On the Peculiarities of the Exchange of Data between Applications in High-Level Languages and Matlab Functions

Alexander V. Krasnovidov
Department of Information and Computer Systems, Emperor Alexander I St. Petersburg State Transport University
St. Petersburg, Russia
alexkrasnovidow@mail.ru

Anatoly D. Khomonenko
Department of Information and Computer Systems, Emperor Alexander I St. Petersburg State Transport University
St. Petersburg, Russia
khomon@mail.ru

Andrew V. Zabrodin
Department of Information and Computer Systems, Emperor Alexander I St. Petersburg State Transport University
St. Petersburg, Russia
zaw2003@yandex.ru

Alexander V. Smirnov
Laboratory of computer aided integrated systems, St.Petersburg Institute for Informatics and Automation of the RAS
Saint Petersburg, Russia
smir@iias.spb.su

Abstract

The possibilities of data exchange between functions in the language of the Matlab system with applications in high-level languages are considered. Programs in high-level languages appropriately to use in conjunction with mathematical packages to collect data, control various technological processes, and perform other similar actions. The instrumental means of organizing such an exchange are characterized. It had been shown the feasibility of integrating applications in high-level languages and the M-function of the Matlab system, if necessary, to organize input / output of large amounts of data.

Introduction

A modern approach to the design of information systems requires maximum information compatibility of various software applications that work in a single information environment.

Currently, a large number of different data processing systems are applied, among which one of the most frequently used is the Matlab system. However, this system is practically no means of interactive interaction with users. In addition, a program written in the Matlab language can be run to perform only inside the own environment. On the other hand, programs developed in high-level C ++ or C # languages provide users with advanced interactive possibilities, but are lack internal means for complex mathematical or statistical data processing. Hence the expediency of combining the computational capabilities of specialized data processing systems with the interactive capabilities of programs in high-level languages. The integration technology of MATLAB functionality allows Windows users to maximize the use of the analytical tools of this software package for the following tasks:

- solving optimization problems of any dimension;
- obtaining reliable results of financial and economic calculations, including on-line;
- three-dimensional visualization of complicated geometric shapes and surfaces having an economic meaning;
- creating business valuation algorithms that require a large amount of computation;
- creation of intelligent project management systems;
- modeling of economic processes of any degree of complexity using the capabilities of the Simulink package.

In: B. V. Sokolov, A. D. Khomonenko, A. A. Bliudov (eds.): Selected Papers of the Workshop Computer Science and Engineering in the framework of the 5 th International Scientific-Methodical Conference "Problems of Mathematical and Natural-Scientific Training in Engineering Education", St.-Petersburg, Russia, 8–9 November, 2018, published at http://ceur-ws.org

The main methods of organizing such an interaction are described in detail in various papers, for example, in [Ada17, Smo05, Kra18] and others. These are the following methods:

1. Program conversion, written in Matlab language, into program in C ++. This method is historically the first.

2. Matlab system interaction with Microsoft Visual Studio using the NET Framework platform. Each of these methods has its own advantages and disadvantages. A detailed comparison of them is given in [Ada17, Bey05]. Here it is worth noting that independent applications use a special mode of operation of the mcc compiler, which leads to difficulties in passing parameters to the Matlab function. In addition, such an application is launched from the command line, which restricts the dialog capabilities. In fact, the mcc compiler generates C ++ code that is equivalent to calling the corresponding Matlab function [Mat09].

The second method is based on the creation of dynamic class libraries according to CLR standards that can be used in any application, which appropriates these standards. Versions of the Matlab system, starting with R2010b and higher, allow creating dynamic libraries which can be used in programs in any programming languages supported by the CLR runtime.

In this case, the M-functions are converted to methods of objects whose classes are declared in the CLR. Further these methods can be called from programs in languages C ++ or C #. This approach allows you to fully utilize the capabilities of the Microsoft system Visual Studio in terms of supporting interactive tools and flexibility in managing the computational process. However, in its implementation should be kept in mind the following features of the classes declared:

- M-functions, unlike C ++ or C # methods, can have several output variables.
- Declared classes can't encapsulate data. They can contain only callable methods.
- Script-files can't be converted to methods and integrated into a dynamic library.

The first of the properties listed is not significant in cases when a call from a C ++ / C # program applies to a method that uses only data transmitted from the calling program, and returns the result only to it (standalone method). The other two properties can lead to difficulties in a various situations, for example, in case of necessity of data exchange between methods contained in the dynamic library. An example of this is a Matlab program that consists of several interrelated M-functions, some of which perform initialization of common variables, for example, set parameters of a certain model, while others perform the actual simulation.

Naturally, the various data exchange between these functions is assumed. Inside the Matlab programming system, such kind of exchange can be implemented using global variables, which could be declared in a Script-file. However, the third mentioned property does not allow their integration into the above dynamic libraries. Hence the need to find other possibilities for data exchange between interrelated M-functions within a single task is following. One of the possible way of solving is the organization of data exchange between interconnected M-functions through the calling program in C ++ or C #. The above way is considered in this article.

1 Organization of the Data Exchange between Matlab-Functions and Programs in High-Level Language

It was shown above that one of the options for organizing the exchange of data between interrelated M-functions is calling a method written in one of the high-level languages (C ++ or C #). In this case, the common data is located in the address space of the calling program and transmitted to the called M-functions as parameters. The general scheme of organizing such kind of interaction is described in detail in [Ada17]. The following peculiarities of data transmission between M-functions and a high-level language program are considered below.

1. Programming languages C and C # are languages with strong data typing. All variables must be declared before they are used. Matlab programming system does not require preleminary declaration of variables and their types.

2. Methods in C ++ and C # applications have or one output parameter returned by the return statement and its type coinciding the method type, or without output parameter (void type). M - functions can return multiple output variables. In general case, the header of the M function has the following form:

function [y1, y2,…,yn] = < function_name > (<Input_Variable_List >).

Thus, the output variables are a vector with elements y1, y2, ..., yn, while all of them, in turn, can be matrices or vectors. Each element can relate to different data types. During the execution of the M - function, all output variables should be assigned values. An example of such a function is the meshgrid function, which is used in conjunction with graphing functions for three-dimensional surfaces. The call to this function is:

[X,Y] = meshgrid(x), where X and Y – output vectors (the result of the function).

It follows from the above that the main point in organizing the interaction between the M-function by the program in the language is the agreement of the types of data transferred and mapping of the set of output variables of M-function to the corresponding variables of C ++ / C # program that has been call and vice versa. To solve this problem, it is proposed to use the .NET MWArray

class library for working with MWArray.dll arrays. This dynamic library contains two namespaces:

- MathWorks.Matlab.NET.Arrays — classes that provide access to Matlab arrays from any .NET CLS compliant language;
- MathWorks.Matlab.NET.Utility — service classes that provide general support for MWArray classes in the Matlab MCR runtime enviroment.

The Namespace MathWorks.Matlab.NET .Arrays contains classes for supporting data conversion between managed types and Matlab types. Each class has constructors, destructors, and a set of properties and methods to access the state of the Matlab main array. The class hierarchy of MathWorks.Matlab.NET .Arrays is shown in Figure 1.

MWArray is an abstract class, the root of an array class hierarchy Matlab. It encapsulates the internal type of Matlab mxArray and provides the ability to access, format and manage the array;

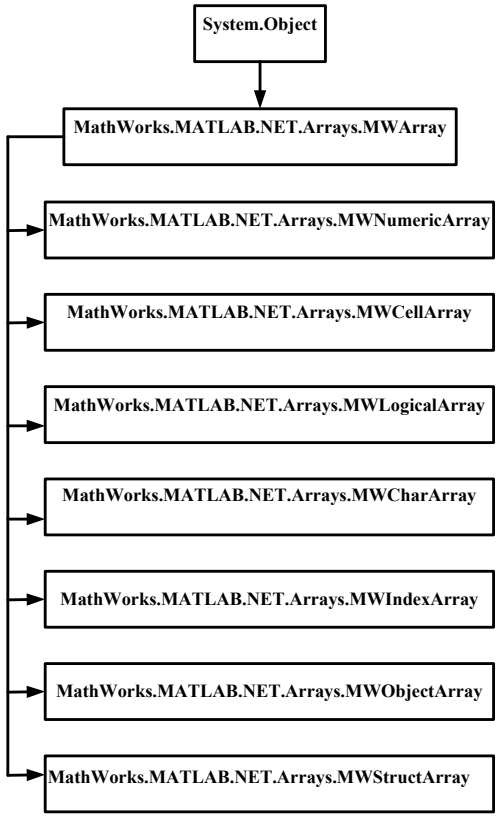

Figure 1: Class Hierarchy_MathWorks .Matlab.NET.Arrays.MWArray

MWNumericArray is a managed representation for Matlab arrays of numeric types. Its Matlab-equivalent is the default array type (vector) of double type, used by most Matlab math functions;

MWLogicalArray is a managed representation for Matlab boolean arrays. Like its equivalent Matlab, MWLogicalArray contains only ones and zeros (true / false);

MWCellArray is a managed representation for Matlab cell arrays. Each element in a cell array is a container that can contain a MWArray or one of its derived types, including another MWCellArray;

MWCharArray is a managed representation for Matlab arrays of character type. Like its equivalent Matlab, MWCharArray supports string creation and string manipulation;

MWIndexArray is an abstract class that serves as the root for the MWArray indexed classes. These classes represent the types of arrays that can be used as input parameters for the array indexing operator [].

MWObjectArray is a special subclass of MWArray that encapsulates its own .NET object in a Matlab array. This object can then be accessed or returned by the Matlab function.

MWStructArray – managed represent-tation for arrays of structures Matlab. Like its equivalent in Matlab, it consists of field names and field values.

Namespace MathWorks.MATLAB.NET. Utility. The service classes in this namespace provide general support for the MWArray classes in the execution environment of the MATLAB MCR component.

The above class hierarchy makes it easy to convert data from the C ++ / C # format to the Matlab format and back, without resorting to explicit type conversion. The following are examples of similar conversions.

- Variables of the double type do not require any conversions.
- One-dimensional array (vector) for transfer to M-function can be formed by the following operators:

```
double [ ] Coeff = {1,2,3,4};//Vector in C++/C#;
    MWNumericArray MTLBCoeff = Coeff; // Vector to pass into M-function as a parameter.
```

- Two-dimensional array for transfer into M - function can be formed by the following operators (*) and (**):

```
double [,] a ={ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }; // Matrix in C ++ / C #; (*)
MWNumericArray arr1 = a;(**)  //Matrix for transfer to M-function as a parameter.
```

- A string variable of type string is converted to Matlab character format for transmission to the M-function using the following operators:

```
string My_String = " This is the C # "string;
MWCharArray mw_ My_String = new MWCharArray(My_String);
```

- Vector type MWArray, returned by M-function in C ++ / C # program; converted to an array of type double using the following operators:

```
MWArray[] Res = null;
// Declaring MWArray Objects
```

```
MWNumericArray descriptor = null; // and MWNumericArray
// select the first element from the MWArray array and convert to
// numeric type MWNumericArray;
descriptor = (MWNumericArray)Res[0];
// convert the array MWNumericArray to an array of type double;
double[,]d_descriptor=(double[,]) descriptor. ToArray(MWArrayComponent.Real);
```

Thus, the MWArray classes are used to process arrays, determine the type, and convert the data. The following are examples of data exchange between M-functions and C # applications.

2 Application for Filtering Experimental Data

Let there be a sequence of numbers xn, which is a series of equidistant measurements of a certain quantity x (t), in which n is an integer (n = 0, 1, 2, ...), and t is considered as a continuous argument. If the sequence yn is calculated by the formula:

$$y_n = \sum_{k=-\infty}^{\infty} c_k x_{n-k} ;$$

then this formula defines a digital non-recursive filter [Hem80].

It can be seen from the formula that the filter operation is completely determined by the set of coefficients c_k.

M-function MainF takes as parameters a set of filter coefficients and the name of the binary file containing the data to be processed.

The returned results are vectors containing the source vector and the vector of processing results.

```
function [ OutMatrix ] = MainF(File,Coeff)
%UNTITLED Summary of this function goes here
%  Detailed explanation goes here
% File - symbolic file name; Coeff is an array (vector) of coefficients.
fid2 = fopen(File,'r');
InV = fread(fid2,'double'); % InV – source vector
fclose(fid2);
Arg = length(InV);
OutV = Filter (T,InV); % Appeal to M - the function that implements the filter
% OutV – output vector
OutMatrix = [InV OutV']; % Formation of a vector -lines of output variables
plot(1:Arg,InV,1:Arg,OutV); % Plotting results
```

M-function MainF is declared as a CLR method of the TFilter class. It is not necessary to declare as the M- method the Filter function, since it is called from a function external to it [Smo05, Bey05]. Since, during data processing, it may be necessary to filter their various variants with different filters, it seems appropriate to organize the input of a file name and coefficient vectors from an application in C ++ / C #.

Such an application should perform the following actions:

1. Receive from the user the name of the file being processed in symbolic form and convert it to the format adopted in Matlab;
2. Receive from the user a vector of filter coefficients in a symbolic (user-friendly) form and convert them to the format adopted in Matlab;
3. Create an object of the TFilter class containing the filtering method.
4. Call the MainF method of the TFilter class and pass parameters (arguments) to it, which are converted to Matlab format.
5. Get the original and processed vector returned by M-function MainF, translating them into the format C #; (in type MWNumericArray – controlled representation of types of Matlab numeric arrays.)
6. Output the received vectors to the listBox visual element (if necessary).

Below are fragments of such an application that perform the above actions.

```
// Declaring Variables
string File_Name; // Name of data file in C format
MWArray [] Res = null; // Declaring MWArray Class Objects
MWNumericArray Vect = null; // and MWNumericArray for work results
// M-functions
double [,] OutV; // Matrix into which the work results will be converted
// M-functions
string SCoeff; // C # format string for entering filter coefficients
double [] Coeff; // Array (vector) to be passed to M function
-------------------------------------------
// Converting the file name from the string type to the Matlab character format
MWCharArray mw_Name = new MWCharArray (File_Name);
-------------------------------------------
```

It was shown above that one of the parameters transmitted to the M-function is an array (vector) of coefficients, which determines the properties of a digital filter. Their values are entered online, so the number of these coefficients is not known in advance. From the point of view of the M-function MainF, this does not matter, since all Matlab arrays are by definition dynamic. However, in C # an array is always a static structure, the size of which must be determined in advance. Therefore, the above transformations (*) and (**) cannot be used directly. To overcome this contradiction, a dynamic structure was created in an application in C # – a

list, which is then converted into an array of the required size.

```
-------------------------------------------------
        List <double> TMP = new List <double> ();
// Template for organizing the list
        // fill the list with entered coefficients
        // .....
        // .....
        Coeff = TMP.ToArray (); // Convert the list
to an array
        MWNumericArray MTLBCoeff = Coeff; //
Transform a numeric array from
        // C # format to a numeric array in the format
        // Matlab
        TFilter Filter = new TFilter (); // Create an
object of class TFilter
         Res = Filter.MainF (1, mw_Name,
MTLBCoeff); // Call the MainF method
        // select the first element from the MWArray
array and convert to
        // numeric type MWNumericArray;
        Vect = (MWNumericArray) Res [0];
        int Size = Vect.NumberOfElements;
        OutV = new double [Size / 2, Size / 2];
        // Initialization of the result matrix
        for (int i = 0; i <Size / 2; i ++)
        {// Filling the result matrix with the
        dimension (2 × Size / 2)
        OutV [1, i] = Vect [i + 1] ToScalarDouble();
        OutV [2, i] = Vect [i + Size / 2 + 1]
        ToScalarDouble ();
        // Display the results in the listBox     visual
element
         listBox1.Items.Add (i.ToString () + "" +
OutV [1, i] + "" + OutV [2, i]);
                }
```

The ToScalarDouble () method of the MWNumericArray class converts scalar Matlab format values to the C # type double. Thus, in this case, an explicit type conversion is required. The result of the application's work is presented in Figure 2. In the above example, the Filter (M-function) method is encapsulated in the MainF method, so the first is not a CLR method – the TFilter class and there is no need to organize data exchange between them.

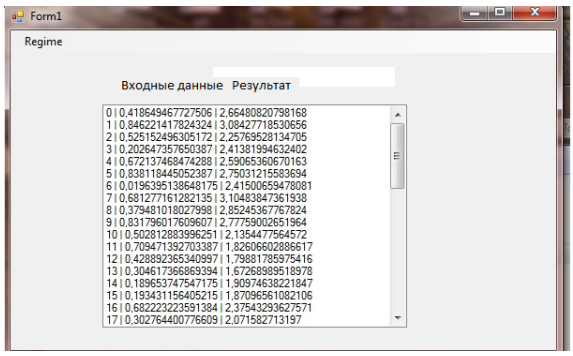

Figure 2: The result of the application working

It should be noted that the use of listBox is not the only option to visualize the results of the application.

3 An Application for Determining the Best access Point for Connecting a Mobile Device to a Local Network

In [Kra16], an algorithm was considered for determining the best access point for connecting a mobile device to a local network, taking into account not only the signal level at its input, but also the bandwidth that can be allocated to the next mobile subscriber when trying to connect it. To solve this problem, a model for selecting such a point was developed. The development was performed with assistance of fuzzy logic apparatus using visual programming in Matlab. Such a model can be saved in a file with the .fis extension and further could be loaded into Matlab-program [Leo05]. After entering the initial data into the model, the actual modeling is performed. The block diagram of the program model is presented in Figure 3.

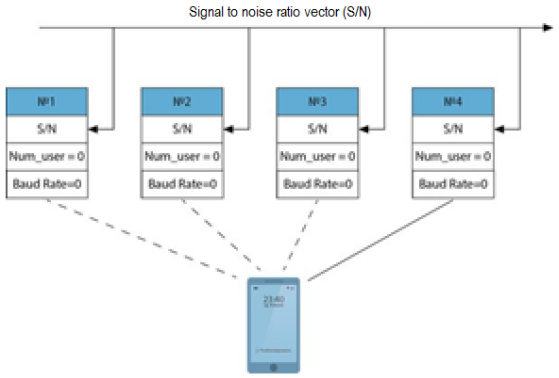

Figure 3: Block diagram of the program model

Each access point is represented by a structure containing the following data fields:

- The sequence number of the access point (double P_Number);
- The current value of the signal level at this point (double Signal_Level);
- Current number of subscribers connected to this point (double Dev_Number);

● The current frame transfer time out of this point (double Baud_Rate).

During model initialization, the indicated structures are combined into a vector, the number of connected subscribers (Num_user), data transfer rate (Baud_Rate) are set equal to zeros, the value of signal / noise levels (S/N) is formed from vector of values of signal / noise levels according to a given distribution law. The current data transfer rate (Baud_Rate) is calculated based in accordance of number of subscribers. Subsequently, after each connection at each point the current number of subscribers is recalculated, the data transfer rate of the point where the connection occurred is calculated. The data of this point is changed and overwritten. The modeling program consists of several M-functions connected by common data and called from a Script-file, the contents of which are presented below.

```
Node = Init1 ('D: \ AVK \ MyC # \ MTLB_Integr \ Fuzzy \ MobileWiFi.fis', N);
Smple = Start_Rnd ();
Step = 1;
while (Step <N)
Res1 = ModelF (Node, Smple, Step);
Node = Res1.ND;
Step = Step + 4;
end;
Print (Res1);
```

M-function Init1 forms an array of structures in accordance with Figure 3 and loads the model file;

M-function Start_Rnd generates the value of the total bandwidth and the values of the S / N levels according to a given distribution law;

M-function ModelF performs simulation in accordance with the above algorithm;

M-function Print provides the output of simulation results in the form of graphs and displays the course of the simulation process.

All listed M-functions are connected by common data: an array of structures shown in Figure 3. For the above Script-file to work, the input of following simulation parameters is required: the name of the model file, the number of simulation cycles, the distribution law parameters, etc. In addition, displaying information on the monitor, in accordance with the principles of the Matlab environment runtime, as well as launching a Script- file is possible only when working in this environment. This implies the expediency of organizing the interaction of the listed above M-functions and an application in C ++ / C #. As in the previous example, these functions must be CLR class methods (in this case, TFuzzy). n fact, such an application must implement the functionality of considered above the Script -file . Such an application should perform the following actions:

1. To get from the user the name of model file in symbolic form and convert it to the format adopted in Matlab.

2. Get from the user the number of simulation cycles, the dimension of the array of structures in a symbolic (user-friendly) form and convert them to the format adopted in Matlab.

3. To create class TFuzzy as a CLR object, containing the methods Init1, Start_Rnd, ModelF, Print.

4. Call the above-mentioned methods of the TFuzzy class in accordance with the simulation algorithm for it parameters (arguments), which are converted to the Matlab format.

Below are fragments of such an application that perform the above actions.

```
// Declaring Variables
---------------------------------------------
string SPar; // one
string File_Name; // 2
double par; // 3
double step; // four
MWArray [] mod = null; // 5
MWArray [] Res = null; // 6
MWArray [] Param = null; // 7
MWStructArray Point = null; // eight
MWStructArray Result = null; // 9
```

Lines 1 ÷ 4 do not need comments, their purpose is similar to similar variables considered in the previous example. The mod object of the MWArray class (line 5) is needed to store the handle of the model file being loaded. This descriptor is an object of the class of the Matlab programming system [Leo05] and passed to the application using the Init1 method of an object named Model belonging to class TFuzzy. The Res object of the class (line 6) of the MWArray is used to store the result of the next simulation step returned by the ModelF CLR method - the Model class. The result is an object of the class of the struct programming system in accordance with Figure 3.

The Param object of the MWArray class (line 7) is used to store the current parameters of the simulation session and belongs to the struct class of the programming system. The Point and Result objects of the MWStructArray class (lines 8÷9) are intended for storing and converting arrays of structures in accordance with Figure 3.

```
---------------------------------------------
TFuzzy Model = new TFuzzy (); // 10
MWCharArray Mod_Name = new
MWCharArray (File_Name);
MWNumericArray Number; // 11
panel1.Visible = false;
mod = Model.Init1 (1, Mod_Name); // 12
Point = (MWStructArray) mod [0]; // 13
Param = Model.Start_Rnd (1); // 14
Step = 1;
while (Step <Par)
{
Res = Model.ModelF (1, (MWArray)
```

```
        Point, (MWArray) Param [0], Step); // 15
        Result = (MWStructArray) Res [0]; // sixteen
        Point = (MWStructArray) Result. GetField
        ("ND"). Clone (); // 17
        Number = Result. GetField ("N"). ToArray
(); // 18
        listBox1.Items.Add ("Connection to the point
No.:" + (Number.ToScalarDouble ()).ToString ());
        Step = Step + 4; // nineteen
        };
        listBox1.Visible = true;
        Model.Print (0, Result); // 20
```

The operator in line 10 creates a Model CLR object of the TFuzzy class. The Number object of the MWNumericArray class is used to convert formats between the application and the corresponding methods of the CLR – Model class (the MWNumericArray class is a controlled representation of the types of Matlab numeric arrays) [Bey05, Mat09]. The operator in line 12 performs model initialization, passing the model file name as a parameter (see previous example). As was shown above, the Init1 method returns a model descriptor, which in this case has the following structure (Figure 4):

As in the previous example, the operator in line 13 selects the first element from the array, but in this case the array consists of structures, so the appropriate type conversion is applied here.

```
name: 'MobileWiFi#1'
type: 'mamdani'
andMethod: 'min'
orMethod: 'max'
defuzzMethod: 'centroid'
impMethod: 'min'
aggMethod: 'max'
 input: [1x2 struct]
 output: [1x1 struct]
 rule: [1x9 struct]
```

Figure 4: The structure of the model descriptor

The operator in line 14 receives a structure containing the current parameters of the simulation session. Operators in lines 15 ÷ 18 work in a loop and perform the following actions:

- Performs model calculation, which consists in determining the best access point at this modeling step (line 15). The ModelF method takes the Point and Param structures as parameters (see above) and returns the result of the Res class struct shown in Figure 5. Since the latter is an array, it is combined with its first element.
- Executes the simulation result at the next step, to the class of the struct (line 16), which is necessary to calculate the model in the next step (line 15).
- Performs deep copying of the current values of the simulation parameters for pass to the ModelF method (line 17) that the model needs to be calculated in the next step (line 15).
- Retrieves the access point number to which the connection occurred at the current simulation step. This is necessary to visualize the order of connections (line 19).
- Performs graphical visualization of simulation results (line 20).

Thus, the use of objects of the class of struct made it possible to organize the storage of common data of the CLR methods of the Model class in an application implemented in C #. The above data structures used in the application are shown in Figure 5, 6 and 7 the results of the application work are shown.

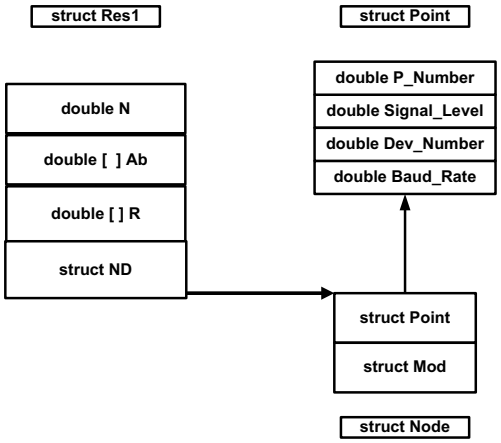

Figure 5: Application data structures

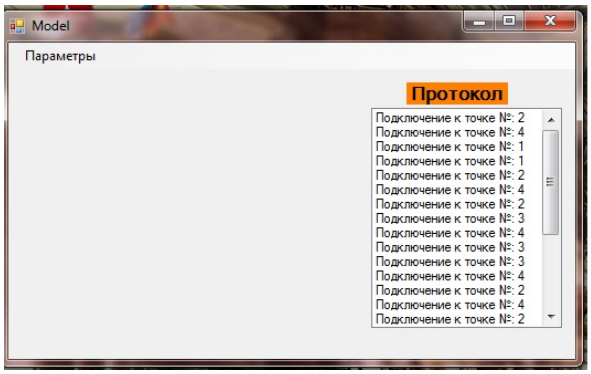

Figure 6: The result of the application with the visualization of intermediate results

Figure 6 demonstrates the dynamics of subscriber connections to access points.

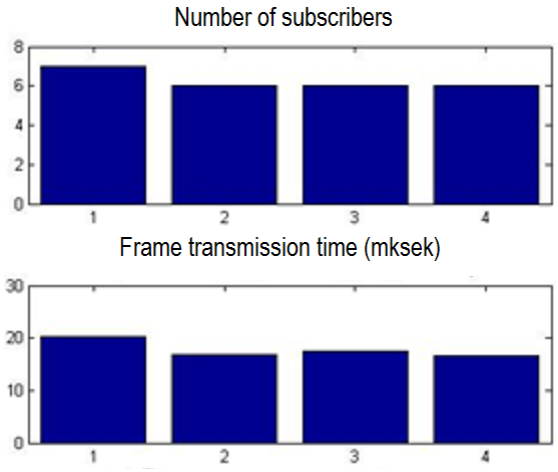

Figure 7: The result of the application with the visualization of the final results

In Figure 7 the results of simulation are displayed. For visualization of the final results used method Print.

Conclusion

The analysis of the considered methods of interaction of the Matlab system with programs in C ++ and C # languages within of the Microsoft Visual Studio application development environment allows to draw the following conclusions:

1. The integration of the capabilities of the Matlab system with applications in high-level programming languages expands the possibilities of developing application software for solving various problems that require both a convenient dialogue with the user, as well as complex mathematical calculations with graphical visualization of results.

2. Due to the impossibility of converting Script files to CLR methods of classes and their integration into dynamic libraries, data exchange between related methods within the CLR class cannot be organized.

3. The exchange of data between interrelated methods can be organized through the application in a high-level language, by means of passing it the appropriate parameters.

4. When transferring parameters by methods, one should take into account that M-functions, unlike C ++ or C # methods, can have several output variables-results. It follows that the main point in organizing the interaction between the M function and the application in a high level language is the coordination of the types of data transferred and the mapping of the set of output variables of the M function to the corresponding variables of the calling program in C ++ / C # and vice versa. To solve this problem, it is proposed to use the .NET MWArray class library for working with MWArray.dll arrays. This library allows you to easily convert data from the C ++ / C # format to the Matlab format and back, without resorting to explicit type conversion.

5. The usage of objects of the class of struct makes it possible to organize the storage of complicated data of the CLR classes methods in an application implemented in C #. In this case, mappings of a set of output variables M - functions to the corresponding variables of an application in C ++ / C # and back can be organized by combining data into structures.

Acknowledgments

The work was partially supported by the grant of the MES RK: project No. AP05133699 "Research and development of innovative information and telecommunication technologies using modern cyber-technical means for the city's intelligent transport system".

References

[Ada17] S. Adadurov, A. Krasnovidow, A. Khomonenko, I. Koroteev. Metody integracii informacionnych system v protsesse razrabotki bezopasnyh prilojeniy [Methods of integration of instrumental systems in the development of secure applications]. // Information security issues. Computer systems. 2017, №4. Pp. 80–86.

[Bey05] Ing Bey, Dzhifeng Ksu. Vzaimodeistvie Matlab s ANSI C, Visual C ++, Visual Basic i Java, [Interaction of MATLAB with ANSI C, Visual C ++, Visual Basic and Java] M.: Williams, 2005. 207 p.

[Hem80] R. Hemming. Tsifrovie filtry. [Digital filters]. // Ed. A. M. Trahtman. M.: Sovietskoe Radio, 1980. 224 p.

[Kra18] A. Krasnovidow, A. Zabrodin, A. Khomonenko. Ob osobennostiah vzaimodeistvia mezhdu sistemoi Matlab I prilozheniiami s pomoschiu tehnologii COM [About the features of the interaction between the Matlab system and applications using COM technology]. // Professional education, science and innovations in the XXI century. Collection of works of the XII St. Petersburg Congress. 2018 Pp, 126–127

[Kra16] A. Krasnovidow. Model akgoritma opredelenia nailutshei tochki dostupa dla podkluchenia mobilnogo ustroistva. [Model of algorithm to determine a best access point to connect a mobile device to the LAN] // Intellectual Technologies on Transport. 2016. № 2. Pp. 36–42.

[Leo05] A. Leonenko. Nechetkoie Modelirovanie v Srede Matlab i fuzzyTECH. [Fuzzy modeling in MATLAB environment and fuzzyTECH] BHV–Petersburg [Link], Saint–Petersburg, 2005 – 736 p.

[Mat09] MATLAB C/C++ Book for MATLAB Compiler 4.5. 2009. – LePhan Publishing, http://www.lephanpublishing.com/MATLABBookCplusplus.html.

[Smo05] N. Smolentsev. Sozdanie Windows-prilozhenii s ispolzovaniem matematicheskih protsedur Matlab [Creating Windows applications using Matlab mathematical procedures]. - M .: DMK_Press, 2008. 456 p., Ill.