ＰＳＥ可視化インタフェース

VISUAL　INTERFACE　FOR　PSE

○梅 谷　征 雄[1)]
Ｙｕｋｉｏ ＵＭＥＴＡＮＩ

１）静岡大学教授　情報学部情報科学科（〒432-8011　浜松市城北３－５－１　umetani@cs.inf.shizuoka.ac.jp）

The method for systematically establishing visual and interactive computational interfaces required for PSE (Problem Solving Environments) are investigated in relation with the programming language model. Four kinds of models (procedural, domain-specific, object-oriented and dataflow) are examined and evaluated using an actual simulation example. The merits and demerits of each model are described in details.

Key Words : PSE, programming language model, simulation

PSE における可視化インタフェースを系統的に構築する方法をプログラミング言語モデルとの関係で考察する。言語モデルとして手続き型、応用向き、オブジェクト指向、データフローの 4 種類を取り挙げ、実問題への適用を通じて各々の得失を明らかにする。

１．はじめに

ＰＳＥ(Problem Solving Environment)が取り扱う物理モデルの数値シミュレーションには、１）特殊な計算ノーハウを必要とする、２）計算規模が大きい、３）モデル合わせのための微妙なパラメータ調整が必要、４）演算精度の監視が必要、などの特徴があり、これに対して、高水準記述からのコード生成、解法エキスパートの活用、高性能計算サーバとのスムーズな連携、計算と可視化の連携、自動精度モニタリングなどの試みが行われている。ここでは、計算と可視化の連携を容易にするための対話インターフェースと、ＰＳＥでサポートすべき機能について検討する。特に言語モデルと対話の関係を考察する。 言語のモデルとして、手続き型、応用向き、オブジェクト指向、データフローの４者を取り上げ，それらの可視化対話の方法を具体例に即して述べ、各々の得失を検討する。それに基づいて、ＰＳＥに適した言語モデルを考察する。

２．言語モデルと可視化のレベル

計算と可視化の緊密な連係を実現するためには、可視化対話のレベルをどこに取るかが重要なポイントとなる。プログラムの実行論理は言語モデルにより異なるので、典型的な言語モデルを取り上げ、それぞれに対する適切な可視化対話方法を探ることとする。

最もポピュラーなものとして手続き型言語モデルがある。これについては FORTRAN や C 言語を構成するデータ構造やソース文のレベルで表示や対話を行なうのが自然なアプローチであろう。文間の中断点の設定／解除、中断点におけるデータの表示やパラメータの変更、先行中断点への復帰などが対話の単位となる。 このモデルの長所は汎用的であることと処理の流れが良く見えることにあるが、反面機能的な対話を行うには細かすぎるので、問題に合わせて中断点ごとにデータの表示方法等をプログラマが工夫する必要がある。

応用分野を指向した高水準言語を使用すればこの欠点はかなりの程度カバーできる。例えば偏微

分方程式により定式化される流体や電磁界のシミュレーションに対しては､境界領域形状､境界条件､空間メッシュ、離散化方式、線形計算アルゴリズムなどのレベルで問題と解法を記述する高水準言語が提案されており[1,2,3]、それらを用いれば応用を志向した可視化と対話が自然に行える。一方、問題が言語の枠組みに収まらないときには適用できないことがこのモデルの弱点である。

第３のモデルはオブジェクト単位に可視化や対話を行なうオブジエクト指向のアプローチであり[4,5]、C++などのオブジェクト指向言語で書かれたアプリケーションには自然に導入できる。この方法の利点はオブジェクト間の参照関係とシミュレータの全体構成を意識した対話ができることにある。オブジェクトにはデータの側面とそれを操作する関数の２つの側面があり、前者からはデータ間の参照関係を、後者からは関数間の呼び出し関係を意識することが出来る。

第４のアプローチは可視化システム ＡＶＳ[6]などで採られているデータフロー指向のモデルである。これはネットワークプログラミングとも呼ばれ、処理手順をデータの変換手順に合わせて構造化する方法を用いる。このモデルでは値が変化したデータに対する処理のみが次々に駆動されることになる。データの変換プロセスに沿って可視化と対話を行なうのが自然なアプローチである。直接データフロー言語で記述されたシミュレータは少ないが、手続き型言語によるアプリケーションをこの手法で構造化することはオブジェクト指向言語などに比べれば容易であり、変換要素の採り方により抽象化のレベルを変えることができる。データフローアプローチはパラメータ変更の影響を見るのに適しており、また並列・分散処理との相性が良い。

３．実例

本章では２章で触れたいくつかのアプローチを具体例に即して述べることとする。取り上げる例題はMOS半導体のデバイスシミュレーション[7]である。これは模式的には図１のような半導体内部の電子密度と正孔分布密度を与えられた不純物濃度と境界条件の元で解き、それに基づきゲート電圧・電流グラフの様なデバイス特性を得るものである。電子密度 n と正孔分布密度 p は、以下の３本の偏微分方程式を連立させ、有限要素法や差分法で離散化近似を行なった後、数値的に解いて求める。

ポアソン方程式　　$e \Delta\psi = q (n - p - Nd + Na)$

電子電流方程式　　$div \ Jn = q \ Urg$

正孔電流方程式　　$div \ Jp = -q \ Urg$

e:Siの誘電率　ψ:電位　Nd - Na:不純物濃度

Jn:電子電流密度　Jp:正孔 電流密度　q:電子電価

Urg:キャリアの再結合の割合

ここで、

$Jn = k T \ \mu n \ nie \ exp(q \psi / (k T)) \nabla (exp (-q \psi / (k T)) n / nie)$

$Jp = k T \ \mu p \ nie \ exp(-q \psi / (k T)) \nabla (exp (q \psi / (k T)) p / nie)$

であり、

$Vt = k T / q , \eta = exp (-q \psi / (k T)) n / nie, \rho = exp (q \psi / (k T)) p / nie$

とおけば、

電子電流方程式　$Vt \ div \ (\mu n \ nie \exp(\psi / Vt) \nabla \eta) = U rg$

正孔電流方程式　$Vt \ div \ (\mu p \ nie \exp(-\psi / Vt) \nabla \rho) = - U rg$

k:ボルツマン定数　T:絶対温度　μn:電子移動度　nie:有効真性キャリア密度　μp:正孔移動度

と表わせる。以後の計算では、$Urg = 0$ を仮定する

(１)　手続き型言語からのアプローチ

上記のような偏微分方程式の数値解は、通常 FORTRAN の様な手続き型言語で解法をプログラムすることにより求めら。プログラムを構成する文の間でデータの表示や対話などを行なうことになる。対話実行を行なう標準的な言語処理ソフトウエアが用意されればプログラマの介入なしに実現できるが、機能的な対話を行うにはレベルが低く過ぎること、境界形状など非言語的な情報の取り扱いが不便であることなどの問題がある。

(２)　応用向き言語からのアプローチ

応用向きの高水準言語 DEQSOL[1]を用いる場合の例をしめす。　DEQSOL ではプログラムが、形状、メッシュ、データ、物性値、境界条件、初期条件、解法手順などの機能別に記述されるので、どのような DEQSOL プログラムにたいしても､汎用的に図２のようなウインドウを使って可視化と対話をおこなうことができる。対話の方法としては、左右のボタンと中央の画面をつかって、境界形状、境界条件、メッシュ、パラメータ値などを変更したり、上部のボタンにより解法手順に対する文単位ならびに式のレベルに立ち入った実行制御が可能である。左右のボタンによるパラメータ値等の変更の効果を直ちに中央の画面で可視化できれば対話の効果が上がる。DEQSOL による解法手順の記述は、線形化した単独の方程式の求解を単位として 3 本の方程式の求解を巡回させる手順となるので、それらの文の間で結果を可視化しながら上部のボタンで Single Step 操作を行えば巡回時の効果をじかに知ることができる。さらに細かくは Operation Step 操作により単独方程式の求解の細部に立ち入ることも出来る。

(３)　オブジェクト指向のアプローチ

オブジェクト指向のアプローチでは、クラス群を定義し、クラスに属するオブジェクトに対する操作の列としてプログラムを構成する。各クラスに属するデータ等のメンバを操作するメソッドを定義することにより、オブジェクトを単位とした対話が可能となる。

図３に本デバイスシミュレータの C++言語によるオブジェクト指向プログラミングの一部を示す。図４にて、SwPointPtr, SwEdgePtr など(以降は Sw*Ptr と略記)は点や縁など形状要素のクラスを、SwFEMMesh はメッシュのクラスを、SwFEMVar は変数や定数のクラスを、SwLinSystem は線形計算のクラスを、SwMatrix と SwVector は行列とベクターのクラスを、Sw*Form は離散化方程式のクラスを、Sw*Bc*ﾊは境界条件のクラスを意味する。また（１）は形状定義の部分、（２）はメッシュ定義の部分、（３）は変数、定数定義の部分、（４）は方程式定義の部分、（５）は境界条件設定の部分、（６）はそれを含んで解法手順を記述する部分である。

ひとたび図３のような記述が与えられるとそこから図４のようなオブジェクト間の引用関係を示すグラフを自動的に構成することが出来る。ここで矢印は、矢印の先にあるオブジェクト内のメソッドが矢印の元のオブジェクトのメンバを引用することを意味する。図４はオブジェクト内にあるデータ間の静的な参照関係とメソッド間の静的な呼び出し関係の両者を示すと考えられる。このグラフを使うとオブジェクト指向プログラムの対話実行を次のように行うことができる。画面上のオブジェクトをクリックしてメソッドを指定すると実行制御は矢印を逆にたどってそれが引用するオブジェクトを次々に呼び出しメソッドが要求する処理を行う。たとえばメッシュの変更が phi などの変数の値に及ぼす影響を知りたい時は、まずメッシュのメソッドを呼び出して変更を行ったのち、phi の値を再計算するメソッドを呼び出せば矢印を逆に辿ってメッシュのデータを引用するので変更の効果が反映される。その過程でオブジェクトの呼び出し過程をトレースして可視化すれば、再計算の詳細を知る事が出来る。オブジェクト指向の利点は常にオブジェクト間の全体的な関連を視野に入れて対話が行えることにある。

(４)　データフロー指向のアプローチ

データフロープログラムはデータの変換過程に着目した図式である。図５に本デバイスシミュレータのデータフロープログラム例を示す。図５にて長方形の箱はデータの変換モジュールであり、箱をつなぐ線はモジュール間に流れる情報を示す。実線はデータ，点線は制御情報である。箱の上ないし

側面から入るのが入力情報、下から出るのが出力情報である。変換モジュールは入力情報のいずれかが変化した時に起動される。DEQSOL 流の汎用モジュールを幾つか用意しておきそれらを使って直接にデータフロープログラムを作成する方法もあるが、FORTRAN 等で書かれた手続き型プログラムを構造化してこのような図式に組み上げることは処理の内容を理解していれば比較的容易にできる。データフロープログラムでメッシュ変更の変数値に及ぼす影響を知るには、メッシュ生成モジュールを修正したのちにそれをクリックして起動すれば良い。入力の変化したモジュールが次々に起動され再計算がなされる。ソルバモジュールが起動されることにより phi, n, p の変数値が更新される。その過程を可視化するには、変換モジュールに加えて可視化モジュールを用意しておき各々の変換モジュールの出力につなげばよい。ＡＶＳ[6]などが採っている方式である。データフロー指向の利点は、データ変換過程が一目瞭然にわかること、並列・分散処理に適合すること、データの一部分のみが変化したときに再計算が効率的に行なえることなどである。

４．プログラマの負担と応答性の見通し

上記のアプローチで示した可視化対話機能を実現するに当たって、ＰＳＥが行うべきこととプログラマが行うべきことの区分けを行い、また対話処理で重要な応答性の見通しについて考察する。

第１の手続き型言語のレベルについては、プログラムテキストからＰＳＥが自動的に対話画面を生成することはそれほど難しくない。しかし標準形式以外の表示画面を用意することはプログラマの仕事となり、ＰＳＥはそれをバインドする機能を持つにとどまるであろう。応答性は言語処理系の作りによるが、コンパイル方式を取った場合はプログラムテキストの変更に対してある程度の遅延を覚悟しなくてはならない。

第２の応用向き言語のレベルもプログラムテキストから自動的に対話画面を構成することは困難でない。応答性は処理系の作り方によるが仮にＪａｖａのような遅延バインディングが許される言語系を用いれば、方程式や境界条件などの変更に対しても高い応答性を確保できる可能性がある。しかし一方では実行時バインディングのオーバヘッドも無視できない。

第３のオブジェクト指向の場合も、プログラムテキストから図４のダイヤグラムを生成しそれに基づく対話制御を行うことはＰＳＥの仕事となる。しかしオブジェクトの設計はプログラマに委ねられているので、表示機能の準備は標準的なものを除けばプログラマの仕事である。この事情は手続き型言語の場合と同じである。変更後の応答性は、表示の対象としたデータに関わる再計算のみを行うので手続き型よりも良いことが期待される。

第４のデータフロー指向の場合は、データフロー型へのプログラムの再構成はプログラマの仕事、それに基づく実行制御はＰＳＥの仕事となる。表示画面の設計も標準的なものを除いてプログラマに委ねられる。オブジェクト指向の場合と異なり変更データが影響する全ての再計算を行うが、データの変化した部分のみで良いので、応答性は多分オブジェクト指向より良いであろう。すなわち４つの中で最も優れていることが期待される。

５．むすび

シミュレーションソフトウエアの可視化／対話化の方策を、手続き型、応用向き、オブジェクト指向、データフローの４つの言語モデルに則して検討し、それぞれの具体的なイメージを提示した。またそれらの機能を実現する時のプログラマの負担と応答性の見通しを検討した。可視化／対話の方策についてはそれぞれ長所と短所があり一概に結論づけられない。プログラマの負担については応用向きが最も軽いと予想されるが、適用範囲が限定されることと裏腹である。応答性についてはデータフローが優れていると思われるが不明な部分も多く実証が必要である。総合的な評価としては手続き型プログラムから割合スムーズに移行できてかつ、ニーズが高いことが予想されるパラメータ感度解析に優れた応答性を期待でき、またネットワークコンピューティングなど今後のシミュレーション形態にも適合するデータフロー指向の可視化対話モデルを推して、ひとまずの結論とする。

謝辞

ＭＯＳデバイスシミュレータの DEQSOL によるプログラム例を提供頂いた日立超ＬＳＩエンジニアリング株式会社の平山裕之氏、ならびにオブジェクト指向プログラム例を提供頂いた(株)日立製作所中央研究所の佐治みゆき氏に感謝の意を表します。

参考文献

1) **佐川暢俊，金野千里，梅谷征雄：数値シミュレーション言語ＤＥＱＳＯＬ，情報処理学会論文誌 30 巻 1 号，** 1989

2) J. R. Rice, R. F. Boisvert : Solving Elliptic Problems Using ELLPACK, Springer-Verlag,1984

3) S. Doi, H. Fujio, K. Sugihara : Automatic parallel program generation for finite element analysis, Quality of Numerical Software, pp.255-265, Chapman& Hall, 1997

4) T. I. Boubez, A.M. Froncioni, R.L.Peskin: A Prototyping Environment for Differential Equations, ACM Transactions on Mathematical Software, Vol. 18, No.1, pp. 1-10, 1992

5) Y. Dubois-Pelerin , T. Zimmermann : Object - oriented finite element programming : An efficient implementation in C++ , Computer Methods in Applied Mechanics and Engineering 108 , pp. 165 – 183, 1993

6) **AVS ユーザーズガイド，クボタコンピュータ株式会社，1992**

7) **村田健郎ほか３名編：工学における数値シミュレーション，pp.119 - 135 ，丸善，昭和 63 年**

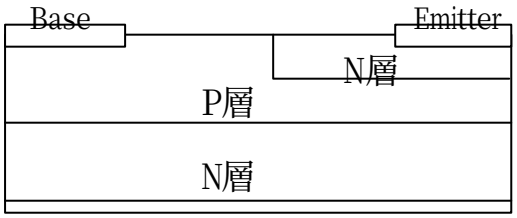
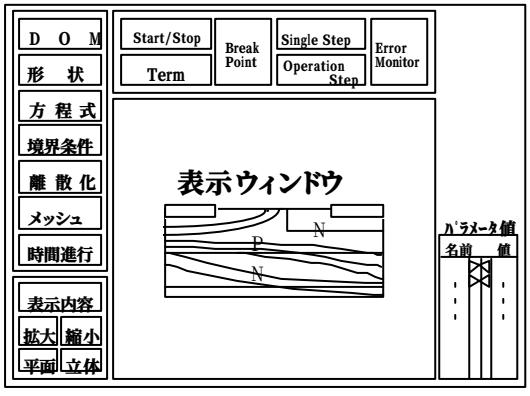


図１MOSデバイス構造模式図



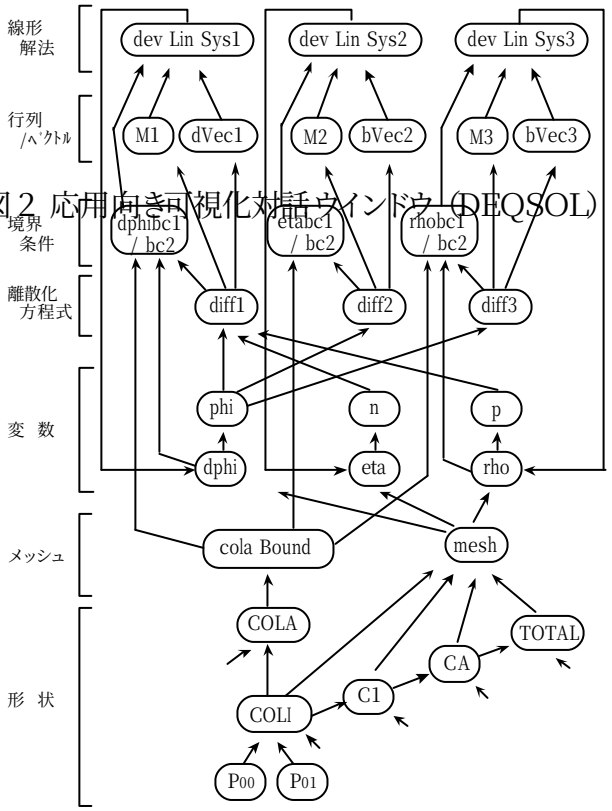


図２ 応用向き可視化対話ウインドウ（DEQSOL）

```
Void main (   ) {
(1)  Sw Point Ptr        P00 (0.0, 0.0) ;
     Sw Point Ptr        P10 (0.0, 0.0) ;
                 ・・・
     Sw Edge Ptr         COL1(P00, P11) ;
                 ・・・
     Sw Quad Ptr         C1(COL1, V10, H01, LEFT1) ;
                 ・・・
     Sw Surface Ptr      CA=C1+C2+C3 ;
                 ・・・
     Sw Surface Ptr      TOTAL = CA+BA+EA ;

(2)  Sw FEM Mesh         mesh  (TOTAL) ;
       mesh. setMesh  (COL1,5) ;
                 ・・・
       mesh. genMeshset (   ) ;
     Sw Subreg                colaBound=mesh. genBound Region (COLA) ;
                 ・・・

(3)  SwFEMVar            phi (mesh),  dphi(mesh)  ・・・
                         n(mesh),  p(mesh) ;
     double              epsd,  nrphi,  ・・・;

(4)  Sw Lin System       dev Lin Sys1,  dev Lin Sys2,  dev Lin Sys3 ;
     Sw Matrix           M1,  M2,  M3 ;
     Sw Vector           bVec1,  bVec2,  bVec3 ;
     devLin Sys1. set Matrix (M1) ;
     devLin Sys1. set RHSVec (bVec1) ;
     devLin Sys2. set Matrix (M2) ;
     devLin Sys2. set RHSVec (bVec2) ;
     devLin Sys3. set Matrix (M3) ;
       devLin Sys3. set RHSVec (bVec3) ;

(6)  While  ( epsd  > 1.0 e-3)  {
       Sw Diff1 Form          diff1  (mesh, es, n, p, br) ;
         M1=diff1. assemble (   ) ;
         bVec1=q  *(n-p-nd+na)-es  *lapl (phi) ;
(5)    Sw First BC1         dphibc1  (dphi, 0.0, COLA+EMIT+BASE)
       Sw Second BC1        dphibc2  (dphi, 0.0, LEFTA+TOPA+RIGHTA) ;
         dev Lin Sys1. apply Dirich BCs (dphibc1) ;
         dev Lin Sys1. apply Neuman BCs (dphibc2) ;
```

図３ C＋＋言語によるMOSデバイスシミュレータの記述例（一部）

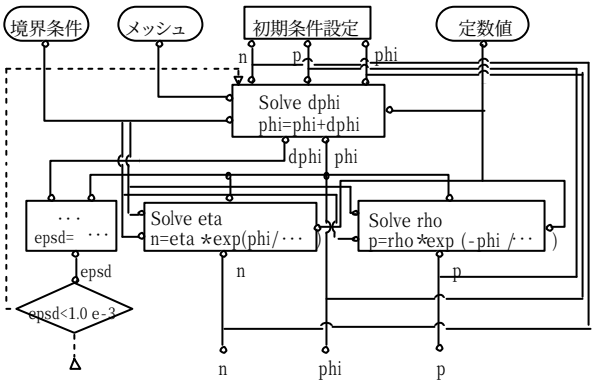
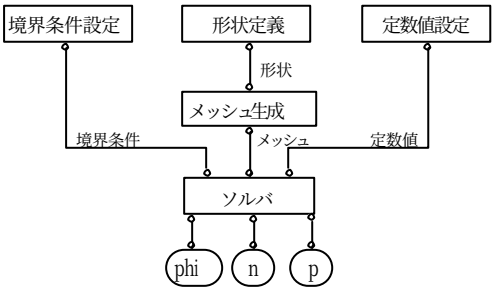


図４ オブジェクト間の引用関係



(a) 全体データフロー

(b) ソルバ部データフロー

図５ MOSデバイスシミュレータのデータフロー図式