A complete software application providing automated measurements storing, monitoring and feedback for dispersed environmental sensors.

P. N. Christias*, A. Maitos**, E. Vogklis***

*Technical Educational Institute of Piraeus – Department of Electronics.
xristias@tee.gr
**Technical Educational Institute of Piraeus – Department of Electronics.
amaitos@teipir.gr
***Technical Educational Institute of Piraeus – Department of Electronics.
e-voglis@otenet.gr

Abstract

The present paper describes the development and the specifications of a software solution that manages information and measurements data, concerning sensor networks. The sensors measure environmental parameters (CO2, CO, Temperature, Humidity and Luminance) and they are installed in geographically dispersed stations. The measurements are collected in a personal computer or data logger installed at every station in the form of text files. A constantly running software scheduler in the measurements PC automatically transmits these files to a main Server via Internet. The measurements files are processed by a software daemon in the main server, the reading values are attributed to the corresponding sensors and finally the daemon stores the measurements information in a database. In addition, the database is designed to hold station, sensor and data files attributes. A Microsoft WinForms application is developed which acts as a client interface. It supports administration tasks on the objects stored in the database. Its main feature is that all the readings per sensor, station or environmental parameter can be monitored through the graphical interface. A separate library is developed which produces graphs showing measurements information, according to the preferable parameters for multiple sensors, either they belong to the same station or not.
All the software components are developed using the Microsoft's .NET platform. DB4o, an object-oriented database is selected to cooperate with the WinForms application and the software daemon for data storage and retrieval. The technology of Microsoft XML Web Services is used in order to receive the files that arrive from each environmental station to the main server. The project is co-funded by the European Social Fund & National Resources – EPEAEK II – ARCHIMIDIS.

Keywords: web-based applications, .NET, Web Services, environmental measurements, sensors networks, object-oriented databases.

1. Introduction

The case study concentrates on the management of readings, relevant data information and metadata, regarding sensors measuring environmental parameters. The sensors are installed in groups, at stations in various geographic locations. Each environmental station is equipped with a personal computer (PC) / datalogger with transmitting capabilities. They serve as a collector for the measurements data which the sensors produce. The data coming from every station, show climatic variations based on specific parameters. They can be used to study outdoors and indoors environmental quality, or contribute to automated fire and flood prevention at remote areas.

The combination and comparison of readings originating from sensors of the same measuring type but located at different stations provide an overall image for the area's micro-climate.

A software application can manage all the relevant information about the environmental stations, the sensors and measurements in a database. A main server hosts the Web services and the database.

The measurements are stored by group in data files inside the station's PC. They are produced by the data loggers to which each sensor is connected. A software mechanism is implemented to automatically send all the data files to the main server. Data is transmitted between the stations PCs and the main server which can be interconnected via a LAN or a WAN. The transmission software module is installed in every station's PC and performs a daily check on the accumulated data files. In case the environmental station is remotely located, the PC automatically connects to the internet and establishes communication with the main server. The files are sent in the form of binary stream and after a successful transmission the PC automatically disconnects from the internet.

For the purpose of serving multiple requests for file transmissions a .NET Web Service is developed which resides in the IIS Web Server, currently installed on the main server machine. The Web Service receives the transmission attempts and the data files as advanced web requests. The measurements files are stored on the main server's hard disk for the total number of the registered stations.

A Windows Service which resides in the main server is designed to constantly check for newly received files from the web service. Each newly arrived file is passed as input argument to a custom method for processing. The processing separates the measurements values and corresponding timestamps for every sensor whose readings are included in the data file. This information is then automatically inserted in the database.

The data related to the physical network of environmental stations is administrated through a graphical user interface. It is a WinForms application from which all the operations in the database are dynamically performed:

1. Administration of the data attributes for the registered environmental stations.
2. Administration of the data attributes for the installed sensors in every station.
3. View of recorded readings filtered by sensor and timestamp.
4. Management of the measurements files which are sent to the main server by each environmental station. The number of columns contained inside the files, the type of data they represent and the values format can be edited.

In addition, a class library was developed for generating custom graphs. Using this feature, multiple sensor readings of one or more environmental parameters are graphically plotted in the same chart, presenting the measurements values for user selected time periods. In the following section, there is an overall description of the parts and functions of the entire application. In sections 3, 4 and 5, implementation issues of the application, the database and the interface are covered in detail. At the end, future approaches and conclusions are discussed.

2. Functionality Issues

The architecture of the proposed solution follows the multi-tier software design [MSDN]. The advantages are enhanced maintainability and scalability. Functionality, components and code are divided into separate tiers. The structural pattern of the application is shown in Fig. 1. The data loggers in every environmental station produce measurements files which are stored as digital files in the station PC‘s hard disk. The software method which is embedded in the business logic layer and is used to receive these files must be invoked. In case the main server is not reachable via a local area network, the file reception method can be invoked by the Web Service running on the main server. The station’s PC automatically establishes connection to the Internet and performs a request to the Web Service. After the files are transmitted it disconnects. A running Windows service has the task of processing all new data files, regardless of the time they arrived. The extracted measurements information is automatically inserted in the database via the aforementioned service. The graphical interface enables management of the properties and measurements data for the stations and sensors.

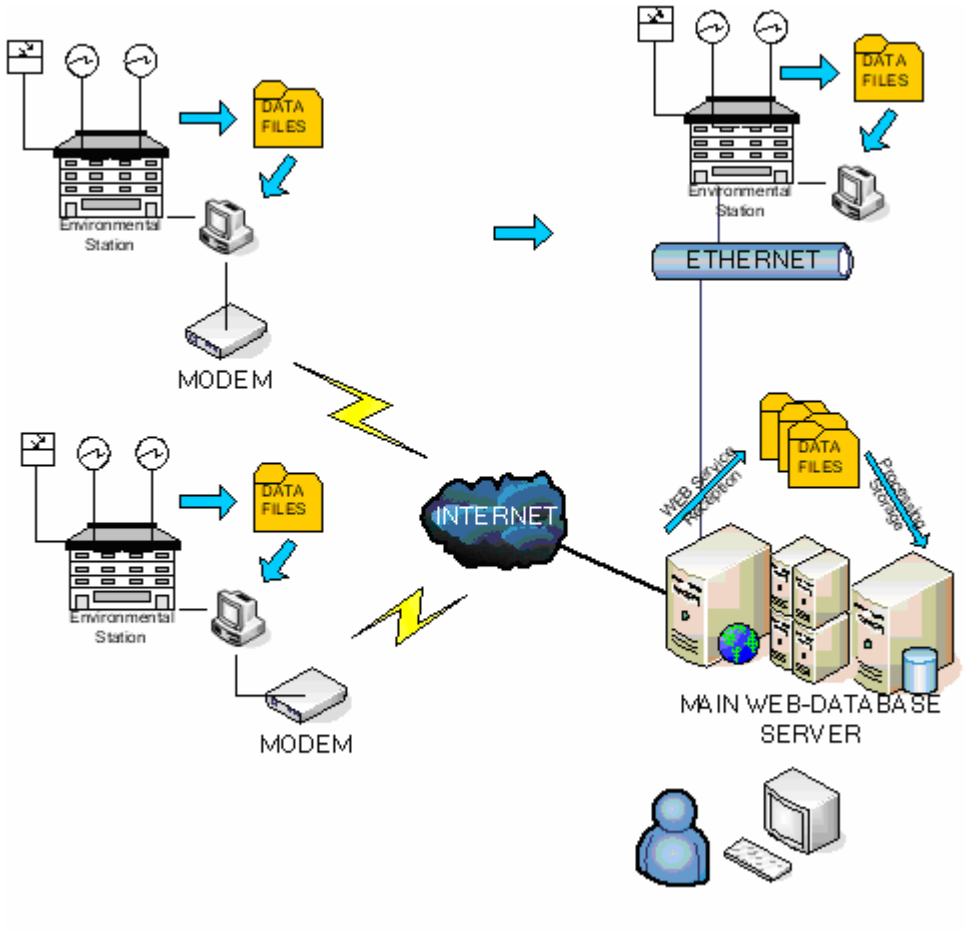

Figure 1. *The structure used for the proposed software application.*

3. The Business Logic Layer

3.1 General

The business logic layer contains all the class libraries which implement the logical and programmatic operations as well as the communication with other tiers and components, which are the database and the client components. Inside the business logic namespace the following parts are included:

1. The class definitions describing the entities of the physical model.
2. The Factory classes which are dedicated in implementing every operation on the objects above.

3. Classes that implement collections (groups of many object instances) of the custom objects.
4. A class library that encapsulates the parameters that must be available as options or choices to the user of the interface. The parameters are grouped in strongly-typed structures and enumerations.

3.2 Custom Objects and Factories

Above every object that is defined in a class in the software package, a prime object exists on top of the class hierarchy called *BusinessObject*. All the objects defined in the solution object model, inherit from the above base class and constitute the custom objects set. Following this design pattern [Gamma et. Al (1995)] the leverage of abstractness is achieved, while overall extensibility is enhanced. At this time the main custom classes that define the required entities for the object model are:

1. Station
2. Sensor
3. SensorMeasurement
4. MeasurementsFile

A detailed description of the attributes for the solution's objects is in Table 1.

The only attributes that are declared inside the *BusinessObject* class are *timeOfCreation* and *timeStamp*.They represent the time when the object was created and the time when some attribute of the object was changed and stored again in the database respectively. Except for the constructor methods which instantiate an object, no method which operates on the database and communicates with any form of client is declared inside a custom object's class definition. Separate classes called Factories contain all the method declarations for the object they represent [Gamma et. Al. (1995)]. That is, a Factory class exists for the *BusinessObject* class (*BusinessObjectFactory*) and for every child class. Factory classes contain static declarations of methods, so no instance of them is necessary in order to be called. They follow the exact inheritance schema of the custom objects. The operational separation of objects caused by the use of Factory classes provides controlled restriction as far the client layer is concerned. The client is only permitted to have object type declarations inside its code and is only permitted to call the Factory static methods for the object type that is referencing. The responsibility for the proper instantiation of objects and returning properly built objects as well as collections of them is assigned to the business logic layer. Another advantage is that the separation of execution members (methods) from their original class definition is very useful for the object database. The reason is that only data related members (attributes) are used for storage. The usefulness of inheritance though, is highly exposed when used in conjunction with Delegates and the new feature of the .NET Framework 2.0, Generics [MSDN].

Table 1. *Members list for all Entities*

ENTITIES	Station	Sensor	SensorMeasurement	Measurements file
ATTRIBUTES	name	name	measurement DateTime	filename
	measurementsFiles FolderName (The desired folder name for hard disk storage)	serialNumber	measurement Value	lineDataBegin (the line in file where measurements data begin)
	Description	description	sensorName	dateColumnNumber (the column where date is marked)
	address	parameterMeasured (CO, CO2,Temperature,etc)		timeColumnNumber (the column where time is marked)
	location	Units (ppm, Celcius, etc)		totalColumnNumber
	isActive (flag declaring whether the station is operational)	primaryDigits (The number of the measurement values decimal digits which are used for calculations)		errorString (the literal specifyingerroneous values in the file)
		isActive (flag declaring whether the sensor is operational)		dateTime Format (the format in which dateime is provided)
				decimalSeparator (the character which is uded as decimal separator in measurement values)

Delegates work the same way as function pointers. They behave as data structures that refer to a static method or to a class instance and an instance method of that class. When a call to the delegate happens, the method passed as input argument is executed. The Object Database API for C# operates on stored objects with exposed delegates that accept methods as input arguments. With the use of a delegate the type

of operation such as insert, update, delete or retrieve is predefined along with the total number and the types of input arguments and the expected return type. Although the signature of the method to supply the delegate input argument is predefined, a restriction exists on the specified data types of input and output. The use of Generics overcomes this restriction. Any type of a custom object can be passed as input arguments or declared as return type. This eliminates the burden of rewriting the same methods which perform the same operations, for each different object at a time. The total gains are generalization of input and specialization in operation at the same time:

1. The methods for the main operations on the database are written only once and they are isolated in the *BusinessObjectFactory* class, maximizing reusability, safety and performance.
2. The above can be executed for any object type in the solution performing any set of instructions on the different attributes of that object.

Along with the Factory classes, Collection classes are developed for every custom object beginning from *BusinessObject*. They are structured in an inheritance tree with *BusinessObjectCollection* being the base collection class. The *BusinessObjectCollection* implements the .NET Framework's *IList* <*T*> interface. The purpose of the Collection classes is to provide strongly-typed collections for groups of custom objects instances. At the time when the client makes a call to a custom's object Factory method, it uses the *BusinessObjectFactory* corresponding method by passing the specific – specially designed for that custom object – delegate for execution. In the case the return value is a collection of custom objects, the correct casting is applied. This is necessary because the *Query()* method for example, returns an *IList* <*T*>. Although at run-time the custom objects type is viewable, in order for the items in the collection to be treated by the client as objects of the specific type, they must be transferred from the collection of <*T*> to a new strongly-typed collection. That is the reason why a Collection class is built for every custom object. The client is permitted to declare custom objects Collections which are instantiated by the return values of the Factory methods. At client-side any processing is possible on each item of the collection. Moreover, every processed collection can be provided back as input argument to a different Factory method.

3.3 Web Service

XML Web Services provide functionality to Web users through a standard Web protocol. In most cases, the protocol used is Simple Object Access Protocol (SOAP) [Short (2002)]. SOAP is the communications protocol for XML Web services. It is a specification that defines the XML format for messages. Web services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a Web Services Description Language (WSDL) document. One of the primary advantages of the XML Web services architecture is that it allows programs written in

different languages on different platforms to communicate with each other in a standards-based way. The web service behaves like a common method. It has input arguments and return value. Furthermore, it is exposed for use on the Web through a URL. The solution's Web Service accepts as input arguments, the filename and the byte array of the file being sent. When the file is successfully received by the web service − this is verified by checking the byte array length before and after transmission − the file is recomposed to its original text format using the byte array. A Boolean true value is sent back verifying successful reception. Data exchange using a file or binary stream is not supported for Web Services. Methods are developed for this data exchange, which decompose the files into byte arrays and recompose them back again after reception.

The benefits the Web Services technology offers are:

1. Information can be exchanged in XML, one of the most widely used protocols.
2. Feedback can by supplied to the environmental stations because web services provide return values.
3. The use of Web Services allows the multi-user and portable expansion of the solution.
4. The extension of the client layer to a web application is easier achieved.

Those benefits will be analyzed in section 5.

3.4 Automated Tasks

The processing of the measurements files on the server and their transmission from the station's PC are operations executed at periodic time intervals. The methods which process the data files are embedded in a Windows Service module. This means the processing task runs repeatedly using the kernel resources as a daemon in a separate system process. The time interval specifying the check frequency on newly arrived files is a configurable parameter and can be altered by the server's administrator in the configuration file of the Windows service (subsection 3.5).

The measurements files transmission operations are included in a standalone package which executes as a scheduled task on the stations PCs. It makes use of the 'wininet.dll' library deriving from the earliest versions of Microsoft Windows. The library's unmanaged methods are called from inside the .NET code to establish, monitor and disconnect an internet connection.

4. The Implementation of the Database

4.1 Technology Description

The database management system (DBMS) is decided to be DB4O an object-oriented Database. Object oriented databases are also called Object Database Management

Systems (ODBMS). Object databases store objects rather than data such as integers, strings or real numbers. An ODBMS makes database objects appear as programming language objects in one or more object programming languages. With traditional databases, data manipulated by the application is transient and data in the database is persisted (Stored on a permanent storage device). Native SQL access is fast, but laborious, requiring a great deal of additional code [Kim, Won (1990)]. The most common approach is custom object-relational mappers which offer a convenient bridge, but they seriously degrade performance. In object databases, the application can manipulate both transient and persisted data. The main advantages and disadvantages of object databases compared to relational are:

Advantages:

- Better concurrency control - A hierarchy of objects may be locked.
- Data model is based on the real world.
- Works well for distributed architectures.
- Less code required when applications are object oriented.

Disadvantages:

- Relational tables are simpler.
- Standards for RDMS are more stable.
- Support for RDBMS is more certain and change is less likely to be required.

The unique design of db4o's native object database engine makes it the ideal choice to be embedded in equipment and devices, in packaged software running on mobile or desktop platforms, or in real-time control systems [DB4o Specs.]. It can be embedded in Java and .NET environments, where no Database Administrator is present. It provides cross-platform portability that liberates users from proprietary vendors' high licensing fees. The major reason for using relational databases today is legacy, i.e. retaining old enterprise data and the set of existing applications relying on it. Db4o is the first to implement Native Queries (NQ). Native Queries lead the industry trend to provide database querying with programming language semantics, validated by Microsoft's LINQ (.NET Language Integrated Queries) project [MSDN]. NQs allow developers to simply use the programming language itself (e.g., Java, C#) to access the database and thus avoid a constant, productivity-reducing context switch between programming language and data access API. Native Queries eliminate all strings from queries. They are a hundred per cent type-safe, object-oriented and refactorable. Errors by the data access code are not thrown until execution. NQs do not accept type-mismatched queries in the first place. Db4o's object-oriented replication (dRS) functionality allows for easy synchronization of data between db4o databases. Objects are stored natively, eliminating the added complexity and performance drain of conversion to other formats such as SQL. Sessions and transactions on data are supported.

4.2 Architectural Design

Regarding the architectural aspect, a centralized design is decided for the data repository as shown in Fig. 1. All the data are stored in a single database which resides in the main server machine. The measurements data are stored in the same database. Another architectural scenario is shown in Fig. 2. A separate database is maintained in every environmental station. The measurements files are processed by the site's PC and stored in the station's database. The user interface specifications require that every station's data is available for combining and viewing together measurements from different sensors in different stations. This architectural scenario puts a restriction because the network connection of the main server with the stations PCs is not always permanent and is very slow in the case of dialup. The acquisition of data each time for a combined request is time-costly and unreliable since a dialup connection is needed to be established. These are the reasons why this design scenario is set aside. In the selected design, the connection is made only once, that is when the station is ready to transmit its measurements data.

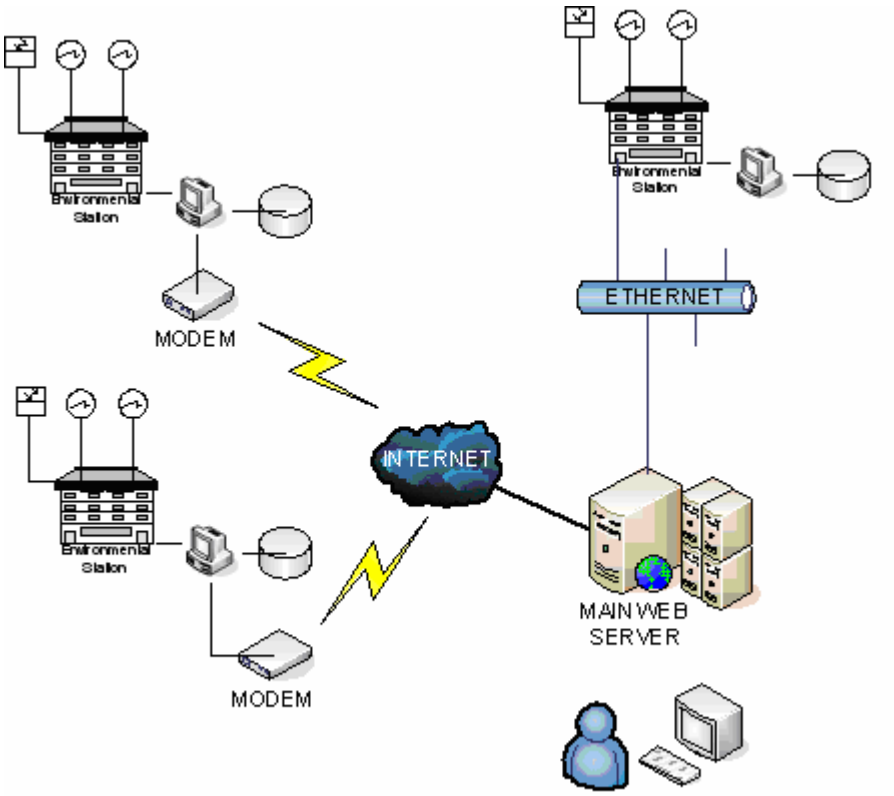

Figure 2. *Separate Databases Architecture*

5. The User Interface

5.1 General Description

A graphical interface is available as a means for administering and using the measurements information. It is a WinForms application and the functionality it offers to the end user is:

- Registering, editing and deleting environmental stations and sensors.
- Viewing measurements values in number format and combined graphs.
- Management of the measurements files each station produces.

The main screen tab (Fig. 3) concentrates the majority of operations. Two grid tables display the data for all the registered stations and their sensors. Depending on the station selection the sensor table is automatically populated. The stations and sensors data elements are initially filled in by the Administrator with the help of graphical forms. By highlighting a sensor and pressing the *View Sensor Measurements* button, the user is redirected to the measurements tab. The selected sensor's measurements can be filtered by date and analytically displayed.

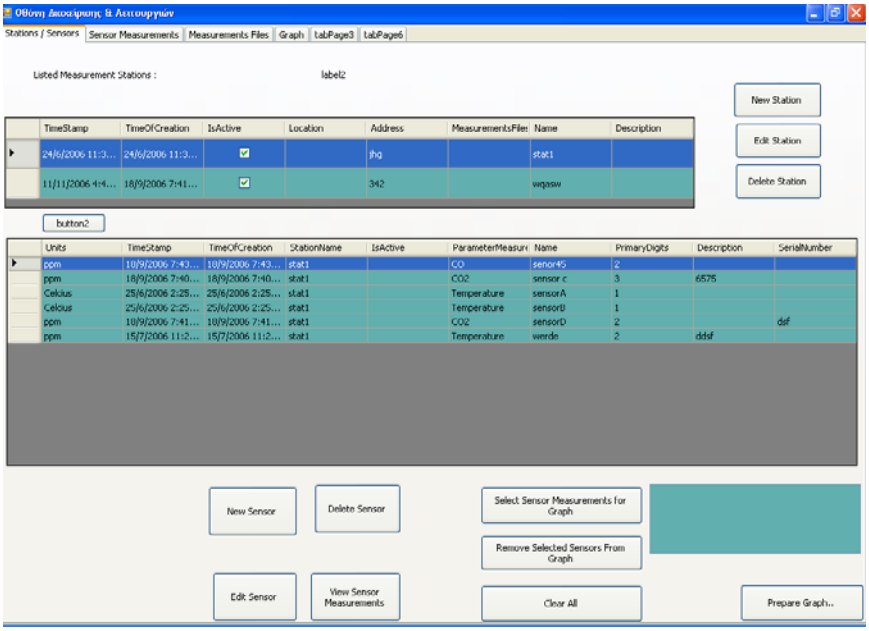

Figure 3. *Main Screen of the Interface*

The most important feature of the interface is the dynamic building of graphs. The graphs are produced by a custom developed class library based on the respective libraries of the .NET Framework. The graphs plot sensor measurements for the selected time window. As with every operation of the interface, the database is used

to retrieve the requested data. The library can display in the same graph curves for multiple sensors of different measuring kinds. Specifically, it supports up to five sensors and two environment parameters plotted in two verticals axis. At run-time, the climaxes for the vertical and horizontal axis are calculated depending on the values to be displayed and the whole graph is built dynamically (Fig. 4).

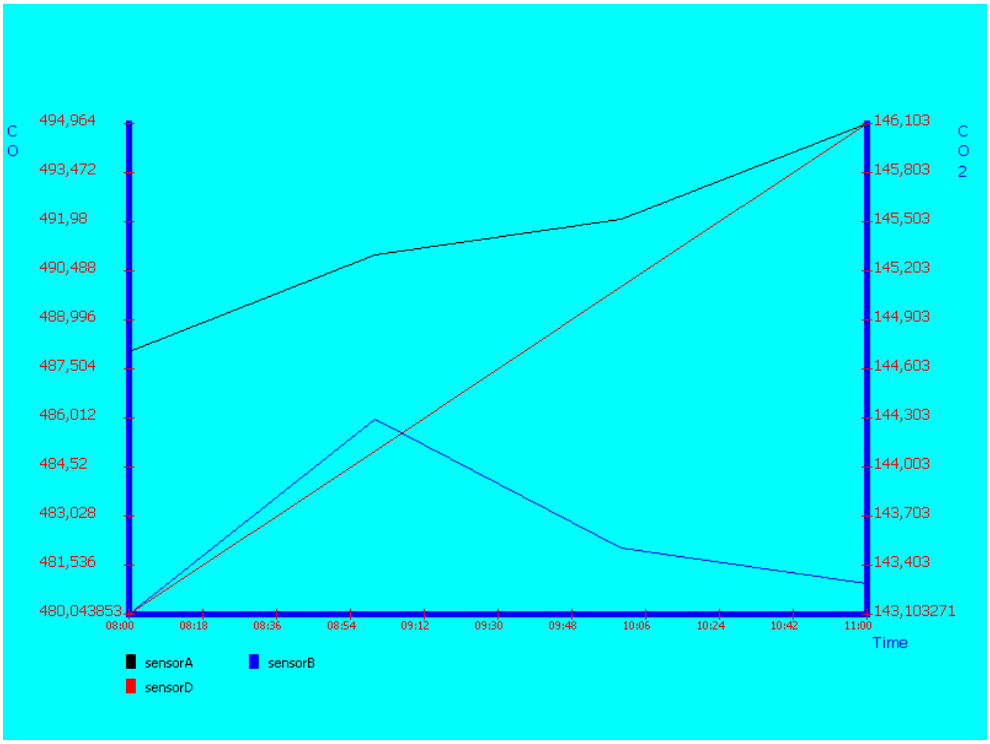

Figure 4. *Dynamic Graph Plot*

6. Forthcoming Expansions and Objectives

The application in its current state allows administration and measurements viewing from a single location, the main server's machine terminal screen. Despite the fact this is adequate for the time being, it is probable that future requirements will arise. The most expected requirement is multiple users requesting simultaneously measurements information. Besides that, the more stations and sensors will be registered in the database, the higher the necessity will be for stricter administrative barriers and access rules. Each station or perhaps a group of stations might have different measuring purposes, thus requiring different administrator and user accounts. This problem is solved by creating a web portal which offers the same functionality as the WinForms application, plus security access control. Viewing

multiple measurements from different locations will be treated as different web requests.

The development of the website using the ASP.NET class libraries is relatively easy, because:

- They have many similarities with the WinForms libraries.
- The information exchange and processing are performed via calls to the Factory custom methods; therefore no extra development concerning the logic body has to be done.

In addition, the proposed architectural design and the specific selection of technologies for the solution's software and database components are chosen in order to support the following scenario. As mentioned in the introduction, the objective after observing the measurements for a given station or a group of sensors is to provide feedback to the environmental stations. The metadata extracted to define the feedback actions will be produced by an algorithm executed on the stored measurements. The feedback can have the form of providing different operation signals to devices which are installed on site. Considering the fact that the application can be extended to monitoring outdoors and indoors environmental factors, signals can be sent to start up or shut down a heating boiler, or a heat pump/air conditioning device as well as and ventilation equipment. The devices will receive signals via a hardware interface with the help of the station's PC. Their functionality will be controlled by an internal PCI card, the serial port, or an intermediate controller which is connected to the PC and translates signals to operating functions. The .NET components have the ability to communicate with the hardware interfaces and pass the appropriate signals depending on the commands the code instructs. The problem that eventually arises is how the feedback messages will reach the stations. This problem gets even more complex when the proposed application is required to offer the same functionality for deployment on mobile devices. From an operational and practical perspective it would be ideal if the administration, measurements monitoring and feedback could be performed not only from a desktop client, but also from a portable computer, a handheld PC or a PDA. For that kind of approach, the portable devices must operate as detached mini client machines (Fig. 5). The user interface is easily transferred to the mobile platforms by creating executable files under the Microsoft's Compact .NET Framework. It is a special variant of the .NET Framework designed specifically for devices with Windows mobile editions installations.

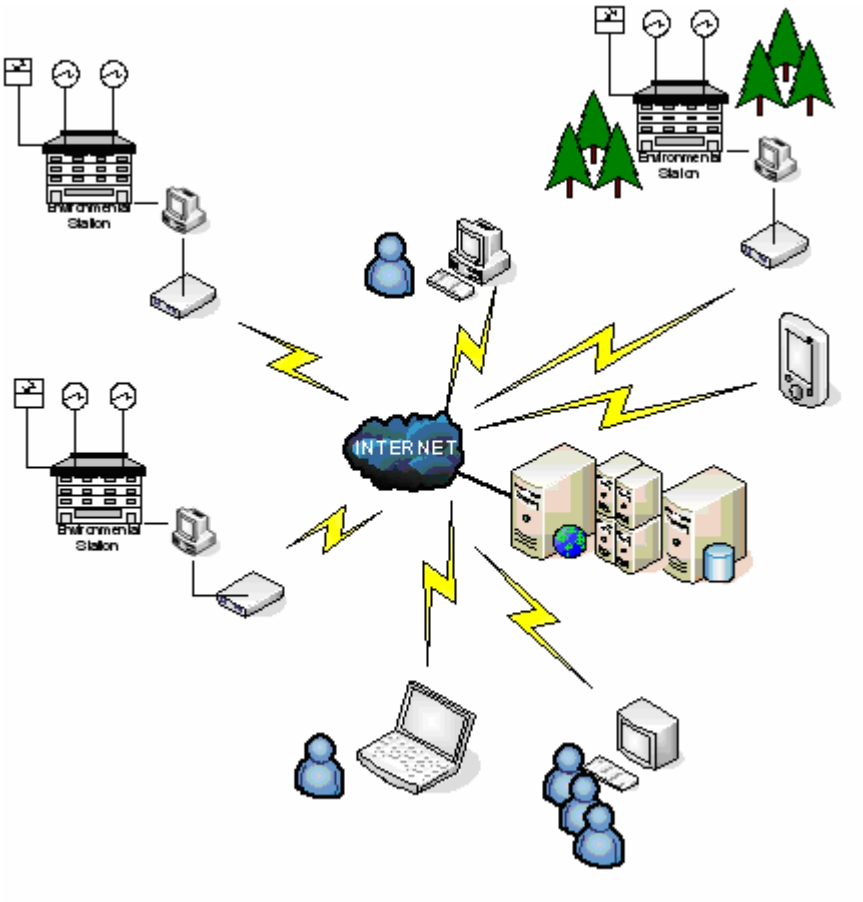

Figure 5. *Web Based Architecture*

What is most importantly needed is the implementation of a feedback mechanism operating similarly with conventional as well as portable platforms. Any platform supporting information exchange using the XML protocol is capable of providing feedback to the stations and receiving measurements data from them, under the proposed solution. Platform interoperability and communication among different software components by means of XML is perhaps the Web Services technology major advantage.

2. Discussion and Conclusions

The value of the developed software application lies not only to the functions it offers. Easy maintenance and platform independence are important parameters

implemented. The application is destined to gather measurements data, monitor and provide feedback for remote sensor networks, provided that an internet connection is available. The choice of Web Services Technology and DB4O, gives the ability to host the entire server on a Linux machine. This can be achieved by using the Mono environment (an open source framework to develop .NET applications) since DB4O also runs on Linux [MONO]. The cost of the solution drops to minimum, providing access to conventional or mobile clients.

References

DB4o. Product Information & Specifications. http://www.db4o.com

Gamma, Helm, Johnson, Vlissides. (1995). Design Patterns.Elements of Reusable Object-Oriented Software. Addison Wesley.

Kim, Won. (1990)Introduction to Object-Oriented Databases. The MIT Press.

MSDN. The Microsoft Developer Network. http://msdn.microsoft.com

MONO Project. http://mono-project.com

Short S. (2002).Building XML Web Services for the Microsoft .NET Platform. Microsoft Press.