Polymorphic Time Domain Computational Electromagnetics

Poman P.M. So

University of Victoria, Victoria, BC, Canada

***Abstract* —** CEM methods such as FDTD and TLM are the de-facto standard for general purpose EM field modeling in the time domain. On the other hand, the Microsoft .NET Framework and the associated C# programming language have become the de-facto standard for software development on Windows. This paper presents the technique for building a time domain CEM object library in C#. This approach could be the basis for creating an open-source standard CEM library.

***Index Terms* —** CEM, FDTD, TLM, time domain analysis, object-oriented approach.

I. INTRODUCTION

Traditional computational electromagnetics (CEM) research does not place sufficient emphasis on object-oriented design and implementation. Classic and recent CEM books [1] – [8] do not address the importance of object orientation at all. As a result, software packages developed by CEM practitioners usually cannot be maintained outside of their respective research institutions. It is hard to imagine engineers around the world would have to re-invent the basic string and math functions before they could start writing codes to solve their design problems. However, when it comes to developing new programs for CEM applications most CEM practitioners have to start from scratch because there is no standard CEM library at their disposal. Using free packages such as NEC [9], TLM3D [10], YatPac [11], and MEEP [12] to solve EM problems is one thing; building new programs base on these packages is a completely different challenge.

In the author's opinion, general purpose CEM methods such as MOM, FEM, FDTD and TLM are mature enough to be placed in an open source standard CEM library. The existence of such a library would not pose unwanted competition to the CEM software industry because the role of CEM industry should be in optimizing the well known modeling methods with proprietary features, in customizing the software with industrial strength graphical user interface front-end, and in interconnecting the field modeling engines to CAD/CAE packages.

Object-oriented paradigm is the key for implementing a standard CEM library. However, Object orientation is not equivalent to programming in Java, C++, and C#. In fact, it is not difficult to find procedure-oriented spaghetti code written in these languages. A truly object-oriented program makes good use of encapsulation, inheritance, and polymorphism. Stroustrup discusses the concept in great details in his authoritative C++ book [13].

The author has illustrated the advantages of an OOP CEM framework in an earlier paper [14]; in order to build a standard CEM library in a reasonably short period of time, existing procedure-oriented CEM codes should be leveraged as much as possible. This paper thus spells out the details of converting a procedure-oriented program to an object-oriented implementation. Since the Microsoft .NET Framework and its associated C# programming language have become the de-facto standard for the Windows software industry, this paper makes use of the C# programming language to apply the OOP techniques to computational electromagnetics.

II. IMPLEMENTATION OF TLM IN C#

The theory of TLM is well described in the literature [1], [4], [15] and [16]. Procedure-oriented implementation of the method can be found in [1] and [10]; a package written in C/C++ has been recently released by Russer *et al*. [11]. These TLM source codes are invaluable resources for CEM researchers who are interested in the TLM method. However, these computer codes are based on legacy modules that are not object-oriented. To illustrate the idea of object-oriented implementation, this paper describes the software technology for converting the TLM_INHO Pascal program in [1] to a reusable class object in C#; the source codes presented in this paper can be downloaded at the CERL website [17].

C# is not the only programming language that is suitable for implementing polymorphic CEM programs. Many CEM professionals may prefer C++ and Java to C# because of the maturity of the two older languages as well as the general availability of third party numerical libraries [18] and [19]. C# is used in this paper because it supports multi-dimensional arrays in a way that is similar to Pascal and FORTRAN. Furthermore, C# supports C++ style operator overloading which is a crucial feature for implementing a complex mathematic library. Finally, C# can leverage the computing power of library modules written in C/C++, Pascal and FORTRAN via the .NET Framework InteropServices. Hence, C# is a serious programming language that CEM researchers may not want to ignore.

The first step to convert the TLM_INHO.PAS program to C# is to map the Pascal data types to the equivalent C# data objects. Pascal's numeric data types such as `integer` and `single` can be translated in a straightforward manner to the C#'s `int` and `float` data types. In both languages, double precision floating point numbers are called `double`.

```
{------- Pascal Code Segment -------}
nx, ny : integer;  {mesh dimension  }
d1, d2 : single;   {normalized freq.}
header : string[80]; {temp. storage }

//-------- C# Code Segment ----------
int nx, ny;      // mesh dimension
float  d1, d2;   //normalized freq
string header;   //temporary storage
```

Listing 1. Code segment using simple data types to illustrate the equivalence of data types between Pascal and C#.

The string types between these two languages are quite different. In Pascal, string is an array of characters; the characters in a string can be manipulated at run-time. In C#, string is an immutable built-in type; a new string must be created if any characters in the old string are to be changed. In addition to that, the C# string is a reference type. When the value of a string variable is assigned to another string variable, only the reference to the string is assigned to the new variable; both variables will refer to the same character string. The code segments in Listing 1 and 2 illustrate the said concepts. When the content of a string is no longer referred by a string variable, the memory location occupied by the defunct string will be recovered by the .NET Framework's garbage collection utility. This garbage collection concept applies to all built-in and user defined reference data types.

Arrays in Pascal and C# are quite similar but the differences have to be noted. In Pascal, array lower bounds can be easily specified. In C#, arrays have a default lower bound of zero; arrays with user specified lower bounds can be created with the static `CreateInstance` class method. In the original TLM_INHO.PAS program, the three-dimensional array for storing voltage impulses is declared as:

```
  v:array[1..5,1..12,1..12] of single;
```

A three-dimensional C# array that allows indices to span through [1..5, 1..12, 1..12] would be:

```
  float[,,] v = new float[6,13,13];
```

The above three-dimensional C# array in fact has 1×13×13 extra entries. To avoid wasting storage, one may use the following statement:

```
float[,,] v = (float[,,])
        Array.CreateInstance(
        typeof(float),
        new int[] { 5,12,12 },
        new int[] { 1, 1, 1 });
```

```
string a, b; // string variables
a = "abc";// assign address of "abc"
b = a;    // to a and b. a and b now
          // refer to the same string
a = "def" // a refer to a new string.
b = a;    // a and b now refer to
          // "def", "abc" becomes
          // inaccessible.
```

Listing 2. Code segment illustrates the reference characteristic of the C# string; the unreferenced `"abc"` string will eventually be garbage collected by the .NET Framework.

Instead of typing the declaration statement above repetitively in a source file, one may define the following generic static method to create three dimensional arrays:

```
public static T[,,] Array3D<T>
       (int x1, int x2, int y1,
       int y2, int z1, int z2){
int[] dim={x2-x1+1, y2-y1+1,z2-z1+1};
int[] lower={x1, y1, z1};
return (T[,,])Array.CreateInstance(
             typeof(T), dim, lower);
}
```

Using this generic method, a three dimensional array of storage for the voltage impulses can be created via the following simple statement:

```
   float[,,] v =
   Array3D<float>(1,5,1,12,1,12);
```

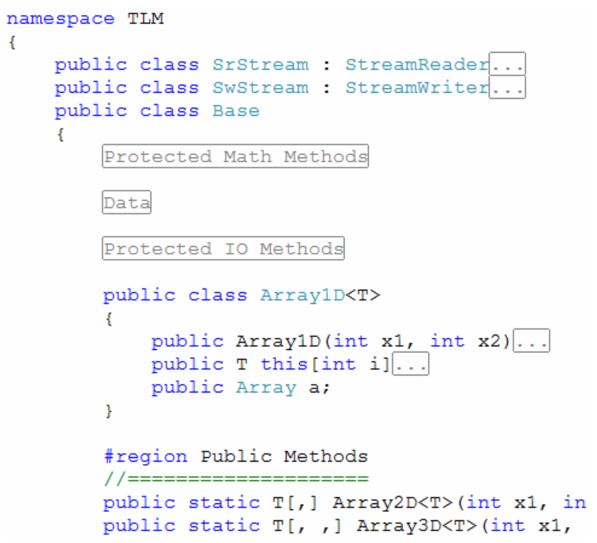
The same technique can be employed to create two-dimensional arrays of any type; the generic method in this case would be:

```
public static T[,] Array2D<T>
(int x1, int x2, int y1, int y2){
 int[] dim ={ x2-x1+1, y2-y1+1};
 int[] lower ={ x1, y1 };
return (T[,])Array.CreateInstance(
              typeof(T), dim, lower);
}
```

Hence, non-zero lower bound two-dimensional arrays of type `char`, `int` and `float` can be declared like:

```
char[,] a =
Array2D<char>(1,5,1,120);
int[,]  b =
Array2D<int>(1,5,1,120);
float[,]c =
Array2D<float>(1,5,1,120);
```

One would expect this technique to work for one-dimensional arrays as well. However, the version 2.0 C# compiler does not allow one-dimensional `System.Array` to be typecast to `T[]`. This compiler deficiency, or bug, can be overcome by using the following simple `Array1D<T>` class.



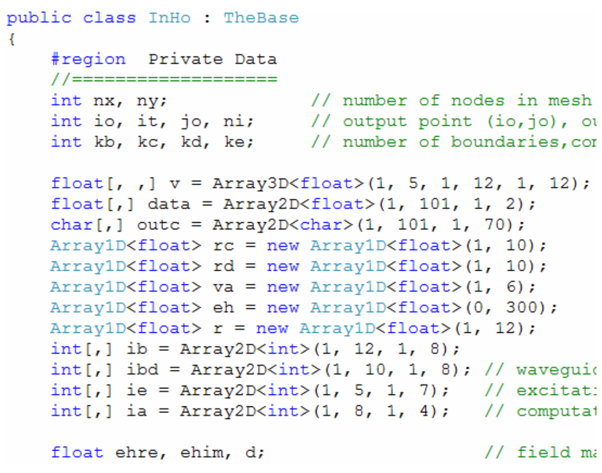
```
namespace TLM
{
    public class SrStream : StreamReader...
    public class SwStream : StreamWriter...
    public class Base
    {
        Protected Math Methods

        Data

        Protected IO Methods

        public class Array1D<T>
        {
            public Array1D(int x1, int x2)...
            public T this[int i]...
            public Array a;
        }

        #region Public Methods
        //=====================
        public static T[,] Array2D<T>(int x1, in
        public static T[, ,] Array3D<T>(int x1,
```

Fig.1. Helper classes for a new TLM_INHO object.

```
public class Array1D<T>{
  public Array1D(int x1, int x2){
    int[] dim ={ x2 - x1 + 1};
    int[] lower ={ x1};
    a = Array.CreateInstance(
        typeof(T),dim, lower);
  }
    public T this[int i]{
      get { return (T)a.GetValue(i);
  }
      set { a.SetValue(value,i); }
    }
    public Array a;
  }
```

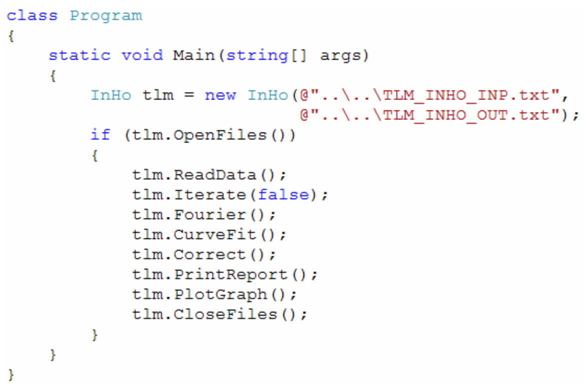
Besides demonstrating C#'s generic class definition, this implementation also illustrates the use of C#'s indexer, `[]`, as well as the accessor (`get`) and mutator (`set`) property methods. With these array creation methods and the `Array1D<T>` class, it is straightforward to transform the TLM_INHO Pascal program to a C# program — all data and procedures crucial to the TLM simulation are placed inside a main C# class; supporting data and utility functions are placed in other general purpose helper classes. A screen shot of the module that contains the immediate helper classes for a new TLM_INHO object is shown in Figure 1. `SrStream` and `SwStream` are text based file IO classes whereas the `TheBase` consists of the previously mentioned static array creation methods and the `Array1D<T>` helper class.

Figure 2 shows the private variables of an inhomogeneous medium TLM class, `InHo`. Since `InHo` is derived from `TheBase`, `InHo` inherits all the data and methods of `TheBase`. As a result, `InHo` can make use of the array creation methods and the one-dimensional array class, `TheBase`, without using the `<class>.<method>` notation. Besides the private data shown in the figure, `InHo` has a number of public methods for data I/O, field simulation, and data processing. Figure 3 shows a main program that makes use of the `InHo` class to implement a single-thread TLM simulation algorithm identical to the original



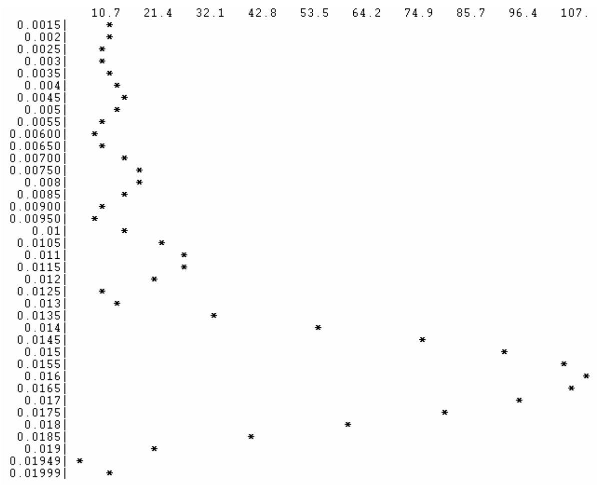
```
public class InHo : TheBase
{
    #region  Private Data
    //====================
    int nx, ny;              // number of nodes in mesh
    int io, it, jo, ni;      // output point (io,jo), ou
    int kb, kc, kd, ke;      // number of boundaries,cor

    float[, ,] v = Array3D<float>(1, 5, 1, 12, 1, 12);
    float[,] data = Array2D<float>(1, 101, 1, 2);
    char[,] outc = Array2D<char>(1, 101, 1, 70);
    Array1D<float> rc = new Array1D<float>(1, 10);
    Array1D<float> rd = new Array1D<float>(1, 10);
    Array1D<float> va = new Array1D<float>(1, 6);
    Array1D<float> eh = new Array1D<float>(0, 300);
    Array1D<float> r = new Array1D<float>(1, 12);
    int[,] ib = Array2D<int>(1, 12, 1, 8);
    int[,] ibd = Array2D<int>(1, 10, 1, 8); // waveguic
    int[,] ie = Array2D<int>(1, 5, 1, 7);   // excitat:
    int[,] ia = Array2D<int>(1, 8, 1, 4);   // computat

    float ehre, ehim, d;                    // field ma
```

Fig.2. The class structure of TLM_IHNO in C#.



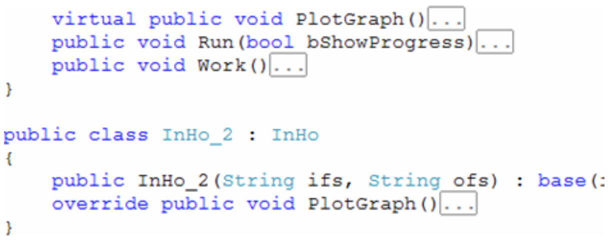
```
class Program
{
    static void Main(string[] args)
    {
        InHo tlm = new InHo(@"..\..\TLM_INHO_INP.txt",
                            @"..\..\TLM_INHO_OUT.txt");
        if (tlm.OpenFiles())
        {
            tlm.ReadData();
            tlm.Iterate(false);
            tlm.Fourier();
            tlm.CurveFit();
            tlm.Correct();
            tlm.PrintReport();
            tlm.PlotGraph();
            tlm.CloseFiles();
        }
    }
}
```

Fig.3. A C# program that employs the TLM_INHO class library.



```
           10.7   21.4   32.1   42.8   53.5   64.2   74.9   85.7   96.4   107.
 0.0015|     *
  0.002|     *
 0.0025|    *
  0.003|    *
 0.0035|     *
  0.004|      *
 0.0045|       *
  0.005|      *
 0.0055|    *
0.00600|   *
0.00650|    *
0.00700|       *
0.00750|         *
  0.008|         *
 0.0085|       *
0.00900|    *
0.00950|   *
   0.01|       *
 0.0105|            *
  0.011|               *
 0.0115|               *
  0.012|           *
 0.0125|    *
  0.013|      *
 0.0135|                   *
  0.014|                                 *
 0.0145|                                               *
  0.015|                                                          *
 0.0155|                                                                  *
  0.016|                                                                     *
 0.0165|                                                                   *
  0.017|                                                            *
 0.0175|                                                  *
  0.018|                                     *
 0.0185|                        *
  0.019|           *
0.01949| *
0.01999|     *
```

Fig.4. The output of the TLM program in figure 3.



```
    virtual public void PlotGraph()...
    public void Run(bool bShowProgress)...
    public void Work()...
}

public class InHo_2 : InHo
{
    public InHo_2(String ifs, String ofs) : base(:
    override public void PlotGraph()...
}
```

Fig.5. InHo_2, a new class derived from InHo.

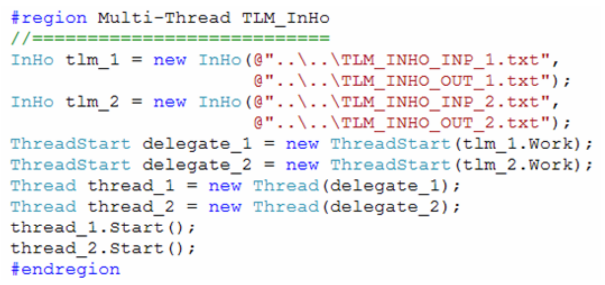
Pascal program. The false argument passed to the Iterate method informs the tlm object it is not necessary to send output to the console during the field simulation process. The original Output (renamed PlotGraph) procedure has been made virtual to support polymorphism.

The new program has been used to analysis the rectangular resonator associated with the original Pascal TLM_INHO program. The output from the current C# implementation, which is identical to the output of the original program, is shown in Figure 4.

The above computation has validated that the Program, InHo, and TheBase classes have been properly implemented in C# using the object-oriented paradigm. The helper classes and array creation methods are general purpose utilities for converting codes in other programming languages to C#.

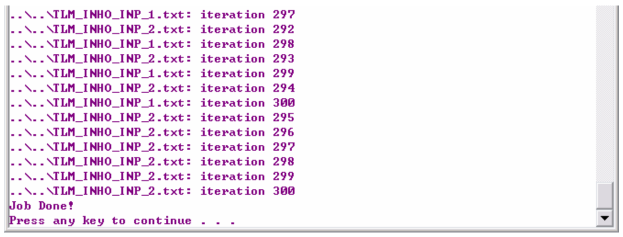
One of the advantages of object-oriented implementation is the ease of leveraging computing power of existing software modules via object inheritance and polymorphism. If the PlotGraph method of InHo is made virtual, a new class, say InHo_2, can be derived from InHo with a new pixel-based graphical PlotGraph method that overrides the original string-based implementation. The code structure of such an implementation is shown in Figure 5. Inheritance and polymorphism are powerful features that do not exist in the traditional procedure-oriented programming paradigm.

Another advantage of object-oriented implementation is the feasibility of instantiating multiple copies of the InHo objects in the Main method. A multiple-engine TLM program can be easily created using the InHo class library; Figure 6 depicts the code segment of a dual-engine simulation program. On a multi-processor computer, the engines will run simultaneously. In order to illustrate the concurrent behaviour of the dual-engine implementation, a true argument is passed to the iterate method so that the tlm objects would print the name of the input file and the iteration number to the console during the field computation process. The screen image in Figure 7 shows interspersed outputs from the two concurrent threads in Figure 6.



```
#region Multi-Thread TLM_InHo
//===========================
InHo tlm_1 = new InHo(@"..\..\TLM_INHO_INP_1.txt",
                      @"..\..\TLM_INHO_OUT_1.txt");
InHo tlm_2 = new InHo(@"..\..\TLM_INHO_INP_2.txt",
                      @"..\..\TLM_INHO_OUT_2.txt");
ThreadStart delegate_1 = new ThreadStart(tlm_1.Work);
ThreadStart delegate_2 = new ThreadStart(tlm_2.Work);
Thread thread_1 = new Thread(delegate_1);
Thread thread_2 = new Thread(delegate_2);
thread_1.Start();
thread_2.Start();
#endregion
```

Fig.6. A multi-thread TLM_INHO code segment in C#.



```
..\..\TLM_INHO_INP_1.txt: iteration 297
..\..\TLM_INHO_INP_2.txt: iteration 292
..\..\TLM_INHO_INP_1.txt: iteration 298
..\..\TLM_INHO_INP_2.txt: iteration 293
..\..\TLM_INHO_INP_1.txt: iteration 299
..\..\TLM_INHO_INP_2.txt: iteration 294
..\..\TLM_INHO_INP_1.txt: iteration 300
..\..\TLM_INHO_INP_2.txt: iteration 295
..\..\TLM_INHO_INP_2.txt: iteration 296
..\..\TLM_INHO_INP_2.txt: iteration 297
..\..\TLM_INHO_INP_2.txt: iteration 298
..\..\TLM_INHO_INP_2.txt: iteration 299
..\..\TLM_INHO_INP_2.txt: iteration 300
Job Done!
Press any key to continue . . .
```

Fig.7. A screen image consists of interspersed outputs from the two execution threads in Fig 5.

Recently released CEM codes, [9] and [12], are mostly implemented in C++. However, as the author has mentioned earlier, programs implemented in C++ are not necessarily object-oriented programs. Programs that do not take advantage of object inheritance and polymorphism are merely class oriented programs. More applications of object-oriented programming paradigm to computational electromagnetics can be found in [20]-[25]. Details about polymorphism and concurrent programming in C# can be found in [26] and [27].

III. CONCLUSION

A C# implementation of the classical two-dimensional TLM algorithm by Hoefer has been presented. Object-oriented features such as encapsulation, inheritance and polymorphism have been demonstrated. The object-oriented paradigm presented in this paper can be used to convert most legacy procedure-oriented CEM programs to modern object-oriented library modules. The technique presented in this paper is equally applicable in the Java environment, or in the C++ world with other operating systems. The author is advocating placing commonly used CEM engines, such as MoM, FEM, FDTD and TLM, in an open source standard CEM library to benefit the EM community at large.

ACKNOWLEDGEMENT

The author wishes to acknowledge the financial support of the National Science and Engineering Research Council of Canada.

REFERENCES

[1] W. J. R. Hoefer (editor T. Itoh), *Numerical Techniques for Microwave and Millimeter-Wave Passive Structures*, Chapter 8, John Wiley & Sons, 1989.

[2] P.P. Silvester and R.L. Ferrari, *Finite Elements for Electrical Engineers*, Second Edition, Cambridge University Press, 1990.

[3] R.C. Booton Jr., *Computational Methods for Electromagnetics and Microwaves*, John Wiley & Sons, 1992.

[4] C. Christopoulos, *The Transmission-Line Modeling Method*, IEEE Press / Oxford University Press, 1995.

[5] T. Itoh, C. Pelosi, and P.P. Silvester, *Finite Element Software for Microwave Engineering*, John Wiely & Sons, 1996.

[6] M.N.O. Sadiku, *Numerical Techniques in Electromagnetics*, Second Edition, CRC Press, 2001.

[7] D.B. Davidson, *Computational Electromagnetics for RF and Microwave Engineering*, Cambridge University Press, 2005.

[8] A. Taflove and S.C. Hagness, *Computational Electrodynamics, The Finite-Differences Time-Domain Method*, Third Edition, Artech House, 2005.

[9] NEC, http://www.si-list.org/swindex2.html.

[10] W.J.R. Hoefer, http://www.cerl.ece.uvic.ca/wjrh/tlm/prog-fortran.html.

[11] P. Russer, http://www.hft.ei.tum.de/php/resYATSIM2.php and http://www.yatpac.org/index.php.

[12] MEEP, http://ab-initio.mit.edu/wiki/index.php/Meep.

[13] B. Stroustrup, *The C++ Programming Language*, Special Edition, Addison-Wesley, 2000

[14] P.P.M. So, "An Object-oriented Framework for Computational Electromagnetics," 22nd *Annual Review of Progress in Applied Computational Electromagnetics*, pp. 219–224, March 12-16, 2006.

[15] P. B. Johns and R. L. Beurle, "Numerical solution of two-dimensional scattering problems using a transmission-line matrix," *Proc. IEE*, Vol. 118, No. 12, pp. 1203-1208, 1971.

[16] W. J. R. Hoefer, "Time domain electromagnetic simulation for microwave CAD applications," *IEEE Trans. MTT*, Vol. 40, No. 7, pp. 1517-1527, 1992.

[17] CERL at UVic, www.cerl.ece.uvic.ca

[18] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C++, The Art of Scientific Computing*, Second Edition, Cambridge University Press, 2002.

[19] JMSL, http://www.vni.com/products/imsl/jmsl/jmsl.html.

[20] D. Kurumbalapitiya and S.R.H. Hoole, "An Object-oriented Representation of Electromagnetic Knowledge," *IEEE Trans. On Magnetics*, Vol. 29, No. 2, pp. 1939–1942, March 1993.

[21] E.J. Silva and R.C. Mesquita, "Data Management in Finite Element Analysis Programs Using Object-oriented Techniques," *IEEE Trans. on Magnetics*, Vol. 32, No. 3, pp. 445–1448,May 1996.

[22] E.Z. Zhou, “Object-oriented Programming, C++ and Power System Simulation,” *IEEE Trans. on Power Systems*, Vol. 11, No. 1, pp. 206 – 215, February 1996.

[23] L. Baduel, F. Baude, D. Caromel, C. Delbe, N. Gama, S. El Kasmi, and S. Lanteri, “A Parallel Object-oriented Application for 3D Electromagnetism,” *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS’04).

[24] C. G. Biniaris, A. I. Kostaridis, D. I. Kaklamani, and I. S. Venieris, “Implementing Distributed FDTD Codes with Java Mobile Agents,” *IEEE Trans. Antenna and Propagation Magazine*, Vol. 44, No. 6, pp. 115 – 119, December 2002.

[25] D. G. Lymperopoulos, D. Logothetis, P. Atlamazoglou, and D. I. Kakalamani, “Using Object-oriented and Literate-Programming Techniques for the Development of a Computational Electromagnetics Library,” *IEEE Antennas and Propagation Magazine*, Vol. 47, No. 3, pp. 31–38, June 2005.

[26] A. Troelsen, *Pro C# 2005 and the .NET 2.0 Platform*, APress, 2005.

[27] J. Richter, *CLR Via C#*, 2ED, Microsoft Press, 2006.

Poman P. M. So received the B.Sc. degree in Computer Science and Physics from the University of Toronto, Canada, in 1985. He received his degrees in Electrical Engineering from the University of Ottawa (B.A.Sc. and M.A.Sc.) and the University of Victoria (Ph.D.) in 1987, 1989 and 1996, respectively.

From April 1997 to June 1998, he was a senior antenna engineer at MDS Space Mission (Ste-Anne-de-Bellevue, Quebec, Canada), formerly known as Spar Aerospace Ltd. His work included high frequency (10 to 40 GHz) antennas and feed components design for commercial satellite systems as well as Ka-band active antenna CAD software development. In October 1993, he was invited to the Ferdinand-Braun-Institut für Höchstfrequenztechnik Berlin, Berlin, Germany, as a research scientist. From August 1990 to February 1991, he was a visiting researcher at the University of Rome in Rome, Italy, and the Laboratoire d’Electronique in Sophia Antipolis, France.

Dr. Poman So joined the Department of Electrical and Computer Engineering at the University of Victoria as an Assistant Professor in 2005. He is a Senior Member of IEEE and a registered Professional Engineer in the Province of British Columbia, Canada. He has twenty years of research and industrial experience in object-oriented computational electromagnetics. He is a co-founder of the Faustus Scientific Corporation and is the creator of the company’s MEFiSTo line of electromagnetic modeling software.

IV. APPENDIX

THEBASE.CS: AN OBJECT-ORIENTED TLM LIBRARY IN C#

```
using System;          using System.Collections.Generic;
using System.Text;     using System.IO;
namespace TLM{
public class SrStream : StreamReader{
    public SrStream(String ifs) : base(ifs){}
    public String ReadString(){
        char a = ' ';   do{ a = (char)Read();}   while (char.IsWhiteSpace(a));
        string buffer = "";
        do{ buffer += a;  a = (char)Read();} while (!char.IsWhiteSpace(a));
        return buffer;
    }
    public float ReadFloat() { return float.Parse(ReadString()); }
    public int ReadInt() { return int.Parse(ReadString()); }
}
public class SwStream : StreamWriter{
    public SwStream(String ofs) : base(ofs){ }
    public string Format<T>(T v, int pl){
        string str = v.ToString().PadLeft(pl, ' ');
        return str.Substring(0, pl);
}   }
public class TheBase{

    // Protected Math Methods
    //=======================
    protected double atan(double v) { return Math.Atan(v); }
    protected double exp(double v) { return Math.Exp(v); }
    protected double cos(double v) { return Math.Cos(v); }
    protected double sin(double v) { return Math.Sin(v); }
    protected double sqrt(double v) { return Math.Sqrt(v); }
    protected double floor(double v) { return Math.Floor(v); }
    // Data
    //=====
    public SrStream sr;
    public SwStream sw;
    public String input_file, output_file;
    // Protected IO Methods
    //=====================
    protected string Format<T>(T v, int pl) { return sw.Format<T>(v, pl); }
    protected float ReadFloat() { return sr.ReadFloat(); }
    protected int ReadInt() { return sr.ReadInt(); }
    public class Array1D<T>{
        public Array1D(int x1, int x2){
            int[] dim ={ x2 - x1 + 1 };   int[] lower ={ x1 };
            a = Array.CreateInstance(typeof(T), dim, lower);
        }
        public T this[int i]{
            get { return (T)a.GetValue(i); }
            set { a.SetValue(value, i); }
        }
        public Array a;
    }
    // Public Methods
    //===============
    public TheBase(String ifs, String ofs){
        input_file = ifs;
```

```
            output_file= ofs;
        }
        public bool OpenFiles(){
            try{
                sr = new SrStream(input_file);
                sw = new SwStream(output_file);
                return true;
            }
            catch (Exception e){
                Console.WriteLine("An error occurred: '{0}'", e);
                CloseFiles();   return false;
        }   }
        public void CloseFiles(){
            if (sr != null) sr.Close();
            if (sw != null) sw.Close();
        }
        public static T[,] Array2D<T>(int x1, int x2, int y1, int y2){
            int[] dim ={ x2 - x1 + 1, y2 - y1 + 1 };
            int[] lower ={ x1, y1 };
            return (T[,])Array.CreateInstance(typeof(T), dim, lower);
        }
        public static T[,,] Array3D<T>(int x1,int x2,int y1,int y2,int z1,int z2){
            int[] dim ={ x2 - x1 + 1, y2 - y1 + 1, z2 - z1 + 1 };
            int[] lower ={ x1, y1, z1 };
            return (T[, ,])Array.CreateInstance(typeof(T), dim, lower);
    }   }   }
```

TLM_INHO.CS: An object-oriented TLM Library in C#

```
using System;
using System.Text;
using System.Threading;
using System.Collections.Generic;
using System.IO;

namespace TLM{
public class InHo : TheBase{
    int nx, ny;          // number of nodes in mesh
    int io, it, jo, ni; // output point (io,jo), output type & num of iters
    int kb, kc, kd, ke; // number of boundaries,computational boxes,dielectric
                        // boundaries & excitation points or lines
    float[, ,] v = Array3D<float>(1, 5, 1, 12, 1, 12);  // voltage buffer
    float[,] data = Array2D<float>(1, 101, 1, 2);
    char[,] outc = Array2D<char>(1, 101, 1, 70);
    Array1D<float> rc = new Array1D<float>(1, 10);  // reflection coef
    Array1D<float> rd = new Array1D<float>(1, 10);  // relative permittivity
    Array1D<float> va = new Array1D<float>(1, 6);   // initial values
    Array1D<float> eh = new Array1D<float>(0, 300); // storage for results
    Array1D<float> r = new Array1D<float>(1, 12);
    int[,] ib = Array2D<int>(1, 12, 1, 8);
    int[,] ibd = Array2D<int>(1, 10, 1, 8); // waveguide, boundaries & codes
    int[,] ie = Array2D<int>(1, 5, 1, 7);   // excitation points and code (115)
    int[,] ia = Array2D<int>(1, 8, 1, 4);   // computation boxes
    float ehre, ehim, d;  // field magnitudes and normalized frequencies
    float pcf, cf, d1, d2, ds;        // normalized frequencies & step size
    float peak, a, cs, max, yo;
    int npt, l, j, m, i, ic, pt, ptp, ptm, nn;  // iteration counters}

    void ReadHeader(){ String header;
        header = sr.ReadLine(); sw.WriteLine(header);
        header = sr.ReadLine(); sw.WriteLine(header);
    }
```

```
void ReadNxNy(){ ReadHeader();  nx = sr.ReadInt();
    ny = sr.ReadInt();        sr.ReadLine();
    sw.WriteLine("{0} {1}", Format<int>(nx, 4), Format<int>(ny, 4));
}
void ReadBound(){  ReadHeader();  kb = 0;
    do{ kb = kb + 1;
        for (m = 1; m <= 8; m++){ ib[kb, m] = ReadInt();
            sw.Write("{0}", ib[kb, m].ToString().PadLeft(4, ' '));
            if (m == 4) sw.Write("    ");
        }
        r[kb] = ReadFloat();    it = ReadInt();      sr.ReadLine();
        sw.WriteLine("{0}{1}",Format<float>(r[kb],16),Format<int>(it, 10));
    } while (it > 0);
}
void ReadDielBound(){ ReadHeader();    kd = 0;
    do{ kd = kd + 1;
        for (m = 1; m <= 8; m++){  ibd[kd, m] = ReadInt();
            sw.Write("{0}", Format<int>(ibd[kd, m], 4));
            if (m == 4) sw.Write("    ");
        }
        rc[kd] = ReadFloat();      it = ReadInt();        sr.ReadLine();
        sw.WriteLine("{0}{1}",Format<float>(rc[kd],18),Format<int>(it,10));
    } while (it > 0);
}
void ReadCompBox(){ ReadHeader();    kc = 0;
    do{ kc = kc + 1;
        for (m = 1; m <= 4; m++){  ia[kc, m] = ReadInt();
            sw.Write("{0}", Format<int>(ia[kc, m], 4));
        }
        rd[kc] = ReadFloat();    it = ReadInt();      sr.ReadLine();
        sw.WriteLine("{0}{1}",Format<float>(rd[kc],22),Format<int>(it,22));
    } while (it > 0);
}
void ReadExcitation(){  ReadHeader();    ke = 0;
    do{ ke = ke + 1;
        for (m = 1; m <= 7; m++){ ie[ke, m] = ReadInt();
            sw.Write("{0}", Format<int>(ie[ke, m], 4));
            if (m == 4) sw.Write("  ");
        }
        va[ke] = ReadFloat();  it = ReadInt();     sr.ReadLine();
        sw.WriteLine("{0}{1}",Format<float>(va[ke],16),Format<int>(it,17));
    } while (it > 0);
}
void ReadFreq(){
    sr.ReadLine();        sr.ReadLine();        d1 = ReadFloat();
    d2 = ReadFloat();     ds = ReadFloat();     sr.ReadLine();
    sr.ReadLine();        sr.ReadLine();        io = ReadInt();
    jo = ReadInt();       l = ReadInt();        ni = ReadInt();
    yo = ReadFloat();     sr.ReadLine();
    sw.WriteLine("Output point is ({0},{1})",
        Format<int>(io,4),Format<int>(jo, 4));
    sw.WriteLine("Number of iterations is {0}", ni);
    sw.WriteLine("Permittivity stub admittance is {0}", yo);
    sw.WriteLine("   D1           D2          Step Size");
    sw.WriteLine("{0}{1}{2}", Format<float>(d1, 8), Format<float>(d2, 8),
                              Format<float>(ds, 18));
}
public InHo(String ifs, String ofs) : base(ifs, ofs) { }
public void ReadData(){
    ReadNxNy();           ReadBound();             ReadDielBound();
```

```
        ReadCompBox();        ReadExcitation();        ReadFreq();
    }
    public void Iterate(bool bShowProgress){
        if (bShowProgress){
            Console.WriteLine("Finished reading input.");
        }
        float a, vx, vy, vxy;
        // CLEAR WORKING SPACE
        for (j = 1; j <= ny; j++){
            for (i = 1; i <= nx; i++){
                for (m = 1; m <= 5; m++){
                    v[m, i, j] = 0;
        }   }   }
        // INITIALIZE EXCITATION POINTS
        for (nn = 1; nn <= ke; nn++){
            for (j = ie[nn, 3]; j <= ie[nn, 4]; j++){
                for (i = ie[nn, 1]; i <= ie[nn, 2]; i++){
                    m = ie[nn, 5];
                    while (m <= ie[nn, 7]){
                        v[m, i, j] = va[nn];      m = m + ie[nn, 6];
                    } v[5, i, j] = va[nn];
        }   }   }
        // Sample Output at time zero }
        switch (l){
            case 3: eh[0] = 2 * (v[1, io, jo] + v[2, io, jo] + v[3, io, jo] +
                           v[4, io, jo] + yo * v[5, io, jo]) / (4 + yo); break;
            case 2: eh[0] = yo * (v[3, io, jo] - v[1, io, jo]);   break;
            case 1: eh[0] = yo * (v[4, io, jo] - v[2, io, jo]);   break;
        }
        for (ic = 1; ic <= ni; ic++){
            // INHOMOGENEOUS SHUNT NODE SCATTERING PROCEDURE
            for (nn = 1; nn <= kc; nn++){
                for (j = ia[nn, 3]; j <= ia[nn, 4]; j++){
                    for (i = ia[nn, 1]; i <= ia[nn, 2]; i++){
                        a = (v[1, i, j] + v[2, i, j] + v[3, i, j] + v[4, i, j]
                            + v[5, i, j] * rd[nn]) * 2 / (rd[nn] + 4);
                        v[1,i,j]=a-v[1,i,j];      v[2,i,j]=a-v[2,i,j];
                        v[3,i,j]=a-v[3,i,j];      v[4,i,j]=a-v[4,i,j];
                        v[5,i,j]=a-v[5,i,j];
            }   }   }
            // SET UP BOUNDARY CONDITIONS
            for (nn = 1; nn <= kb; nn++){
                for (j = ib[nn, 3]; j <= ib[nn, 4]; j++){
                    for (i = ib[nn, 1]; i <= ib[nn, 2]; i++){
                        vxy = v[ib[nn, 6], i, j];
                        v[ib[nn, 6], i, j] = r[nn] * v[ib[nn, 5], i
                        + ib[nn, 8], j + ib[nn, 7]];
                        v[ib[nn, 5], i + ib[nn, 8], j + ib[nn, 7]] =
                        r[nn] * vxy;
            }   }   }
            // PERFORM IMPEDANCE MODIFICATIONS AT AIR-DIELECTRIC BOUNDARIES
            if (ibd[1, 1] != 0){
                for (nn = 1; nn <= kd; nn++){
                    for (j = ibd[nn, 3]; j <= ibd[nn, 4]; j++){
                        for (i = ibd[nn, 1]; i <= ibd[nn, 2]; i++){
                            vx = v[ibd[nn, 6], i, j];
                            vy = v[ibd[nn, 5], i + ibd[nn, 8], j + ibd[nn, 7]];
                            v[ibd[nn, 6], i, j]
                                = -rc[nn] * vy + (1 + rc[nn]) * vx;
                            v[ibd[nn, 5], i + ibd[nn, 8], j + ibd[nn, 7]]
```

```
                             = rc[nn] * vx + (1 - rc[nn]) * vy;
            }   }   }   }
            // INTERCHANGE IMPULSES AMOUNG NODES
            for (nn = 1; nn <= kc; nn++){
                for (j = ia[nn, 3]; j <= ia[nn, 4]; j++){
                    for (i = ia[nn, 1]; i <= ia[nn, 2]; i++){
                        a = v[3, i, j];          v[3, i, j] = v[1, i, j + 1];
                        v[1, i, j + 1]           v[4, i, j] = v[2, i + 1, j];
                        v[2, i + 1, j] = a;
            }   }   }
            // Sample Output
            switch (l){
                case 3: eh[ic] = 2 * (v[1,io,jo] + v[2,io,jo] + v[3,io,jo] +
                     v[4, io, jo] + yo * v[5, io, jo]) / (4 + yo);    break;
                case 2: eh[ic] = yo * (v[3, io, jo] - v[1, io, jo]);  break;
                case 1: eh[ic] = yo * (v[4, io, jo] - v[2, io, jo]);  break;
            }
            if (bShowProgress){
                Console.WriteLine("{0}: iteration {1}", input_file, ic);
                Thread.Sleep(1);
    }   }   }
    public void Fourier(){
        float ra, rb, cs, u, uk, ehre, ehim, ehmod, d;
        npt = 0;      max = 0;      ra = 0;      rb = 6.283184f;     d = d1;
        while (d <= d2){
            ehre = 0;   ehim = 0;   uk = (float)exp(-d * ra);        u = uk;
            for (ic = 0; ic <= ni; ic++){
                cs = ic * rb * d;
                ehre = (float)(ehre + (eh[ic] * cos(cs) * uk));
                ehim = (float)(ehim - (eh[ic] * sin(cs) * uk));
                uk = uk * u;
            }
            ehmod = (float)(sqrt(ehre * ehre + ehim * ehim));
            npt = npt + 1;             data[npt, 1] = d;
            data[npt, 2] = ehmod;      d = d + ds;
        }
        for (j = 1; j <= npt; j++){
            if (data[j, 2] > max){
                max = data[j, 2];      peak = data[j, 1];        pt = j;
    }   }   }
    public void CurveFit(){
        float ai, bmax;
        if (pt > 1 && pt < npt){ ptp = pt + 1;        ptm = pt - 1;
            ai = ((data[ptm,2]-max)*(data[ptm,1]-data[ptp,1])-(data[ptm, 2]-
                 data[ptp, 2]) * (data[ptm, 1] - data[pt, 1]));
            ai = ai / ((data[ptm,1] - data[pt,1])*(data[pt,1]-data[ptp,1])*
                 (data[ptm, 1] - data[ptp, 1]));
            bmax = (data[ptm,2]-max)/(data[ptm,1]-data[pt,1])-ai*(data[ptm,1]
                 + data[pt, 1]);
            peak = -bmax / (2 * ai);
    }   }
    public void Correct(){
        float a, p;
        // VELOCITY ERROR CORRECTION
        p = (float)(3.14592653 * peak);
        cf = (float)(p / atan(sqrt(2.0) * sin(p)));
        pcf = peak / cf;
    }
    public void PrintReport(){
        sw.WriteLine();
        sw.WriteLine("        RESULTS");
        sw.WriteLine("        -------");
        sw.WriteLine("{0} points plotted.", Format<int>(npt, 4));
        sw.WriteLine("  Maximum plotted field magnitude:");
        sw.WriteLine("{0}", Format<float>(max, 14));
        cf = 1 / cf;
        sw.WriteLine("  The velocity correction factor (Do/D) is {0}", cf);
        sw.WriteLine(
            "  MESHSIZE/MESH WAVELENGTH     MESHSIZE/FREE SPACE    FIELD
      MAGNITUDE");
        sw.WriteLine("           (D)                     (Do)    WAVELENGTH
(EHMOD)");
        sw.WriteLine(
            "  ------------------------   -----------------------  -----------
      ----");
          for (j = 1; j <= npt; j++){
            sw.WriteLine("        {0}                     {1}              {2}",
                    Format<float>(data[j, 1], 8),
                    Format<float>(data[j, 1] * cf, 8),
                    Format<float>(data[j, 2], 8));
        }
        sw.WriteLine();
        sw.WriteLine("Maximum field value at D  = {0}", peak);
        sw.WriteLine("      corresponding to Do = {0} <=== FINAL RESULT", pcf);
    }
    public void PlotGraph(){
        float u;
        sw.WriteLine("                        Graph of EHMOD vs. D");
        sw.WriteLine("   D                              EHMOD");
        u = max / 70;
        if (u == 0)
            Console.WriteLine("No plot generated - all values equal zero.");
        else{
            sw.Write("         ");
            for (j=7; j<=63; j+=7) sw.Write("   {0}",Format<float>(j*u,4));
                sw.WriteLine("   {0}", Format<float>(max, 4));
            for (j = 1; j <= npt; j++){
                data[j, 2] = data[j, 2] / max;
                data[j, 2] = (float)(floor(70 * data[j, 2]));
                for (i = 1; i <= 70; i++){
                    outc[j, i] = ' ';
                    if (data[j, 2] == i) outc[j, i] = '*';
                }
                sw.Write(" {0}|", Format<float>(data[j, 1], 7));
                for (i = 1; i <= 70; i++){ sw.Write(outc[j, i]); }
                sw.WriteLine();
    }   }   }
    public void Run(bool bShowProgress){
        ReadData();         Iterate(bShowProgress);        Fourier();
        CurveFit();         Correct();                     PrintReport();
        PlotGraph();
    }
    public void Work(){
        if (OpenFiles()){   Run(true);  CloseFiles();
    }   }
}   }
```

TLM.CS: An object-oriented TLM Program in C#

```
using System;                  using System.Text;        using System.Threading;
using System.Collections.Generic;                        using TLM;
```

```
namespace CS_TLM{ // This is the main program to test out the TLM class
    class Program{
        static void Main(string[] args){
            // Single Thread TLM_InHo
            //========================
            InHo tlm = new InHo(@"../../TLM_INHO_INP.txt",
                                @"../../TLM_INHO_OUT.txt");
            if (tlm.OpenFiles()){
                tlm.Run(false);    tlm.CloseFiles();
            }
            // Multi-Thread TLM_InHo
            //=======================
            InHo tlm_1 = new InHo(@"../../TLM_INHO_INP_1.txt",
                                  @"../../TLM_INHO_OUT_1.txt");
            InHo tlm_2 = new InHo(@"../../TLM_INHO_INP_2.txt",
                                  @"../../TLM_INHO_OUT_2.txt");
            ThreadStart delegate_1 = new ThreadStart(tlm_1.Work);
            ThreadStart delegate_2 = new ThreadStart(tlm_2.Work);
            Thread thread_1 = new Thread(delegate_1);
            Thread thread_2 = new Thread(delegate_2);
            thread_1.Start();   thread_2.Start();
            while (thread_1.IsAlive || thread_2.IsAlive){ Thread.Sleep(1); }
            Console.WriteLine("Job Done!");
}    }    }
```