Dynamic Adaptation of a Math Service Using TRAP.NET

Technical Report FIU-SCIS-2008-07-01
July 2008

Javier Ocasio Pérez, Pedro I. Rivera-Vega,
Dept. of Electrical and Computer Engineering
University of Puerto Rico Mayagüez Campus
ocasio.javier@gmail.com, privera@uprm.edu

S. Masoud Sadjadi, and Fernando Trigoso
School of Computing and Information Sciences
Florida International University
sadjadi@cs.fiu.edu, perudise@gmail.com

Abstract

This technical report details a case study done for the TRAP.NET project (which stands for Transparent Reflective Aspect Programming in Microsoft's .NET Framework). TRAP.NET provides dynamic adaptation for software programs written in .NET.

1. Introduction

This work comes from the interest to use, test and take measurements of TRAP.NET with an appropriate application. It was identified that key applications that could tremendously benefit from TRAP.NET, are the ones that are a server side, since they need to be constantly running and available (to offer a service), and also ones which involve databases (large amount of date which need to be available at all time). For showing TRAP.NET capabilities, a simple Math service was designed which calculates the Cosine of a number, and that can be adapted to use several (more efficient) modifications of the algorithm, while the service is running. With these goals in mind two implementations of the Math Service were developed, one using .NET Remoting and a console client and another Web Service version using Windows Internet Information Services (IIS) and ASP.NET.

Research Tasks

Development of Math Service Prototype (Using .NET Remoting)

The MathService prototype consists of the following solutions:

1- *MathServer*, which contains the TRAP.NET, TRAP.NET.ConsoleAddin, TRAP.NET.Generator, TRAP.NET.Composer and MathServer (Windows Application) projects

2- *MathSource*, which contains the MathSource Class Library project (Interface)

3-*CosineClient*, which contains the CosineClient Console Application project

4- *MathDelegates*, which contains the MathDelegates Class Library project

How to use:

Run the adapt-ready version of the MathServer(MathServer.exe.AdaptReady.exe), run the CosineClient (Console application), and run the Composer to change the CosineLimit method (which will change the 'behavior' in the cosine function, that is it will be able to compute more terms in the series that simulates the function, hence giving more accurate results). The CosineClient writes a log, which will serve to plot both the approximate cosine and the cosine from the Math Library; this will help to gather data to compare the quality of the function before and after the dynamic adaptation.

Details:

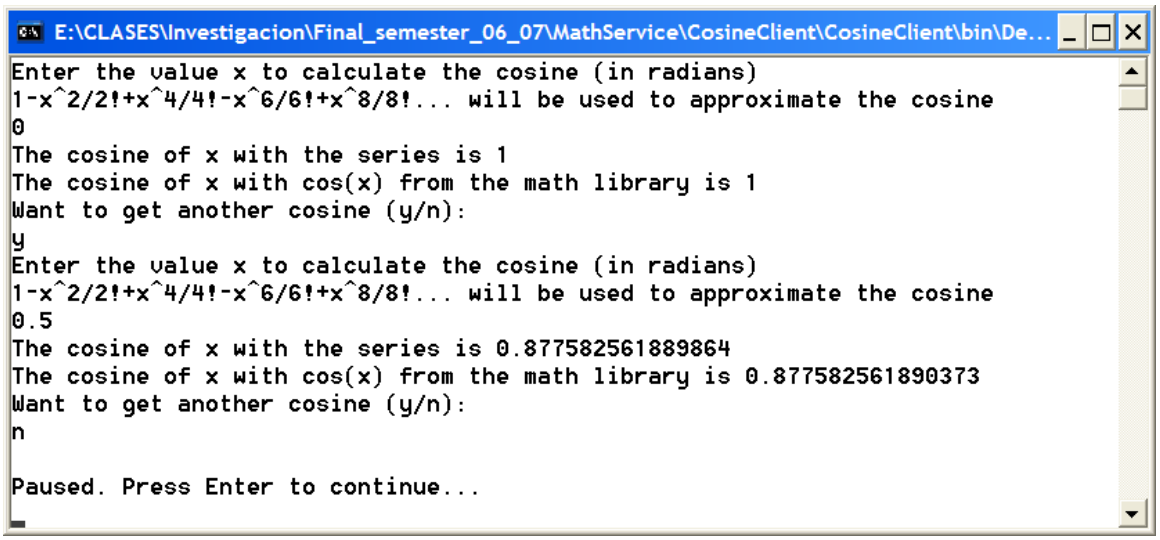
The series: $1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}...$ will be used to approximate the cosine of [1]. The purpose of approximating the cosine of x is to modify the algorithm of the series while the application is running, and have more accurate results without stopping the Math Server and hence have the Cosine Client application constantly available. The algorithm to calculate the cosine approximation is expressed as follows:

```
public double Cose(double x)

{

        int n = 0;

        double a = 0;

        // this method is what makes possible the dynamic adaptation

        int limit = CosineLimit();

        while (n <= limit)

        {

                a += Math.Pow(x, 2 * n) * (Math.Pow(-1, (n)) / nf(2 * n));

                n += 1;

        }

        return a;

}

public double nf(int u)

{
```

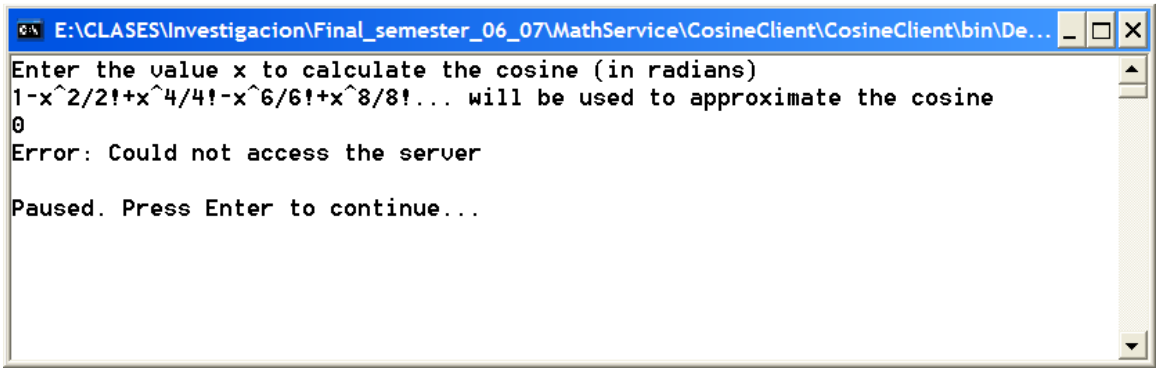
```
        int count = 1;
        double g = 1;
        while (count <= u)
        {
            g *= count;
            count++;
        }
        return g;
}

[AdaptReady(true)]
public int CosineLimit()
{
        return 5;
}
```

Where Cose is the method that contains the main logic of the approximation, where n represents the number of terms in the series; the algorithm uses the methods nf and CosineLimit. The method nf calculates the Factorial of a number and CosineLimit is the method with the adapt-ready attribute ([AdaptReady(true)]) that will make possible the dynamic adaptation of the cosine algorithm by letting the Composer change its code to reflect how many terms will be calculated in the series. These methods are part of the class *MathMethods*, which is the Remoting Object that will provide the math services through .NET Remoting [2]. *MathMethods* class is part of the MathServer Windows Application.

Using the *MathSource.IMathService* interface, the CosineClient Console application is able to invoke the Cose method through the *MathMethods* Remoting Object. The CosineClient application prompts the user to enter the value x to calculate the cosine, when the value is read, the application tries to connect to the MathServer, if the server is running the x value is sent and the cosine approximation received. If the MathServer is not running an appropriate error message is displayed (see **Output 1** and **Output 2**).



Output 1: Results for CosineClient when the MathServer is running



Output 2: Results for CosineClient when the MathServer is not running

The CosineClient writes a log that is used to plot graphs of the Cosine values in Microsoft Excel. The data in the loginfo.txt log is distributed in columns in the format: Time Value AproxCosine MathCosine, and this data is used to plot the cosine of both the approximation algorithm and the Math library cosine and also to plot relationships of the percentage difference of the calculated values with respect to Time and the x value (in radians). Several data was collected for the Cosine from the x range [0, 10], first without adapting the MathServer application and then with adaptation, all this while both the CosineClient and MathServer where running. **Figure 1** show a plot of the cosine function that was calculated with the Math Library provided with the .NET framework, this is the ideal shape that the approximation should obtain if it has sufficient terms. **Figures 2 and 3** show the plot of the cosine with the approximation algorithm without adaptation, that is with CosineLimit method giving original n=5. In this case, the approximation gives accurate results up to $x \approx 3$, but deviates from the expected values abruptly from $x \approx 6$ (for more precise results see Excel_files\test3.xls included with the Code Solutions).

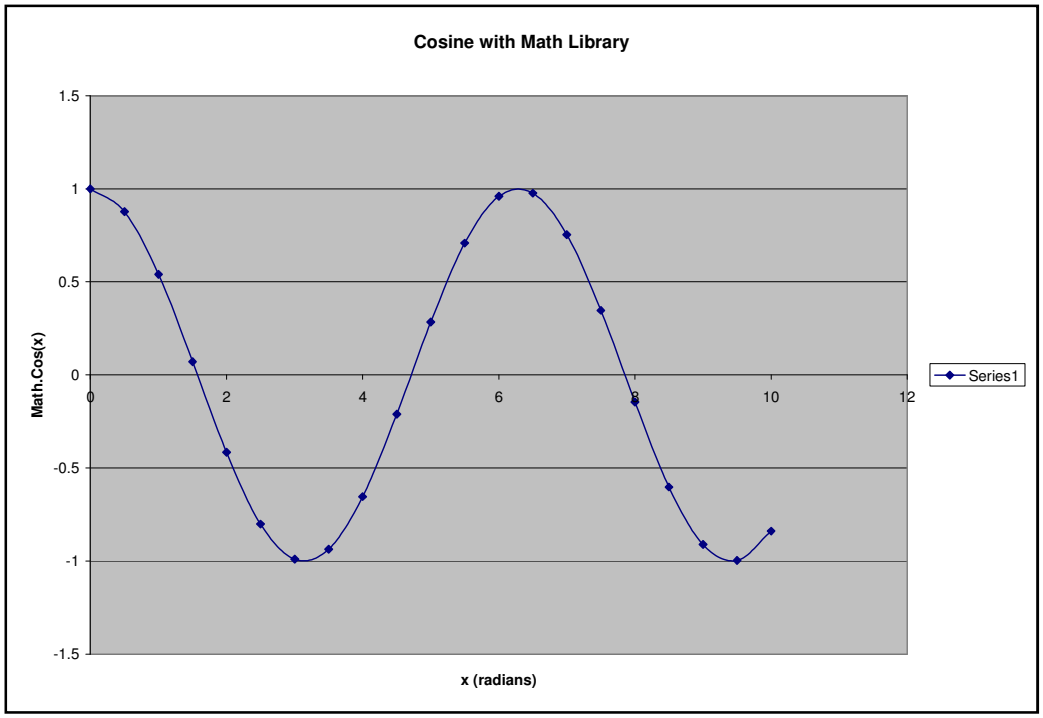


Figure 1: Cosine with Math Library (x range [0, 10])

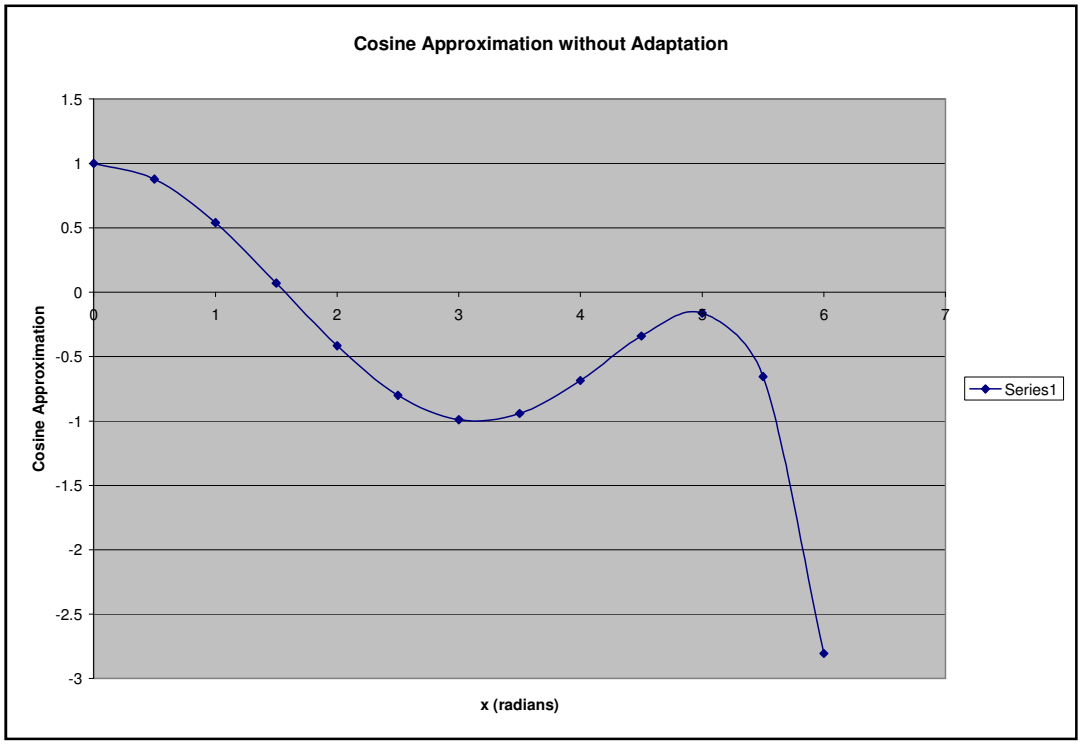


Figure 2: Cosine Approximation without Adaptation (x range [0, 6])

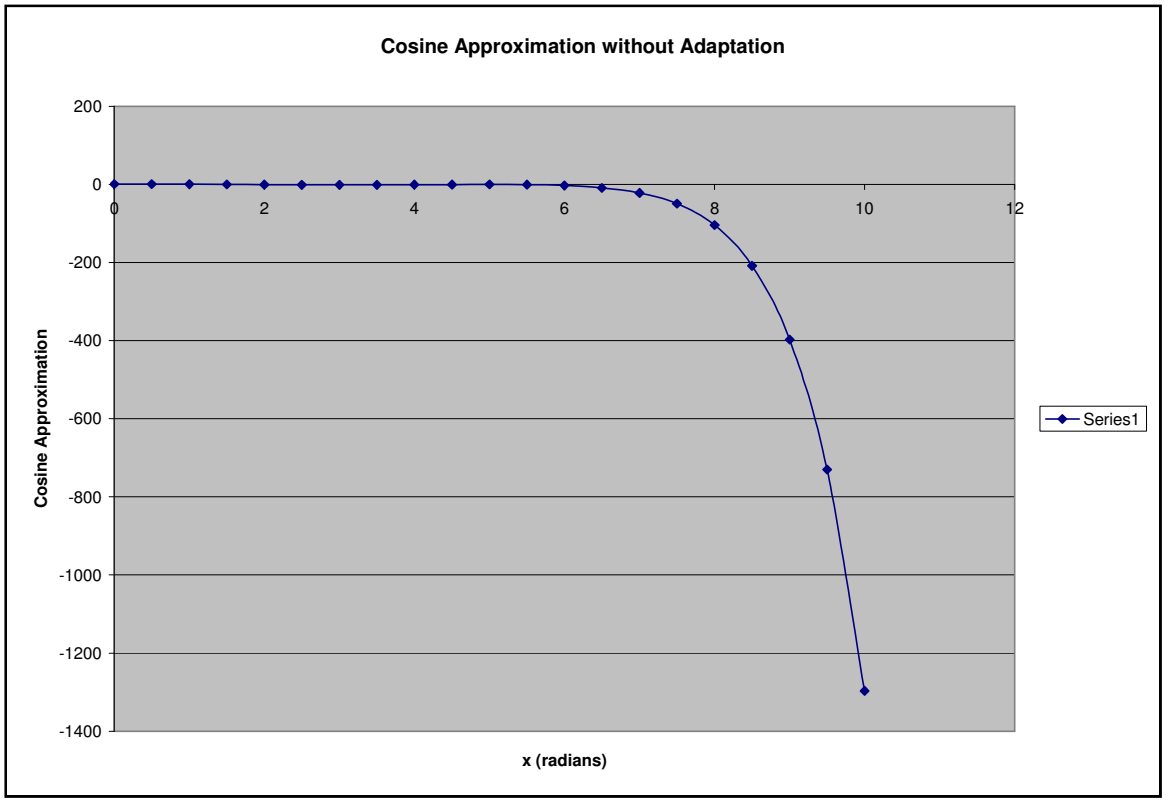


Figure 3: Cosine Approximation without Adaptation (x range [0, 10])

In order to obtain more accurate results and not stop the running MathServer, TRAP.NET comes in handy. Since the running MathServer is the Adapt-Ready version, the TRAP.NET Composer can be used to dynamically change the code from the CosineLimit method and have more terms in the cosine algorithm. To test the changes in 'behavior' of the approximation algorithm, the MathDelegates Class Library is used, this library contains the new implementations of the CosineLimit which will be loaded through the Composer. The MathDelegates library contains the methods SmallLimit, MediumLimit, LargeLimit and JumboLimit, which will give term limits of 10, 30, 100 and 300 respectively. When the Composer is run and the MathDelegates library loaded, a delegate method can be selected to adapt the CosineLimit method. **Figure 4** show the Composer with SmallLimit delegate method selected. **Figure 5** shows the TRAP.NET log (inside the MathServer application \bin folder); this log contains information on the events and XML files used for the adaptation process [3].

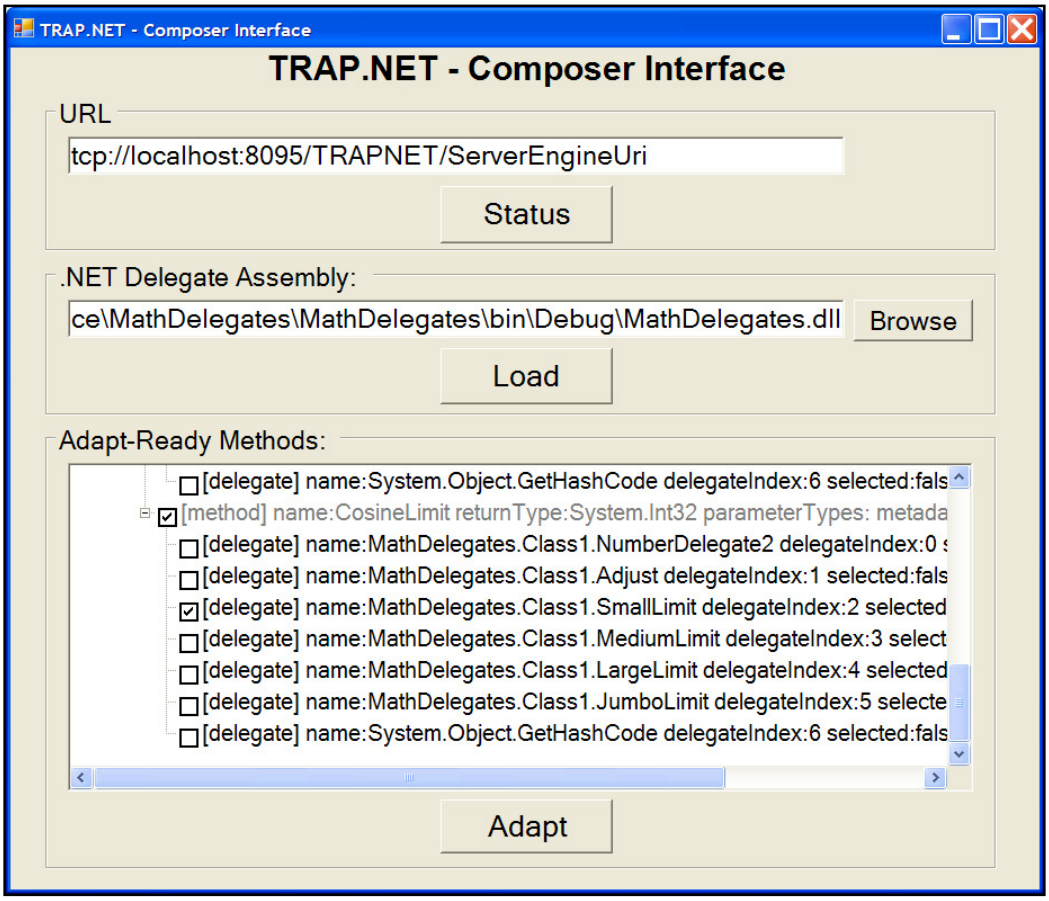


Figure 4: TRAP.NET Composer with SmallLimit delegate method selected

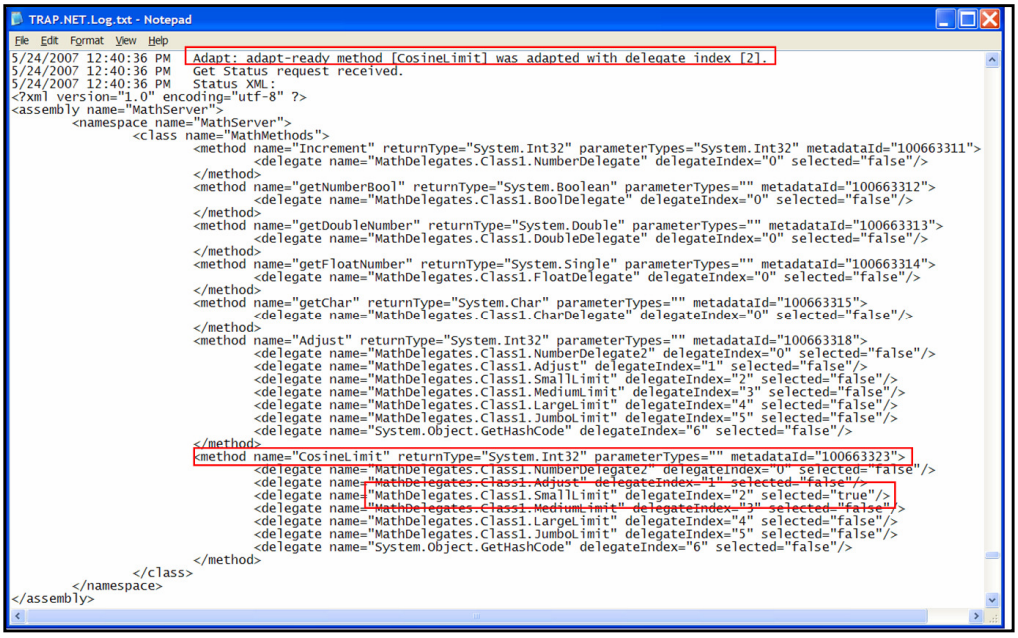


Figure 5: TRAP.NET Log

The SmallLimit delegate method inside the MathDelegates Class Library looks as follows:

```
// Small limit for Cosine series terms
public int SmallLimit()
{
    return 10;
}
```

After the adaptation process the CosineLimit method inside MathServer.MathMethods should look as follows:

```
[AdaptReady(true)]
public int CosineLimit()
{
    if (IsAdaptationEnabled)
    {
        return 10;
    }
    else
    {
        return 5;
    }
}
```

Figure 6 show the plot of the cosine with the approximation algorithm with SmallLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=10. In this case, the approximation gives accurate results up to $x \approx 7$. **Figure 7** show the plot of the cosine with the approximation algorithm with MediumLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=30. In this case, the approximation gives accurate results up to $x \approx 10$. **Figure 8** show the plot of the cosine with the approximation algorithm with

LargeLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=100. In this case, the approximation also gives accurate results up to $x \approx 10$ (and even higher, but the sample was taken up to x=10).

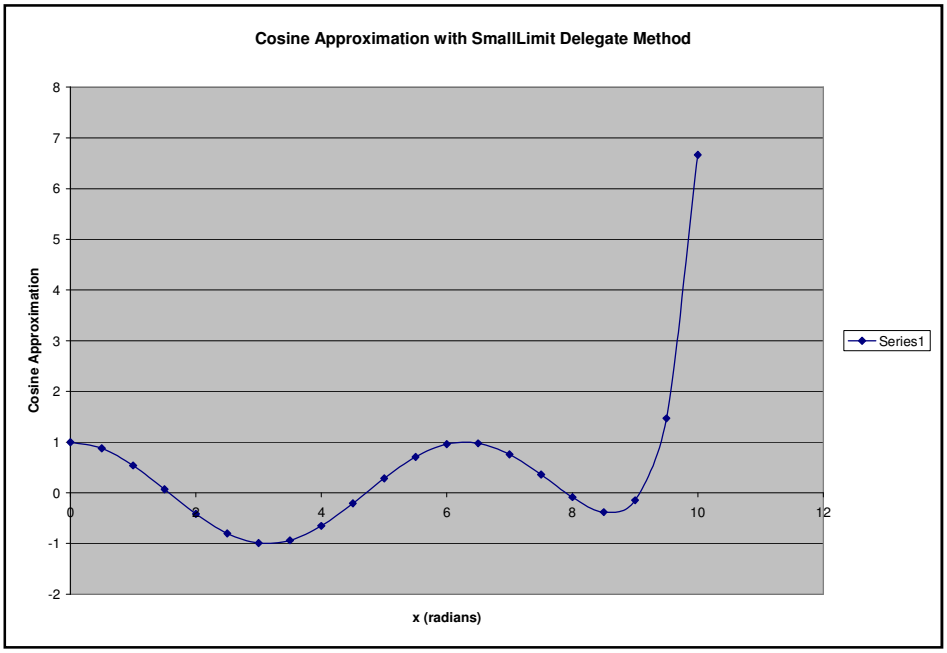


Figure 6: Cosine Approximation with SmallLimit Delegate Method (x range [0, 10])

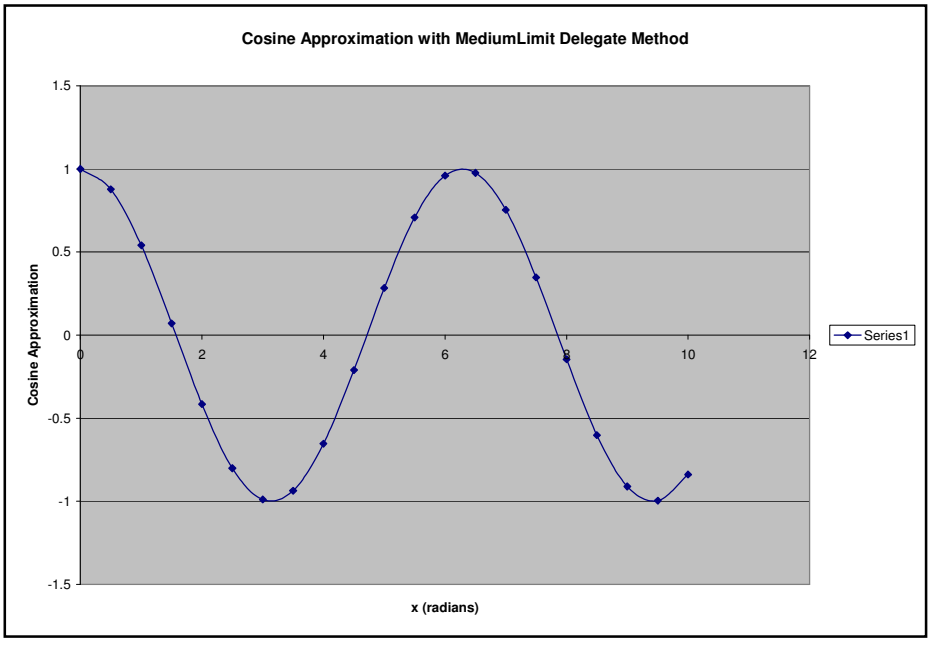


Figure 7: Cosine Approximation with MediumLimit Delegate Method (x range [0, 10])

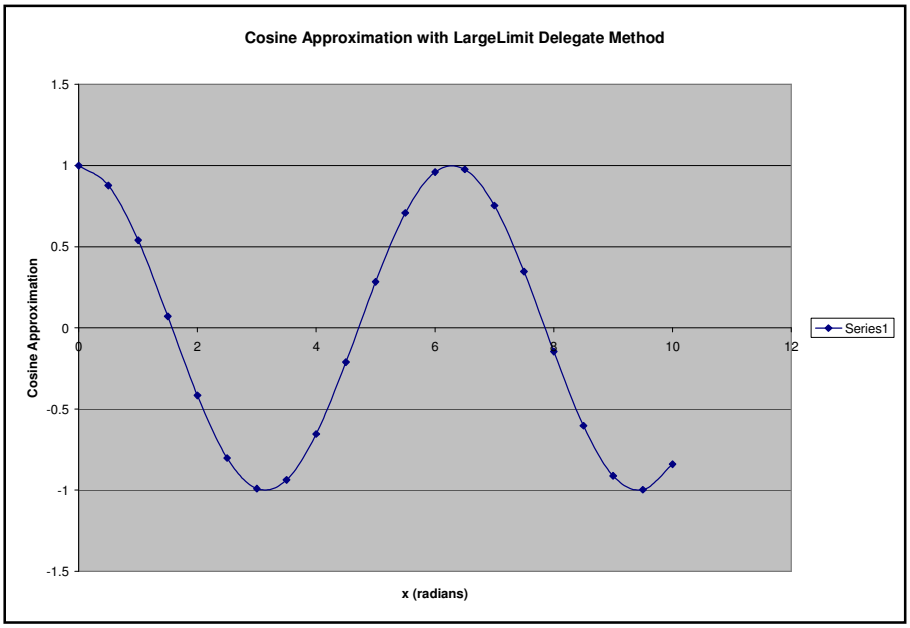


Figure 8: Cosine Approximation with LargeLimit Delegate Method (x range [0, 10])

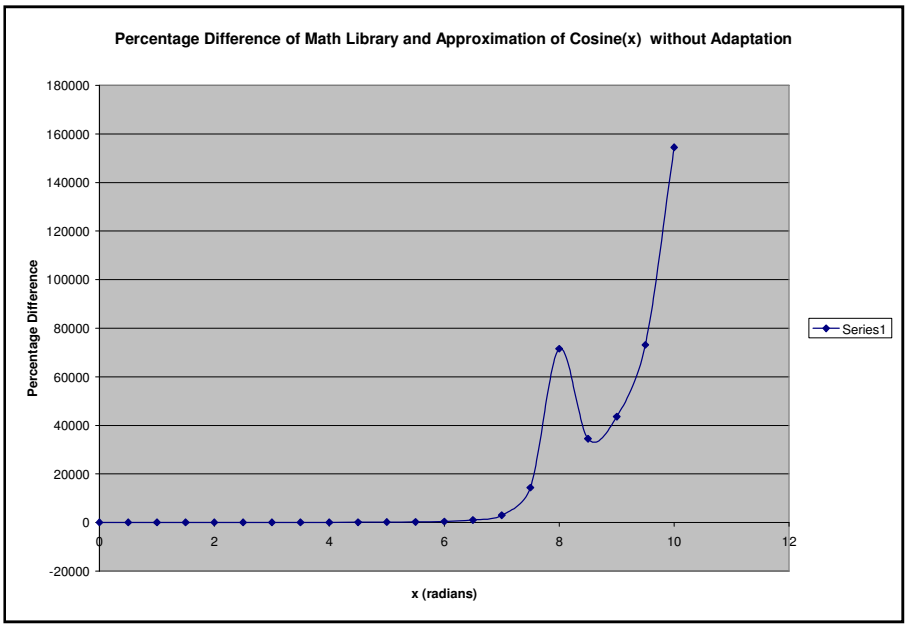


Figure 9: Percentage Difference of Math Library and Approximation of Cosine(x) without Adaptation

Percentage difference plots were also used to measure the difference from the cosine approximation value to the cosine calculated with the Math Library of the .NET Framework. The following formula was used to calculate the percentage difference:

$$pDiff = \left| \frac{\cos_{library} - \cos_{approx}}{\cos_{library}} \right| * 100 .$$

Figure 9 show the plot of the Percentage Difference of Math Library and Approximation of Cosine(x) without Adaptation, that is with CosineLimit method giving original n=5. In this case, the percentage difference gives acceptable results up to $x \approx 3$, but deviates from the expected values abruptly from $x \approx 6$.

Figure 10 show the plot of the Percentage Difference of Math Library and Approximation of Cosine(x) with SmallLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=10. In this case, the percentage difference gives acceptable results up to $x \approx 7$. **Figure 11** show the plot of the Percentage Difference with MediumLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=30. In this case, the percentage difference gives acceptable results up to $x \approx 10$ (note that in x=10, the percentage difference is 9.08082E-11%). **Figure 12** show the plot of the Percentage Difference with LargeLimit Delegate Method, that is with CosineLimit method dynamically adapted, giving n=100. In this case, percentage difference also gives accurate results up to $x \approx 10$ (and even higher, but the sample was taken up to x=10).

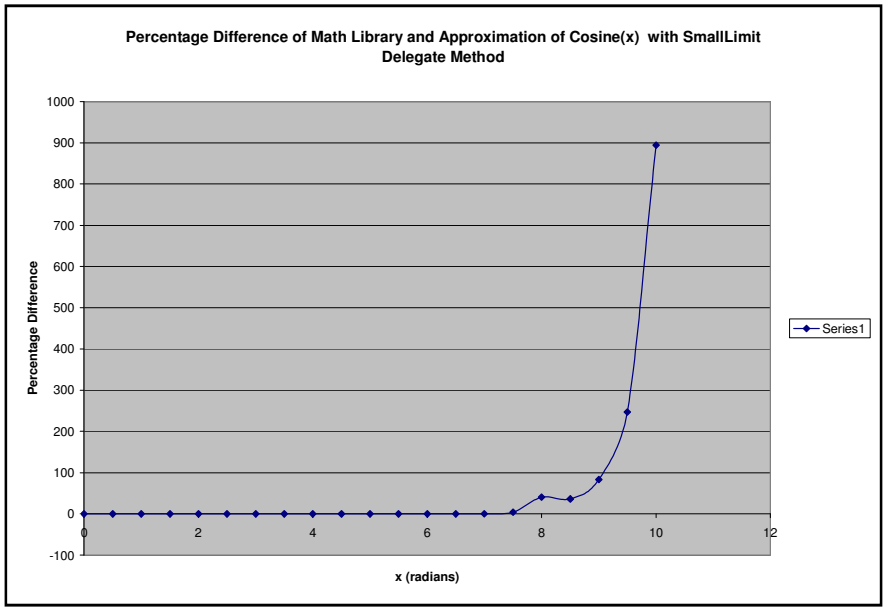


Figure 10: Percentage Difference of Math Library and Approximation of Cosine(x) with SmallLimit Delegate Method

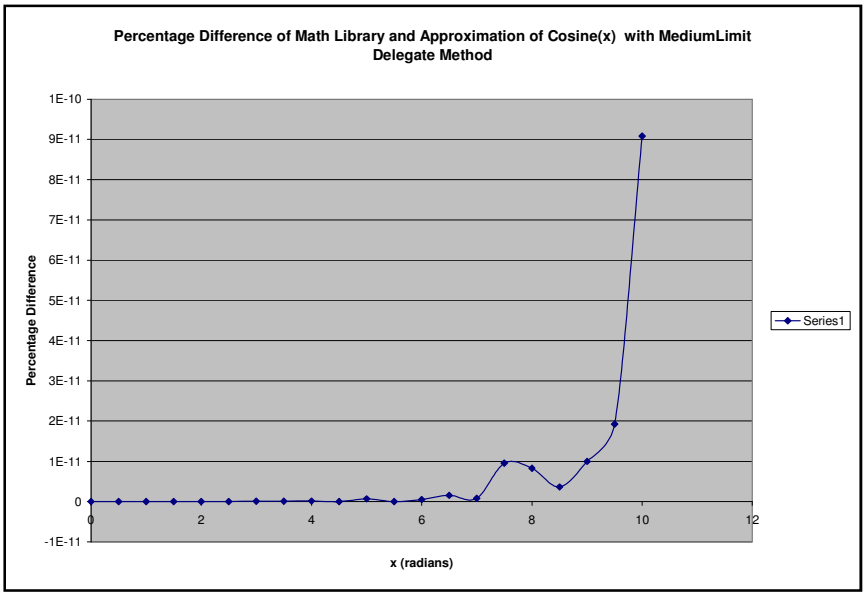


Figure 11: Percentage Difference of Math Library and Approximation of Cosine(x) with MediumLimit Delegate Method

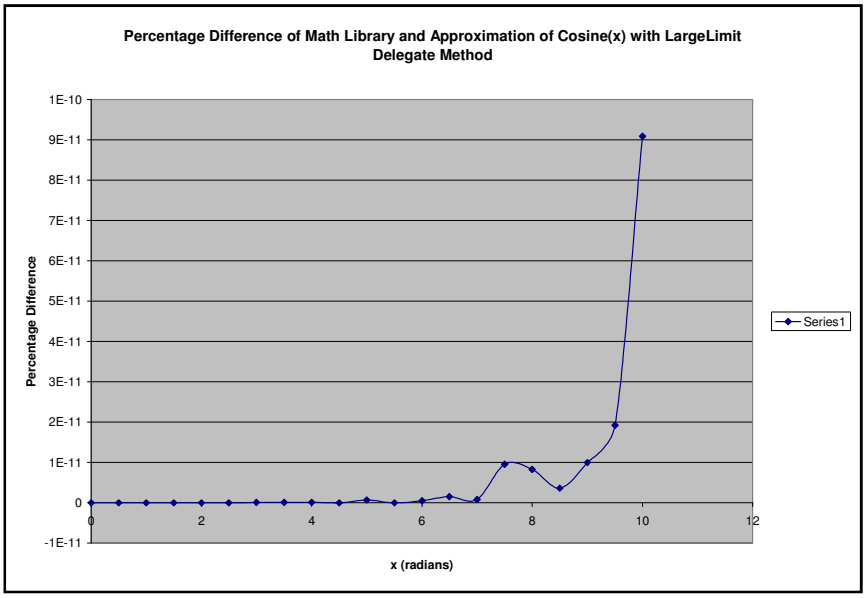


Figure 12: Percentage Difference of Math Library and Approximation of Cosine(x) with LargeLimit Delegate Method

Figure 13 show the plot Percentage Difference vs. Time without adaptation from [12:35 PM, 12:40 PM) and with adaptation from [12:40, 12:49 PM] (x range [0, 10]). In this case, it is evident that the percentage difference begins to get smaller as the adaptation process began at 12:40pm, and keeps getting smaller as CosineLimit methods get dynamically adapted with more

terms. Here the dynamic adaptation of the SmallLimit delegate occurred at 12:40:36 PM, the adaptation with MediumLimit occurred at 12:43:34 PM, the adaptation with LargeLimit occurred at 12:47:01 PM, these times were given by the TRAP.NET Log.

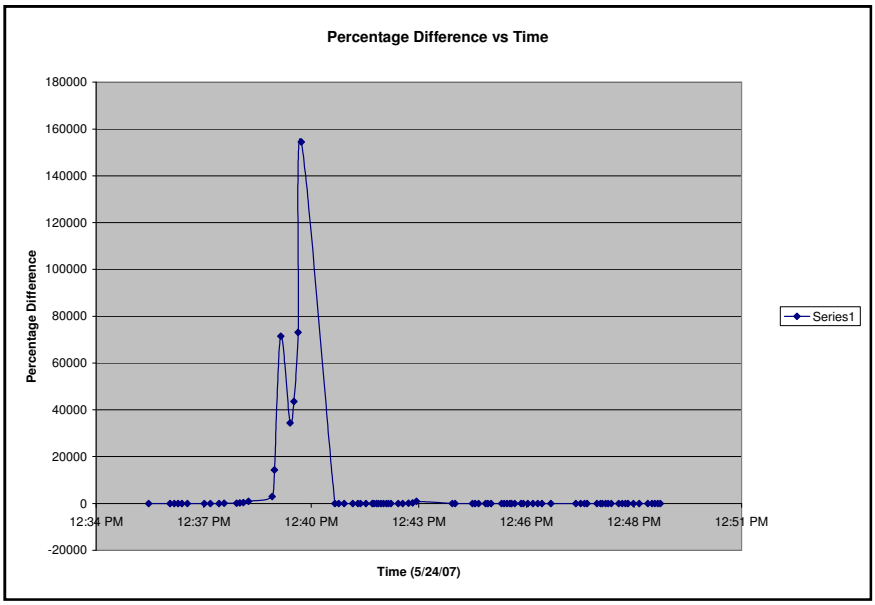


Figure 13: Percentage Difference vs. Time without adaptation from [12:35 PM, 12:40 PM) and with adaptation from [12:40, 12:49 PM] (x range [0, 10])

Development of Math Web Service Prototype (Using Windows IIS and ASP.NET)

The MathWebService prototype consists of the following solutions:

1- *CosineService*, which contains the CosineService Class Library project

2- *pracweb* website, which contains practice2.aspx (the cosine calculation page), Default.aspx (default page), Default.aspx.cs (default page source code), /bin directory with precompiled dynamic link libraries (CosineService.dll, MathSource.dll).

3- *MathServer*, which contains the TRAP.NET, TRAP.NET.ConsoleAddin, TRAP.NET.Generator, TRAP.NET.Composer and MathServer (Windows Application) projects

4- *MathSource*, which contains the MathSource Class Library project (Interface)

5- *MathDelegates*, which contains the MathDelegates Class Library project

6- *CosineServiceTest*, which contains the CosineServiceTest Console Application project (for testing purposes)

How to use:

The user must have Windows IIS installed in his/her computer. The user must put the pracweb folder inside C:\Inetpub\wwwroot. Then using the Internet Information Services Manager, the user must convert pracweb to an application root, for this do the following (adapted from [4]):

1- Open IIS Manager and browse to the Default Web Site.

2- Expand the Default Web Site node and look for the subdirectory that you want to designate as an application root, for our case, pracweb.

3- Right-click the pracweb directory and then click Properties.

4- On the Directory tab, in the Application Settings section, click Create.

5- In the Application name text box, type the name of the application, for our case pracweb and then click OK.

Run the adapt-ready version of the MathServer(MathServer.exe.AdaptReady.exe), run the pracweb web application, for this using a web browser type the URL: http://localhost/pracweb/practice2.aspx. To change dynamically the behavior of the web application run the Composer to change the CosineLimit() method (which will change the 'behavior' in the cosine function, that is it will be able to compute more terms in the series that simulates the function, hence giving more accurate results).

Details:

The MathWebService prototype is similar to the MathService explained earlier (see *Development of Math Service Prototype (Using .NET Remoting)*). The main difference is that now the Math service is transformed into a web application. The idea is that the web application must be constantly available for Internet users, so its modification should occur in a dynamic manner.

To avoid compilation of source code every time a source code is changed (compilation of entire code, even when the changes are only is small parts of the code), pre-compiled dynamic link libraries are included in the \bin directory of pracweb. One of this DLL is CosineService, this library acts like an interceptor (or Interface) that forwards methods calls to MathServer. Using the *MathSource.IMathService* interface, the CosineService library is able to invoke the Cose (discussed earlier) method through the *MathMethods* Remoting Object. The pracweb web application prompts the user to enter the value x to calculate the cosine, when the value is read, the application calls the Approximation method in CosineService and this method tries to connect to the MathServer, if the server is running the x value is sent to the Approximation

method, and from here to the practice2.aspx page where the cosine approximation is presented. If the MathServer is not running an appropriate error message is displayed (see **Figure 14** and **Figure 15**).

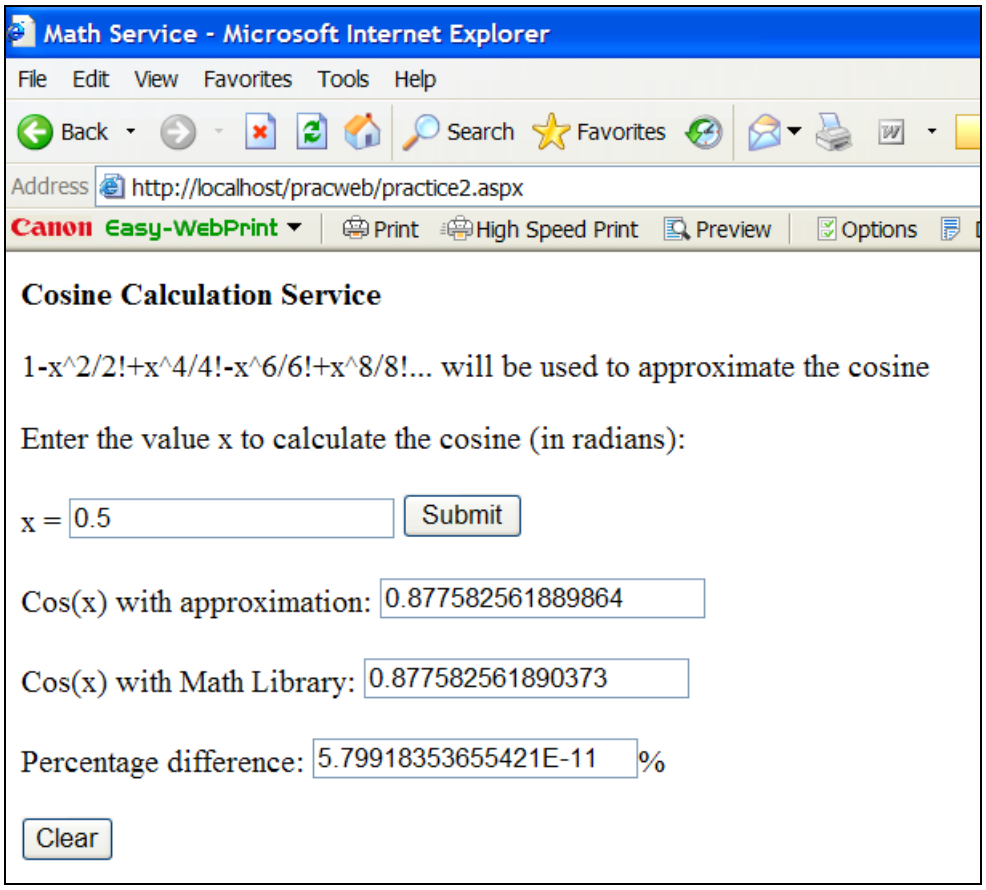


Figure 14: Results for Cosine Calculation Service when the MathServer is running

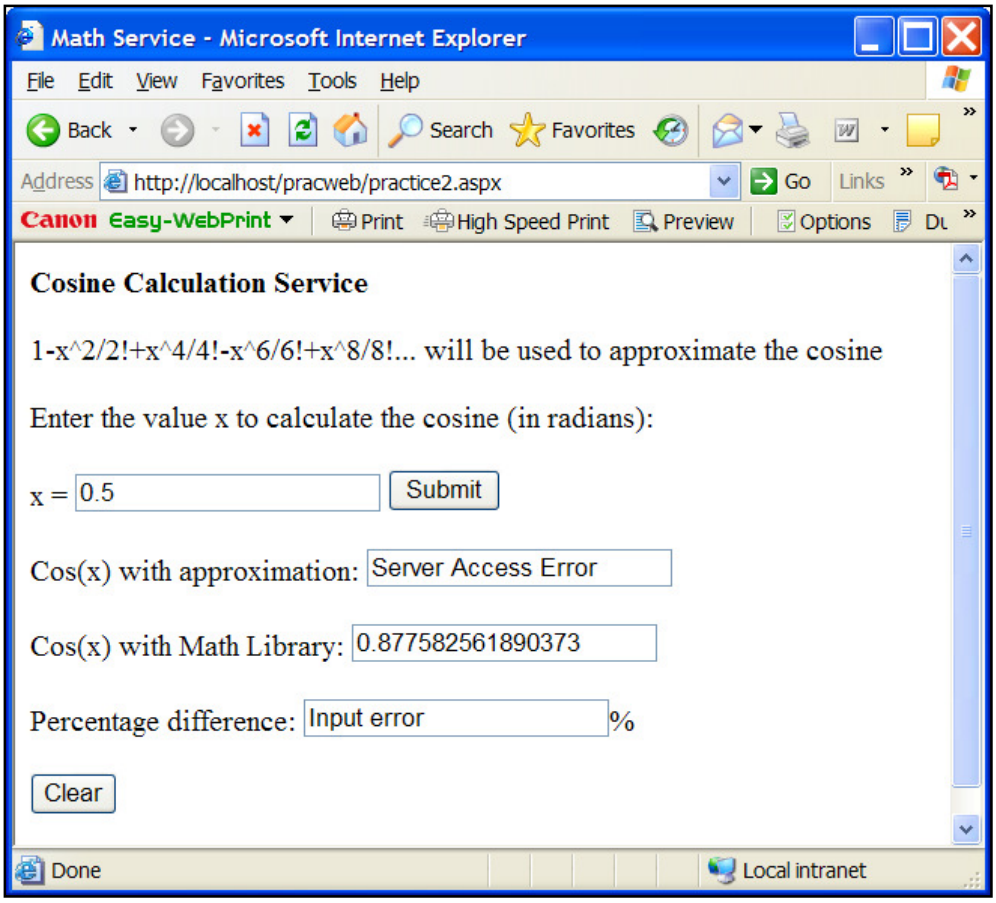


Figure 15: Results for Cosine Calculation Service when the MathServer is not running

The following ASP.NET code contains the functions that access the CosineService Library and that implement the functionality of the buttons in the practice2.aspx web page:

```
<%@ Import Namespace="CosineService"%>
<script runat="server">
   void Submit_Click(object sender, EventArgs e)
   {
      approxValue.Text = Cosine.Approximation(xValue.Text);
      mathValue.Text = Cosine.MathLibraryValue(xValue.Text);
          pDiff.Text = Cosine.PDifference(mathValue.Text,
```

```
                                        approxValue.Text);
    }
    void Clear_Click(object sender, EventArgs e)
    {
        xValue.Text = "";
        approxValue.Text = "";
        mathValue.Text = "";
        pDiff.Text = "";
    }
</script>
```

The Submit_Click function calls the Approximation method in CosineService Library, which receives the results from the adapted results from the MathServer Application. The Clear_Click function simply clears the data inside the textboxes of the web page. The operation and results of the dynamic adaptation of the Cosine algorithm are the same as those discussed in the MathService explained earlier (see *Development of Math Service Prototype (Using .NET Remoting)*), including all the plots and the percentage differences obtained.

Seminars/Conferences Attended

Laboratory Safety

By: Roberto Torres, Health and Safety Office UPRM

Date: February 20, 2007

Time: 10:45 AM

Place: Physics 229, UPRM

Critical thinking

By: Dr. Dimaris Acosta, Biology Department

Date: March 1, 2007

Time: 10:45 AM

Place: Physics 204, UPRM

Jr. Tech 2007 presentation of UPRM projects

By: PRLSAMP students, Dr. Efraín O'Neill and Engineering Mentors

Project: *Dynamic Adaptation of Software with TRAP.NET*

Date: March 8, 2007

Time: 10:45 AM

Place: Physics 319, UPRM

Puerto Rico Interdisciplinary Scientific Meeting, 42nd Junior Technical Meeting (JTM/PRISM 2007)

Oral Presentation: *Dynamic Adaptation of Software with TRAP.NET*

Date: Saturday, March 10, 2007

Time: 11:40 AM

Place: E-242, Inter American University at Bayamón, P.R.

Engineering Ethics a Comparative Perspective

By: Dr. Juan Lucena

Date: March 29, 2007

Time: 10:45 AM

Place: Dr. Ramón Figueroa Chapel Amphitheatre, UPRM

Research articles discussion round tables

By: PRLSAMP students, Dr. Efraín O'Neill and Engineering Mentors

Article: *Using Logical Data Models For Understanding And Transforming Legacy Business Applications, IBM Systems Journal*

Date: April 26, 2007

Time: 10:45 AM

Place: Physics 319, UPRM

Conclusion

During this research experience it was demonstrated that the TRAP.NET System, through its Composer could communicate with a server, and be able to change code with it, and in this way affect client applications, all this while both the client and the server applications where running. This goal was achieved with a Console Math Service application using .NET Remoting and a Web Service version using Windows Internet Information Services and the ASP.NET technology. This proofs that dynamic adaptation of software, especially in long running systems is a viable solution. Nevertheless, there are still many areas that need more research, like safeness in adaptation, analysis of the dependencies of applications, state transition of involved objects or components in adaptation. Also, the application of TRAP.NET to existing applications, for example open-source software is a remaining challenge.

Also, as part of the research experience, I benefited from various seminars and workshops, some of them sponsored by PR-LSAMP and others recommended by Dr. Rivera-Vega. One of the most important was the Puerto Rico Interdisciplinary Scientific Meeting, 42nd Junior Technical Meeting, in which I had the chance to give an oral presentation of our team project entitled: *Dynamic Adaptation of Software with TRAP.NET*. Another important event was the Research articles discussion round tables activity in which I had the chance to present the article: *Using Logical Data Models for Understanding and Transforming Legacy Business Applications, IBM Systems Journal.*

During the past semesters I worked remotely with the TRAP.NET Research group in Florida International University, with the creator of the TRAP concept, Dr. Sadjadi and the Project Manager Fernando Trigoso. I was also under the supervision of Dr. Pedro I. Rivera-Vega, here in the University of Puerto Rico at Mayaguez. During this under-graduate research experience I received support from the personnel of both universities, both online support with the FIU team and personal meetings with Dr. Pedro Rivera-Vega in UPRM.

Thanks to this experience I had the chance to work closely with faculty mentors, and graduate students, which helped me, get a perspective of graduate school and research life. It also served to improve my social interaction and communication skills, and it signified an important step toward graduate school.

Acknowledgements
This work was supported in part by the National Science Foundation grant REU-0552555 and IBM. Any opinions, findings and conclusions or recommendations expressed in this material are

those of the author(s) and do not necessarily reflect those of the National Science Foundation nor IBM.

References

[1] The Sine and Cosine series

http://www.ucl.ac.uk/Mathematics/geomath/level2/series/ser11.html

[2] Remoting with C# and .NET: Remote Objects for Distributed Applications, David Conger, Wiley; 1 edition (January 3, 2003)

[3] TRAP.NET Summer 2006 Advance Topics, Status Report, Fernando Trigoso, Allen Lee, Tuan Cameron, Ana Sanchez.

[4] Walkthrough: Creating ASP.NET Web Application Root Directories in IIS

http://msdn2.microsoft.com/en-us/library/ha2y9493(vs.80).aspx

[5] Getting Started - Creating Web Sites

http://msdn2.microsoft.com/en-us/library/xasa2c3y(VS.80).aspx

[6] Introducing ASP.NET 2.0, Anand Narayanaswamy

http://www.c-sharpcorner.com/UploadFile/anandn_mvp/IntroducingASP.NET2.005292006112235AM/IntroducingASP.NET2.0.aspx

[7] Local IIS Web Sites

http://msdn2.microsoft.com/en-us/library/ckk1e6z4(VS.80).aspx

[8] Keeney, J., "Completely Unanticipated Dynamic Adaptation of Software", Ph.D. Thesis, Department of Computer Science. 2004. Trinity College Dublin, Dublin, Ireland.

https://www.cs.tcd.ie/John.Keeney/pubs/JohnKeeney_PhD_Thesis.pdf

[9] Long Running Services and Service-Oriented Architecture

http://dev2dev.bea.com/pub/a/2004/05/soa_rao.html

[10] S. Masoud Sadjadi, Philip K. McKinley, and Betty H.C. Cheng. Transparent shaping of existing software to support pervasive and autonomic computing. In Proceedings of the first Workshop on the Design and Evolution of Autonomic Application Software 2005 (DEAS'05), in conjunction with ICSE 2005, St. Louis, Missouri, May 2005.