

手続き型言語に適用可能なモジュールのバインド方式について

荻原 剛志

京都産業大学コンピュータ理工学部

オブジェクト指向のスタイルによらない手続き型言語を用いた開発では、抽象度の高い記述が難しいためにモジュール間の依存度が高い。本稿では共有変数を使って複数のモジュールを連携させる言語機構、タップを提案する。タップを利用することによって、具体的な手続き名を互いのモジュールに記述する必要がなくなり、独立性を高めることが可能である。また、オブジェクト指向におけるデザインパターンであるオブザーバパターンを手続き型言語でも利用できるようになる。

A Binding Mechanism for Procedural Programming Languages

Takeshi Ogiwara

Faculty of Computer Science and Engineering, Kyoto Sangyo University

In procedural programming style, software modules tend to depend each other. Comparing to object-oriented programming, it is difficult to describe highly abstracted modules. This paper shows a new binding mechanism called "tap", which can connect modules each other using a shared variable. With taps, independency of modules can increase, because procedure/function identifiers do not have to be written in other modules. Additionally, one of object-oriented design patterns, observer pattern is available with taps.

1. はじめに

C言語をはじめとする手続き型（本稿では非オブジェクト指向の言語の意味で使う）のプログラミング言語は、組み込み用機器などを中心に現在でも広く利用されている（図1）。これらの言語では、複数のモジュールを相互に関連づけようとする場合、少なくとも一方のソースコードに他方の手続き名などの固有の情報を記述することが普通である。しかし、モジュールを結合させるためのグルーコードはモジュールの独立性を低下させる要因となる。ソフトウェアの改変や既存のソースコードの再利用の際には、不要なグルーコードを削除したり、新しい用途

向けの別のグルーコードを書き足したりする作業が必要となる。

一方、オブジェクト指向言語では、クラスの継承機構などを利用した抽象度の高い記述が行えることから、独立性の高い記述が可能なデザインパターンが広く用いられている。さらに、リフレクションを利用してソースコードの改変なしに既存のモジュールを連携させるコンポーネント指向の開発手法も利用されている。しかし、これらの方法を手続き型言語にそのまま適用することは難しい。

本稿では、NeXTstepのアプリケーション開発に用いられたターゲット-アクションパラダイム、および Mac OS X のアプリケーション開発

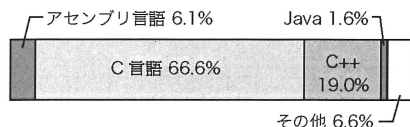


図1 組み込み開発における使用言語（文献[1]より）

で利用されている Cocoaバインディングの概念に注目し、共有変数を介してモジュール間の通信を行う方法を提案する。

この方法を利用することにより、手続き型言語を用いた開発においてモジュール間のグルーコードを削減し、再利用性の高いモジュールを実現できる。

この手法では具体的な接続関係の情報はソースコードに記述せず、リンク時の追加情報として与える。このため、接続関係が変化してもコンパイルし直す必要がない。また、関数呼び出しのように一対一の関係に限定せず、複数のモジュール間で情報を共有できる。また、特別なライブラリを利用せず、実行コードの量を増加させることがない点も特徴として挙げることができる。

2. モジュール間の連携

2.1 ターゲット-アクションパラダイム

ターゲット-アクションパラダイムは NeXT社の NeXTstep[2]、OPENSTEPにおける GUIアプリケーションの構築に用いられた技法であり、現在でも Apple社の Mac OS X および iPhone で使用されている。この技法はアプリケーション記述言語である Objective-C の特徴を利用している。

Objective-C はメッセージ名を決定するメッセージセレクタと引数を使って、どんなオブジェクトにもメッセージを送信できる。この機能を利用して、GUI部品にメッセージセレクタとターゲットのオブジェクトを登録しておき、部品がクリックなどの操作を受けた場合にターゲットにメッセージを送るようにできる。

図2は概念を表したもので、ボタンオブジェクトのターゲットはカスタムオブジェクトである ObjA、メッセージセレクタは activate: である。ボタンをクリックすると ObjA にメッセージ activate: が送られる。メッセージの引数は

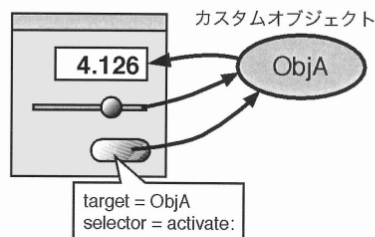


図2 ターゲット-アクションパラダイムの概念

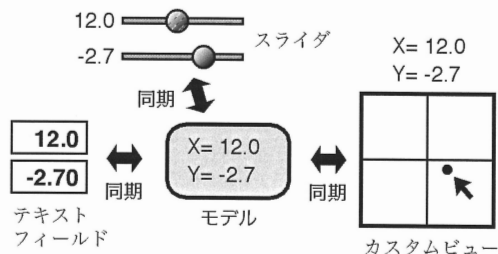


図3 Cocoaバインディングの概念

ボタンオブジェクト自体である。また、図2の矢印で示したような、GUI部品とオブジェクトの間の関連づけは支援ツールを用いることによって行うため、その部分に関してはソースコードをまったく記述しなくてよい。

2.2 Cocoaバインディング

Cocoaバインディング[3]は Mac OS X で導入されたソフトウェア開発手法で、モデルと呼ばれるオブジェクトの属性値の変化に応じて複数のオブジェクトに対応する動作を行わせることができる。

ターゲット-アクションパラダイムでは一対一のオブジェクトの関係しか表現できなかったが、Cocoaバインディングではあるモデルに対して複数のオブジェクトを対応づけておき、どのオブジェクトが値を変更した場合でも他のオブジェクトに対して対応した動作を起動させることが可能である。

図3は概念を表したもので、点の座標を表すモデルとその値を表示、操作できる3種類のオブジェクトが関連づけられている。どのGUI部品を操作しても座標値の変化がモデルに反映すると同時に他のオブジェクトの表示を自動的に変更する。これらのオブジェクトの関連付けも

支援ツールで行えるため、ソースコードに記述する必要はない。

なお、この機能は Objective-C のランタイムシステムが持つキー値監視という機構を利用しているため、Objective-C あるいは通常のオブジェクト指向言語の記述のみで実装を模倣することは容易ではない。

3. 提案する言語機構

3.1 タップの宣言

以下で提案する言語機構をタップと呼ぶ。タップの記述は他のモジュールと共有する変数の宣言と、タップの変更に伴って動作する手続きから成る。手続きはなくてもよい。モジュールの外部の情報は記述しない。

タップを利用するモジュールは、外部変数と同様にして以下の宣言をソースコード中に記述する。ただし、以下で記述するタップの構文は現時点での案であり、確定したものではない。

```
#tap type nameOfTap;
```

または次のように記述する。タップと共に記述されるコードはそのモジュールに固有のもので、共有はされない。

```
#tap type nameOfTap {  
    実行可能なコード  
}
```

プログラム内の複数個のタップがひとつの変数を共有できるが、それらのタップの名前は同じである必要はない。ひとつのモジュールに複数個のタップがあってもよい。共有される変数の実体がどのモジュールにあるのかについて留意する必要はない。

タップ名は変数名、関数名と同じ名前空間に属する。従って、C 言語では異なるモジュール間での名前の衝突の危険がある点に注意が必要である。

3.2 値の参照

プログラムのコード中で、共有している変数の値を使いたい場合は式として以下のように記述できる。

```
#( nameOfTap )
```

3.3 通知

他のモジュールに対して何かのイベントを知らせたい場合、タップに対して以下のようにして通知を発する。パラメータの値がタップの新しい値となる。

```
#nameOfTap( 値 );
```

タップの宣言で指定した型が構造体の場合、メンバを指定することができる。

```
#nameOfTap.メンバ( 値 );
```

通知が送られたら、そのタップを共有している複数のモジュールでは、タップとともに記述されている実行可能なコードが実行される。ただし、タップに通知を発したモジュールではコードは実行されない。

3.4 タップによる共有の例

図4にタップによる変数の共有例を示す。例には3つのソースファイルと合計4つのタップがある。タップを結合する情報が別に与えられており、シンボル `info` と `statA`、および `cell` で変数を共有することが指定されているとする。一方、シンボル `statB` には共有の指定はない。

このソースコードをリンクして作成したプログラムでは、シンボル `info`、`statA`、`cell` は同一の変数を参照する。例えば `info` に対して通知を発すると、`src2` の `statA` に対応するコードが実行される。`src3` の `cell` には対応するコードが

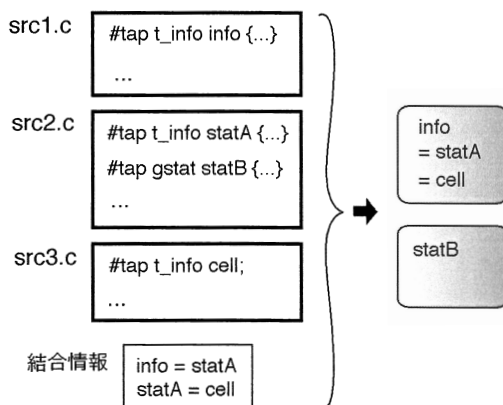


図4 タップによる変数の共有

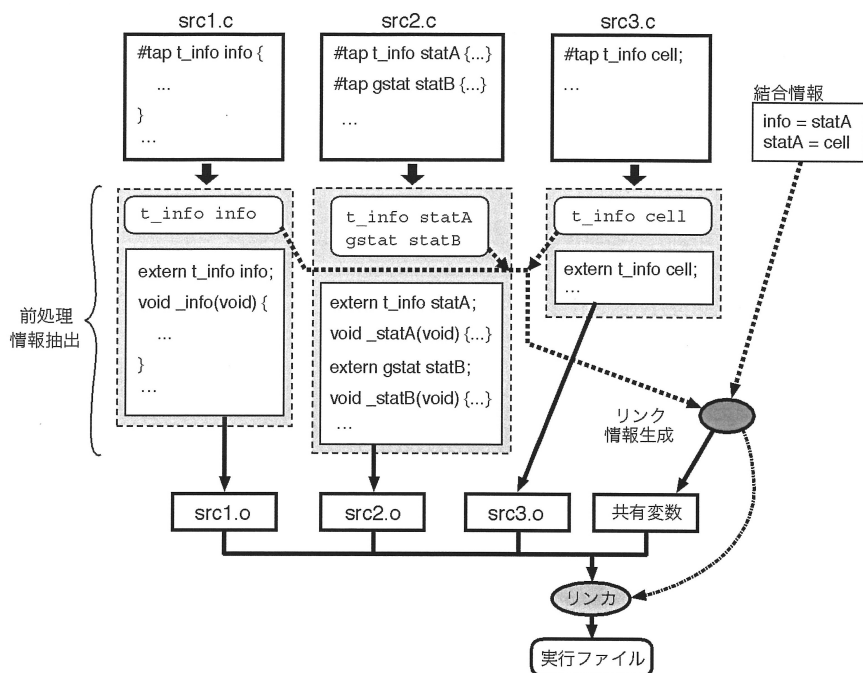


図5 タップを利用するためのコンパイル／リンク手順

指定されていないため、通知によって手続きが起動されることはない。

シンボル `statB` に対応する変数は共有されない。従って、`statB` に対して通知を発しても何の動作も起動されない。

4. タップの実装

4.1 前処理を前提とした実装方法

図5にコンパイルの前処理とした実装方法の概要を示す。この方法では、タップの記述を含むソースファイルに前処理を行い、タップの情報を抽出するとともに通常のCコンパイラで処理できるソースファイルを出力する。得られたソースファイルにはタップで指定した変数に対する `extern` 宣言と、通知に対応して起動する関数が生成されており、そのままコンパイルが可能である。

一方、抽出された情報にはタップ名と対応するデータ型、通知に対応する関数名（生成したもの）が含まれる。この情報と、ソースコードとは別に用意するタップ間の結合情報を照合して、どのタップがどんな型の変数を共有するかを調べることができる。

この結果から、必要な変数のデータ型と個数を得て、共有変数とコールバックのためのデータ構造（後述）を含むオブジェクトコードを生成する。このオブジェクトコードと、前処理によって得られたソースファイルをコンパイル、リンクすれば、結合情報の通りにタップが変数を共有した状態の実行ファイルを得ることができる。なお、オブジェクトコードの生成には、アセンブリ言語のコードを使う方法とC言語のソースを合成する方法が考えられる。C言語のソースを用いる方が汎用性があるが、リンクスクリプトなどを利用してリンクに対して付加的な情報を渡す必要がある。

4.2 共有変数のデータ構造

上で述べたように、前処理済みのソースファイルでは、タップは単なる外部変数として見える。タップを使って他のモジュールに通知を発するために次のように実装する。

タップが共有する変数は、変数として値を格納する領域の直後に、その変数を共有しているタップが通知を受け取った時に実行する関数へのポインタのリストを持つ（図6）。共有変数の

共有変数
関数ポインタの個数
関数ポインタ 1
関数ポインタ 2
⋮
関数ポインタ n

図6 タップの共有変数と関数ポインタ

大きさは分かっているため、実装上の問題はない。いずれかのタップが通知を発すると、共有変数の値を書き換えた後、関数ポインタを先頭から順番に呼び出す。ただしこの時、通知を発したタップに関連する関数は呼び出さない。

通知を受け取って手続きを実行するタップがひとつもないか、タップが共有されていない場合には関数ポインタの個数は0となる。この場合、共有変数へのアクセスは関数呼び出しを伴わずに実行可能である。

5. タップを用いたプログラム例

5.1 GUIを用いたプログラム

組込みシステムでは、オブジェクト指向言語を用いずにGUI、または類似のインタフェースを用いたシステムを実現しなければならない場合もある。このようなシステムでは、各部品とコールバック関数を逐一割り当てたり、すべての部品を一元管理するモジュールを定義するなど、さまざまな手法がとられるが、多くのグルーコードを記述しなければならない例も少なくない。

このようなシステムにおいて、部品とイベント処理のルーチンでタップを利用できれば、図

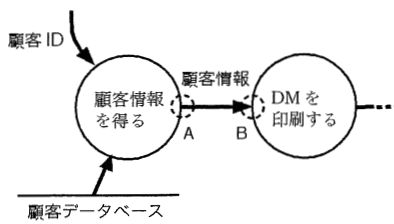
2、図3のようなインタフェースを実現することが可能になる。

5.2 構造化設計に基づくシステム

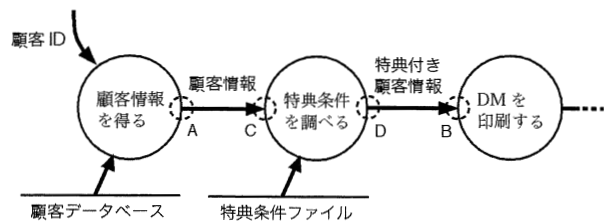
構造化設計に基づくシステムでは、基本的にデータフローダイアグラム(DFD)のプロセスに相当する部分を手続きとして実現し、データフローは手続き呼び出しとして実装する。この方法ではデータフローのどちらかを呼び出し側、どちらかを呼び出される側と決めてプログラミングする必要がある。特に呼び出し側では、呼び出す手続きをコード内に明示しなければならないため、その部分がグルーコードとなってしまう。このようなプロセス間でタップを用いる方法を考える。

図7では、最初に(1)のDFDのような設計だったとする。これを、学生割引や誕生日などの特典が受けられるかどうかのチェックをしてから次の処理を行うように変更したとする。(2)が変更後のDFDである。ここで「顧客情報」と「特典付き顧客情報」が実際には同じデータ構造を持っているとすると、(1)の点線の丸で示したA-Bのデータ入出力のインタフェースは、(2)におけるA-CおよびD-Bと同じである。

変更前、「顧客情報を得る」が「DMを印刷する」を呼び出す関係にあり、呼び出しがタップAとタップBを利用して行われていたとすれば、図に示す変更は非常に簡単に行うことができる。新たなプロセス「特典条件を調べる」もタップC、Dを用いて実装し、タップの接続を変更するだけでよい。以前からある「顧客情報を得る」と「DMを印刷する」を変更する必要はない。



(1) 変更前のデータフローダイアグラム



(2) 変更後のデータフローダイアグラム

図7 DFDの変更とタップの再接続

6. 議論

6.1 本手法をどのように使うか

タップは 2.2 で説明したように、Mac OS X の Cocoa バインディングを意識した記述手法である。Cocoa バインディングで共有されるモデルは、オブジェクト指向の MVC (Model-View-Controller) パターンのモデルに相当する。タップを利用して MVC に近いプログラミングスタイルを実現することができる。

また、やはりオブジェクト指向プログラミングで広く利用されているデザインパターンである、オブザーバパターン（マルチキャスト）を手続き型のモジュールで実現する手段としても利用できる。

本手法のタップを利用すると、前節の例で示したように関数名を互いのソースコードに明示せずにプログラミングを進めることができる。特に、比較的簡単なインタフェースを持ち、互いに接続を変更できる部品のなモジュールが数多くある場合に有効性が高い。

さらに、グローバル変数を抽象化するために利用することができる。プログラムのあちこちからアクセスされていた変数をタップで書き表すことによってモジュール単位にまとめることができる。可視範囲が限定されることによってプログラムが理解しやすくなり、不必要な変数を整理、削減することでモジュールの独立性を高める効果が期待できる。

6.2 オブジェクト指向手法との比較

本研究は、オブジェクト指向のスタイルによらない、従来型の手続き型プログラミングにおけるモジュールの独立性向上に関する提案である。オブジェクト指向のアプローチが適用可能な問題領域であれば、オブジェクト指向を採用した方が広範なメリットが期待できるだろう。例えば、C 言語などの手続き型言語をベースとしながら、オブジェクト指向に基づくプログラミングが可能な開発プラットフォームもある。代表的な例として GTK+[4] や GObject[5] を挙げることができる。

一方で、オブジェクト指向のスタイルで開発する必要のない用途や、すでに手続き型で記述されたソフトウェア資産が存在する場合などに

は本手法が適用できる。前述したように、本手法には（前処理系が必要であるが）特別なライブラリは必要がなく、プログラムの規模が増大することはない。既存のソフトウェアに対し、適用できる部分にのみ可能な範囲で適用するという使い方もできる。

6.3 タップへの動的な関係づけについて

本稿で提案したタップの概念は、図5のようにリンクによって静的に定まるものに限定している。もちろん、共有変数と関数ポインタのリストを動的に管理可能なデータ構造を用意してタップを動的に関連付けることも原理的には可能である。

オブジェクト指向におけるオブザーバパターンではオブザーバは動的に追加、削除が可能である。また、C# 言語[6]におけるデリゲートには、起動されるメソッドを動的に追加できる機能がある。

タップの共有、あるいはタップへの関数の関連づけを動的に追加、削除する可能性については今後も検討が必要である。ただし、機構は複雑化するものの、それに見合ったメリットが得られるかは不明である。

6.4 タップの結合の連鎖

タップは、同じ値を持つものが結合しあってモデルを共有するという概念に基づいている。このため、あるタップの値が変化した時に手続きを起動して、同様な値を持つ別のタップの値も変化させるというプログラムを記述することは考えられる。例えば同じ株価を円とドルで表すような場合である。

このような共有関係の連鎖が閉路を形成してしまうと、タップが相互に呼び合い続けて停止しないおそれがある。ただし、再帰的な呼び出しを許容する言語では発生しうる現象であり、決定的な防止策はない。最終的にはソフトウェアのデザインが妥当かどうかにかんして依存する。

6.5 マルチスレッドへの対応

あるタップに対する通知が複数のモジュールで手続きを起動することがある。現在は関数のリストを順に呼び出す実装となっているため、すべての手続きは通知が発せられたスレッドで実行される。しかし、マルチスレッドプログラ

ミングでは状況に応じて別のスレッドで手続きを起動したいという場合も多い。

本手法は具体的なスレッドモデルを前提としていないため、今回の提案にはスレッドへの対応は含めていない。今後、pthreadなど、広く利用されているスレッドモデルへの対応を検討する必要がある。

複数のスレッドからタップの共有変数にアクセスする場合には、相互排除の仕組みも導入する必要がある。また、タップは双方向の利用が可能な機構であるため、マルチスレッドでの利用においてはAdaのランデヴとの類似も意識すべきであろう。

7. おわりに

本稿では、非オブジェクト指向言語によるプログラミングにおいて、モジュール間の連携を容易に実現する言語機構であるタップを提案するとともにその具体的な実装方法を示した。

提案手法はシンプルな構造と原理に基づいているためにさまざまな応用や拡張を考えることができ、今後検討すべき点は多岐にわたる。

今後はタップを用いたプログラミングを実用化するため、C言語に対する前処理系を開発して、有効性の実証を行う予定である。

謝辞

本研究の一部は京都産業大学総合研究支援制度の助成を受けたものである。

参考文献

- [1] 独立行政法人情報処理推進機構, 2008年版組込みソフトウェア産業実態調査報告書 (2008).
- [2] NeXT Computer, Next Development Tools, Addison-Wesley (1991).
- [3] Apple Inc., Introduction to Cocoa Bindings Programming Topics, <http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaBindings/CocoaBindings.html> (2007).
- [4] The GTK+ Team, The GTK+ Project, <http://www.gtk.org/> (2008).
- [5] The GNOME Project, GObject Reference Manual, <http://library.gnome.org/devel/gobject/unstable/index.html> (2008).

- [6] Microsoft Corporation, C# Programming Guide, [http://msdn.microsoft.com/en-us/library/67ef8sbd\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/67ef8sbd(VS.80).aspx) (2008).