CFD 入出力における XML 形式の利用

XML application on CFD data format

城之内忠正，四日市大，三重県四日市市萱生町 1200, jyo@yokkaichi-u.ac.jp:
Tadamasa Jyonouchi, Yokkaichi Univ., Kayou-cho 1200 Yokkaichi-shi, 511-0912 JAPAN:

XML data format in CFD IO is proposed. For simplification, and to demonstrate the availability and reuse ability, well-formed XML document is adopted to CFD input and output data, which includes document type data, for examples, purpose conclusion etc. We applied XML format to compressible flow calculation around cylinder by Cartesian grid method.

1. はじめに

CFD の入出力データを再利用し，データの相互利用を進めるためには，入出力のデータ形式を，利用する側で予め知っている必要がある．データ形式を統一すれば再利用の問題はなくなるが，様々な計算現場でのデータ形式の統一は難しい．むしろ，データの中にデータ形式の説明を書き込んでおけば，その情報から応用に適したデータ形式に変換して使うことができる．

XML はタグを使ったマークアップによってデータを記録するデータ形式であり，人間がデータの意味を理解しやすい上に，コンピュータでデータを処理できるという特徴を持っている．つまり，人間とコンピュータにとってわかりやすいデータである．この XML の特徴から，データの再利用やデータの交換に向いていることがわかる．本研究のねらいは XML 形式を CFD 入出力データに応用することによって，データの再利用を進めることにある．また，Java コンピューティングを CFD に応用する上で，ネックになっていることは，オブジェクトの保存に関連したファイル入出力である．XML の入出力への応用は，長期保存に耐えるデータを実現するものと期待される．

2. タグの定義方針と XML パーサ

XML は文書に対するマークアップ言語という側面とデータをマークアップする形式という側面がある．タグを定義する方針としては，文書アプリケーションに対しては，「タグを除いても情報が失われないこと」という方針が基本であろう．それに対して CFD の入出力で利用する場合，パラメータや計算格子等のデータを記録する手段であり，典型的なデータアプリケーションである．このような場合，XML は階層的データ構造の格納庫とみなせる．データアプリケーションとして XML を利用する場合，データをタグで囲むか，タグの属性で定義するか迷うところである．今回の応用例では，**「階層構造を（将来的にも）持たないデータは，属性とする」**というタグ定義方針を採用した．その理由は，属性のほうがデータを取り出しやすいからである．また，XML データを解釈してデータを取り出す役割は XML パーサが行う．入出力のようにデータを一度処理すればよい場合は SAX パーサを利用すれば十分である．今回は軽量の SAX パーサである MinML[1] を採用した．また，データ出力としては，オブジェクトを永続化する（ファイルに保存する）手段として XML 形式を採用している JSX[2] ライブラリを利用することにした．

3. 応用例

図 1 のような円柱周りの流れを計算する場合の入力データを例に，XML 形式のデータについて説明する．この例では直行格子法による領域分割型ソルバーであるソフト部品を使って，一様流部品に円柱部品（図 1 の中央部分）を組み込んでいる．この入力データをリスト 1 に示すが，部品の組み合わせやパラメータの他にコメントや結果・結論といった文書データも組み込めることがわかる．　そういう意味で，人間とコンピュータの両者が理解しやすいデータであろう．

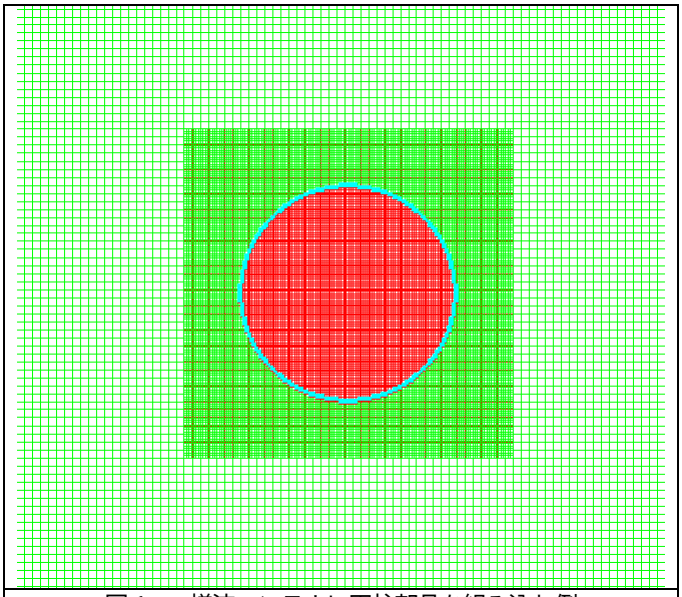

図 1　一様流コンテナに円柱部品を組み込む例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<数値実験>
<模型>
  <部品 id="0" comment="一様流コンテナ"
    geometry="CFD.webCompo.UniformFlow"
    flowSolver="CFD.webCompo.EulerSolver"
    nx="100"   ny="100" disp="#0000FF"
    width="800" height="800">
      <部品 id="1" comment="円柱コンテナ部品"
        geometry="CFD.webCompo.AroundCylinderCompo"
        flowSolver="CFD.webCompo.EulerSolver"
        nx="140"    ny="140"   disp="#FF0000"
        imin="30"   jmin="30" imax="70"jmax="70">
      </部品>
  </部品>
</模型>
<計算ケース id="2">
  <計算パラメータ   mach="1.8"   cfl="0.6"   stage="4"
                    accuracy="1"   color="red" />
  <説明>1 次精度 SFS による並列計算</説明>
  <結果>直行格子の計算の割にはなめらか</結果>
  <結論>直行格子の境界条件の与え方に工夫の余地あり</結論>
</計算ケース>
</数値実験>
```

リスト　1　　円柱周りの流れの入力データ

参考文献

(1) http://www.wilson.co.uk/xml/minml.htm
(2) http://www.csse.monash.edu.au/~ bren/JSX/

付録 A　SAX による入出力プログラム

```
import   java.awt.*;
import   java.io.*;
import   org.xml.sax.*;
import   uk.co.wilson.xml.MinML;

/** SAX による入力データの取り出し*/
public   class inputDataParser   extends   MinML
                implements      Universal{
    private     StringBuffer   sbuff;
    private     String          id = null;
    private     String          geom, flow, cl;
    private     int      nx, ny;
    private     int      imin, jmin, imax, jmax;
    private     int      width, height;
    private     DomainContainer   root, dcom;
    private     DomainComponent   com;

/** XML ファイルを開き解析を始める */
    public BatchRun parseData(String file){
        try{
            FileInputStream     fin =
                    new FileInputStream(file);
            InputStreamReader isr =
                    new InputStreamReader(fin);
            super.parse((Reader)isr);
        }catch(Exception e){
            System.out.println("err "+e);
        }
      return new BatchRun(root, width, eight);
    }
/** XML 文書の解析開始 */
    public void startDocument()
                            throws SAXException{
        sbuff = new StringBuffer();
        sbuff.setLength(0);
    }
    /** 解析終了 */
    public void endDocument()
                            throws SAXException{
        sbuff = null;
    }
    /** 入力データの要素の解析 */
    public void startElement(String name,
     AttributeList amap) throws SAXException{
        String   tmp, value, key=null;
        if(name.equals("部品")){
          for(int j=0;j<amap.getLength();j++){
            if(amap.getName(j).equals("id"))
                key=amap.getValue(j);
            }
            if(key.equals("0")){
              getAllValues(amap);
              com   = new DomainComponent();
              com.setDomainComponent(flow,
                      geom, Color.green, nx, ny);
              root = (DomainContainer)com;
            }else{
              getAllValues(amap);
              com   = new DomainComponent();
              com.setDomainComponent(flow,
                       geom, Color.green, nx, ny);
              dcom = (DomainContainer)com;
              root.addDomainContainer(dcom,
                        imin, jmin, imax, jmax);
            }
        }else if(name.equals("計算パラメータ")){
          for(int j=0;j<amap.getLength();j++){
            tmp     = amap.getName(j);
            value = amap.getValue(j);
          }
        }
    }
/** 要素の終了 */
    public void endElement(String name)
            throws SAXException{
        sbuff.setLength(0);
    }
/** 字句解析 */
    public void characters(char[] ch, int start,
              int length) throws SAXException{
        sbuff.append(ch, start, length);
    }
/** 属性値の取り出し */
private void getAllValues(AttributeList amap){
        id = geom = flow = cl = null;
        nx = ny = imin = jmin = imax = jmax = 0;
        for(int j=0; j<amap.getLength(); j++){
          if(amap.getName(j).equals("id"))              id     =
              amap.getValue(j);
           if(amap.getName(j).equals("geometry"))
            geom = amap.getValue(j);
           //以下文字情報の取り出す繰り返し
           …
          if(amap.getName(j).equals("nx"))
    nx=new Integer(amap.getValue(j)).intValue();
      //以下文字を整数で取り出す繰り返し
        …
        }
    }
}
```

この付録 A では，軽量 XML パーサ MinML を使って入力データの取り出しを行っている．SAX では開始タグに遭遇する度に startElmnet()メソッドが呼び出され，終了タグの度に endElement()メソッドが呼び出される．呼び出されるメソッドの中で必要なデータを取り出せばよいわけであり，入力データのような一度しか使わないデータの読み取りに SAX は向いている．なお，文字データから数値を取り出す部分が煩雑であるが，これは Java の癖なので我慢するしかない．

付録B　JSXによるデータの書き出し

```
//#### メッセージループ
public void menuLoop(){
  gkick.start();
  //標準文字入力ストリームの生成
  BufferedReader  lineread  =
          new BufferedReader(
         new InputStreamReader(System.in));
  String menu = "Enter command: stop,"
              +" start, save, load, exit, view";
  while(true){     //入力のループ
    System.out.println(menu);
    String buff = "";
    try{
      buff = lineread.readLine();
      if(buff.equals("stop")){
        dcom.stopFlow();   //アプレットの制御
      }else if(buff.equals("start")){
        dcom.startFlow();
      }else if(buff.equals("save")){
        dcom.stopFlow();
        System.out.println("Enter file name,");
        System.out.println("    for example: test.da");
        buff = lineread.readLine();
        FileOutputStream       fout =
                          new   FileOutputStream(buff);
        if(buff.lastIndexOf(".xml") > 0){
          // JSXでxml形式に保存する
          ObjOut out = new ObjOut(false, fout);
          out.writeObject(dcom); //書き出し
          out.flush();
        }else{//オブジェクトの直列化（バイナリで保存）
          ObjectOutputStream obfout =
                            new
                           ObjectOutputStream(fout);
          obfout.writeObject(dcom);
          obfout.flush();
        }
        fout.close();
      }else if(buff.equals("load")){
        dcom.stopFlow();
        System.out.println("Enter file name,");
        System.out.println("    for example: test.da");
        buff = lineread.readLine();
        FileInputStream    fin =
                           new FileInputStream(buff);
        //JSXを使ってXMLファイルからロード
        if(buff.lastIndexOf(".xml") > 0){
          ObjIn in = new ObjIn(fin); //入力ストリーム
          dcom = (DomainContainer)in.readObject();
                    //読み込みの時にキャストが必要
        }else{//オブジェクトの直列化から読みこむ
          ObjectInputStream obfin =
                              new ObjectInputStream(fin);
          dcom = (DomainContainer)obfin.readObject();
        }
        fin.close();
        dcom.startFlow();
      }else if(buff.equals("exit")){
        dcom.stopFlow();
        System.exit(0);
      }else if(buff.equals("view")){
        dcom.stopFlow();
          System.out.println("Enter grid or contour");
          buff = lineread.readLine();
          if(buff.indexOf("g") >= 0){//計算格子
            canvas.setDrawProperty(gridTag, null);
          }else{
            canvas.setDrawProperty(contourTag,
                                   pressureTag);
          }
            dcom.startFlow();
          }else{
            System.out.println("Input error");
          }
        }catch(Exception e){
          System.out.println("Exception: "+e);
        }
    }
```

付録 B は，オブジェクトという構造化されたデータの入出力部分のプログラムコードである．複雑な入れ子関係にある流れの特性データ等が，一命令でファイルに保存することが出来る．従来，パラメータや流れ場の特性量を個々に記録していた方法に比べると簡単になっている．

ただし，注意しなければいけないのは，オブジェクトの直列化を利用して保存すると，バイナリファイルとして保存されてしまうことである．オブジェクトの直列化によるバイナリファイルは，大きさが小さくなる点などよい面もあるが，２つの点で問題が発生する．ひとつはバイナリなので生成したプログラムでしか読めないこと，もうひとつはプログラムを修正しオブジェクトの構成を変更すると以前作ったファイルが読み込めなくなってしまう点である．この問題は XML を使えば対処できる．テキストデータはエディタで読めばいつでも読めるし，プログラムの修正によるデータ構造の変化は XSLT 等を使ったデータ変換を行えばよい．

しかしながら，テキストデータになるため，データ量は数倍に増えてしまうため用途に応じて使う方が賢明であろう．