情 報 システム 63－5
（1997．3．19）

Nuts—柔軟な部品間結合をサポートするコンポーネントアーキテクチャ

上田哲郎
筑波大学経営政策科学研究科企業科学専攻
日産自動車(株)総合研究所 社会・商品研究所
久野靖
筑波大学経営政策科学研究科企業科学専攻

オブジェクト指向技術を利用したソフトウエア開発が注目され、数多くのクラスライブラリやフレームワークの試みが提案されている。C++言語によるオブジェクト指向開発環境を見た場合、コンポーネントの組立に傾倒しているものはプログラミング的融通性に欠き、一方で言語レベルでの汎用性を保存しているものは自由度が高い半面、コンポーネントレベルでの結合が強く構造と制御の柔軟性、可読性に欠けるという傾向がある。本研究で取り上げるNutsクラスライブラリはC++言語の範疇で言語としての自由度を妨げることなしにコンポーネント指向のプログラミングを可能にするアーキテクチャである。

Nuts—An Architecture Supporting Flexible Combination between Components

Tetsuro Ueda
Graduate School of Systems Management, The Univ. of Tsukuba
Nissan Motor Company Co.,Ltd.
Social and Advanced research laboratory
Yasushi Kuno
Graduate School of Systems Management, The Univ. of Tsukuba

Software development environments incorporating object-oriented techonologies are getting popular in these days, and a large number of class libraries and/or framework drafts have been proposed. In the case of o-o development environment, those that focus on component composition are not flexible enough, and those that aim for general programming parts enforce strict framework rules and are difficult to understand. In this parper we describe Nuts class library — a flexible, general, and easy-to-use application toolkit. Nuts incorporates stacked component architecuture, which provieds simple and comprehensive programming model.

はじめに

大規模で複雑なソフトウエア開発にはオブジェクト指向技術が欠かせなくなっている。

オブジェクト指向を用いたソフトウエアモジュールの再利用は、粒度の小さいクラスライブラリから、フレームワークと呼ばれるソフトウエアの骨組みの部分にまで及んでいる。アプリケーションフレームワークを用いたソフトウエア開発のアプローチでは、

1. アプリケーション領域の一般的なフレームワークからそのアプリケーション固有の差分のみを記述することでソフトウエアの骨組みを構築し、

2. その骨組みにクラスライブラリ部品、あるいはアプリケーション固有部品 (クラスライブラリから派生) を差し込む

ことでソフトウエア開発を効率化する。

これまでにいくつかのアプリケーションフレームワークが提案されてきたが、プログラミング言語本来の自由度を妨げることなく、同時に再利用性、可読性を高めたものは少ない。

例えば C++を用いたグラフィックスアプリケーションフレームワークとして有名な ET++ [3] は、C++の言語の枠内であるが可読性は低く、フレームワーク自体の理解にかなりの労力を要する。

本研究で取り上げる Nuts は、C++を用いた汎用的なアプリケーションフレームワーク構築のためのコンポーネント指向クラスライブラリである。そのアーキテクチャは、ソフトウエアの構造と制御の記述に一貫したモデルを採用することで、可読性を高め理解のためのハードルを低くしている。また構造の自在な変更を可能にし、構造と制御を分離することで制御の柔軟性を保証するものである。別の言い方をすれば Nuts は、C++プログラマーのためのホワイトボックス再利用技術でありながら、コンポーネント指向の再利用を可能にするものである。

以下では、まず最初に構造と制御のモデル化について説明する。つぎに既存のアプリケーションフレームワークと比較しながら Nuts のソフトウエア構築手法を例をあげて説明する。また Nuts を用いた具合的なアプリケーションとして Nuts 部品を組み上げてソフトウエアを構築するツール Nuts/Builder を説明する。最後に今後の改良点を取り上げる。

1 Nuts の構造と制御のモデル

1.1 Nuts の構造モデル

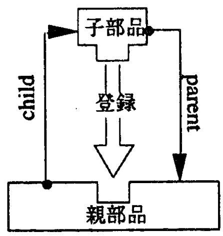

図 1: 部品の親子関係

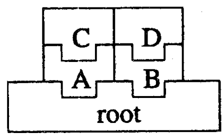

図 2: Nuts の構造モデル

Nuts ではソフトウエアの構造を以下のようにモデル化した。

プログラム＝部品の積み上げ、ととらえる。部品の積み上げは階層構造を伴う。一つの部品上に、機能の異なった (あるいは全く同じ) 子部品を一つ以上登録して使うことができる。逆に子部品は唯一つの親部品を持つ。親子部品間には図1のような相互参照関係がある。これにより部品の積み上げ構造は図2のように木構造になる。

唯一親を持たない部品がルート部品である。ルート部品は、アプリケーションクラスに属する特殊な部品であり以下のような特徴を持つ、

- アプリケーションを構築する際に必ず一つ存在しなければならない。
- すべての子孫部品から参照される。
- アプリケーション全体への特殊なサービス
(イベント処理, ガベージコレクション等)
を行う。

従ってNutsのオブジェクトモデルの構築とは、部品木を記述することである。従来、新規オブジェクトの確保はプログラムの奥深くの所在の知れない関数の中に埋もれてきた。しかし、オブジェクトの関係を親子関係を基本にした部品木にモデル化すると、その関係の記述はプログラムの外に抜き出すことができる。オブジェクトは固有の名前を持つ。この名前により、図2は次のように記述することができる。

```
root has A and B.
A has C.
B had D.
```

本方式では構造の記述がプログラムの外に分離されるため構造の理解がしやすく、また変更も容易である。

1.2 Nutsの制御モデル

Nutsでは、構造モデルに従って組み立てられた部品同士が直接隣接していなくても、相互にメッセージ通信を行うことができる。そのため、組み立てた構造により不当に強い成約を受けることなく、柔軟な部品間結合を有したプログラムを記述することができる。これを制御モデルと呼んでいる。具体的に制御モデルには次の3つのメソッドが用意されている。

1. sendMessageメソッド:名前を指定し、部品木に沿った葉方向への深さ優先探索によりその名前にマッチする部品を探索する。送信が成功した場合、そのメッセージを最後に受理した部品への参照が返される。マッチする部品がなく、全ての部品を走査したメッセージは破棄される。
2. passMessageメソッド:sendMessageと同じく名前による部品木の探索。探索方向は部品木に沿った根方向への探索となる。
3. sendEventメソッド:あらかじめイベント要求を登録した部品に対して、対応したイベント発生時にメッセージが送信される。この処理はルート部品によって行なわれる。

sendMessage,passMessageの各メソッドの名前による探索には次のパターンがある。

- "名前":指定した名前と同じ名前を持つ部品に一致
- "/名前":指定した名前のクラスに所属する部品に一致
- "//名前":指定した名前のクラスまたはそのサブクラス以下に所属する部品に一致
- 上記のものの論理式による組み合わせ

これらの認識処理は図3に示す一連のメソッドによって扱われる。最終的にすべてのメッセージはrecognizeメソッドでハンドリングされるので、recognizeメソッドの内容を見れば部品の制御の流れを把握できる。

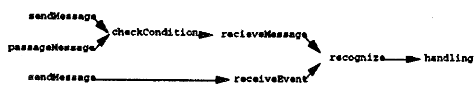

図3: メッセージ授受の流れ

上記の基本的な制御に加え、redirect,passというメソッドによりメッセージの横取りができる。redirectは横取り先を名前によって指定し、上記と同じ方法で再探索を行う(探索が失敗すると横取りは起こらない)。passでは横取り先を参照によって指定する(参照が無効だとエラーとなる)。 またメッセージパッシ

ング以外に、部品の親子関係により自動的に挿入される制御がある。これをコンテキストに依存した制御と呼ぶ。例えばボタン部品の上に置かれたコマンド部品は、ボタンが押されると自動的にその実行メソッドが呼ばれる。別のクラスの部品であればその部品にあったメソッドが呼ばれる。このように親子間の関係により動的に適切なメソッドが選択されて実行される。

これらの枠組みを使って、プログラムの開発からオブジェクトモデルと制御モデルを抜き出しテンプレート化することにより、コーディングは極めて機械的な作業として行える。オブジェクトモデルと制御モデルは独立しており、プログラム構造の作成,変更が容易である一方、構造の変化によってプログラム制御が影響を受けることはない。また、どのような複雑なオブジェクトモデルを作り上げたとしても、制御モデルが遠隔なオブジェクト間の通信を確保してくれるため柔軟な制御を可能にしている。

2 Nuts の部品とクラス階層

Nuts の各クラスは次のような規約に従う。

- 部品は固有の名前を持ち、部品名を返すメソッドを持つ
- クラス名を返すメソッドを持つ
- メッセージがそのクラスで認識されない場合にクラス階層を逆のぼるメソッドを持つ
- そのクラスがあるクラスのサブクラスかどうか調べるメソッドを持つ
- newNuts クラス名 (親部品名, 部品名) というスタティックメソッドによってインスタンスを生成できる

多くの制御モデルが文字列による動的なクラス識別を行うため、特にクラス名を返すメソッドは重要である。Nuts のクラス規約が保たれるように、規約に沿ったクラスヘッダファイルを自動生成するツールが用意されている。このツールは新規クラス名とスーパークラス名をキーにして Nuts のひな型クラスから新規クラスのヘッダファイルを生成する。プログラマは新規クラスの作成を必ずこのツールに委ねなければならない。Nuts の基底クラス階層は図 4のようになっている。

図 4: Nuts の基底クラス

Core クラスはすべてのクラスの基底クラスである。そのコンストラクタは親部品への参照を要求し、Nuts の構造モデルに従って部品間の親子関係を構築する。またメッセージ受信のための仮想関数をもつ。Manager クラスは子部品を持つことができる部品の基底クラスであり、メッセージを子部品に配送する仮想関数を持つ。Applicaion クラスはルート部品としての基本的な性能を持ったクラスでイベント処理などの機能をもつ。

3 Nuts によるアプリケーションの構築

3.1 初期構造の記述と初期化の流れ

ソフトウエアの初期構造は図 5のように一つのファイル (プロジェクトファイルと呼ぶ) に記述される。

まず最初にルート部品が確保される。そのうえに順次機能部品を積み上げていく。記述は"newNuts クラス名 (親部品名, 部品名)" という形に統一されているので構造は速やかに理解できる。

さらに初期構造の記述はプロジェクトファイルに集中しているので、従来のプログラミングのようにプログラムの各所に点在している宣言をプログラマーが追って理解する必要はない。

また構造の記述が機械的なものになっているため後で述べるアプリケーションビルダーによる構築に容易に発展することができる。

```
//app.C
newNutsApp(app);
newNutsShell(app,shell);
newNutsStdout(app,stdout);
```

図 5: プロジェクトファイルの例

3.2 ET++との比較

ET++では hello world アプリケーションは図 6 のように記述される [16]。

```
//include files...
class HelloDialog : public Dialog
{
MetaDef(HelloDialog);
HelloDialog() : Dialog(){ }
VObject *DoMakeContent();
};

VObject *HelloDialog::DoMakeConent()
{
return new Matte(newNuts
TextItem("Hello_World"));
}

class Hello : public Application
{
public:
MetaDef(Hello);
Hello(int argc,char *argv[])
: Appclication(argc,argv){ }
Manager *DoMakeManager(Symbol){
return newNuts HelloDialog();}
};

main(int argc,char *argv[])
{
Hello app(argc,argv);
reurn app.Run();
}
```

図 6: ET++を用いた hello world

一方 Nuts を用いた場合は、図 7のように記述される。

```
//include files
newNutsMotifApp(hello);
newNutsMotifToplevel(hello,top);
newNutsApplicationShell(top,shell);
newNutsDialog(shell,dialog);
newNutsLabel(dialog,Hello_World);
```

図 7: Nuts を用いた hello world

この例からもわかるように、Nuts の構造記述は非常にシンプルで部品が木構造に組み上がることさえ知っていれば構造の理解は容易である。

一方 ET++ではオブジェクトの確保が特定の hook メソッド (virtual constructor) のオーバーロードで実現されていることを知っていたとしてもその理解は容易でない。

また、Nuts では部品の参照関係が部品間の親子関係のみに画一化されているため、クラス毎の参照関係を各々知っている必要がない。ET++では参照関係を確立する際、オブジェクトへのポインターを引き数で渡すためコンストラクターに様々な形があるの対し、Nuts では親オブジェクトへのポインターとオブジェクトの名前を引き数で渡す形に統一されている。

さらに、ET++がクラス継承を基本にした仮想関数のオーバーロードによってフレームワークの拡張を行うのに対し、Nuts は部品のコンポジションを積極的に用いるアプローチである。

これはアプリケーションフレームワークのひな形を作ろうとしたとき大きな違いを生む。メニューバーをもつアプリケーションのひな形を例にとると、ET++ではデフォルトのメニュー構成に対して hook メソッドをオーバーロードした個別のクラスを派生させなければならない。

一方 Nuts はプロジェクトファイルでいろいろなメニュー部品を組み合わせたものを用意することでメニューバーを必要とする様々なアプリケーションのひな形を容易に作ること

	行数	固有部品種数
app1	4898	37
app2	7645	68
app3	465	5
app4	846	9
app5	1958	20
Nuts/Builder	6749	57

表 1: Nuts を用いた開発例

ができる。また ET++では main() で明示的にアプリケーションの初期化メソッドを呼ばなければならないが、Nuts では隠れた main() がルート部品を自動的に探し出して決まった順序で初期化ルーチンをコールする。ユーザーは固有の部品を作った場合、各初期化フェーズの初期化ルーチンのみを適切にオーバーロードすればよい。

4 Nuts によるアプリケーション開発

Nuts ライブラリは 199 種類の汎用部品からなり、そのコード量は C++で 29000 行である。グラフィックス部品は X Window,OSF/Motif Widget をカプセル化したものである。その他に部品群のグループを管理する部品や、プロセス間通信を担う部品、ニューラルネットのノードを構築するための部品などがある。

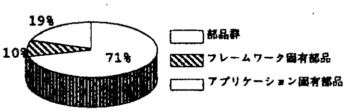

図 8: Nuts ライブラリを用いたコード比率

アプリケーションの適応例はビジネス系を中心に 5 例ほどあり開発データの内訳は表 1 のようになっている。また平均的コード比率は図 8のようになっている。おおまかに言って開発効率は部品の再利用により 3 倍になったと言える。

5 Nuts アプリケーションビルダーへの発展

前述のようにプログラムの初期状態はプロジェクトファイルに機械的に記述されるのみであるので、ビジュアルなツールを容易に開発することができた。Nuts/Builder(図 9) は部品バンクからワークシート上に必要な部品をドラックして積み上げていくことにより、簡単にプロジェクトファイルを生成できるツールである。Nuts/Builder 自体が Nuts 部品 381 個を積み上げることによって構築されている。開発データは表 1に示すとおりであり、開発工数には 1 人月を要した。

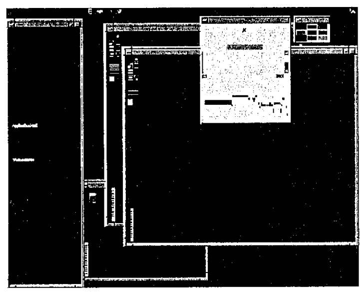

図 9: Nuts/Builder の画面イメージ

6 おわりに

最近、デザインパターンを利用したソフトウエアアーキテクチャの分解がもてはやされている [8]。Nuts をデザインパターンに照らして検討した結果、のいくつかを含んでいることがわかっている。

例えば NutsXOperator (図 10) は、NutsXObject (X-Window の drawabel) を画面上で操作するための部品である。NutsOperator の派生クラスには NutsXScroller がある。NustXScroller 上にのせられた NutsXObject の部品は、画面をマウスでドラッグすることでスクロールされる。これは、デザインパターンの Decorator パターンに相当する。

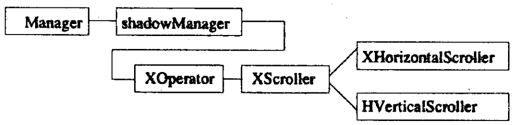

図 10: NutsOperator クラス

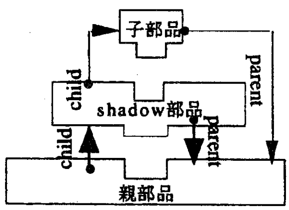

図 11: NutsShadowManager の親子関係

Decorator パターンに当てはまる部品は、本来親子関係の成り立つ部品間に割り込む形になるため、子部品からみた親部品の条件 (例えば XWindow 部品の親は XWindow 部品でなければならない) を妨げることになる。そこで Decorator 部品は、NutsShadowManager クラスの部品として派生される。

NutsShadowManager クラスは、NutsManager クラスの派生クラスで、その親部品からは見えるが子部品からは存在しないようにみえる部品を提供する (図 11)。

これにより NutsShadowManager クラスから派生したクラスの部品は、部品の親子関係を気にすることなしにどこにでも挿入するすることが可能になる。

デコレーションは複数組み合わせることができる。Nuts の場合はそれらの部品を相互に積み上げるだけで目的のデコレーションが得られる。例えば、NutsXScroller の派生クラスには NutsXHorizontalScroller と NutsXVerticalScroller が存在する。それぞれスクロールの方向を水平、垂直に限定したものである。これらの部品を図 12のように積み上げると NutsXScroller と同じ機能を提供することができる。

一方で、ET++がマルチプラットフォームに対応するために Abustoruct Factory パターンなどを採用しているのに対して、Nuts ではその辺りの配慮に欠けていることが判明した。今後改良していきたい。

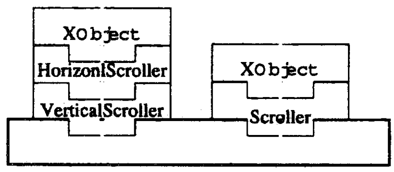

図 12: デコレーションの組合せ

これまでのアプリケーション構築経験から言えば、Nuts は C++によるソフトウエア開発の期間と工数を削減するうえで非常に有効であった。また、C++を C と同じようにしか利用できていない初心者にも十分活用が可能であったと同時に、C++のパワーユーザーにとっても満足できるものであった。今後は各部品の洗練化、機能部品の充実を図り、適応事例を増やしていく予定である。

参考文献

[1] A.J.PalayW.J.Hansen. *The Andrew Toolkit-an overview.* 1988.

[2] Thomas Eggenschwiler Andreas Birrer. *Financial software desing patterns.* 1996.

[3] E.Gamma etc. A.Weinand. *ET++ - an object-oriented pplicaiton framework in C++.* Nov 1988.

[4] David A.Wilson. *Programing with MacApp.* Addison-Wesley Publishing Company,Inc, 1990.

[5] Thomas Eggenschwiler and Erich Gamma. *ET++SwapsManager: Using Object Techology in the Financial Engineering Domain.* 1992.

[6] Ralph E.johnson. *Documenting Frameworks using Patterns.* 1992.

[7] Peter W.Mandany etc. *Exeriences building An Object-Oriented System in C++.* March 1991.

[8] Erich Gamma. オブジェクト指向における再利用のためのデザインパターン. SOFTBANK.

[9] Wizard Grady Booch and Michael Vilot. *The Design of the C++ Booch Components.* Oct 1990.

[10] John M.Vlissides and Mark A.Linton. *Unidraw:a framework for building domain-specific graphical editors.* Nov 1989.

[11] Dougleas E.Leyens Peter W.Madany. *A C++ Class Hierarychy for Building UNIX-Like File Systems.* Oct 1988.

[12] Vincent F.Russon Ralph E.Johnson. *Resusing Object-Oriented Designs.* May 1991.

[13] Nayeem Islam Roy H.Campbell. *Choise,Frameworks and Refinement.* Oct 1991.

[14] Nayeem Islam Roy H.Campbell. *Designing and Implementing choices : an Object-Oriented System in C++,* Vol. 36. Sep 1993.

[15] W. プリー佐藤啓太. デザインパターンプログラミング. トッパン, 1996.

[16] 佐藤啓太. クラスライブラリ自由自在. トッパン.

[17] 青木幹雄. コンポーネントウエア：部品組み立て型ソフトウエア開発技術, 第 37 巻. Jan 1995.