オブジェクト指向設計による分子動力学計算プログラム

石　井　　晃

(平成6年6月21日受理)

1．序　言

最近，ソフトウェア開発の現場に於いてオブジェクト指向という考えに基づいたプログラム開発が行われるようになってきている。これは従来の構造化プログラミングと違ってソフトウェアでシュミレートする対象をまず「オブジェクト」という構成要素に分解し，このオブジェクトについてデータとそのデータを処理する関数を付加して一つのまとまったプログラム単位にする方法である。しかし，昔から計算機を多用して来た科学技術分野ではこのオブジェクト指向に基づいたプログラム開発の理解及び実行は十分とはいえない。それはオブジェクト指向を標榜したプログラム開発環境のほとんどが，主に大量の文書や顧客リストなどの処理を念頭に置いて整備されてきていることからもわかる。

しかしながら，元々オブジェクト指向プログラミングが構造化プログラミングを批判する形でできたのは，大規模プログラムの開発を容易にするという理由であり，その点では特に対象分野を限るものではない。批判の対象にされたのは FORTRAN に附いて言えば COMMON 文や subroutine の引数という形でのデータのやり取りは甚だデバックがしにくいという点である。たとえば対象とする問題自体をいくつかの部分に分割し，それをそれぞれサブルーチンという形で分割管理するのは可能であるが，メインプログラムの中で，それらのサブルーチンの引数はむき出しになってしまい，従ってどの部分でトラブルが起こってもメインプログラムを経由してその影響は全プログラムに及ぶ。それを防ぐために COMMON 文の中で変数を受け渡しすれば，一応その変数を操作できるプログラムは局所的に限定されるが，この場合はどのプログラムがどの COMMON 文を使用しているのかを掴むのが甚だ難しくなる。つまり変数の受け渡しは地下に潜ってしまい，エディタの検索機能で探し回ることになる。

さらに，構造化プログラミングは現場での大規模なソフトウェア開発に於いて致命的な欠陥がある。それは通常開発現場では一つのソフトウェアが複数の人間によって開発されるということである。従って，一人の人間が書いたプログラムに別の人間がコードをつけ加えて機能を拡張することが日常茶飯事であることを念頭に置く必要がある。これは特にソフトウェア会社だけの問題ではな

く，大学の研究室においても毎年新しい学生が卒業論文制作，修士論文制作，あるいは博士論文の準備として入れ替わり立ち替わり開発プロジェクトに携わるのであるから，大学研究室におけるソフトウェアの開発の状況はソフトウェア開発会社よりさらに意思の疎通という意味では厳しいといえる。そうした常に入れ替わる複数の人間が一つのプログラムをいじっていくとすると，たとえばCOMMON 文を用いてどのプログラムからどこへデータが流れていったのか，掴むのは非常に難しくなる。

例を挙げよう。たとえば次のようなプログラムがあったとする。

```
program main
common/A/x, y, z
subroutine alpha (t)
subroutine beta (t)
        :
        :
end
subroutine alpha (t)
common/A/x, y, z
        :
return
end
subroutine beta(t)
common/A/x, y, z
        :
retuen
end
```

ここにあげたプログラムでは，A という common block が alpha, beta という二つのサブルーチンで使われている。ところが，ここで別の学生が，次のようなプログラムを作ったとしよう。

```
subroutine crash (t)
common/A/x, y, z
x=x+1
t=t
return
end
```

このプログラムは基礎データである common block A の中身を勝手に書き換えるものであるから，その学生がこのプログラムを全体のプログラムのどこに挟むのかで，先の alpha, beta というサブルーチンの実行結果は全く保証されなくなってしまう。この学生に悪気はなかったとはいえ，このプログラムを全体のプログラムの中の随所にいれてしまったとしたら，第三者がこのことに気づくのはかなり難しい。

つまり，従来の構造化プログラミングの欠陥とは，このような不完全なサブルーチンを挟み込むことによるデータの破壊がいとも簡単に行われることにある。特に大量のエラーメッセージを抱えて，ともかくも動かそうと妙に悪あがきしたときに，ついに犯しがちなことでもある。オブジェクト指向設計はこうしたことを未然に防ごうというプログラム設計の思想である。従って科学技術系の研究室では研究室で大切に開発しているプログラムを保護するためにも，是非必要な設計法といえる。

そこで，本論の目的は，そうしたオブジェクト指向によるプログラムの設計・開発をいかに科学技術，特に物理科学の分野に導入するのかを分子動力学のプログラムを例にとって考察することにある。

2．設計方針

2.1　オブジェクト指向

オブジェクト指向設計と構造化プログラミングとは必ずしも相反する設計法でないことをまず明言しておこう。構造化プログラミングが一人のプログラマーで十分目が届く範囲，たとえば千行から二千行のプログラムを書く場合には良く機能するのに対し，オブジェクト指向設計はとうてい一人では見渡せない範囲，たとえば数万から数十万行のプログラムのコーディングで威力を発揮する。

オブジェクト指向はプログラム設計の思想であるので，どのプログラム言語でも，たとえば FORTRAN77 でも可能である。しかし，先ほどのような意図しないデータ書き換えのようなものを防ぐ意味ではそれなりのデータ保護機能の備わったプログラム言語であることが望ましい。オブジェクト指向言語を標榜するものはふつうオブジェクトに相当するものとしてクラスというものを定義し，このクラスが，いわば変数とその変数の取り扱いに関する関数の集合体となる。それぞれのクラス内の局所変数を定義することによってクラス内ではいちいち関数の引数や COMMON 文などを使わなくても変数を引用することができる。また，自分のクラス以外のプログラムから参照できる変数とそうでないものとに分けることができるため，先に述べた意図せぬデータの破壊をかなり防ぐことができる。なぜなら，クラスの外からは参照できないデータを参照するためにはその対象となるクラス自身のコーディングを変更する必要があり，それを実行するのは明らかに「意図

した破壊」しかあり得ないからである。このようなデータ参照の保護を情報隠蔽と呼び，多くのオブジェクト指向言語に備わった機能である。つまり，情報隠蔽は研究室の学生に，自分がいじっていい部分といけない部分を明確に区分させ，もしその学生のプロジェクトが失敗してもいつでも元のプログラムは自動的に（バックアップということでなくても）守られることになる。

さらに，オブジェクトとしてなるべく対象とする現実に合わせてオブジェクト化していくと，プログラムが読みやすくなるというメリットもある。たとえば本論文で対象とする分子動力学なら，一つの原子を一つのクラスとして表現するというのはかなりわかりやすいクラス化であろう。そのクラスのデータメンバとして波動関数があり，また，そのクラスの関数として原子に働く力を計算する関数がある，という具合である。

オブジェクト指向言語の重要な特徴として，派生クラスという概念がある。これはあるクラスに一部機能をつけ加えたり，また，変数を追加したりして，そのスーパーセットを派生クラスとして新たに定義することである。たとえば，学生に研究室で開発しているプログラムのクラス自身はいじらせず，もっぱらその派生クラスについてコーディングさせていれば，元のプログラムを全くいじらずにいろいろプログラム開発に伴うテストコーディングができる。もし，これがFORTRAN77 であれば，テストとして動作させるためには今までに作られているサブルーチンについても引数を追加したり，COMMON BLOCK を追加したりしなければならず，バックアップファイル以外で加えた変更点を元に戻すのはかなり大変な作業となる。しかも，この機能の追加の作業が何週間も続くとなると，同じ名前のサブルーチンで引数や COMMON 文の異なるバージョンが研究室内に同時に存在することとになり，そのバージョン管理はきわめてやっかいなことになる。派生クラスを用いるというやり方を取ると，元になっているクラスはそのままで派生クラスにのみ変更を加えるので，その派生クラスにその学生の名前にちなんだ名前でも仮につけさせておけば，バージョン管理の心配などは全く無用で，かなり思い切った機能変更の試みなども安心してできることになる。

2.2　使用言語

オブジェクト指向のプログラミング言語としては Smalltalk, C++, Effel などいろいろなものが存在している。最近処理系が出回り始めた Fortran90 なども FORTRAN77 と比べてかなりオブジェクト指向を意識した仕様変更をしている。分子動力学のように完成までに長期の開発作業が必要になるプロジェクトの場合は開発途中で変更できないプログラミング言語の選択はかなり慎重に行わなければならない。従来からある財産を生かすという立場では FORTRAN 系統の処理系の選択も十分に考えられる。しかし，ここでの分子動力学プログラム開発は 1 からの開発であるため，逆に過去の遺産を気にせずに最良の選択をすることができる。そこで，こうした処理系の中で，ここで

は，C++ を選択することにする。

C++ は C 言語の上位互換にあたり，C 言語に関する過去の遺産を継承できる利点がある。さらに，FORTRAN 同様に複素数を実数同様にサポートするなど，C 言語に比べると科学技術計算に関する機能は大幅に強化されている。また，M. A. Ellis and B. Straustrapの“The Annotated C++ Reference Manual”[1]と ANSI による基準があるため，その範囲内でプログラミングをしていれば機種依存性は考えなくていい。

現在のコンピュータ使用では何らかのウインドウシステム（X window, MS Windows, OS/2, Machintosh OS, etc）の使用を念頭に置いた方がいいが，このユーザーインターフェイスとしてのウインドウ自体がC++でプログラミングされているため，分子動力学計算の部分もC++ でコーディングしておけば，あらゆるウインドウシステムに移植が容易になる。現在のところ，C++ はパソコンからハイエンドのワークステーションまで動作しているが，ダウンサイジングとともに事実上あらゆる科学技術計算を行う上でC++ は利用可能であると考えられる。

本論文に於いて以下に掲載するプログラムについては，ANSI 2．1 以降の基準のC++ であれば，動作する。しかし，将来的な発展を考えれば，テンプレートや例外処理の利用が望ましく，ANSI3.0の基準に達したコンパイラの利用が望ましい。具体的に開発に用いた開発環境は Borland C++ 4.0 for DOS, Windows and WindowsNT（英語版）である。特に，ユーザーインターフェイスの部分は Borland C++ 4.0に付けられているクラスライブラリー Object Windows Library 2.0（OWL2.0）を用いて書かれている。しかし，分子動力学計算本体のプログラムコードの動作は Microsoft Visual C++ Development System for Windows 1.5（英語版）及び Symantec C++ Professional 6.1 for Windows（英語版）の上でも確認している。また，ユーザーインターフェイスの部分を Visual C++ や Symanted C++ に添付されている Midrosoft Foundation Class を用いてコーディングしてもほぼ同様に実現することができる。

以上，プログラムの開発に直接用いた機種は DELL OptiPlex 466/LV であるが，ANSI3.0 の基準を満たしている IBM AIX C++/6000 1.0 及び IBM C Set++/6000 2.1 上でも計算本体の部分は動作するので，これに AIX 上のユーザーインターフェイス開発環境を組み合わせれば，IBM のワークステーション上への移植が可能である。さらにほかの機種へも ANSI 3.0 の基準を満たす C++ コンパイラ（例えば GNU GCC と G++）が用意されていれば移植は可能である。

3．分子動力学の理論

3．1　Pseudopotential 法

近年注目されている，第一原理からの分子動力学計算にCar-Parrinello 法[2]と呼ばれる方法がある。この方法は計算規模は大きいがかなり精密に計算することができる。この Car-Parrinello 法において電子状態は pseudopotential 法で計算する。一般に結晶の電子状態を計算するには結晶内の電子の波動関数を何等かの基底関数で展開して計算する。数学的に扱い易い平面波で展開するのが一番いいのだけれども，電子の感じる原子ポテンシャルは中心部で急激に大きく（負に大きく）なっているので，そこを正確にしようとすると非常にたくさんの平面波を加える必要が出てきて実用的ではなくなる。pseudopotential 法は原子の中心部分のポテンシャルをもっと滑らかで，かつ波動関数はなるべく本物に近いものを再現するよう偽者のポテンシャルを作り電子状態を計算する方法である。古くは固有値さえ再現されていればいいとして箱型などかなり適当な擬ポテンシャルを用いていたが，最近では第一原理的なきちんとした擬ポテンシャルを作って計算するようになってきた。

Hartree-Fock 近似による波動関数は次の方程式で計算される，

$$\hat{H}_\nu^{HF}\phi_\nu=\left[-\frac{1}{2}\nabla^2-\frac{Z}{r}+\hat{V}^{HF}\right]\phi_\nu=\varepsilon_\nu\phi_\nu$$

ここで，

$$\hat{V}^{HF}(\vec{x})=\sum_c\left[2\hat{J}_c(\vec{x})-\hat{K}_c(\vec{x})\right]$$

はコア電子とのクローン相互作用および交換・相関相互作用である。

$$\hat{J}_c(1)=\int\phi_c(2)\frac{1}{r_{12}}\phi_c(2)\,d\vec{x}_2$$

$$\hat{K}_c(1)\Psi(1)=\phi_c(1)\int\phi_c(2)\frac{1}{r_{12}}\Psi(2)\,d\vec{x}_2$$

Phillips-Kleinman [3]はこの Hartree-Fock 方程式を次のように書き改める。

$$\left[-\frac{1}{2}\nabla^2-\frac{Z}{r}+\hat{V}^{HF}+\hat{V}^{PK}\right]\chi=\varepsilon_\nu\chi$$

ここで，

$$\hat{V}^{PK}=\sum_{c}(\varepsilon_{v}-\varepsilon_{c})|\phi_{c}\rangle\langle\phi_{c}|$$

は擬ポテンシャルであり，χは価電子的で，且つコア電子の軌道と直交する事を要求されないある関数である。従ってχは次のうように表す事が出来るであろう。

$$\chi=\phi_{v}+\sum_{c}a_{c}\phi_{c}$$

ここでa_cはパラメータであり，これを適当に選んで関数Xがノードを持たないようにする。しかしながら，この方法では擬波動関数χが一意にはならない。

より良い擬ポテンシャルを選ぶためには，擬ポテンシャル自身がそれを求めるときの価電子の波動関数に依存してしまってはならない。なぜなら，もし依存していると，対象とするエネルギーを変えるたびにいちいち擬ポテンシャルを計算しなおさなければならなくなる。そこで改めてHartree-Fock 方程式を見てみると，交換・相関相互作用の項が価電子の波動関数に依存してしまっている。特にコア電子の軌道のある領域での振る舞いに依存してしまっている訳である。

Phillips-Kleinman の計算したLiの波動関数の図を見ると，原点付近での挙動，即ちコア電子軌道の領域での振る舞いはS, P, D, の別があり，かつ，２Ｓと３Ｓ，２Ｐと３Ｐの振るまいがよく似ている事に気付く。つまり，角運動量別に交換相互作用の部分の振る舞いを考えれば，いろいろなエネルギー領域で同じ擬ポテンシャルが成り立つであろうというという予想が出来る。

そこで，有効なポテンシャルとして次の形に書く事にしよう。

$$V_{eff}=\sum_{l=0}^{\infty}V_{l}(r)|l\rangle\langle l|$$

ここで，$|l\rangle$は実際には$|lm\rangle$で，必要なだけ球面調和関数の和を取る。この$|lm\rangle\langle lm|$は射影演算子になっており，こちらで選ぶ波動関数の基底（普通は平面波に採るが）から，それぞれの角運動量成分を抜き出す形になっている。ある程度以上大きな角運動量成分については同じポテンシャルを用いて近似する。

$$V_{l}=V_{\lambda+1}\qquad for\ l>\lambda$$

従って，$\Delta V_1=V_1-V_{\lambda+1}$とかくと，有効ポテンシャルは次のように書ける事になる。

$$V_{eff}=V_{\lambda+1}(r)+\sum_{l=0}^{\lambda}\Delta V_l(r)|l\rangle\langle l|$$

実際の計算ではｓとｐ，場合によってはｄまで擬ポテンシャルを作り，それより大きな角運動量成分はｄあるいはｆについてのポテンシャルで近似する事になる。

3．2　Norm-Conserving Pseudopotentiol

Harmann, Schlüter Chiang[4,5]によって提示された Norm-Conserving Pseudopotential では，擬ポテンシャルを決定する条件として，次の関係式に着目した。

$$2\left[(r\varphi)^2\frac{d}{d\varepsilon}\frac{d}{dr}\ln\varphi\right]_R=4\pi\int_0^R\varphi^2r^2dr$$

これは適当な球面 $r=R$ より内側の波動関数のノルムさえ元の真の波動関数と一致していれば，対数導関数の $r=R$ での一致は自動的に保証されるという関係式である。そこで，このＲより外側では真のポテンシャルに一致して内側ではずっと滑らかになっている擬ポテンシャルを作って擬波動関数も外側では本物と完全に一致するように作り，その時にノルムが真の波動関数と擬波動関数でＲより外側では一致するようにする。そうすればこの関係式から，バンド構造の計算に必要な $r=R$ における波動関数の対数導関数を一致させる事が出来る。Hamann, Schlüter and Chiang によるMo の場合の真の波動関数と擬波動関数の図に見るように内側で節が無く滑らかになっており，しかもノルムは一致している。

具体的には真のポテンシャルをまず用意し，それを $r=R$ より内側で滑らかになるように変えた擬ポテンシャルに直す。その時の関数として

$$V_{1l}^{Ps}(r)=\left[1-f\left(\frac{r}{r_{cl}}\right)\right]V(r)+c_lf\left(\frac{r}{r_{cl}}\right)$$

とする。ここで $V(r)$ は真のポテンシャルであり，

$$f(x)=\exp(-x^{3.5})$$

がカットオフ関数になる。

ここからノルムを合わせる手続きに入る。まず上の擬ポテンシャル $V_{1l}^{Ps}(r)$ から計算された擬波動関数 $W_{l1}(r)$ に対し，

$$w_{2l}(r)=\gamma_1\left[w_{1l}(r)+\delta_{lg_l}\left(\frac{r}{r_{cl}}\right)\right]$$

なる第二の擬波動関数を計算する。そしてこれが

$$\gamma_l\int_0^\infty w_{2l}^2(r)\,ar=1$$

となるように規格化する。そうしておいて $W_{2l}(r)$ について動径方向の Schrödinger 方程式を逆に解く事よって擬ポテンシャル $V_{2l}^{Ps}(r)$ を求める訳である。

3.3 Separable pseudopotential

1982年，Kleinman and Bylander [6] は Hamann, Schlüter and Chiang の擬ポテンシャルを改良して記憶容量が著しく少なくて済むようにした。Hamann, Schlüter and Chiang のままの形であると，擬ポテンシャルを平面波で挟んだとき

$$\langle e^{i\vec{k}\cdot\vec{r}}|\,lm\,\rangle V(r)\langle\,lm\,|\,e^{i\vec{k}'\cdot\vec{r}}\rangle\propto\int j_l(kr)\,V(r)\,j_l(k'r)\,r^2drP_1(\cos\theta_{kk'})$$

という形になるが，これでは波数について二重に和を取らなければならなくなる。平面波の数をｎ個，Brillouin zone の中の点をｍ点取るとするとこれは $\frac{mn(n+1)}{2}$ となるが，通常ｎは1000，ｍは20ぐらいであるので，これは著しいオーバーヘッドを与える。Kleinman-Bylander はこの積分を二つに分けてこの問題を解決することを提案した。つまり，

$$\int j_l(kr)\Psi_c(r)\,r^2dr\ \int j_l(k'r)\Psi_c(r)\,r^2drP_1(\cos\theta_{kk'})$$

と分離するわけである。

そのため，擬ポテンシャルを

$$V(r)=V^{local}(r)+\sum_{l,m}\Delta V_1(r)|l,m\rangle\langle\,l,m|$$

と書いたときに，ここであるエネルギー E_l に付いて擬波動関数を $\varphi(r)$ とすると，

$$\Delta V_l^{NL}(r)=E_l|\,\xi_l\rangle\langle\,\xi_l|$$

とする。このとき，

$$\xi_l(r)=\frac{\Delta V_l(r)\,\varphi_l(r)}{\langle \varphi_l|\,\Delta V_l^{\,2}|\,\varphi_l\rangle^{\frac{1}{2}}}$$

とする。こうするとすぐわかるように

$$\Delta V_l^{NL}(r)\,\varphi_l(r)=\Delta V_l(r)\,\varphi_l(r)$$

となる。しかも，この定義だと積分は分離され，波数の和はそれぞれについて単独に取ればいいことになる。

しかし，この巧妙な方法にも落とし穴がある。すぐに気付くのは，しかるべきエネルギーのは波動関数に依存するため，対象とする電子のエネルギーをいろいろ変えたときにきちんと真のポテンシャルをエミュレーションしているかどうか，つまり，transferbility が甚だ怪しくなると言うことである。この点は Gonze, Käckel, Scheffler [7]によって指摘され，対数導関数を見ると ghost-state がとんでもないところに出現する。この対処法は，その点に気を付けて擬ポテンシャルを作れ，という事である。

3．4　Ultrasoft Pseudopotential

いよいよ本題である Ultrasoft Pseudopotential に話を移そう。これは1990年に D.Vanderbilt [8]によって提案されたものである。これまで見てきた Norm-Conserving Pseudopotential (NCP) ではノルムを保存させるため，前に掲げた Mo の例でもわかるように決して擬ポテンシャル自身それほどソフトにはなっていない。従って，d電子やf電子などを扱おうとしてもまったくソフトになっておらず，従って full potential の計算とさして違わない平面波の数が必要となって事実上遷移金属を交えた計算が出来なかった。しかし，その制限は外から見た原子の電荷密度を本物と同じに見せるためにやむを得ない制限であった。それは self-consistent に擬波動関数を計算するときや Car-Parinello 法で動力学をするときの原子間の Helmann-Feynmann 力を正確に計算するためにはやむを得ない制限と思われてきた。ところが，D. Vanderbilt はこの制限は必ずしも不可欠なものではないことを示したのである。

ではまず D. Vanderbilt の論文に添って，NCP を書き直すことから入ろう。運動エネルギー演算子をTと書き，まず all-electron の場合の波動関数（真の波動関数）は

$$\left[T+V_{AE}(r)-\varepsilon_i\right]\left|\,\Psi_i\right\rangle=0$$

と書ける。一つのエネルギー固有値のところだけで参照する場合，先の Kleinman-Bylander のやり方に従えば

$$|\chi_i\rangle \equiv (\varepsilon_i - T - V_{local})|\varphi_i\rangle$$

と定義して，nonlocal ポテンシャルを

$$V_{NL} \equiv \frac{|\chi_i\rangle\langle\chi_i|}{\langle\chi_i|\varphi_i\rangle}$$

と定義する。$r=R$ の外側では $\varphi_i=\psi_i$ であることから $|\chi_i\rangle \equiv (\varepsilon_i - T - V_{local})|\varphi_i\rangle$ は明らかにRの内側だけに局在した関数である。すぐわかるように

$$V_{NL}|\varphi_i\rangle = |\chi_i\rangle$$

であり，従って

$$(\varepsilon_i - T - V_{local} - V_{NL})|\varphi_i\rangle = 0$$

なる Schrödinger 方程式を満たす。

もし，複数の固有値を参照して擬ポテンシャルを導くなら（通常はせいぜい２つであるが），その固有値を ε_i, ε_j として，まず

$$B_{ij} \equiv \langle\varphi_i|\chi_j\rangle$$

という行列を定義し，更にそれを使って

$$|\beta_i\rangle = \sum_j (B^{-1})_{ij}|\chi_j\rangle$$

という関数を定義する。これを使って nonlocal pseudopotential を次のように定義する。

$$V_{NL} \equiv \sum_{ij} B_{ij}|\beta_i\rangle\langle\beta_j|$$

こうすれば，すぐに計算してわかるように

$$\begin{aligned} V_{NL}|\varphi_n\rangle &= \sum_{ij} \mathrm{B}_{ij}|\beta_i\rangle\langle\beta_i|\varphi_n\rangle \\ &= \sum_{ij} B_{ij}|\beta_i\rangle \sum_k (B^{-1})^*_{jk}\langle\chi_k|\varphi_n\rangle \\ &= \sum_{ij} B_{ij} \sum_{j'} (B^{-1})_{j'i}|\chi_{j'}\rangle \sum_k (B^{-1})^*_{jk} B^*_{kn} \\ &= |\chi_n\rangle \end{aligned}$$

となる。従って，

$$V_{NL}|\varphi_n\rangle = (\varepsilon_n - T - V_{loc})|\varphi_n\rangle$$

であるから，

$$(\varepsilon_n - T - V_{loc} - V_{NL})|\varphi_n\rangle = 0$$

となる。ここで

$$\begin{aligned} B_{ij} &= \langle\varphi_i|(\varepsilon_j - T - V_{loc})|\varphi_j\rangle \\ &= \int_0^R dr\, u_i(r)\left[\varepsilon_j + \frac{1}{2}\frac{d_2}{dr^2} - \frac{l(l+1)}{2r^2} - V_{loc}(r)\right] u_j(r) \end{aligned}$$

である。ただし，$u_i(r) = r\varphi_i(r)$。

Norm Conserving Pseudopotential であれば，ここで本物の波動関数と擬波動関数とで

$$Q_{ij} = \langle\Psi_i|\Psi_j\rangle_R - \langle\varphi_i|\varphi_j\rangle_R$$

なる量を定義すれば，$Q_{ij} = 0$ である。しかし，Vanderbild の擬ポテンシャルでは $r = R$ においての波動関数と擬波動関数の対数導関数の一致のみを問題にし，ノルムは保存させない。従って，当然 Q_{ij} はゼロにならない。まず，重なり演算子として

$$S = 1 + \sum_{ij} Q_{ij}|\beta_i\rangle\langle\beta_j|$$

を定義しよう。そうして，non-local pseudopotential を

$$V_{NL}=\sum_{ij}(B_{ij}+\epsilon_j Q_{ij})|\beta_i\rangle\langle\beta_j|$$

と定義する。すると，これに対して，

$$(T+V_{loc}+V_{NL}-\epsilon_i S)|\varphi_i\rangle=0$$

という擬波動方程式が成立する。ここでエネルギー固有値に相当するところに重なり演算子がかけられていることに注意されたい。

従って，ノルムにこだわらずに滑らかな擬ポテンシャルを勝手に作って，それに対して擬波動関数を計算すればよいのだが，ノルムが保存していないと，電荷の保存が怪しくなり，従って分子動力学でもっとも重要な原子と原子の間に働く力の計算が問題になってくる。こうした場合にノルムは

$$\langle\varphi_i|S|\varphi_j\rangle_R=\langle\Psi_i|\Psi_j\rangle_R$$

と計算される。従って，この重なり演算子Sを入れていれば，外から見た電荷は保証されるわけである。

以上をとりまとめれば，一個の原子に対する真のポテンシャルがあったとして，それとある適当な半径 $r=R$ で対数導関数が真の波動関数に（関心のあるエネルギー領域で）等しい擬波動関数を与える適当な滑らかな擬ポテンシャルを見つけておけば，後はそれについてノルムを計算すれば本質的には Car-Parrinello の方法が可能になる[9,10]。

4．分子動力学プログラム

4.1　クラスライブラリー

C++ ではクラスライブラリーという形で一度作ったプログラムコードの再利用をすることができる。先に述べたようにオブジェクト指向言語ではデータとその手続きをひとかたまりにしてクラスという，いわばCにおける構造体や Fortran90 に於けるモジュールを拡張したものをプログラムの基本的な単位とする。そのクラスに適当な初期条件を与えてメモリ上に展開したものをそのクラスのインスタンスという。クラスライブラリーとは，そうしたこれまでに作ったプログラムコードを良く設計された複数のクラスのまとまりとして保存し，新しいプログラムの製作の時にはこれ

まで作ったプログラムのうち必要なものを必要な場所でインスタンスとして展開し，再利用する。従って，分子動力学のクラスライブラリーは開発とともに次第に多くのクラスを含むように発展させることができ，かつ古いクラスをインスタンスとして生成することで，以前のバージョンのプログラムコードも変更なしに実行できる。

この特徴は分子動力学クラスライブラリーの移植性にも貢献する。数値計算をそれだけでひとまとまりのクラスライブラリーとし，それぞれのウインドウシステムでは用いた部分のクラスをインスタンスとして生成すれば，そのままそのウインドウシステム上で分子動力学計算が稼働することになる。グラフィカル・ユーザー・インターフェースは市販の表計算ソフトなどとは違ってそれほど複雑なものとはならないから，それぞれのウインドウシステムごとに多少のプログラミングによって，分子動力学クラスライブラリーを用いた分子動力学計算ソフトウェアができあがることになる。

分子動力学の場合，たとえば原子を Atom というクラスにしてしまうのは一つの方法である。しかし，それではあまり広すぎるので，ここでは一個の原子をいくつかのクラスにどう分けるかを考える。これがこの論文の主題であり，そしてこれから発展していく分子動力学計算ライブラリーの全体を決める骨格部分となる。ここでは一つの原子について，その norm-conserving pseudotential や ultrasoft pseudopotential を計算することを念頭に置いて，まず真のポテンシャルに対して動径方向の波動関数を計算するという計算としては簡単な問題に絞って考察していくことにする。

4.2　グリッドクラス

Schrödinger 方程式で，動径方向の波動関数はポテンシャルが決まっているとすれば解くのはそれほど難しくない。求める量は波動関数であり，必要なのは動径方向の座標の値とポテンシャルの値である。ここで，一つのクラスの中にこの計算のすべてを押し込むことも可能である。たとえば次のようにクラスを作ることができる。

```
class Wavefunction
 {
      public :
           complex wavefunction [N] ;
      private :
           double     grid [N] ;
           double     potential [N] ;
      public :
           SolveEq () ;
```

}

このクラスでは波動関数の値と，それを求める命令である SolveEq () が外から参照できるパブリックであり，グリッドとポテンシャルはプライベートメンバになっている。たとえば学生に対する演示として波動関数を計算してみせるだけならこれで十分であろう。しかし，一度真の波動関数について波動関数を計算した後，それとある適当な半径r＝Rのところで対数導関数が等しい擬波動関数を求めていく作業を考えると，動径方向の波動関数を求める部分はなるべく共通のコードで，つまり，コードの再利用ができることが望ましい。その意味でいえば，上のクラスは落第で，違うポテンシャルについてはこのクラスのコード全体をエディタでコピーした上で異なるポテンシャルに合わせて一部コードを書き換える必要がある。それも，SolveEq () 本体の一部を書き換えることになる。これでは再利用とはいえない。

従って，グリットもポテンシャルもこうしたクラスのメンバーとしてではなく，独立したクラスとしてこれをインスタンスとして生成することにする。さらに，グリッドを定数で刻む場合と指数関数で刻む場合，さらには Harmann-Skillmann グリットのようなものなど，いろいろなグリッドが考えられるので，グリットとポテンシャルも切り放すことにする。そうすると，グリットの違いを引数としてグリッドのインスタンスを取るときの signiture で区別して扱う polymorphism が可能になる。まず，等間隔で刻むグリッドのクラスは次のようなものである。

```
const unsigned int Ngrid＝500 ;
const unsigned int N2grid＝Ngrid＊2 ;

class SConstGrid    {
      private :
            int T ;
      public :
            int Z, L, N ;      //charge, angular momentum,
                               //quantum number
            double maxR, minR, dR, hR ; //max., min, and difference of
                               //the grid
            duble gridR [Ngrid] ;      //grid
            double grid2 [N2grid] ;  //half grid for Runge-Kutta

      public :
            SConstGrid (int, int, int, int) ;      //constructor
```

```
        ~SConstGrid () {} ;      //destructor
        void makeGrid () ;       //procedure to make grid
        void maxGrid () ;        //procedure to get maximum
} ;
```

4.4　ポテンシャルクラス

ポテンシャルのクラスはポテンシャルを作るに当たってのグリッドをグリッドクラスから取り，その刻みに従ったポテンシャルをデータとして蓄えるクラスである。その際に，フルポテンシャル以外に norm-conserving pseudopotential, ultrasoft pseudopotial なども同じように解きたいため，まず，抽象基底クラスとして SPotentiol を定め，それからの派生クラスとしてフルポテンシャルを扱う SFullPot を作る。こうすると，別に SNormconserve, SUltrasoft などが派生クラスとしてつくれることになる。こうした場合のメリットは，実際にこれらのクラスのインスタンスを生成したときに，記憶場所として基底クラスの SPotential のメンバが先に並び，その後にそれぞれの派生クラス特有のメンバが並ぶ。従って，別のクラスで SPotential＊v とインスタンスを生成したときにはｖの中身が実際には SFullPot であろうが別のであろうが同じように扱えることになる。

実際の定義を示そう。

```
class SPotential {     //ポテンシャル基底クラス
        public :
                int type ;
                double V [N2grid] ;
        public :
                SPotential () {} ;
                virtual ~SPotential ()
                        { delete [] V ;} ;
                const SPotential & operator=(const SPotential & pot) ;
        //insert operation
} ;
class SFullPot : public SPotential { //フルポテンシャルのクラス
    public :
        SConstGrid＊r ;     //グリッドクラスのインスタンス
    public :
        SFullPot () ;     //default constructor
```

```
        SFullPot (SConstGrid*) ; //constructor
        ~SFullPot () {delete r ; delete [] V ;} ;  //destructor
          void makePotential (SConstGrid*r) ;  //ポテンシャルを作る
} ;
class SNormconserv : public SPotential {  //norm-conserving pseudopotential
    public : SConstGrid*r ;  //グリッドクラスのインスタンス
    public :
        SNormconserv (SConstGrid*r) {} ;
        ~SNormconserv () {} ;
        void makePotential (SConstGrid*r) ;  //ポテンシャルを作る
} ;
class SUltrasoft : public SPotential {    //ultrasoft pseudopotential
    public :    SConstGrid*r ;  //グリッドクラスのインスタンス
    public :
        SUltrasoft (SConstGrid*r) {} ;
        ~SUltrasoft () {} ;
        void makePotential (SConstGrid*r) ;  //ポテンシャルを作る
} ;
```

まず，基底クラスの SPotential でポテンシャルの値をメンバとして持つ。こうしておけば，実装でどのポテンシャルについての派生クラスでインスタンスを構成しても，利用するコードの方で SPotential を用いると宣言しておけば，そのコードはどのポテンシャルについても共通に使える。

また，実際にポテンシャルを生成する関数である makePotential は，すべて引数として SConstGrid*をとっている。こうしておけば，グリッドとして等間隔ではなく，たとえば指数関数グリッドなどを用いるときでも，別に makePotential (SExpGrid*r) をクラスメンバー関数として追加すれば，signiture から C++コンパイラは二つの makePotential を区別してくれる。

4.5　波動関数クラス

グリッドクラスとポテンシャルクラスのインスタンスを含む形で波動関数クラスを構成する。波動関数自体は Runge-Kutta 方で計算するものとする。

```
class SWaveFunction
 {
    public :
```

```
        double E ;  //エネルギー
    protected :
        double P, Q, dR, R ;
    public :
        SConstGrid * r ;
        SFullPot * v ;
    public :
        double      wave [Ngrid] ; //波動関数
        double      dwave [Ngrid] ;     //波動関数の一階導関数
    protected :  //内部処理用
        doublewave_for [Ngrid] ;  //波動関数
        double      dwavefor [Ngrid] ;  //波動関数の一階導関数

    public :
        SWaveFunction () {} ;
        SWaveFunction (int, int, int, int) ;
        SWaveFunction (SConstGrid *, SFullPot *) ;
        ~SWaveFunction () {} ;
        virtual void makeWave () ;
    protected :  //波動関数の計算で用いる関数
        doubl      define_target_E () ;
        void getInitValue_for () ;
        void solveWave () ;
        void runge_kutta_for (int i) ;
        double      dwave1 (int i) ;
        double      dwave2 (int i) ;
} ;
```

実装に当たっては public メンバと protected メンバの使い分けに注意されたい。一般に外部から利用されるもののみを public メンバとし，それ以外は protected か private とする。ここで波動関数の計算をさせる makeWave () は public であり，外から波動関数の計算を促すことが出来るが，その中身は solveWave () そのものであり，つまり実際の計算をさせるプログラムコードの実体は外部から干渉できないようになっている。

4.6　ユーザーインターフェイス

ユーザーインターフェイスはそのコードがC++で記述されている限り，どのウインドウシステムについてもほぼ似たように実装できる。上で挙げた例で言えば，SWaveFunctionのインスタンスを生成して，その中のwave [Ngrid] についてグラフを描くなり，適当な処理をして行けばいいのである。幸い，MS-Windows, OS/2, Machintosh OS, X-Window, MS-WindowsNTなど，現在身の回りでメジャーに使われているウインドウシステムのほとんどはC++によるコーディングが可能である。ここではその一例として，MS-Windows 3.1及びWindowsNT (Win32s) で動作する，波動関数のグラフを描くユーザーインターフェースの例をObject Windows Library 2.0によってコーディングしたものを付録に示す。ほぼ同じ構造で，Microsoft Foundation Class 2.0 (2.5) でコーディングすることもできる。

4.7　今後の拡張

ここで問題としたのは一個の原子についての波動関数の計算プログラムと，それを擬ポテンシャルに再利用していく場合のクラス構成である。実際に分子動力学計算を行うにはこれらをまとめてSAtomという一つのクラスとし，そこにその原子の位置と擬波動関数，及びその原子に働く力（Hellmann-Feynmann force）のデータを集めておくのが便利である。そのうえで，実際に単位格子なり，スラブ計算で電子状態を計算していく部分ではSAtomクラスのインスタンスを扱う原子の数だけ生成することになる。その際にすべての原子にわたって共通の操作を繰り返す必要性から，クラスのインスタンスを扱う一般的なコンテナクラスがC++コンパイラーにつけられていること，それに付随して，不可欠というほどではないにしてもテンプレート機能の装備が望ましい。その意味で，現在メジャーなコンパイラの一つ，Microsoft Visual C++ 1.0/1.5は望ましい開発環境ではないが，今年のうちに発売が予定されている次のver2.0において対応することがアナウンスされているのでこれから開発を進めていく分には大きな問題ではないであろう。メモリの問題についても次第に解決されていく方向である。一般に小さなプロトタイプをパソコンで動作確認した上でワークステーション上で本格的に稼働させることが望ましいので，その意味でも分子動力学計算の本体をユーザーインターフェースから完全に切り放しておくことが肝要である。

5．結　論

本研究においては，オブジェクト指向に基づいた完全なクロスプラットフォームによる分子計算のプログラム開発の重要性を指摘し，それを実現する分子動力学計算プログラムの基本設計を提示した。C++を用いてコーディングし，原子の波動関数の計算と擬波動関数の計算をpolymorphism

を利用してクラス化する方法を示した。

謝　辞

本研究で示したプログラムコードのうち，原子の動径方向の波動関数を計算する Runge-Kutta 法の部分は逢坂豪先生が以前BASICを用いてコーディングしたものを C++ に移植したものである。

参考文献

(1) M. A. Ellis and B. Stroustrup, The Annotated C++ Reference Manual (Addison-Wesly, 1992) (邦訳：注解 C++ リファレンスマニュアル，トッパン，1992)
(2) R. Car and M. Parrinello, Phys. Rev. Lett. 55 (1985) 2471
(3) J. C. Phillips and L.Kleinman, Phys. Rev. 116 (1959) 287
(4) D. R. Hamann, M.Schlüter and C. Chiang, Phys. Rev. Lett. 43 (1979) 1494
(5) G. B. Bachelet, D.R.Hamann and M. Schlüter, Phys. Rev. B26 (1982) 4199
(6) L. Kleinman and D. M. Bylander, Phys. Rev. Lett. 48 (1982) 1425
(7) X. Gonze, Peter Käckell and M. Scheffler, Phys. Rev. B41 (1990) 12264
(8) D. Vanderbilt, Phys. Rev. B41 (1990) 7892
(9) K. Lassonen, R. Car, C. Lee and D. Vanderbilt, Phys. Rev. B43 (1991) 6796
(10) K. Lassonen, A. Pasquarello, R. Car, C. Lee and D. Vanderbilt, Phys. Rev. B47 (1993) 10142

付録　分子動力学計算のプログラム実装部分

本文で紹介しているグリッドクラス，ポテンシャルクラス，波動関数クラスの各メンバ関数の実装部分を示す。また，ユーザーインターフェースの実例を Borland Object Windows Library 2.0 クラスライブラリを用いて示す。

```
// Sakyu Objective Molecular Dynamics Class Library
//                        version 1.00     May 9, 1994
//     Calculate radial grid, potential of half-gride,
//     and wave. . .
# include<iostream.h>
# include<math.h>
# include "atomlib. h"
```

```
SConstGrid : : SConstGrid (int z, int l, int n, int t=4)
 {
      Z=z ;
      L=l ;
      N=n ;
      T=t ;
      for (int i=0 ; i <Ngrid ; i++)
      {
           gridR [i]=0.0 ;
      }
      for (i=0 ; i <N2grid ; i++)
      {
           grid2 [i]=0.0 ;
      }
      minR=0.00001 ;
}
void SConstGrid : : maxGrid ()
 {
      double am, a0, dd, a ;
      am=2.0*T*2.30259259 ; //log (10)=2.30259259
      a0=am ;
      dd=2.*N ;
      a=am+dd*log (a0) ;

      while ((a-a0)> 0.0000001*a0)
      {
           a0=a
           a0=am+dd*log (a0) ;
}
maxR=0.5*a*N/Z ;
}
void SConstGrid : : maxGrid()
```

```
{
    dR=(maxR-minR)/Ngrid ;
    hR=dR*0.5 ;     //increment of the grid
    gridR [0]   =grid2 [0]=minR ;
    for (int i=1 ; i <Ngrid ; i++)
    {
        gridR [i]   =gridR [i-1]+dR ;
    }
    for(i=1 ; i <N2grid ; i++)
    {
        grid2 [i]   =grid2 [i-1]+hR ;
    }
}
const SPotential & SPotential : :operator=(const SPotential & pot)
{
    if (& pot !=this)
    {
        for (int i=0 ; i <N2grid ; i++)
        {
            V [i]     =pot. V [i] ;
        }
        type=pot. type ;
    }
    return *this ;
}
SFullPto : :SFullPto(SConstGrid* cg) :r(cg)
{
    type=FULLPOT ;
    r-> maxGrid () ;
    r-> makeGrid () ;
}
void SFullPto : :makePotential (SConstGrid* r)
```

```
 {
      for (int i=0 ; i <N2grid ; i++)
      {
            V [i]   =-(r-> Z)/r-> grid2 [i] ;
      }
}
SWaveFunction : : SWaveFunction (int Z, int L, int N, int T=4)
 {
      SConstGrid* r=new SConstGrid (Z, L, N, T) ;
      SFullPot*   v=new SFullPot (r) ;
//  v-> makePotential (r) ;
//  E=define/target/E (r) ;
//  makeWave (r, v) ;
}
SWaveFunction : : SWaveFunction (SConstGrid *rr, SFullPot *vv) : r (rr), v (vv)
 {//You shouid call this constructor after you construct rr and vv.
}
void SWaveFunction : : makeWave ()
 {
      E   =define_target_E () ;
      getInitValue_for () ;
      solveWave () ;
      for (int i=0 ; i <Ngrid ; i++)
      {
            wave [i]   100*wave_for [i] ;
            dwave [i]   100*dwavefor [i] ;
      }
//  normalize () ;
}
double SWaveFunction : : define_target_E ()
 {
      double Er=1. ;
```

```
        double Energy=-0.5*(r->Z)*(r->Z)/(r->N)/(r->N)*Er;
//  E=-0.5*pow(r->Z,2.)/pow(r->N,2.)*Er;
        return Energy;
}
void SWaveFunction::getInitValue_for()
 {  //Initial condition of Runge-Kutta, forward
        double R1;
        R1=sqrt(-2.*E)*(r->minR);
        P  =exp(-R1)*pow((2.*R1),r->L+1.);
        Q  =-P*sqrt(-2.*E);
        dR  =((r->maxR) - (r-minR))/Ngrid;
        return;
}
void SWaveFunction::solveWave()
 {
        R  =r->gridR[0];
        dR =r->dR;
        wave/for[0]=P;
        dwavefor[0]=Q;
        for(int i=0; i<Ngrid; i++)
        {
                wave_for[i]=P;
                dwavefor[i]=Q;
                runge_kutta_for(i*2);
                R  =R+dR;
        };
}
void SWaveFunction::runge/kutta/for(int i)
 {
        double PP  =P;
        double QQ  =Q;
        double p0=dR*dwave1(i);
```

```
        double q0=dR*dwave2 (i) ;
        P   =PP+0.5*p0 ;
        Q   =QQ+0.5*q0 ;
        double p1=dR*dwave1 (i+1) ;
        double q1=dR*dwave2 (i+1) ;
        P   =PP+0.5*p1 ;
        Q   =QQ+0.5*q1 ;
        double p2=dR*dwave1 (i+1) ;
        double q2=dR*dwave2 (i+1) ;
        P   =PP+p2 ;
        Q   =QQ+q2 ;
        double p3=dR*dwave1 (i+2) ;
        double q3=dR*dwave2 (i+2) ;
        P   =PP+(p0+2.*(p1+p2)+p3)/6. ;
        Q   =QQ+(p0+2.*(p1+p2)+p3)/6. ;
}
double SWaveFunction : :dwave1 (int i)
 {
        double f=Q+(r-> L+1)/(r-> grid2 [i]) *P ;
        return f ;
}
double SWaveFunction : :double SWaveFunction : :dwave2 (int i)
 {
        double f=2.*(v-> V [i]-E) *P-(r-> L+1)/(r-> grid2 [i]) *Q ;
        return f ;
}
void SWaveFunction : :normalize ()
 {
        double S=0.0 ;
        for (int i=1 ; i <Ngrid-2 ; i=i+2)
        {
            S=S+wave [i-1] *wave [i-1]+4.*wave [i] *wave [i]+wave [i+1] *wave [i+1] ;
```

```
        }
        S=S*dR/3. ;
        double C=sqrt(S) ;
        for(i=0 ; i <Ngrid ; i++)
        {
                wave [i]=wave [i]/C ;
        }
}
double SWaveFunction : : GetMaxValue ()
 {
        double max=wave [0] ;
        for(int i=0 ; i <Ngrid/2 ; i++)
        {
                if(wave [i]> max)
                {max=wave [i] ;}
        }
        return max ;
}
```

OWL 2.0 によるユーザーインターフェース部分のコーディング

ヘッダー部分と実装本体を合わせてあるが，現実は別のファイルにしている。

```
/*   atom4app.cpp        OVERVIEW
     ==========
      Class definition for atomwin4App (TApplication).        */
# include <owl¥owlpch.h>
# pragma hdrstop
# include <owl¥editfile.h>
# include <owl¥opensave.h>
# include "atom4app.rh"          //Definition of all resources.
// {{TApplication=atomwin4app}}
Class atomwin4App : public TApplication  {
private :
        TEditFile *Client ;        //Client window for the frame.
```

```
        TOpenSaveDialog : :TData FileData ;        //Data to control open/saveas standard dialog.
public :
        atomwin4App () ;
        virtual ~atomwin4App () ;
        void OpenFile (const char *fileName=0) ;
// {{atomwin4AppVIRTUAL_BEGIN}}
public :
        virtual void IntiMainWindow () ;
// {{atomwin4AppVIRTUAL_END}}
// {{atomwin4AppRSP_TBL_BEGIN}}
protected :
        void CmHelpAbout () ;
        void DrawWave () ;
// {{atomwin4AppRSP_TBL_END}}
  DECLARE_RESPONSE_TABLE (atomwin4App) ;
} ; // {{atomwin4App}}
# include <owl¥owlpch.h>
# pragma hdrstop
# include <owl¥vbxctl.h>
# include "tmwn4abd.h"          //Definition of about dialog.
# include "tmywindw.h"
// {{atomwin4App Implementation}}
//Build a response table for all messages/commands handled
//by the application.
//
DEFINE_RESPONSE_TABLE1 (atomwin4App, TApplication).
// {{atomwin4AppRSP_TBL_BEGIN}}
        EV_COMMAND (CM_HELPABOUT, CmHelpAbout),
        EV_COMMAND (CM_DRAWWAVE, DrawWave),
// {{atomwin4AppRSP_TBL_END}}
END_RESPONSE_TABLE ;
//FrameWindow must be derived to override Paint for Preview and Print.
```

```
//
class SDIDecFrame : public TDecoratedFrame {
public :
        SDIDecFrame (TWindow *parent, const char far *title, TWindow *clientWnd, BOOL
trackMenuSelection=FALSE, TModule *module=0) :
                TDecoratedFrame (parent, title, clientWnd, trackMenuSlection, module)
        { }
        ~SDIDecFrame ()
        { }
} ;
atomwin4App : : atomwin4App () : TApplication("Sakyu Objective Molecular Dynamics")
 {
        //Common file file flags and filters for Open/Save As dialogs. Filename and directord are
        //computed in the member functions CmFileOpen, and CmFileSaveAs.
        FileData.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
        OFN_OVERWRITEPROMPT ;
        FileData.SetFilter ("ALL Files (*.*) | *.* | ") ;
        //INSERT >> Your constructor code here.
}
atomwin4App : : ~ atomwin4App ()
 {
        //INSERT >> Your destructor code here.
}
//atomwin4App
//=====
//Application intialization.
//
void atomwin4App : : InitMainWindow ()
 {
        Client=new TEditFile (0, 0, 0) ;
                SDIDesFrame *frame=new SDIDecFrame (0, GetName (), new TMyWindow) ;
                //TMyWindow in this new operation is a key parameter to link
```

```
  //this Mainframe with the TMyWindow cl              ass of tmywindw.cpp
      nCmdShow=nCmdShow ! =SW_SHOWMINIMIZED ? SW_SHOWNORMAL : nCmdShow ;
      //Assign ICON w/this application.
      frame-> SetIcon (this, IDI_SDIAPPLICATTON) ;
      //
      //Menu associated with window and acceletor table associated with table.
      //
      //Associate with the accelerator table.
      frame-> Attr.AccelTable=SDI_MENU ;
      MainWindow=frame ;
}
//==========
//Menu Help Abou atomwin4.exe command
void atomwin4App : : CnHelpAbout ()
 {
      //
      //Show the modal dialog.
      //
      atomwin4AboutDlg (MainWindow). Execute () ;
}
int Owlmain (int, char* [])
 {
      TBIVbxLibrary   vbxSupport ;       //This application has VBX controls.
      atomwin4App     App ;
      int             result ;
      result=App.Run() ;
      return result ;
}
void atomwin4App : : DrawWave ()
 {
                    //INSERT >> Your code here.
}
```

```
/*TMyWindow.cpp        OVERVIEW
      =====
      Class definition for TMyWindow (TWindow ).        */
# include <owl¥owlpch.h>
# pragma hdrstop
# include <owl¥window.h>
# include <owl¥dc.h>          //device context
# include <classlib¥arrays.h>
# include "tnptll_d.h"
# include "atom4app.rh"          //Definition of all resources.
#include "atomlib.h"          //分子動力学計算プログラムを引用指定
typedef TArray <TPoint> TPoints ;
typedef TArrayIterator <TPoint> TPointsIterator ;
// {{TWindow=TMyWindow}}
class TMyWindow : public TWindow {
public :
            TDC *graphDC ;
            TPoints*          Line ;
            TPoints*Wave ;
            SConstGrid*r ;        //分子動力学計算のグリッドクラスのインスタンス
            SFullPot*   v ;       //ポテンシャルクラスのインスタンス
            SWaveFunction*   w ;        //波動関数クラスのインスタンス
protected :
            int Z ;
            int L ;
            int N ;
public :
      TMyWindow (TWindow* parent=0, const char far* title0, TModule* module=0) ;
      virtual ~TMyWindow () ;
      void Paint (TDC&, BOOL, TRect&) ;
      void drawXaxis (TDC*) ;
      void drawYaxis (TDC*) ;
```

```
// {{TMyWindowRSP_TBL_BEGIN}}
protected:
        void DraWave () ;
        void EvLButtonDown (UINT modKeys, TPoint & point) ;
    void InputZ () ;
    void EvRButtonDown (UINT modKeys, TPoint & point) ;
    void InputL () ;
    void InputN () ;
    void SetInputs () ;
// {{TMyWindowRSP/TBL/END}}
DECLARE_RESPONSE_TABLE (TMyWindow) ;
} ;      // {{TMyWindow}}
# include <owl¥owlpch.h>
# include <owl¥inputdia.h>
# pragma hdrstop
# include "atomlib.h"
//
//Build a response table for all messages/commands handled
//by the application.
//
DEFINE_RESPONE_TABLE1 (TMyWindow, TWindow)
// {{TMyWindowRSP_TBL_BEGIN}}
    EV_COMMAND (CM_DRAWWAVE, DrawWave),
    EV_WM_LBUTTONDOWN,
    EV_COMMAND (CM_INPUT_Z, InputZ),
    EV_WM_LBUTTONDOWN,
    EV_COMMAND (CM_INPUT_L, InputL),
    EV_COMMAND (CM_INPUT_N, InputN),
    EV_COMMAND (CM_SETINPUT, SetInputs),
// {{TMyWindowRSP_TBL_END}}
END_RESPONSE_TABLE ;
// {{TMyWindow Implementation}}
```

```
TMyWindow : : TMyWindow (TWindow * parent, const char far * title, TModule * module) :
        TWindow (parent, title, module)
 {
        //Override the default window style for TWindow.
        Attr.Style | =WS_BORDER | WS_CAPTION | WS_CHILD | WS_MAXIMIZEBOX |
        WS_MINMIZEBOX | WS_VISIBLE ;
                //INSERT >> Your constructor code here.
                Z=25 ;
                L=1 ;
                N=2 ;
                Init (parent, 0, 0) ;
                graphDC=0 ;
                Line=new TPoints (10,0,10) ;
                Wave=new TPoints (10,0,10) ;
}
TMyWindow : : ~ TMyWindow ()
 {
                Destroy () ;
                //INSERT >> Your destructor code here.
                delete graphDC ;
                delete Line ;
                delete Wave ;
}
void TMyWindow : : DrawWave ()
 {
        //INSERT >> Your code here.
        if ( ! graphDC)
          {
                SetCapture () ;
                graphDC   =new TClientDC (*this) ;
                graphDC-> SetMapMode (MM_HIMETRIC) ;
                TPoint p0   =TPoint (-500,5000) ; //origin of the coordinate
```

```
graphDC-〉 SetWindowOrg(p0) ;
            drawXaxis (graphDC) ;
            drawYaxis (graphDC) ;
                SConstGrid *r=new SConstGrid (Z, L, N, 4) ;
                r-〉 maxGrid () ;
                r-〉 makeGrid () ;
                SFullPot *v=new SFullPot (r) ;
                v-〉 makePotential (r) ;
                for (int i=0 ; i 〈N2grid ; i++)
                {
                      if (v-〉 V [i]〉 -6000 && v-〉V [i] 〈-6000)
                      {
                int px=(10000*(r-〉 grid2 [i])) ;
                            int py=1.*(v-〉 V [i]) ;
                            TPoint p=TPoint(px,py) ;
                            switch (i)
                            {
                                  case 0 :
                                        graphDC-〉 MoveTo (px,py) ;
                                        break ;
                                  default :
                                        graphDC-〉 LineTo (px,py) ;
                                        break ;
                            }
                            Line-〉 Add (p) ;
                      }
                }
                SWaveFunction *w   =new SWaveFunction (r,v) ;
                w-〉 makeWave () ;
                double max   =w-GetMaxValue () ;
                for (i=0 ; i 〈Ngrid ; i++)
                {
```

```
                        int px=(10000*(r->gridR[i]));
                        int py=(5000*(w->wave[i]/max);
                        TPoint p=TPoint(px,py);
                        switch (i)
                        {
                                case 0:
                                        graphDC->LineTo(px,py);
                                        break;
                                default:
                                        graphDC->LineTo(px,py);
                                        break;
                        }
                        Wave-Add(p);
                }
        }
}
void TMyWindow::Paint(TDC & dc, BOOL, TRect &)
 {
        BOOL first  =TRUE;
        TPointsIterator i(*Line);
        TPointsIterator j(*Wave);
                dc.SetMapMode(MM_HIMETRIC);
                TPoint p0  =TPoint(-500,5000); //origin of the coordinate
dc.SetWindowOrg(p0);
                drawXaxis(& dc);
                drawYaxis(& dc);
        while(i)
                {
                        TPoint p=i++;
                        if(!first)
                                dc.LineTo(p);
                        else
```

```
                {
                        dc.MoveTo (p) ;
                        first=FALSE ;
                }
        }
    while (j)
        {
                TPoint q=j++ ;
                if ( ! first)
                        dc.LineTo (q) ;
                else
                {
                        dc.MoveTo (q) ;
                        first=FALSE ;
                }
        }
}
void TMyWindow : : drawXaxis (TDC* DC)
  {
        TPoint porg=TPoint (0, 0) ;
        TPoint pend=TPoint (6000, 0) ;
        DC-> MoveTo (porg) ;
        DC-> LineTo (pend) ;
}
void TMyWindow : : drawYaxis (TDC* DC)
  {
        TPoint ybottom   =TPoint (0, -6000) ;
        TPoint ytop   =TPoint (0, 6000) ;
        DC-> MoveTo (ybottom) ;
        DC-> LineTo (ytop) ;
}
void TMyWindow : : EvLButtonDown (UINT modKeys, TPoint & point)
```

```
 {
      TWindow : : EvLButtonDown (modKeys, point) ;
      //INSERT 〉〉 Your code here.
      if (greaphDC)
      {
            ReleaseCapture () ;
            Line-〉 Flush () ;
            Wave-〉 Flush () ;
            Invalidate () ;
            delete graphDC ;
      }
}
void TMyWindow : : InputZ ()
 {
      //INSERT 〉〉 Your code here.
      char inputZ [6] ;
      if ((TInputDialog (rhis, "Atomic Number" , "Input a new Z number" ,
                                          inputZ, sizeof (inputZ))). Exec () =IDOK)
      {
            Z     =atoi (inputZ) ;
      }
}
void TMyWindow : : EvRButtonDown (UINT modKeys, TPoint & point)
 {
      TWindow : : EvRButtonDown (modKeys, point ;
            //INSERT 〉〉 Your code here.
            Invalidate () ;
}
void TMyWindow : : InputL ()
 {
      //INSERT 〉〉 Your code here.
            char inputL [6] ;
```

```
        if ((TInputDialog (this, "Angular Momentum", "Input a new L number",
                                        input, sizeof (inputL))). Execute () == IDOK)
        {
            L     =atoi (inputL) ;
        }
}
void TMyWindow : : InputN ()
 {
    //INSERT >> Your code here.
    char input [6] ;
    if ((TInputDialog (this, "Main Quantum Numher", "Input a new N number",
                                        inputN, sizeof (inputN))). Execute () == IDOK)
    {
        N     =atoi (inputN) ;
    }
}
void TMyWindow : : SetInputs ()
 {
    //INSERT >> Your code here.
    TINPUTALL_DIALOG adialog=new TINPUTALL_DIALOG (this, IDD_INPUTALL, 0) ;
    adialog. SetLvalue (L) ;
    adialog. SetLvalue (N) ;
    if (adialog. Execute () == IDOK)
    {
        L=adialog. GetLvalue () ;
        N=adialog. GetNvalue () ;
    }
}
```