Partskit: GUIプログラミングのための回路モデル化

5C－7

大鎌 広　畠 俊一　藤原 祥隆
北見工業大学

1　まえがき

GUIプログラミングに伴う困難さ [1] を軽減するため、本稿ではGUIプログラミングのフレームワークを提案し、そのシステムコンセプトについて議論する。提案するフレームワークは特定の機能を持った電子回路を汎用的な電子部品から構成することから類推した方式であり、Partskit と称する。

2　Partskit コンセプト

電子回路は、多くの汎用的な単機能の電子部品を通信線で接続することで構成される。

Partskit では電子回路の構成と同様に

- 入出力ポートを持った部品
- 各部品の入出力を接続する通信線
- 部品の通信線の接続の記述

から GUI アプリケーションプログラムを構成する。Partskit では、上記の 3 つの構成要素の全てが C++ 言語で記述されている。

本稿では "部品" という用語を Partskit 上の固有のソフトウェア要素を指すために使う。部品は次の性質を持つソフトウェア要素 (オブジェクト) である。

1. 有限個の入出力ポートを有している。入出力ポートは各々の部品中で一連の番号で識別される。入出力ポートを通じてのみ、他の部品と通信できる。
2. 各部品は一つのコントロールフローからなる逐次的動作 (Action) と固有のメモリを有する。
3. 各部品は静的なオブジェクトである。部品の lifetime は長く、アプリケーションプログラムの lifetime と同じである。
4. 各部品の Action は (疑似的に) 並行に動作する。
5. 通信データは少数の型の逐次データである。
6. 各部品はアプリケーションプログラム中でユニークな識別名を持つ。

Partskit: Circuit modeling for GUI programming
Hiroshi OHKAMA, Syunichi HATA, Yoshitaka FUJIWARA
Dept. of Computer Sciences Kitami Institute of Technology
Koen-cho 165 Kitami-shi 090

並行に動作する部品を通信線で接続することで一つのアプリケーションプログラムを作成する。部品間の接続はアプリケーションプログラムのプログラミング時に固定し、動作中に変更はしない。この制限を設けることで、このアプリケーションを部品の接続図として記述できる。Partskit では部品および接続トポロジが静的であることを重視している。

通信しあう逐次動作するプロセスの集合として、一つのアプリケーションプログラムを構成するという点で, UNIX で単機能のプログラムをパイプで結んで所望の機能を実現する手法や、Hoare の CSP [2] と近縁のプログラム技法である。

GUIプログラムはマウス、キー、等の装置やウインドウとウインドウポインタが発生するイベントに対するイベント駆動型プログラムである。イベントの発生場所や種類が非常に多いため、逐次的なプログラミング言語でのプログラムを難しくしている [1]。

Partskit では イベントを発生する装置やウインドウに一つの部品が対応していて、各々のイベントの監視をそのイベントを発生する (装置に対応している) 部品の Action が受け持つ。Action は C++による逐次的な動作記述であり、逐次的なプログラミング方式のみを習得したプログラマも理解しやすいと思われる。

Partskit では、汎用の部品を用意した上で、次の手順でアプリケーションを開発する。

1. 汎用の部品の中からアプリケーションに必要な部品のセットを選ぶ。
2. 汎用の部品だけで必要な機能を満たせなければ、既存の部品をクラス継承して、新たな部品を作成する。
3. それらの部品を接続して、コンパイルし、所望のアプリケーションプログラムを得る。この時、部品の接続を支援するプログラムを使用できる。この支援プログラムをアプリケーションビルダとよぶことにする。

3　逐次的言語でのGUIプログラムの問題

GUI プログラムを逐次的手続き言語で記述する場合、次の構造になる。しかしながら、このプログラム

スタイルでは以下の問題が起こる。

```
main()
{
  初期設定 //widget の生成, コールバックの登録
  while(1){  //イベントループ
      event=イベントの収得
      event の発生した場所と種類に応じた
                              手続きの実行
  }
}
```

1. 各イベントに対応する手続きがイベントループ中に混在する。そのためイベントと手続きの対応が不明確になったり、イベントループの記述が長くて読みにくくなる。

2. イベントを発生するオブジェクト間の通信の流れや全体像を把握しにくい。

3. event の型をそろえる必要がある。そのため、イベント収得ルーチンで未対応のイベント発生装置を使用する時に、プログラムを拡張するのが困難になる。

4. イベントが発生していない時の処理が特殊扱いになる。

5. event の発生した場所と種類に応じた手続きの実行は短時間に終了するようにコーディングしなければならない。そうしなければ一つの要求を受けると他の要求を受け付けない事になる。それゆえ本来一つのコントロールフローで記述するのが自然な処理であってもコントロールフローを分断する必要が生じる。このためにコーディングが困難になる場合がある [1]。

GUI プログラミングではツールキットと呼ばれるライブラリを使う事で上記の問題のうち 1 と 2 の一部だけが解決できる。ツールキットではイベントの場所と種類に応じて、呼び出すべき関数 (コールバック関数、イベントハンドラ) をイベントループ実行前にあらかじめ登録する。そして、イベントループからの関数の呼び出し機構はツールキットで用意した関数をプログラマに使用させる事で、1 番の問題を解決している。

2 番の問題は widget と呼ばれるオブジェクトを標準化し、widget 間の通信はコールバック関数の登録時に通信の必要な widget のポインタを同時に登録する事である程度対処している。しかしこの対処には次の問題がある。

- ポインタを使う事で、型のチェック機能を無効にしている。
- ある widget が二つ以上の widget と通信する時は、大局変数を使うか、通信のためだけに、その複数の widget のポインタの構造体を用意し、そのポインタをコールバックの登録時に渡すプログラムテクニックを使う必要がある。結局 widget 間の通信の把握は難しくなりがちである。

4 Partskit による解決

Partskit では上記の問題は次のように解決する。

各イベントに対応する手続きは、部品の Action で実現し、長い一つのイベントループは存在しない。

各々のイベントは対応する部品があり、部品間の通信の流れや全体像は、部品の外部で記述される。またアプリケーションビルダを使い、流れの全体像を回路図のように図示できる。各部品の動作記述中で他の部品を直接指定しないため、動作を含めてライブラリにすることが可能であり、再利用しやすい。

イベントは各々の部品の Action が監視するので、型をそろえる必要もなく、新たな装置を追加できる。

Action は lifetime が長く、コントロールフローの途中で、他の部品の Action に制御を移したり、他の部品との通信が可能なので、自然なコントロールフローを分断しない。

5 むすび

GUI フレームワークとして Partskit を提案している。Partskit は電子回路のように、各々が機能と入出力端子をもった部品を通信線で結合して希望するプログラムを作成する。Partskit では従来 GUI プログラムで利用されてきた toolkit でのコントロールフローの分断の問題や例外的なイベントの処理の困難が生じないことについて論じた。

参考文献

[1] 増井俊之: "GUI ベースのプログラミング," bit 別冊 ビジュアルインタフェース —ポスト GUI を目指して— ,pp.45-64, 共立出版,1996

[2] R. E. Filman, D. P. Friedman: "Coordinated computing — Tools and techniques for distributed software," McGraw-Hill,Inc., 1985