International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:11, 2013

The Implementation of the Multi-Agent Classification System (MACS) in Compliance with FIPA Specifications

Mohamed R. Mhereeg

***Abstract*—**The paper discusses the implementation of the MultiAgent classification System (MACS) and utilizing it to provide an automated and accurate classification of end users developing applications in the spreadsheet domain. However, different technologies have been brought together to build MACS. The strength of the system is the integration of the agent technology with the FIPA specifications together with other technologies, which are the .NET widows service based agents, the Windows Communication Foundation (WCF) services, the Service Oriented Architecture (SOA), and Oracle Data Mining (ODM). The Microsoft's .NET widows service based agents were utilized to develop the monitoring agents of MACS, the .NET WCF services together with SOA approach allowed the distribution and communication between agents over the WWW. The Monitoring Agents (MAs) were configured to execute automatically to monitor excel spreadsheets development activities by content. Data gathered by the Monitoring Agents from various resources over a period of time was collected and filtered by a Database Updater Agent (DUA) residing in the .NET client application of the system. This agent then transfers and stores the data in Oracle server database via Oracle stored procedures for further processing that leads to the classification of the end user developers.

***Keywords*—**MACS, Implementation, Multi-Agent, SOA, Autonomous, WCF.

I. Introduction

VARIOUS methods for classification of End Users Computing (EUC) activities have been proposed in the literature [1]-[3], [6], [9]. Although these methods provide classification solutions of end users, these classification schemes rely on the use of questionnaires both paper-based and web-based or interview techniques to gather the necessary data for analysis. These gathering techniques suffer from a number of problems such as validity (the degree to which the gathered data is correct or true, and represent the development activities of a user precisely) and reliability (the degree to which the gathered data can be relied on to categorize end users). Additionally, classification techniques normally require revisiting applications users have been developing in order to monitor and record any changes to knowledge and skills users gain and improve over a period of time, this is a time and effort consuming when it is managed manually, especially if the process requires monitoring and data collection for a long period of time. Classification results could suffer from lack of accuracy (the capability of providing the correct measurement or classification) when large amounts of data are involved. This paper proposes a new automated and distributed data gathering and classification system based on the software Multi-Agent technology.

This technique will save time and effort required to complete and return questionnaires and avoids going through unnecessary interviews. The .NET Windows Service based agents will be utilized to automate the monitoring and data gathering process that leads to the classification of end user developers. The Monitoring Agents (MA) were configured to execute automatically, without any user intervention as windows service processes in the .NET web server application of the system. Additionally, these agents function autonomously to provide continuous and periodic monitoring of excel spreadsheet workbooks. These agents listen to and read the contents of workbooks development activities in terms of file and author properties, function and formulas used, and Visual Basic for Application (VBA) macro code constructs.

Data gathered by the Monitoring Agents from various resources over a period of time will be collected and filtered by a Database Updater Agent (DUA) residing in the .NET client application of the system. This agent then transfers and stores the data in Oracle server database via Oracle stored procedures for further processing that leads to the classification of the end user developers. The .NET WCF services were used for the distribution of the agents over a network to satisfy the monitoring and classification of the multiple users approach. Spreadsheet applications will be used as end users workbench in order to evaluate the system; the reason is that many companies rely on spreadsheets as a key tool in their financial reporting and operational processes. As a result, the use of spreadsheets is an integral part of the information and decision-making framework for these companies [8]. Oracle data mining classification algorithms: Naive Bayes, Adaptive Bayes Networks, Decision Trees, and Support Vector Machine were utilized to analyze the results from the data gathering process in order to automate the classification of excel spreadsheet developers.

Thus MACS is an attempt to remove the organization's reliance on the human collection of data typically required by most business applications, and replace it with a tool that is capable of providing automatic, consistent and accurate data gathering and classification of spreadsheet developers.

M. R. Mhereeg was with Glamorgan University, Pontypridd, CF37 1DL, UK. He is now with the Department of Computer Science, Tripoli University, Tripoli, Libya, (e-mail: mmhereeg@msn.com).

II. Multiagent System Based On SOA

The introduction of web services allowed the developers to create applications that can communicate across platforms and programming languages over a network. Web services use a set of open standard protocols based on XML to expose functions (or methods) on the web as web services and allow clients to discover and make the synchronous calls to the exposed methods.

The Windows Communication Foundation (WCF) that is part of .NET Framework 3.0 introduces an extension to web service technology [7], [10]. The main advantages of this platform are that the WCF services and clients can participate in asynchronous service calls that enables the application to continue operating while the method call runs and, the ability to communicate over a variety of supported protocols, including HTTP, TCP, named pipes, and MSMQ. .NET 3.0 and newer versions offer a great support for applications built in Service Oriented Architecture (SOA) [4], [5]. The Service Oriented Architecture paradigm is presented in Fig. 1.

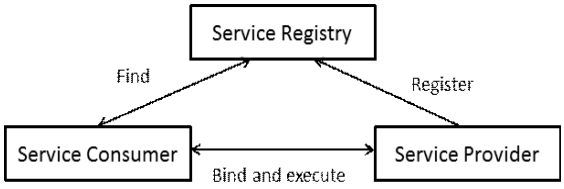

Fig. 1 Service Oriented Architecture Schema

In the SOA a system consists of the following three components:

Service Provider: is an application that exposes a service, accepts and executes requests from consumers.

Service Consumer: is an application that accesses a service over a transport protocol and, executes its functions.

Service Registry: is a directory that keeps information about exposed services, such as services' addresses, names, parameters, and returned values.

Comparing this paradigm with the MultiAgent architecture proposed by FIPA, one can find that SOA's *Service Registry* plays the same role as FIPA's *Directory Facilitator*, and the *Service provider* and *Service Consumer* are both the agents. These agents according to FIPA are the programs that expose and consume some services. The role of Message Transport System is assigned to the web services or its extensions (WCF Services). The MultiAgent Classification System (MACS) has been built in .NET Framework based on SOA. This gives the system more elastic and moreover the interaction between the agents is simpler and its safety is on the upper level [4], [5].

Fig. 2 presents the skeleton of the proposed architecture of the MACS multi-agent system based on SOA. The *User Agent* uses the *Service Registry* to manually register and unregister the services of MACS during the design phase of the system. On launch, the *User Agent* queries the *Service Facilitator* requesting the services available in the system for consumption, the agent then supplies the *Database Updater Agent* with the names and addresses of these services. The *Database Updater Agent* starts to communicate with the *Monitoring Agents* using the supplied addresses of the WCF services they expose that is to retrieve the gathered data. Data is then processed by the *Database Updater Agent* and uploaded to the data mining tools for classification purposes.

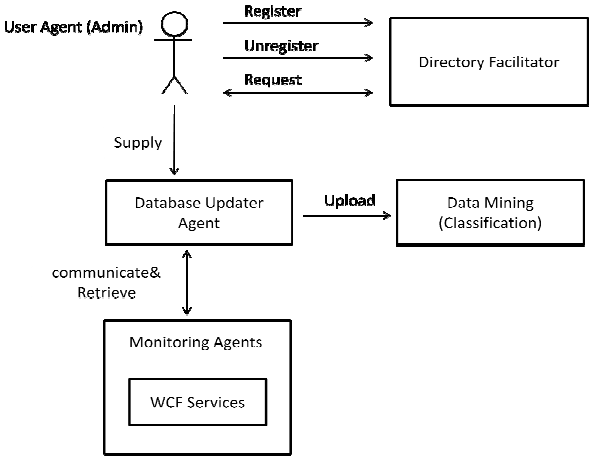

Fig. 2 The Skeleton of the Proposed Architecture of MACS

III. The Implementation Details

The main software applications that have been used for the implementation of the prototype are Microsoft's .NET Framework, Visual Studio.NET 2008, and Oracle Server Database. .NET framework was chosen as a development environment within which the agents can be created and communicate to achieve their goals. It provides a feature-rich application that consists of two parts: the Common Language Runtime (CLR) and a unified set of class libraries. These two components provide the execution engine for building .NET applications. The CLR is significant in making the .NET framework, platform and language independent. It also provides other important services such as security, memory management, and exception handling. The class library includes many attractive features, including Consol Applications, Graphical User Interface (GUI) applications (Windows Forms) for smart client applications, Windows Services, and Windows Communication Foundation services which are essentials for developing MACS multi-agent system.

Windows services were chosen as they enable developers to produce long-running executable applications that run in their own windows sessions. This concept was used in the construction of the Monitoring Agents (MAs) that reside in the server-side application to monitor and collect data from end user excel spreadsheets.

Windows Communication Foundation services technology that ships as part of the .NET framework 3.0 and newer versions has been used for the distribution of the agents over the University of Glamorgan's network. This technology is also used to manage the communication between the agents

because of the built-in support for the multi-protocol request-response communication, and because of the provision of WCF/SOAP reliable messaging transfer between the agents. Additionally, the .NET framework enables the asynchronous service calls participation between the WCF clients and services. A high-level performance comparison report [11] between WCF and the other Microsoft distributed communication technologies like: ASP.NET Web Services, Web Services Enhancements (WSE), .NET Enterprise Services, and .NET Remoting shows that in most cases, the performance is significantly better for WCF over the other technologies.

Visual C#.NET that is provided by Visual Studio.NET was chosen as the programming language for the development of the prototype because it is an object-oriented language, and was created specifically for .NET platform. Thus it has the best support for .NET applications. Oracle 10.2g was used as a storage device to store the gathered and analysed data over the life-cycle of the system. The .NET Data Provider for Oracle was the component used to provide access to Oracle Database. Oracle Data Mining is the technology used for the classification of the end users.

The prototype may be divided into the server-side and the client-side software applications as illustrated in Fig. 3. The sever-side consists of a number of Monitoring Agents running in different computers with appropriate local data repositories. The client-side application consists of the Database Updater Agent, the Service Facilitator to simplify the access between the agents in both sides of the prototype, and Oracle Server Database.

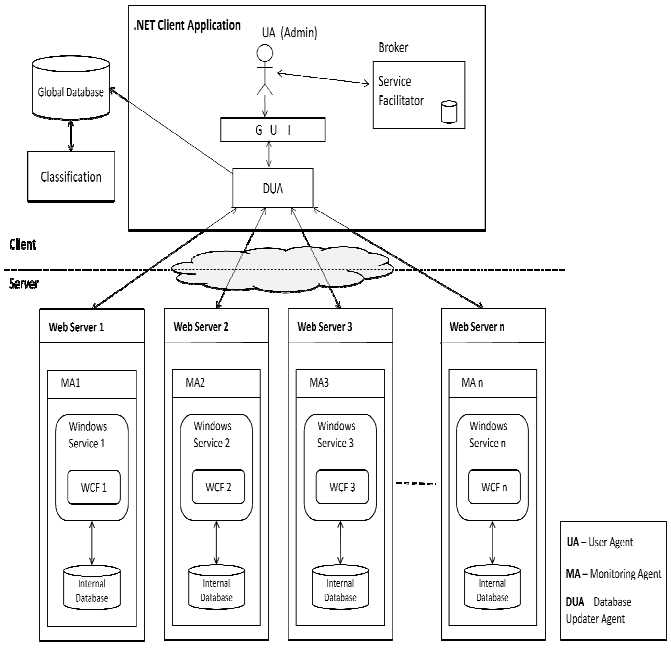

Fig. 3 MACS Architecture

A. The Server-Side

The Monitoring Agents are installed manually on a number of computers on the network of the University of Glamorgan, one MA per computer. However, the number of MAs (Service Providers) that can be installed is unlimited. These agents are identical and achieve exactly the same task of monitoring and data collection in different machines. Every agent has its own local data store within which the gathered data is stored. The agents are configured to start automatically when the computer within which they are installed boot. They can also be paused, stopped, and restarted manually using Visual Studio .NET Server Explorer. Thus this gives the user control over the agents and the capability to choose if they do not want to be monitored.

WCF services are hosted within the MAs for service exposure and to manage the communication with the Database Updater Agent residing in the client-side. The communication is based on XML SOAP messages, and TCP is used as the transfer protocol between the client-side and server-side agents. All the WCF services exposed by the MAs are registered manually in the Service Facilitator in order to simplify the access and ensure the availability of services. The following subchapter describes in details the implementation of the Monitoring Agent.

1. The Monitoring Agent

The MA is configured to start automatically when the user logs on and exist as an autonomous windows service process. Because it is a windows service based agent, it will execute continuously and run in the background as long as the computer is switched on. Using the FileSystemWatcher class supplied by .NET, the MA will detect and monitor any excel spreadsheets can be accessed by the user during its lifetime, and store the collected data in a local data store, Fig. 4 shows the basic algorithm of the MA.

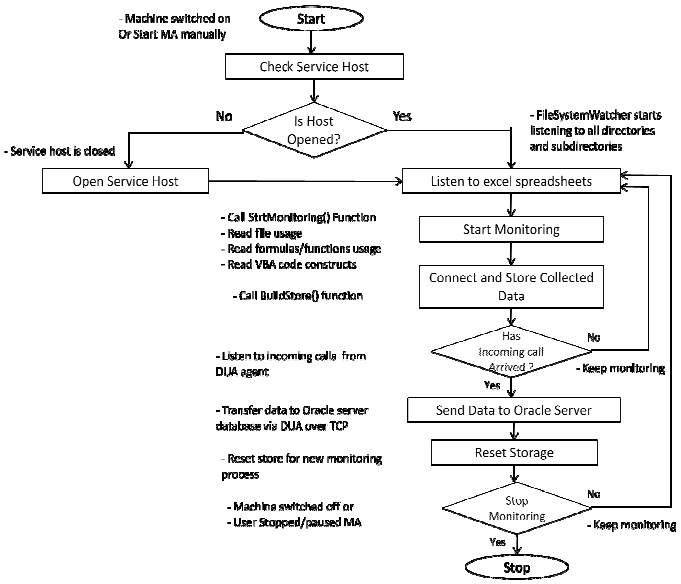

Fig. 4 The Basic Algorithm of The MA

The WCF technology is used to provide the communication

abilities between the MAs and the DUA agent in the client-side application. A WCF service is built and hosted in every MA agent. The WCF service's interface is defined that consists of one exposed method marked by OperationContract; Fig. 5 shows the service contract and the service class that implements it. After the interface and its method are defined, they are encapsulated in a class that implements the interface. The method returns a value of type string and responsible for the data return to the client-side application. The method has another task of clearing the *ArrayList* data store, thus the store can be reused for the monitoring activities to come.

```
// step 1: Define a service contract.
[ServiceContract(Namespace = "http://AutoService_StreamTCP_MiniNet.ServiceModel.Agents")]
public interface IAutomation_StreamTCP_MiniNet
{
    [OperationContract]
    string[] RetRangeArrayList();
}

// Step 2: Create service class that implements the service contract.
public class AutomaticService_StreamTCP_MiniNet : IAutomation_StreamTCP_MiniNet
{
    public string[] RetRangeArrayList()
    {
        String[] myArr = (String[])returnArrayList().ToArray(typeof(string));
        ClearArrayList();
        return myArr;
    }
}
```

Fig. 5 WCF Service Contract and the Service Class

A ServiceHost is constructed that is responsible for opening a communication channel for accepting the incoming calls, provided with an address where the service is located along with the service contract supported by the address, and provided with the supported communication protocol, which is the TCP protocol in this case. When the ServiceHost opens the channel for the service, this allows the DUA agent to make the asynchronous calls to invoke the method provided by the service. Thus the contents of the store can be transferred to the client-side application for further processing.

On launch of the MA, the *OnStart ()* procedure executes to check the availability of the service host that is responsible for accepting the incoming calls. If the host is closed, it is opened automatically by the MA. A communication exception is thrown if any communication problems occur (Fig. 6 shows checking a WCF ServiceHost).

```
public static ServiceHost svcHost = null;
protected override void OnStart(string[] args)
{
    if (svcHost != null)
    {
        svcHost.Close();
    }
    try
    {
        Uri baseAddress = new Uri("http://J345-MM.uni.glam.ac.uk:9000/ServiceModelAgents/Service");
        svcHost = new ServiceHost(typeof(AutomaticService), baseAddress);
        // Open the service host to accept incoming calls
        svcHost.Open();
        // Start Listening ...
        MA_Listen();
    }
    catch (CommunicationException commProblem)
    {
        Console.WriteLine("There is a communication problem. " + commProblem.Message);
    }
}
```

Fig. 6 Checking a WCF Service Host

Once this is achieved, the MA starts to listen to any percepts can be risen by users in the environment. When the user launches an excel spreadsheet, the agent automatically detects and opens that spreadsheet (see Fig. 7), opens all its workbooks and reads: the file properties (File Usage), all formulas the spreadsheet contains and VBA code constructs (Object and Code Usage) [12]. This is however the first part of the goal. The MA continuous to monitor the spreadsheet as long as the development process is still running, that is to read any updates the user makes. When the user hits the button *Save* the MA reads: the updated property details, that is in order to record when changes have been made, this achieved via using the property *DateLastSaved*. It also records the entire newly developed Object and Code usage patterns.

```
private Excel.Application ExcelObjf = null;
private void ReadFile(string FileName)
{
    try
    {
       // Construct an Excel Application Object
       ExcelObjf = new Excel.Application();
       // Here is the call to Open a Workbook in Excel
       Excel.Workbook theWorkbook = ExcelObjf.Workbooks.
       Open(FileName.Trim(), 0, true, 5,"", "", true,
       Excel.XlPlatform.xlWindows, "\t",false, false, 0, true, 1, 0);
       // Get the collection of sheets in the workbook
       Excel.Sheets sheets = theWorkbook.Worksheets;
       Excel.Worksheet workSheet = Excel.Worksheet)
       theWorkbook.ActiveSheet;
       foreach (Excel.Worksheet Sheet in sheets)
       {
          for (int i = 1; i <= 300; i++)
          {
            Excel.Range Range = Sheet.get_Range("A" + i.ToString(),  "IV" +
            i.ToString());
                    System.Array mValues= System.Array)Range.Cells.Formula;
                    int ArrayLen = mValues.Length - 1;
                    string[] ExcelArray = new string[ArrayLen + 1];
                    string[] ExcelArray1 = new string[ArrayLen + 1];
                    ExcelArray1 = ConvertToStringArray(mValues);
          }
       }
    }
    TheWorkBook = null;
    }
    catch (NullReferenceException nre)
    {
        Console.WriteLine(nre.ToString());
    }
}
```

Fig. 7 Opening a Spreadsheet

Once the monitoring task is complete, the MA connects to the *ArrayList* data store to be used for storing the gathered activity details. However, the store is made available to the DUA incoming calls any time during the process for data transfer, even if the MA is busy running monitoring activities. The WCF is the method by which the DUA agent accesses the *ArrayList* store in the server-side application. Thus it is essentially the interface to the data. Data collected from the ArrayList by the DUA is filtered and saved in Oracle server database for further processing. On completion of the monitoring and data transfer process, the MA reacts to either User logs out or user stops the MA manually. The OnStop() method then executes and closes the ServiceHost as shown in Fig. 8 before the MA stops.

```
protected override void OnStop()
{
     if (svcHost != null)
     {
          svcHost.Close();
          svcHost = null;
     }
}
```

Fig. 8 Stop MA Agents

The functionality of the MA is decomposed into different C# files created all together in a solution explorer that Visual Studio .NET provides. One represents the code needed to listen to and open the excel spreadsheets, and then expose the WCF service for data transfer. One represents the code needed to get the File properties, one represents the code needed to get formulas the spreadsheet contains, one represents the code needed to get VBA Code Constructs, and one represents the code needed to get the regular expressions. An xml configuration file is also created in a separate file to play the role of the communication with the client-side application. This partition enables the easy maintenance of the existing files and the introduction of any new features without affecting the rest of the code. These files are as the following:

- *WCF_HostIn_WS.cs*: contains C# code and represents the main code file for the project. It is used to achieve several tasks: (1) listens to and monitors any excel files being copied manually, opened for updating purposes, or created new in the local path C:\. (2) Executes methods from the other code files during the monitoring process, such as methods that retrieve the file properties, methods that retrieve formulas used, and methods that retrieve code structure patterns. (3) Store the collected data into the ArrayList local store. (4) Opens a communication channel with the UDA agent for the data to be transferred to the client application for further processing.
- *FileProperties.cs*: contains C# code used to read the file properties from the excel file under processing. The file uses the DSOFile.dll component provided by Microsoft to achieve this task.
- *Functions.cs*: contains C# code used to collect the type and number of all functions and formulas the excel spreadsheet under processing may contain. This file has the capability to read data from 249 different excel functions and formulas.

CodeConstructs.cs: contains C# code that scans through a VBA code and reads the features the code consists of, such as the Procedures, Functions, For loops, Do loops, and If statements a user has developed.

RegularExpressions.cs: contains C# code used for matching standards with the code constructs for string parsing and replacement. It scans through VBA code matching texts with patterns, that is in order to find and replace strings that take a defined format. For example, Regular Expressions have been used to parse Public/Private Sub components, Public/Private Function, Do Loops, and For Each components and so on.

- *App.config*: a configuration file is created to configure where the WCF service that every MA exposes is located, that is in order for the DUA to find it. This configuration allows providing the endpoint of the service and the service behaviour data at the point of deployment instead of the design time. Fig. 9 shows part of the configuration file of a service hosted in a MA running in a machine with a computer name *J345-MM.uni.glam.ac.uk*.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="AutoService_StreamTCP_MiniNet.ServiceModel.Agents.AutomaticService_StreamTCP_MiniNet"
               behaviorConfiguration="AgentServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress="http://J345-MM.uni.glam.ac.uk:9000/ServiceModelAgents/Service"/>
          </baseAddresses>
        </host>
        <!-- this endpoint is exposed at: net.tcp://lJ345-MM.uni.glam.ac.uk:8000/ServiceModelAgents/Service  -->
        <endpoint address="net.tcp://J345-MM.uni.glam.ac.uk:8000/ServiceModelAgents/Service"
                  binding="netTcpBinding"
                  bindingConfiguration="StreamedTCPBinding"
                  contract="AutoService_StreamTCP_MiniNet.ServiceModel.Agents.IAutomation_StreamTCP_MiniNet" />
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />
      </service>
    </services>
    <!--For debugging purposes set the includeExceptionDetailInFaults attribute to true-->
    <behaviors>
      <serviceBehaviors>
        <behavior name="AgentServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Fig. 9 app.config Configuration File

The <system.serviceModel> section contains the configuration information of the service. The <services> section contains the details of the service implemented. The <service> element specifies the namespace and class that implement the service.

The <endpoint> element provides details of the service that the DUA requires in order to find and communicate with the service. The endpoint consists of three important pieces of information: an address, binding, and contract. The address refers to the location that the MA hosting the service uses to advertise the service. The binding element specifies the mechanism and protocol used to access the WCF web service. A contract defines an agreement of how DUA should communicate with the service. An extra element has been added called *BindingConfiguration* as the streamed technique is used for the data transfer over TCP rather than the default buffered technique over HTTP.

The <behaviours> element indicates that a service metadata is published for the service. The attribute *HttpGetEnabled* is set to "**true"** to allow the metadata to respond to metadata request made by the client using HTTP/GET request. This means that the client application can retrieve the metadata of the service, so that the DUA can use the information provided by the metadata to find/access the service.

2. Detecting File Changes

The FileSystemWatcher is a component provided by Microsoft .NET Framework, which allows developers to connect to directories and watch for specific changes occur to the files and subdirectories within it. This component is utilized by every MA to monitor the local hard drive C:\ and detect any excel spreadsheet changes. However, the MAs have the capability to monitor any hard drives exist in the

monitored computer like drives D, E, F…etc. depends of the partition of the machine. The User Agent is responsible for changing the code via replacing the path C:\\ (*watcher.Path = "C:\\";*) with any other paths the MAs are required to monitor, for instance, when changing from path C:\\ to D:\\, the code-line should look like (*watcher.Path = "D:\\";*)

Upon start, the MA instructs the FileSystemWatcher class to start the monitoring process and then the data collection process that is managed by the MA follows whenever a new excel file is created or an existing file is updated. Fig. 10 shows the FileSystemWatcher initialization.

As shown below, a FileSystemWatcher object is created and its properties are set to include all subdirectories of drive C:\. There is a restriction on the file type that it looks only for changes on files with the extension *.xlsx*; this prevents the event handler of being called unnecessarily. The event handler is added to identify the method needs to be called whenever a file is created or updated. Every time the event handler is executed, it calls the method named *Start_Monitoring()* that collects the File Usage attributes, Object and Code Usage Patterns, and some other features like charts and pivot tables and then the local store is populated with the collected data. The *EnableRaisingEvents* property is set to "**true**" to initiate the monitoring process.

```
public void MA_Monitor()
{
    FileSystemWatcher watcher = new FileSystemWatcher();

    watcher.Path = "c:\\";
    watcher.NotifyFilter = NotifyFilters.FileName | NotifyFilters.LastWrite
                         | NotifyFilters.DirectoryName;
    // Watch excel 2007 spreadsheets
    watcher.Filter = "*.xlsx";
    watcher.IncludeSubdirectories = true;
    // Add event handlers.
    watcher.Created += new FileSystemEventHandler(OnChanged);
    watcher.Changed += new FileSystemEventHandler(OnChanged);
    // Begin watching.
    watcher.EnableRaisingEvents = true;
}
// Define the event handlers.
private void OnChanged(object source, FileSystemEventArgs e)
{
    // Monitor when a file is created or changed
    Start_Monitoring(source, e);
}
```

Fig. 10 FileSystemWatcher Initialization

B. The Client-Side

The client-side application initiates the connection with the server-side application and consumes the WCF services that its agents expose. The WCF client is the component used to manage the communication. The UA is responsible for retrieving the metadata manually from the WCF services and use it to create a WCF client that the DUA (i.e. Service Consumer) can use to access the services. These services must be up and running in order for the metadata to be obtained. The UA utilizes the *ServiceModel Metadata Utility* tool provided by .NET Framework to achieve this task. This command-line tool obtains the metadata from the services running in the web server application and generates the required client proxy in C# language. In addition to creating the client proxy, the tool also generates the configuration file that enables the DUA agent to connect to the services using their endpoints this file provides. The following example shows how the tool can be used with the appropriate switches to generate the proxy file together with the configuration file for the service located in the below URI address:

svcutil.exe /language:cs /out:generatedProxyStreamTCP.cs /config:app.config http://J345-MM.uni.glam.ac.uk:9000/ Service ModelAgents/Service /async

The /async switch generates an asynchronous client version of the service. This can be done for any number of services the client application wants to connect to. A metadata for multiple services can be obtained at the same time using the same Svcutil.exe command shown above via just adding the addresses of the services following each other. Fig. 11 shows the outcome of the command.

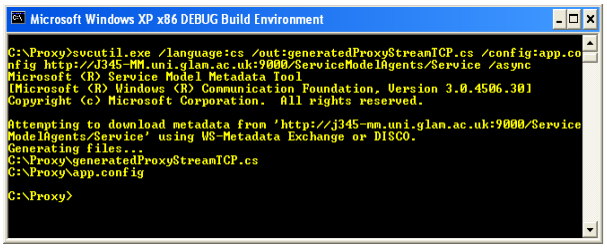

Fig. 11 The Svcutil.exe Command

The *generatedProxyStreamTCP.cs* C# proxy file and the *app.config* configuration file are both added manually to the client project within the Solution Explorer in Visual studio .NET. These files are the keystone that will be relied on to manage the communication process.

A .NET Windows Form based GUI has been constructed. This GUI is used by the UA to query the SF database for the available services the DUA can request data from. The *Call Service* button is used to start the DUA agent to call the selected service in the ComboBox area. Fig. 12 shows a successful call to the service named J345_mm.

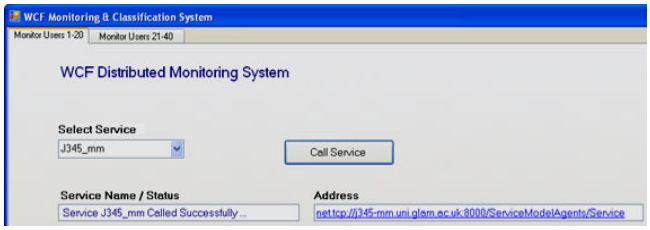

Fig. 12 The GUI of MACS

This process can be done asynchronously for all the services exposed by the MAs in the web server application. Data collected is analyzed by the DUA and stored in Oracle tables using Oracle Stored Procedures. The classification process of the data is the topic of the next paper followed by the Evaluation and Results.

1. The DUA Agent

The DUA is responsible for accessing the MAs in the server-side application through the WCF services they expose

to retrieve the outcome of the monitoring process. However, after the WCF client has been created that includes information about the types and methods that are exposed by the services; the DUA creates an instance of the client and invokes the methods asynchronously. Making asynchronous calls is important for the client-application; this avoids blocking the GUI thread while the method call runs. Thus, it is possible for another call to be made while the current one is still running. This provides WCF applications even more flexibility to maximize throughput balanced against interactivity [13]. In order for the DUA to call the WCF services asynchronously, it creates a *Callback* function that needs to be called when the asynchronous operation is complete. The DUA then calls the service and the *Callback* function is specified in the call. Once this is done and a successful call is made, the *Callback* function executes to retrieve the result (i.e. contents of the MAs' local stores. Fig. 13 shows an asynchronous call to a service named J345_mm.

```
public void service_J345_MM()
{
    Automation_StreamTCP_MiniNetClient client = new Automation_StreamTCP_MiniNetClient();
    IAsyncResult arRetRangeArrayList = client.BeginRetRangeArrayList(RetRangeArrayListCallback, client);
}
public void RetRangeArrayListCallback(IAsyncResult ar)
{
    String[] result = (((Automation_StreamTCP_MiniNetClient)ar.AsyncState).EndRetRangeArrayList(ar));
    ........
    SendResult(result);  // Send result to Oracle table
    ........
    Filtering(result);   // Send result for filtering
}
```

Fig. 13 Asynchronous Callback to the Service J345_mm

All services have to be up and running in the server-side in order for the DUA to make a successful retrieval of data. If a service for example is stopped (i.e. the MA hosting the service is not running), a communication exception is thrown; this appears in the GUI as an indication of failed connection. For successful calls, the length of the variable *result* shown in code 5.7 is measured, if the string is empty, a message will appear in the GUI indicating that no records have been retrieved. If the *result* contains records, a copy of the records is stored in Oracle table using *SendResults* function that is in order to keep track of the history of development of end users. Another copy of the results is sent to the function named *Filtering* for filtering purposes. This process involves searching through the records, finding all the spreadsheets that a particular user developed during the monitoring process, and selecting the last updated version of every spreadsheet. The reason behind searching for the last updated versions is that because these spreadsheets contain all the features that end users developed over time before closing the spreadsheets. In other words, the users are expected to save their work a number of times before they finish via hitting the *Save* button. Thus, the last updated version of every spreadsheet is the most valuable one. Other measurement techniques like complexity and size of the spreadsheets could be used as alternatives to select the spreadsheets that will take part in the classification phase of the project. These techniques were dismissed because it was found that a number of spreadsheets developed by the same users are exactly the same in terms of complexity and size, as some users click the button *Save* twice or more without adding any new features to their applications. Users in some other cases may open and close their applications without making any changes; this again yields spreadsheets with the same complexity and size. Thus searching for the last updated version of every spreadsheet for every individual user has resolved the problem. These records are stored in a different oracle table to be used in a later stage in the data mining process.

In the end of both cases of filtering and non-filtering processes, the DUA builds up the parameter sets required to use Oracle Stored Procedures and populate the parameters accordingly.

A connection to Oracle database is then made, and data is uploaded into the appropriate tables via the execute method of the connection object. Fig. 14 shows the basic algorithm of the DUA.

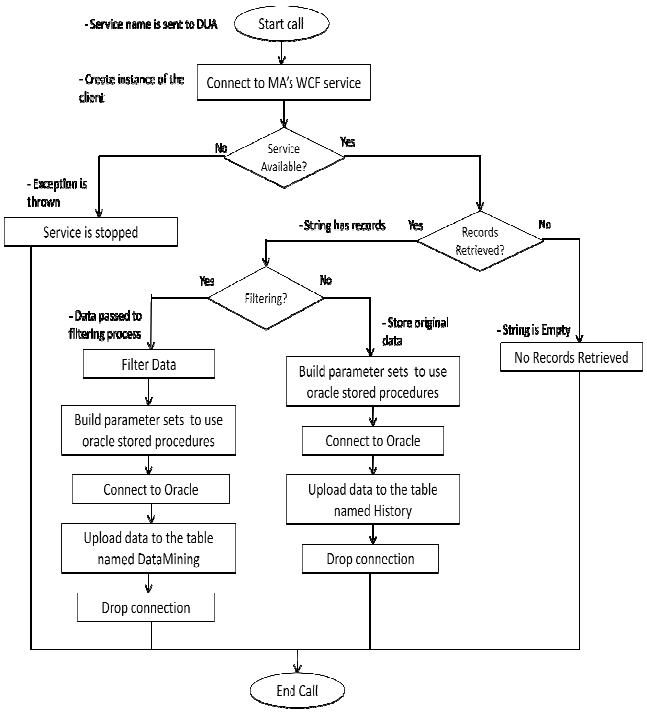

Fig. 14 The Basic Algorithm of the DUA

2. The Service Facilitator (SF)

In FIPA specifications, the Directory Facilitator (DF) is an optional component of the MultiAgent System. In contrast, in MACS the Service Facilitator (SF) is considered as mandatory because the system is being implemented with respect to SOA architecture. The SF is being used to advertise the WCF services of the system the MAs expose. The SF database is built in Oracle. It keeps the information about the services, like the services' names, addresses and returned values. On launch of the client application, the SF responds to the UA queries requesting the services registered in the system. Fig. 15 shows querying the SF for registered services.

```
// Connect to Oracle Server Database
OracleConnection objConn = new OracleConnection("User Id=sys; Password=pword; Data Source=or1;");
OracleCommand objCmd = new OracleCommand();
OracleCommand objCmd1 = new OracleCommand();
objCmd.Connection = objConn;
objCmd1.Connection = objConn;
// Get the Numer of services available
objCmd.CommandText = "get_count_SF_tab";
objCmd.CommandType = CommandType.StoredProcedure;
objCmd.Parameters.Add("return_value", OracleType.Number).Direction = ParameterDirection.ReturnValue;
try
{   objConn.Open();
    objCmd.ExecuteNonQuery();
    count = objCmd.Parameters["return_value"].Value;
}
catch (Exception ex)
{  MessageBox.Show(ex.ToString());  }
try
{   objCmd1.CommandText = "select * from Agents_Expose_Services_tab";
    OracleDataReader dr = objCmd1.ExecuteReader();
    while (dr.Read())
    {   for (int j = 0; j < count; j++)
        collect[i, j] = dr.GetString(j);
        i += 1;  }
}
catch (Exception ex)
{  MessageBox.Show(ex.ToString());  }
    ..............
    ..............
objConn.Close();
objConn.Dispose();
```

Fig. 15 Querying the SF for Registered Services

A list of all available services is provided and appeared in a *ComboBox* area on the GUI. The UA can then instruct the DUA to communicate with the desired services to achieve the task of the data collection. The registration and unregistration of services is managed manually by the User Agent during the design stage of MACS. New services can be registered and unwanted current services can be unregistered to/from the SF as MAs are added or removed from the system. The information the SF provides allows the DUA to communicate with the desired services, instead of calling all or discarded services.

3. Oracle Server Database 10g Release 10.2

The data collected from the server-side application via the DUA is deposited in an Oracle Server Database. This data will form the basis for classifying the spreadsheet developers' competency using Oracle Data Mining tool. However, .NET Framework Data Provider for Oracle is the technique used to upload the data into the appropriate tables. ODP.NET is used because it enables .NET to access Oracle in a more efficient, flexible, and more stable manner than other techniques like OLE DB, and ODBC. The data is received and inserted into the tables using Oracle Stored Procedures.

IV. Conclusion

It has been demonstrated that the .NET Windows Service based agents can be utilized together with the Microsoft's FileSystemWatcher component to function automatically and autonomously to provide continuous and periodic monitoring of Excel spreadsheet development activities. The development of monitoring agents in .NET Framework and the existence of the WCF services produced agents that are fully distributed over the WWW. The creation of the WCF services proxies added an extra advantage, is that these agents are made accessible by the client application through the WCF services they expose without worrying about how the services are delivered or where they are hosted. This technology also provided the system with the capability to obtain a reliable end-to-end streaming transfer between SOAP endpoints. MACS has been built based on SOA, this supplied the system with more flexibility and the interaction between the agents is simplified. The existence of the Service Registry that SOA provides made it even easier for the DUA to find and communicate with the agents. The decomposition of the agents into C# classes enables the easy maintenance of a specific class without affecting the rest of the agent's code. This also provides the flexibility of introducing new classes to an agent as the service it exposes expands.

References

[1] Agentcities - A Global, Collaborative Effort To Construct An Open Network Of On-Line Systems Hosting Diverse Agent Based Services, Http:// Www.Agentcities.Org, (Last Visited 2010).

[2] Ahmed, S. (2007) Analysis Of Workplace Surveillance In A Quest For An Ethical Stance. *Journal Of Business Systems, Governance And Ethics*, 2, 4, 15-26.

[3] Bellifemine, F., Poggi, A. & Rimassa, G. (1999) Jade - A Fipa-Compliant Agent Framework. *Proceedings Of The 4th International Conference On The Practical Applications Of Agents And Multiagent Systems Paam99.* The Practical Application Company Ltd., Pages: 97-108.

[4] Blackwell, A. (2002) What Is Programming? *14th Workshop Of The Psychology Of Programming Interest Group.* Brunel University.

[5] Brigham, F. & Daves, R. (2009) *Intermediate Financial Management,* Usa, South-Western, Cengage Learn

[6] Cammarata, S., Mcarthur, D. & Steeb, R. (1983) Strategies Of Cooperation In Distributed Problem Solving. *Proceedings Of The Eight International Joint Conference On Artificial Intelligence, Pages 767-770.*

[7] Chris, P., Dennis, M., Shawn, C. & Amit, B. (2007) *Pro Wcf: Practical Microsoft Soa Implementation,* Usa, Apress.

[8] Codasyl End-User Facilities Committee Status Report. Information Management Two,North Holland. 1979, 137-163.

[9] Cotterman, W. & Kummar, K. (1989) User Cube: A Taxonomy Of End Users. *Communication Of The Acm,* 32, 1313-1320.

[10] Dictionary Of Computing. Oxford University Press, 1983.

[11] Gupta, S. (2007) A Performance Comparison Of Windows Communication Foundation (WCF) With Existing Distributed Communication Technologies. Http://Msdn.Microsoft.Com/En-Us/Library/Bb310550.Aspx. Microsoft Corporation - MSDN, (Last Visited, 2010).

[12] Mhereeg, R. Mohamed. (2012), "The Development Of The Multi-Agent Classification System (MACS) In Compliance With Fipa Specifications", *World Academy Of Science, Engineering And Technology, Waset, France,* Pp.1285-1291, PISSN 2010-376x, EISSN 2010-3778

[13] MSDN. (2010) Synchronous And Asynchronous Operations. Http://Msdn.Microsoft.Com/En-Us/Library/Ms734701.Aspx. Microsoft Corporation - Msdn, (Last Visited, 2010).