.knobx, sld.knobx);
    // スライダが操作されたら正方形を描画する
  }
  if (resetRect(&sFMEM) == PEN_DOWN) {
    sld.knobx = 0;
    // 正方形にタッチされたら値を 0 に設定する
    draw_slider(&sld); // スライダを描画する
  }
}
```

図 3 タッチパネルのイベントを得て動作するループ部分
Fig. 3 A loop to get events of the touch panel.

© 2013 Information Processing Society of Japan

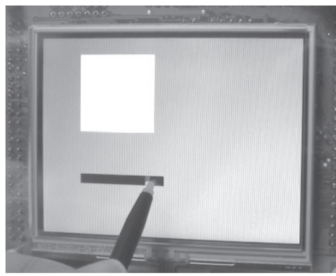

図 4　プログラムの動作例
Fig. 4　Operation of the system.

```
typedef CovalTypeOf(UI32) UI32coval;
extern UI32coval slcov;    // coval of the slider
extern UI32coval sqcov;    // coval of the square
extern UI32coval dgcov;    // coval of the digits
sliderBody sld;            // struct for sliders
// バインドの前にコールバック関数を設定しておく必要がある
CovalBind(&slcov, &sqcov); // 3つの coval をバインドする
CovalBind(&slcov, &dgcov);
/* 中略 */
while(1) {
  if (GC_Slider(&sld, &sFMEM) == PEN_DOWN) { // スライダが操作されたら
    CovalAssign(&slcov, sld.knobx);          // coval に値を設定する
  }
  if ( resetRect(&sFMEM) == PEN_DOWN ) { // 正方形にタッチされたら
    CovalAssign(&sqcov, 0);              // 値を 0 に設定する
  }
}
```

図 5　イベントに応じて coval の値を更新
Fig. 5　Shared value of the coval is updated according to events.

に格納されているので，この値を使って関数 drawRect で正方形を描く．また，関数 resetRect は，描画された正方形の左上付近がタッチされた場合，正方形を最小化すると同時に PEN_DOWN を返す．このときにスライダも最小値に戻して描画しなおすようにしている．**図 4** に画面の例を示す．

4.2　coval の適用の例

図 3 では，2 つの GUI 要素が互いに反応しあうようにするため，関数を呼んで再描画しあうコードになっている．

これを，coval を利用して書き換えたものが**図 5**，および**図 6** のコードである．ここではさらに，スライダの現在値を文字で表示する機能も追加した．図 5 では，スライダ，正方形と文字列のそれぞれに対応する coval を利用しているが，これらに対応するコールバック関数の定義，設定は図 6 で行っている．図 5 ではこれらの coval をバインドして，スライダか正方形にタッチがあれば，対応する coval に対して値の更新を行う．coval の値の更新には CovalAssign マクロを使っているので，バインドされている coval のうち，更新した以外の coval のコールバック関数が起動される．たとえば，スライダが変更された場合には新たな大きさの正方形が描画され，その値が文字列として描かれる．**図 7** に動作の概念を示す．

図 6 で，点線で区切られている部分は別々のファイルと

```
UI32coval slcov;       // coval of the slider

static int cb_slider(UI32coval *cv, sliderBody *sl)
{   // 第2引数にはスライダのポインタが渡る
    sl->knobx = CovalValue(cv); // coval の値を取得
    draw_slider(sl);            // スライダを再描画
    return 0;
}
/* 中略... 次の行でコールバック関数を設定 */
  CovalSetCallback(&slcov, cb_slider, &sld);
............................................................
UI32coval sqcov;       // coval of the square

static int cb_rect(UI32coval *cv, void *_) {
    UI32 v = CovalValue(cv); // coval の値を取得
    drawRect(v, v);          // 正方形を再描画
    return 0;
}
/* 中略... 次の行でコールバック関数を設定 */
  CovalSetCallback(&sqcov, cb_rect, NULL);
............................................................
UI32coval dgcov;       // coval of the digits

static int cb_digit(UI32coval *cv, void *_) {
  /* 省略．画面上に現在値を文字列で描く */
}
/* 中略... 次の行でコールバック関数を設定 */
  CovalSetCallback(&dgcov, cb_digit, NULL);
```

図 6　coval とコールバック関数の定義
Fig. 6　Definition of covals and their callback functions.

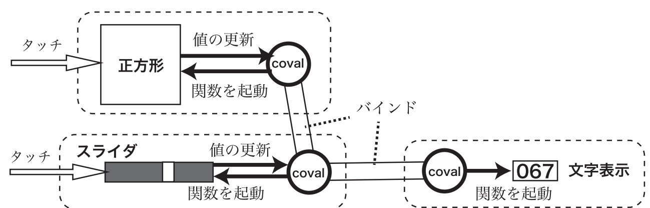

図 7　図 5，図 6 の動作概念
Fig. 7　An illustration of Fig. 5 and Fig. 6.

しておくこともでき，コールバック関数は外部から隠蔽される．また，coval がバインドされているかどうかは，図 5 の部分の動作には影響しない．たとえば，画面に文字を描く機能は，変数 dgcov をバインドすれば追加でき，バインドしなければ表示はされない．

このように，各部品には coval の値の更新とコールバック関数について記述しておき，必要な場合にバインドするだけでよい．部品が相互に呼び出しあうようなコードを記述したり，部品のつながり方が変更された際に部品の振舞い自体を書き換えたりする必要はない．

5.　coval を利用したソフトウェア設計

5.1　構造化モデリング手法と coval の利用

ここでは，手続き型言語で開発を行う場合に広く利用さ

© 2013 Information Processing Society of Japan

図 8　タンクの概念図
Fig. 8　An illustration of the tank.

図 9　水位制御のデータフロー図
Fig. 9　A data flow diagram of the tank.

図 10　coval を用いたデータフロー図
Fig. 10　A data flow diagram with covals.

図 11　coval を用いた構造図
Fig. 11　A structure chart with covals.

れている構造化モデリング手法 [12] に基づき，coval によるデータバインディングの利用を考慮に入れて設計を進める方法を提案する．ここで用いるモデリング手法は，データフロー図（DFD）に制御フローの概念を導入し，リアルタイムシステムの制約にも対応できるように拡張したもの [14] である．

提案手法では，DFD に記述されたデータフローのうち，プロセス間で共有されている値を表しているものを見つけ出す．このようなデータフローは coval で実装できることが多い．そのようなデータフローは coval の利用を表す記法で書き改め，構造図においても coval を共有するモジュールが明示されるようにする．

5.2　モデリングの例題

例題として，タンクの水位を管理する制御システムのシミュレータを考える．このタンクにはつねに水が蓄えられ，必要に応じて取水口から水が取り出される．水位がタンクに設定された最低値を下回ると給水栓が開き，タンクへの給水が行われる．水位が設定された最高値に達すると給水は停止する．このシミュレータでは，取水栓を開く動作を，利用者がボタンをタッチすることで表す．取水栓，給水栓を流れる単位時間あたりの水量は決まっているものとする．このタンクの概念図を**図 8** に示す．

シミュレータは一定の時間間隔でタンクの水位と取水栓，給水栓の状態を調べ，次の動作と状態を決めるものとする．この考え方に沿って記述した DFD を**図 9** に示す．実線の矢印はデータフロー，破線の矢印は制御フローを表す．

5.3　DFD と構造図におけるバインディングの記述

図 9 の DFD では，プロセス 1 とプロセス 3 が取水栓の状態を共有し，プロセス 4 とプロセス 5 が給水栓の状態を共有している．また，プロセス 3，4，5，6 の 4 つはタンクの水位を共有している．これらを coval で実装することに決めて図 9 を書き直したものが**図 10** である．ここでは平行な二重線で coval を表している．coval からプロセスに向かう破線の矢印は，値の更新によってプロセスが活性化されることを示す．

図 10 をもとに，構造図に変換したものが**図 11** である．ここでも coval は平行な二重線で示している．coval とモジュールの関連は破線で示され，手続き呼び出しではなく，バインディングで関連づけられることを表現している．ただし，コールバック関数を呼び出すことが決まっている場合には，coval からモジュールへ向けて矢印を書く．モジュール 4 は親モジュールから直接呼び出されるのではなく，coval の値が更新されることで起動される．

この構造図を参照することによって，coval によるバインディングが利用可能なモジュールを意識してコーディングを行うことができる．

通常は構造図の段階でモジュール間の呼び出し関係が決定されるが，いったん決められた呼び出し関係は後から変更しにくい．coval によるバインドで実現された部分は独立性が高く，変更や追加が容易である．また，双方向の情報交換や，多数個のプロセス間の情報共有といった柔軟性を取り入れることもできる．

たとえば，シミュレータに給水栓の状態を表示するように追加要求がなされた場合，従来の設計では，いずれかのモジュールを変更して描画のための手続き呼び出しを行うようにする必要がある．一方，coval を用いた実装では，coval を備えた描画用モジュールを作成し，給水栓の状態を表す coval にバインドするだけでよい．

© 2013 Information Processing Society of Japan

6. 議論

6.1 関連研究

共有変数を使ってモジュール間を連携させる仕組みを持つ言語として，仕様記述言語 SpecC [15] がある．SpecC は C 言語を拡張した文法を持ち，生成したコードを実行可能である．SpecC ではプログラムの振舞いの単位をビヘイビアと呼ぶ．ビヘイビアは情報の出入り口であるポートを備え，複数のビヘイビアがポートを経由して変数を共有できる．また，ポートをチャネルで結合することでビヘイビア間の連携を記述できる．このような構造はプロパティ間のバインドと似ているが，SpecC はコンパイル済みの部品間の接続を動的に変更する手段を提供しておらず，データバインディングとして利用することはできない．

事象の発生を伝達するデザインパターンとしてオブザーバパターンが広く知られている．オブザーバパターンは C 言語などの手続き型言語でも実装でき [11]，組込みシステムのミドルウェアの一部として利用する提案 [16] もある．オブザーバパターンや，複数のクライアントが値を共有するためのデータバスパターン [17] などは，データバインディングの機能と重なる部分が多いように見える．しかし，モジュール間の連携動作のために，値の共有の方法と，値の変更にともなって自動的に手続きを呼び出す仕組みを一体として提供することが本稿でのデータバインディングの目的であるのに対し，これらのデザインパターンはより一般的な機能の枠組みを提供しているにすぎない．したがって，これらのデザインパターンは，データバインディングの実装に利用することは十分考えられるが，データバインディングそのものの機能を提供するものではない．

6.2 coval を利用することの得失

データバインディングは，プログラムのロジック部分と UI 部品，あるいはデータソースとの間で利用するとメリットが大きい．本稿でも，タッチパネルによる割込みを契機に動作するプログラムの例を取り上げ，動作を確認した．

coval によるバインディングは手続き呼び出しと異なり，値を更新する側，手続きが呼び出される側は 1 つには限定されず，双方向，あるいはマルチキャスト的に制御を伝播させることも可能である．いったん coval を使って機能を実現していれば，その coval に対して異なるモジュールを関連づけたり，複数のモジュールを対応させたりすることができる．しかもその際，既存のコードには変更を加える必要がない．既存のソースコードの一部を coval を使った実装に書き換えることも容易である．

一方，coval への操作によって起動されるコールバック関数の有無や処理内容は隠蔽されるうえ，バインド先はソースコードからは分からない．このため，手続き呼び出しのみのプログラムに比べ，実際の動作における処理時間や負荷が分かりにくいという問題がある．特に，リアルタイム制約のあるシステムではこの点に関して注意が必要である．

6.3 他のデータバインディング手法との比較

オブジェクト指向開発における代表的なデータバインディングの例は 2.1 節で述べた．これらの手法では，オブジェクトのプロパティをそのままバインドの対象にできるのに対し，提案手法では共有したい値をあらかじめ coval 構造体で定義しておかなければならない．これは本手法の対象が C 言語であり，オブジェクト指向言語のようにクラスや実行環境に手を加えることができないため，やむをえない点である．

オブジェクト指向言語では，あるオブジェクトの直接のプロパティだけではなく，プロパティ名をドット（.）で次々に連結した表記を用いて，バインドするプロパティをより柔軟に指定できる．このような表記をプロパティパス [3]，あるいはキーパス [5] などと呼び，文字列での指定も可能である．coval はメタデータに相当する情報を持たず，動的な解釈実行の仕組みも備えていないため，パスによる表記には対応できない．ただし，coval 構造体と何らかの識別子を関連づける連想配列のような仕組みを導入すれば，実行中でも識別子を指定して coval を特定できるようになる．このような仕組みを使って，ソースコードの外部にバインディングの指示を記述することも可能である．

オブジェクト指向のデータバインディング手法には，単に値の更新を伝えるだけではなく，型の変換や，簡単な演算を自動的に行うものもある．coval にはそのような機能は備わっていない．しかし，coval グループ間を連結するために用意した coval bridge [1] という仕組みを利用することで，型の変換や演算を行わせることも可能である．本稿では詳細は省略する．

6.4 コールバック関数の起動方法について

ある coval のコールバック関数が呼び出された結果，別の coval に影響が及ぶという関係が循環してしまうと，処理が収束しなくなる可能性がある．また，コールバック関数が次々に呼び出される関係が木構造を構成する場合がある．このとき，関数の呼び出し方法は深さ優先となるが，この動作が必ずしも設計の意図に沿っているとは限らない．

このような状況は，連携しあうオブジェクト間でも発生する可能性があり，coval に特有の問題ではないが，防止，あるいは軽減のための方策は必要である．1 つの方法として，コールバック関数の呼び出しを遅延するための実行キューを用意し，制御に用いることが考えられる．

なお，本稿で示した coval の実装では，並列処理環境での実行について特別な仕組みは用意しておらず，スレッドセーフではない．コールバック関数の並列動作や，スレッド（タスク）間の同期・通信の機能の必要性については今

© 2013 Information Processing Society of Japan

後の検討が必要である．

6.5 構造化モデリングによる設計と実装

本稿では，coval の利用を想定して DFD と構造図による設計を進める方法を提案した．

ただし，提案した手順や記法には改善の余地が多分に残されている．特に，バインディングによる動的な連携や，複数の coval が同じコールバック関数を使う場合などを構造図で記述することは難しい．UML なども利用することも含め，概念を整理するための分かりやすい記法が望まれる．

7. おわりに

本稿では，手続き型言語で利用可能なデータバインディングの仕組みとして coval を提案し，実際に coval を用いてソフトウェア開発が行えることを示した．さらに，構造化モデリングの手法に coval を組み合わせて設計を進める方法について提案を行った．

今後は実用的なシステム，規模の大きなシステムに対しても coval の有効性を確認することと，同時に，性能面でのより詳細な評価が必要である．

謝辞 本研究は科学研究費（基盤（C）22500038）の助成を受けたものである．

参考文献

[1] 荻原剛志：共有変数を用いたバインド方式の提案とソフトウェア開発への応用について，情報処理学会ソフトウェア工学研究会報告，Vol.2011-SE-171, No.23, pp.1–8 (2011).

[2] 荻原剛志：共用変数によるバインド機構を用いた組込みシステムの開発手法について，電子情報通信学会ソフトウェアサイエンス研究会報告，Vol.111, No.481, SS2011-58, pp.7–12 (2012).

[3] Microsoft: Data Binding Overview (.NET Framework 4.5) (online), available from ⟨http://msdn.microsoft.com/en-us/library/ms752347.aspx⟩ (accessed 2012-11-19).

[4] Oracle (Hommel, S.): Using JavaFX Properties and Binding (JavaFX 2 Tutorials and Documentation) (online), available from ⟨http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm⟩ (accessed 2012-11-19).

[5] Apple: Introduction to Cocoa Bindings Programming Topics (online) (2009), available from ⟨http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaBindings/CocoaBindings.html⟩.

[6] Adobe: Using ADOBE FLEX 4.6 (online) (2012), available from ⟨http://help.adobe.com/en_US/flex/using/flex_4.6_help.pdf⟩.

[7] Jeremy Ashkenas, DocumentCloud Inc.: Backbone.js (online), available from ⟨http://backbonejs.org⟩ (accessed 2012-11-19).

[8] Oracle: Java Architecture for XML Binding (The Java Tutorials) (online), available from ⟨http://docs.oracle.com/javase/tutorial/jaxb/index.html⟩ (accessed 2012-11-19).

[9] Objective Systems: XBinder v2.2 C/C++ User Manual (online) (2011), available from ⟨http://www.obj-sys.com/xbinder-manuals.shtml⟩.

[10] Fowler, M.: *Patterns of enterprise application architecture*, Addison-Wesley (2003).

[11] Douglass, B.P.: *Design Patterns For Embedded Systems In C*, Elsevier (2011).

[12] SESSAME WG 2：組込みソフトウェア開発のための構造化モデリング，翔泳社 (2006).

[13] 山崎尊永ほか：今すぐ使える！ H8 マイコン基板 増補版，CQ 出版社 (2010).

[14] Hatley, D.J. and Pirbhai I.A.: *Strategies for real-time system specification*, Dorset House Pub. (1987).

[15] Dömer, R., Gajski, D.D. and Gerstlauer, A.: SpecC Methodology for High-Level Modeling, *9th IEEE/DATC Electronic Design Processes Workshop* (2002).

[16] Bellebia D. and Douin, J.-M.: Applying patterns to build a lightweight middleware for embedded systems, *Proc. 2006 Conf. Pattern Languages of Programs* (*PLoP '06*), ACM, Article 29, DOI: 10.1145/1415472.1415506 (2006).

[17] Douglass, B.P.: *Real-Time Design Patterns*, Addison-Wesley (2003).

荻原 剛志 （正会員）

平成 2 年大阪大学大学院基礎工学研究科博士後期課程修了．同年大阪大学情報処理教育センター助手．平成 5 年奈良先端科学技術大学院大学情報科学センター助教授．平成 7 年神戸大学工学部助教授．平成 17 年高知工科大学工学部教授．平成 19 年大阪大学大学院情報科学研究科特任教授．平成 20 年 4 月より京都産業大学コンピュータ理工学部教授．ソフトウェア工学，深層暗号等の研究に従事．Mac OS X，iOS 上のソフトウェア開発に興味を持つ．工学博士．IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．

© 2013 Information Processing Society of Japan