WAMDAS: A Web Service-Based Wireless Alarm Monitoring and Data Acquisition System for Pharmaceutical Plants

by
Edilberto García Rodríguez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGUEZ CAMPUS
2005

Approved by:

_____	_____
Manuel Rodríguez-Martínez, Ph.D. Chairperson, Graduate Committee	Date
_____	_____
Néstor Rodríguez, Ph.D. Member, Graduate Committee	Date
_____	_____
Jorge Ortíz, Ph.D. Member, Graduate Committee	Date
_____	_____
José Cruz Cruz, Ph.D. Representative of Graduate Studies	Date
_____	_____
Isidoro Couvertier, Ph.D. Chairperson of the Department	Date

Abstract

Typical IT infrastructures used in manufacturing companies such as pharmaceutical plants are based on enterprise servers, with vast capacities for large data sets, managed with relational database systems, with powerful capabilities for query processing. In addition, workstations running diagnostics and control applications that monitor critical status conditions in manufacturing equipments are located in far locations throughout the plant. These workstations are wired to sensors that gather information from the equipments. This organization makes it difficult and expensive to integrate these systems efficiently in order to monitor status and alarm information from equipments. To mitigate this problem, we have developed the Wireless Alarm Monitoring and Data Acquisition System (WAMDAS), a Web Service-based wireless system to monitor status of equipments, and process critical alarms triggered by contingencies that occur during operational conditions. WAMDAS integrates data sources residing in workstations and enterprise servers with their traditional DBMS always running. These data sources contain status information and alarms generated from plant's equipments. In our work, we show that our system is scalable, efficient and capable of managing the typical number of status requests, alarms and acknowledgment messages per hour that can occur at a pharmaceutical plant

Resumen

Infraestructuras de tecnología informática (IT) regularmente utilizada en plantas farmacéuticas están basadas en servidores de alto rendimiento para manejar tipos de datos, con capacidad poderosa para el procesamiento de "queries". En adición, estaciones de trabajo con aplicaciones de diagnóstico y control para monitorear información crítica de estatus de los equipos de manufactura, están localizadas en áreas remotas de la planta. Estas estaciones de trabajo están alambradas a sensores que recolectan información de los equipos. Esta organización hace difícil y costosa la integración de los sistemas de una manera eficiente para monitorear estatus y alarmas de equipos. Para mitigar este problema, hemos desarrollado un sistema inalámbrico de monitoreo de alarmas y adquisición de datos (WAMDAS). WAMDAS es un sistema inalámbrico "Web Service-based" para monitorear estatus de equipos, y procesar alarmas generadas por contingencias que ocurren durante operaciones normales en una planta farmacéutica. WAMDAS integra las distintas fuentes de datos que residen en las estaciones de trabajos y servidores de la empresa las cuales a su vez administran los sistemas de bases de datos. Estas fuentes de datos contienes información referente a la información de estatus y alarmas generadas por los distintos equipos. En este trabajo, demostramos que nuestro sistema es extensible, eficiente y capaz de manejar el número típico de pedidos de estatus, alarmas y mensajes para atender alarmas por hora de los que podrían ocurrir en una planta farmacéutica.

To Zuly, Mar, Papi, for their unconditional support.

Acknowledgements

First, I will wish to thank God for giving me the health and strength to accomplish this work. Second, I thank my family for the unconditional support given to me, specially my father, Roberto Garcia (papi), who has been my best friend, my best supporter and the person that encourages me to stick to my goals and complete them. In addition, to my little daughter, Zulymar Garcia (titerona), with her love and beauty inspires my work and my life. Without her, I would not have the necessary determination to complete my efforts. Thanks to my lovely wife, Marlene Sonera (mar), for her love, understanding, help, patience, sacrifice and contribution to make me a better student. I also want to thank my advisor, Manuel Rodríguez, for his help, comments, suggestions, specially his concern and comprehension by giving credit to my sacrifice as full time worker in Vega Baja and as a student in Mayagüez at the same time. In addition, I want to thank the other members of my dissertation committee, Nestor Rodríguez, Jorge Ortíz and José Cruz, for their comments and suggestions. Besides, my complete gratitude to the Engineering Department of Pfizer, Vega Baja specially to engineer José Fong for his support and contributions to this thesis. Also, thanks to my company Advanced Control Services, Inc and to my boss, Victor Taveras, for his good will and patience through my years of student. Finally, I would like to thank the Department of Electrical and Computer Engineer of the University of Puerto Rico at Mayagüez for giving me the opportunity to study and be graduated from such as prestigious educational center.

Table of Contents

List of Tables

List of Figures

List of Acronyms

SCADA	Supervisory Control and Data Acquisition
HMI	Human Machine Interface
PLC	Programmed Logical Controller
PDA	Personal Digital Assistant
PC	Personal Computers
OS	Operating System
DBMS	Database Management System
XML	Extensible Markup Language
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery, and Integration of Web Services

CHAPTER 1

INTRODUCTION

1.1 Overview

The emergence of wireless technology such as Bluetooth and IEEE 802.11x family for handheld devices has provided companies with a variety of options to organize and maintain their data using dynamic wireless networks. In addition, XML and Web services technologies provide innovations in terms of heterogeneous data integration and ubiquitous communication. An important application of these technologies is their use to monitor status and manage critical alarms raised by various equipments in a manufacturing plant. These status and alarm information about equipments are very important for the manufacturing production of a pharmaceutical plant. If an alarm is not acknowledged and the problem resolved in time, it can cause serious damage to the equipment, lost of the active product or even multi-million dollar fines by environmental regulatory agencies.

Traditional networks used in manufacturing companies such as Pharmaceutical plants are based on enterprise servers with vast capacities to store large data sets, running production DBMS software (e.g., Oracle or MS SQL Server), with powerful capabilities for query execution. In addition, the clients connected to these networks are regular workstations with high computational capacity, continuously available power supply, and only connected by traditional wiring methods, always ready for data transmission and processing. In a traditional pharmaceutical plant network configuration, as shown in Figure 1.1**,** a centralized server for data storage is connected to software systems known as Supervisory Control and Data Acquisition Control Systems (SCADA). A SCADA is software dedicated to monitor status from equipments and to display alarms for the operators within the area. Also, the sever component is connected to a chlorine totalizer used to monitor the chlorine concentration of a water tank. In addition, the SCADA is connected to a Programmed Logical Controller (PLC) in order to monitor, control and report alarms of equipments.

However, the servers on this type of network configuration cannot incorporate efficiently larger quantity of devices like PLCs, sensors and SCADA systems, since these servers do not have an appropriate protocol(s) or technology for data exchange with neither the PCLs nor the sensors. This leads to ad-hoc and highly cost solutions including expensive software architectures, drivers and wiring to manage each source of data, and interface with the sensors and PLCs.

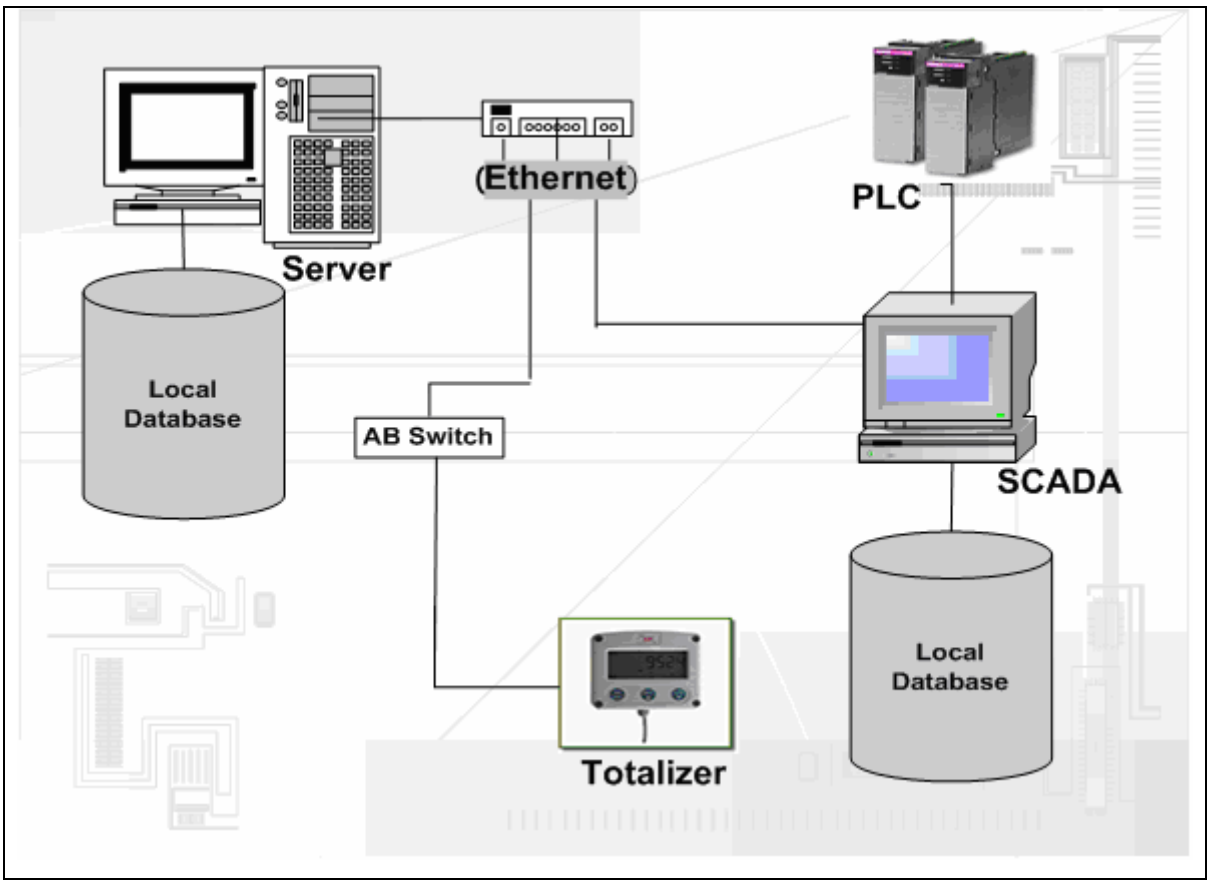

Figure 1.1: Traditional Network Configuration at Pharmaceutical Plants

Other assumptions on the networks of these types of manufacturing companies are that the data are not distributed since they rely on centralized servers for data storage. Many of these stand-alone systems (a.k.a. "Information Islands") are located within areas with distinct Human Machine Interfaces (HMIs), different local owners, and DBMS with heterogeneous data formats. Furthermore, the wiring process invested each time that a new client (i.e. Workstations, HMIs) is installed, and its associated configuration costs, create real issues

with management on these stand-alone systems and the data generated by them. In particular, it becomes critical to track the various data items, such as alarms, status notification, and access logs being generated by the system. Another important issue is the problem on how to integrate these data items in an efficient and uniform way, since the data sets are heterogeneous due to different software architectures and heterogeneous data formats. In addition, physical limitations of devices and equipments connected to these clients make the operator's job very expensive in terms of work force hours since they have to be almost all the day logging data on each system or equipment, and analyzing them. It would be more cost-effective to have this personnel doing technical work associated with preventive maintenance for the equipment deployed in the premises of the company.

Today, existing software solutions used at manufacturing plants, especially at pharmaceuticals plants, consist of different software systems monitoring status of equipments and alarms generated by these equipments. These software solutions used to monitor status and alarms of equipments are deployed on different areas in the pharmaceutical plant, thus making the system distributed in terms of data sources. To monitor those remote equipments it is necessary to have at least two operators. One of them is making roundtrips twice by shift to gather status and alarms information. The other one is confined in a control room monitoring other equipments connected to a Supervisory Control and Data Acquisition System (SCADA). Thus, a centralized integration site is used to gather the information from the various equipments. To complicate matters, each time that new equipment needs to be installed (e.g., a water pump), software and sensors to monitor such equipment must be deployed. This leads to highly costly solution in terms of wiring installation and more work for operators. Each time that remote equipment needs to be monitored by operators, new expensive Human Machine Interfaces (HMIs), or SCADA systems are installed in those far locations as shown in Figure 1.2. For example, a PanelView HMI is needed to display status and alarm information from potable water tanks and well pumps located in the Pump House area. In another place, the Warehouse area uses a SCADA system to gather status and alarm information such as room temperature and humidity.

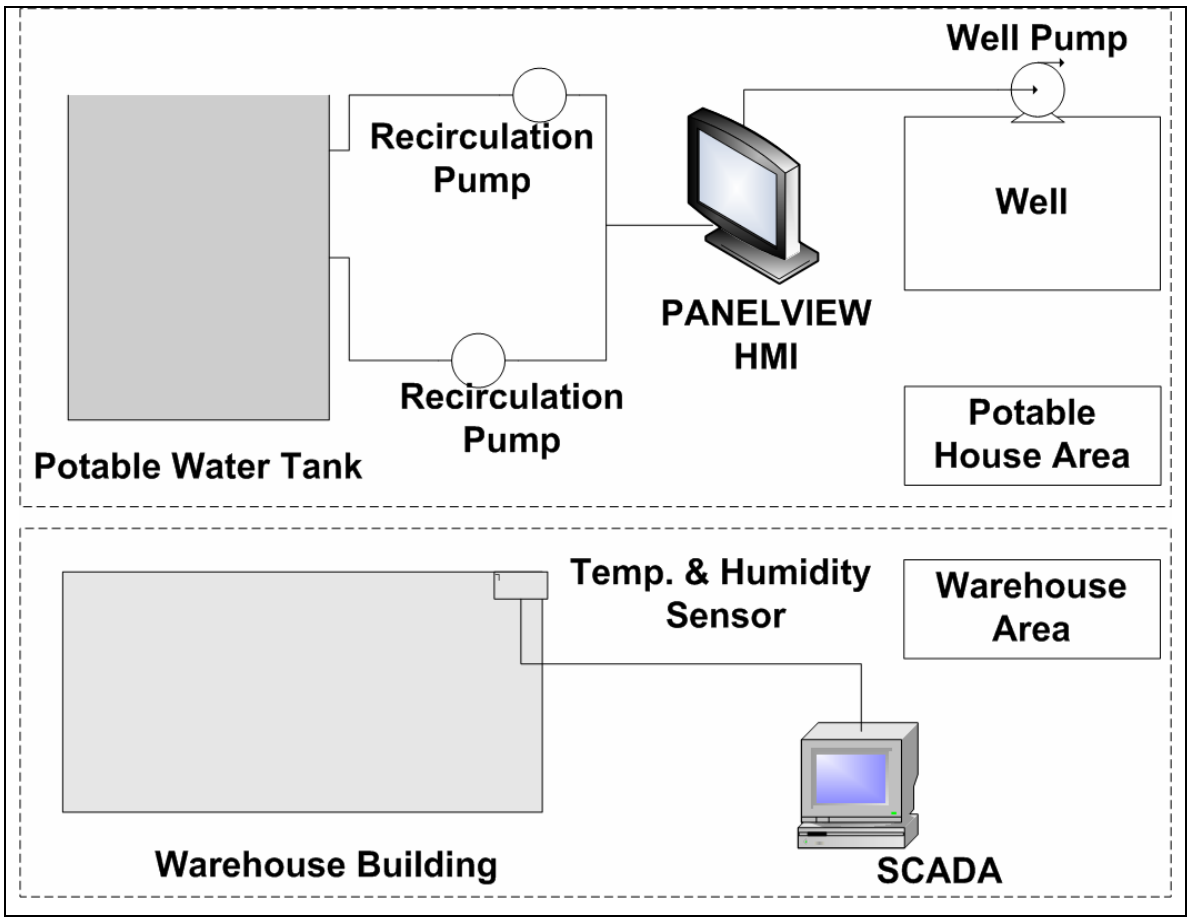

Figure 1.2: Distinct Software Architectures

Our research is also motivated by how an operator's job can be optimized. For example, in a typical pharmaceutical working environment (see Figure 1.3), an operator's regular job is to verify the status of all system equipments, perform data logging of each verification, monitor possible alarms messages of these equipments, and perform preventive maintenance.

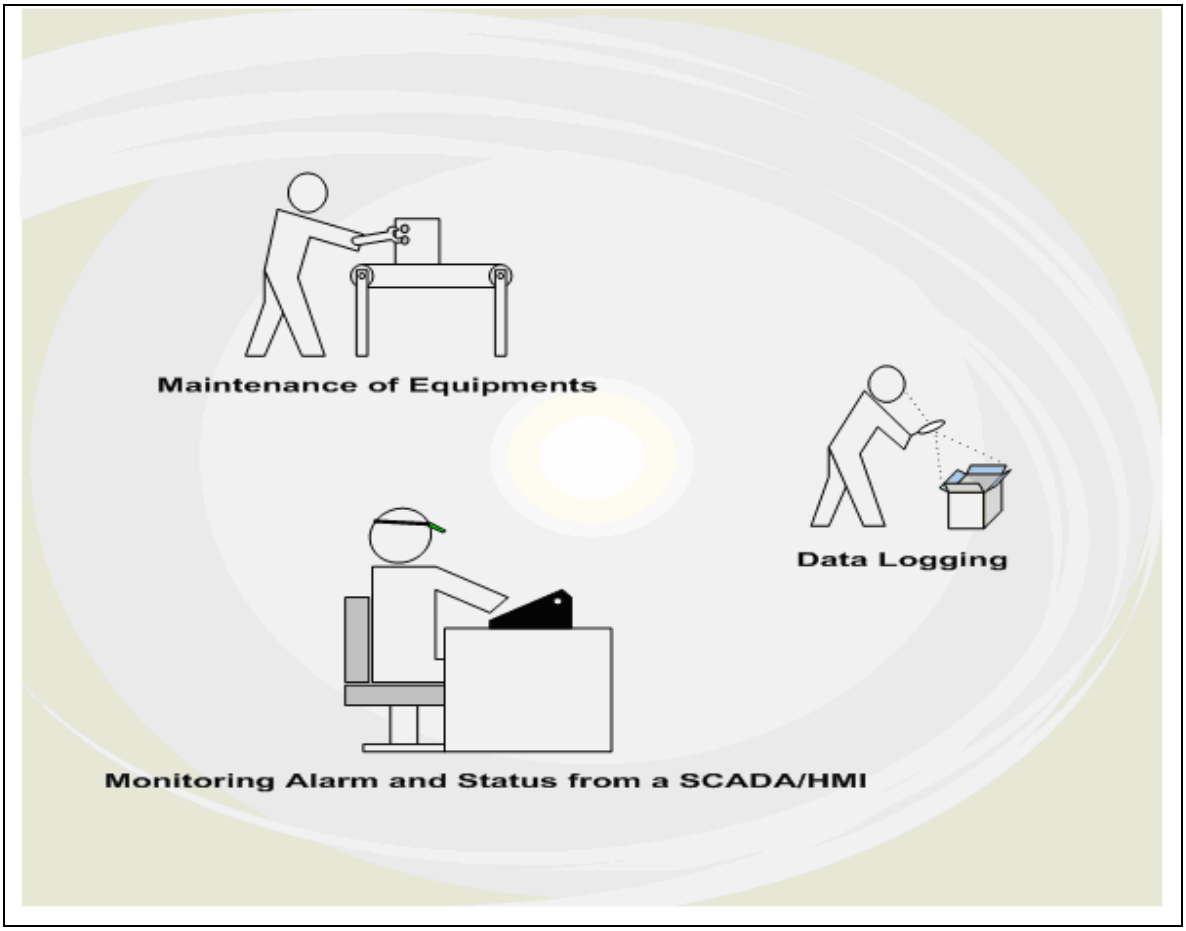

Figure 1.3: An Operator's Regular Job

As we mentioned before, at least one operator must be gathering status information from system equipments at least one or two times during his/her regular eight-hour shift. This status information includes equipment operating mode, component temperature, chlorine concentration, header pressure, and many others. Meanwhile, when one or more equipments generate an alarm, the operator must acknowledge these alarm messages as soon as possible. After the operator acknowledges the alarm(s), he/she must proceed to correct the problem with the equipment(s). In addition, another operator must be confined during his/her regular shift in a control room monitoring system equipments status and logging alarms data from a SCADA system. The remaining hour's shift of an operator is related to preventive maintenance tasks, equipment troubleshooting and optimization. These leads to at least one operator within an eight-hour regular shift must be confined inside a control room waiting for alarms and status information from all equipments, and another operator performing

roundtrips to remote locations, at least twice by shift, to gather status and alarms information from equipments. This scheme reduces the operator's job to only a monitoring position instead of a productive and dynamic job position.

1.2 Problem Statement

Given the scenario presented in section 1.1, a system solution to integrate data such as equipments' status and alarms critical information from remote equipments is necessary. However, a system solution to integrate alarms information and status of equipments in a pharmaceutical plant does not exist. This alarm information and status of equipments must be available near real-time to operators, process control engineers and supervisors in order to react quickly to any danger situation. In addition, we want to optimize an operator job by release them from being positioned in front of a computer waiting for alarms to happen or as in other cases, performing unnecessary roundtrips for verifying and logging status information of remote system equipments. Also, we want to eliminate unnecessary round trips to remote areas in the plant to gather data from equipments and acknowledge alarms (see Figure 1.4).

For this, we need a software solution capable of delivering, in near real-time fashion, alarms and status information to any operator no matter his/her location in the plant. This software must be capable to integrate data such as status and alarms from remote HMIs, SCADAs, sensors and PLCs. This data must be first-hand information for operators, process control engineers and supervisors in order to troubleshoot critical equipments. It also may help to prevent environmental contaminations due to equipments malfunctioning by providing critical alarms information to the appropriate personnel. Failure in recognizing an alarm notification from critical equipments can cause highly expensive fines, rounding in millions of dollars due to equipment malfunctioning. In a worst case scenario, it can cause an involuntary plant shutdown. In both cases, the pharmaceutical plant is in risk of lossing a hugge amount of money, and being even penalized by a regulatory agency (e.g. FDA) with a temporary or permanent closing of the facilities.

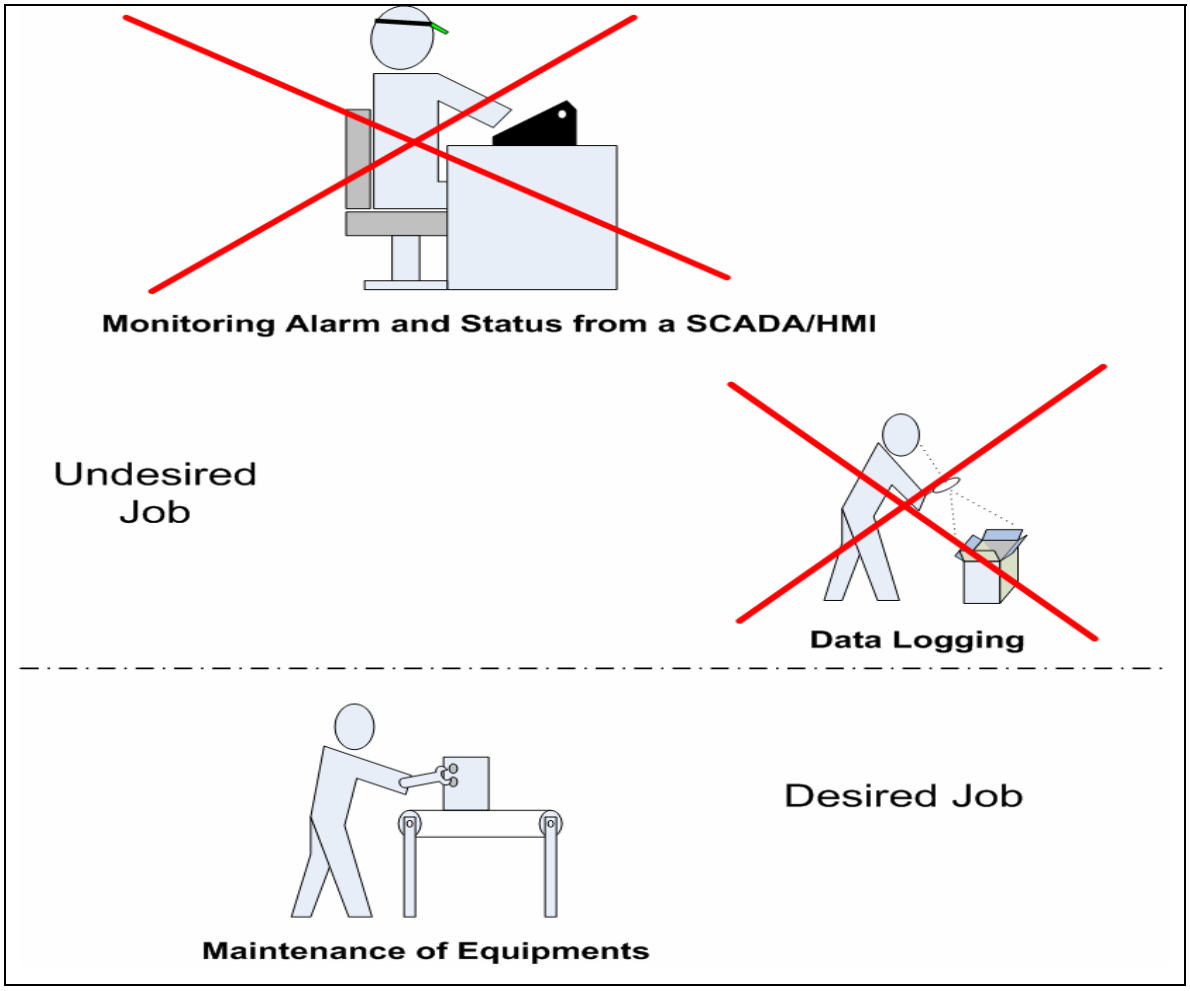

Figure 1.4: An Operator's Regular Job Optimized

1.3 Proposed Solution

The proposed problem solution in this research project is the Wireless Alarm Monitoring and Data Acquisition System (WAMDAS). WAMDAS is a robust Web Service-based, database middleware, system to integrate data sources residing on PDAs, mobile laptops, workstations and enterprise servers with their traditional DBMS always running. All these devices are assumed to be deployed on the premises of a pharmaceutical plant, which has wire-based and wireless LANs to interconnect the devices.

WAMDAS is designed to manage dynamic environments like pharmaceutical plants, where different software architectures, hardware diversity and costly wiring solutions are a

limiting factor for effective system integration. In WAMDAS, handheld and mobile devices will be treated as *Peer* data sources and monitoring tools, capable of delivering content to other sites in the system such as internal Web servers and local authorized computers. At the same time, these handheld and mobile devices will be able to fetch data from all the nodes connected to the middleware system. In addition, WAMDAS will allow mobile data sources to access local and remote resources such as sensor data from remote HMIs, including Water Treatment Pumps, Turbines, Supervisory Control and Data Acquisition Control Systems (SCADA,) and other important data sets. Furthermore, WAMDAS could help to facilitate layout changes (e.g., rellocations of SCADAs, HMIs or sensors to other places in the plant) by reducing expensives wiring installation. WAMDAS natural wireless infrastructure allows for easy movement of equipment around the plant, without much delay or loss of information. To the best of our knowledge, none of the existing commercial or in-house software solutions implemented at pharmaceutical plants has provided a well-defined framework to manage alarm notifications and status information from diverse data sources with different architectures and data formats that match WAMDAS. In short, we designed WAMDAS to monitor and acquire remote data from critical equipments at a pharmaceutical plant.

The WAMDAS system is composed of the following elements:

- **Web Services** – responds to HTTP requests made by operators, process control engineers or supervisors listed on the system. They are responsible for the process of alarms and status information

- **Smart Device Application** – Lightweight, powerful application deployed on Pocked PCs for the use of system's clients to request status information and to receive first-hand alarms information from remote equipments.

- **Database System** – stores the data and metadata of status and critical alarms from the equipment. In our case we assume that a, relational database (i.e., Microsoft SQL Server) is used on each site.

1.4 Research Objectives of this Project

The main objective of this thesis is to design, develop and implement the WAMDAS framework. WAMDAS is a middleware system that can monitor critical devices at a manufacturing plant. The system will help operators, engineers, supervisors and other users to improve their work environment and time frames. At the same time, the system will help optimize the data management of the control systems by using a robust wireless infrastructure and efficient algorithms for query processing and data integration.

The specific objectives of WAMDAS are as follows:

1. Design a system architecture for a wireless distributed system for monitoring and integrating all the critical information within a manufacturing plant. The critical information includes alarms and status information of the following system equipments: Cooling Towers, Water Injection (Reverse Osmosis System), Potable Water System including main pumps, Chillers and Turbines equipments.

2. Implement alarm-monitoring services to speed up the time frame of critical alarm delivery and management. The main objective of these services is to reduce downtime of any critical equipment (e.g. Water Tank Level set point) which can lead to major damages and loss of revenue due to a malfunction of any equipment component.

3. Develop a robust system using the C# language and XML Web services technology where all the data will be managed in XML universal standard format.

4. Use SOAP (Simple Object Access Protocol) protocol for the wireless communication in order to add scalability and portability to the system. SOAP is a lightweight protocol that encapsulates the message in an appropriate XML format capable of being received in any platform regardless of the operating system or architecture.

The benefit of using SOAP is that it provides a standard, extensible, compensable framework for packaging and exchanging XML data in a network-independent manner. Also, adds scalability by using the HTTP protocol and is therefore very scalable in its native form.

1.5 Summary and Contributions

The most important contributions of this project are summarizes as follows:

1 The development of WAMDAS, a Web Service-based, middleware system, which allows operators, engineers and supervisors to get first hand status and critical alarms information from remote equipments.

2 Enabling Web Services to become database middleware systems capable of serving and request data information in a wireless fashion on behalf of the client.

3 The implementation of a Status and an Alarm protocol to monitor critical equipments at a pharmaceutical plant.

4 A performance evaluation that shows that our system is more efficient than traditional centralized architectures usually implemented at pharmaceutical plants.

1.6 Structure of this Thesis

The reminder of this thesis is structure as follows. Chapter 2 discusses related work that serves to develop this thesis. Chapter 3 presents an overview of the WAMDAS architecture and its components. In Chapter 4, we present a detailed description of WAMDAS operation including the Status and Alarm protocols. Chapter 5 presents the performance evaluation of this thesis. Finally, chapter 6 presents a summary of our contributions, conclusions and future work related to WAMDAS.

CHAPTER 2

RELATED WORKS

We present a review of related work relevant to the WAMDAS system. The areas are 1) Peer-to-Peer Distributed Systems, 2) Client-Server architecture, 3) Middleware System, and 4) Web Services technology.

2.1 Peer-to-Peer Systems

2.1.1 Peer-to-Peer Database Systems

Peer-to-Peer (P2P) database systems like for example R* [9, 10] have architectures that show how highly coupled systems work and interact efficiently with each other, assuming that all the nodes are always available when data is needed and computers architectures are similar between peer sites. Another well-known P2P database system is Mariposa [15]. In Mariposa, as well as in R*, each site is an autonomous database server site. In these types of systems the database management system forms a federation of sites that have agreed to share their data and query processing capabilities There is no central authority that dictates what data are stored at each site, who should access the data, or how to access the data. Also, each site is assumed to run the same DBMS, or uses the same communications protocol to interporate between them. Each site is willing to cooperate in solving the query, and will make its best effort to perform the task assigned to it as efficiently as possible. It is assumed that each site is a full-fledge database server with high storage capacity and powerful query processing engines.

2.1.2 Peer-to-Peer File Systems

In recent years new P2P file systems applications such as BitTorrent [3], Kazaa and Emule, among others, are making P2P systems once again a very popular architecture for Internet applications developers. This popularity is due to several factors such as a) availability of free, open-source P2P software, b) increase in computer hardware power, c) improvement in Internet connectivity to homes, small businesses, and universities, and d) simpler, more efficient protocols to locate and access data in the P2P system.

A P2P system consists of N numbers of peers sharing computational resources such as CPU cycles, disk space, data files, music files and movies. Each peer has interchangeable roles of client and server since they can request and serve data to other peers in the network.

P2P are not without some drawbacks in their use. Some of the most common problems in P2P systems are security and availability of services. Security can be a drawback in P2P since all peers are traditionally considers bonafide members of the network providing full access to shared directories. Also, the availability of services can be unpredictable on P2P system since a server dedicated to serve some type of data or process queries request from peers might not available at some point time. Instead, peers are confined to the availability of others members and the willingness of them to participate in the network. Thus, results of operations in P2P, particularly query processing, might yield variables results from execution to executions due to the variable nature of peers providing results to generate the final solution to the client requests.

2.2 Client-Server Model

Another popular DDBMS architecture is the Client-Server Architecture [7, 8], which requires full collaboration from each server and their respective client(s). The fundamental assumption of these systems lies in the concept that the data sources reside on heavyweight enterprise servers. These enterprise servers have the specific role of serving data to client sites. Meanwhile, the client sites role is to help the users to get the data produced by the server sources. The server provides storage and query processing for all clients connected to it. Clients have caching capabilities and query processing to process the data in their caches. This latter feature provides offloading capabilities by allowing clients to execute part of the query and reducing server load. This results in increase server throughput. In addition, clients provide the user-interface for the users to access and manage the data provided by the server(s).

In general, distributed join processing [2] and data caching [4] have been the major focus of research on these systems. However, the assumptions in client-server architectures are that both, the client as well as the server sites run heavy DBMS software with powerful query processing capabilities.

2.3 Middleware Systems

A Middleware [1, 5, 11, 12, 13] System, also known as Heterogeneous Database system, is a middle layer of software that interconnects the client applications with the data sources. Still, most of these systems are based on the assumption that data sources reside on enterprise servers. Figure 2.1 depicts the architecture commonly used to organize middleware systems. Typically, the server application used at the middleware layer to service client requests is called the integration server. The integrating server provides client applications with a uniform view and common access to the data available in each source. Meanwhile, the

translator layer can be built out of wrappers, or database gateways, which are software modules that allow a single-site database server to gain access to the data managed by a remote database server. The gateways gives the local DBMS an access method mechanism to the data stored in the remote database server, and this local DBMS becomes an integration server for its clients.

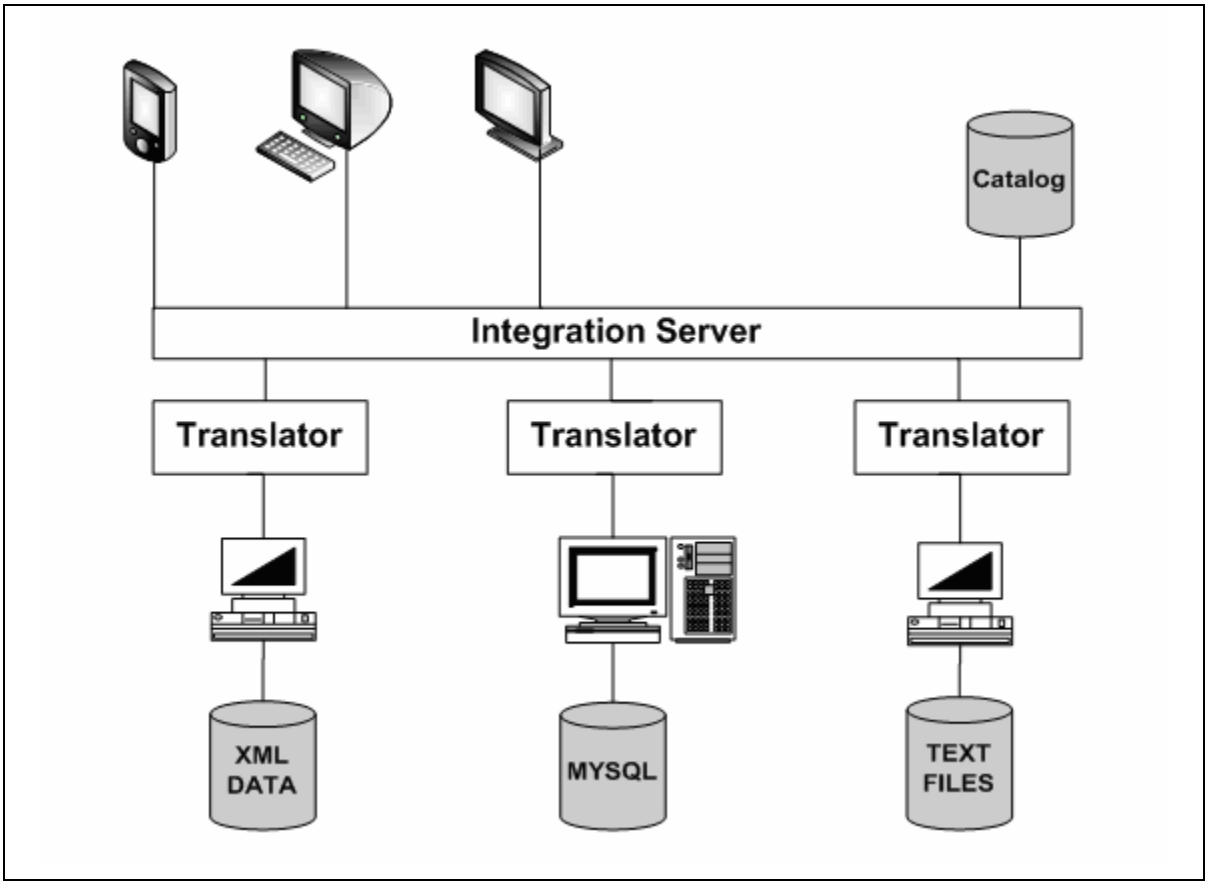

Figure 2.1: A typical Middleware Architecture

The most interesting type of middleware system is the Mediator system [6], in which a server application called the mediator acts as the integration server for the clients application.. The mediator server depends on the functionality of wrappers to gain access to the information stored in each remote data site. A wrapper extracts the information from a data source and translates them into the global data model specified by the mediator. Usually, the integration server runs close to the client application. On the other hand, wrappers can be run either at the integration site or at the data sources site. This type of Middleware system

provides to DDBMS scalability, heterogeneous data integration, and queries execution capabilities, by providing clients connected to a network a uniform view of the data available in each server and encapsulating the process required for this.

2.4 Web Services technology

The term *Web services* describes a standardized way of integrating Web-based applications using the XML, SOAP [16], WSDL [17] and UDDI open standards over an Internet protocol backbone. A Web services is an RPC application that runs over HTTP. One of the important assets of Web services is the accomplishment of ubiquitous interoperability by communicating different software architectures and operating systems. A Web service allows heterogeneous systems to work together as a single Web of computation. XML is used to tag the data, SOAP [16] is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available. Used primarily as a means for businesses to communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall.

Unlike traditional client/server models, such as a Web server/Web page system, Web services do not provide the user with a GUI. Web services instead share business logic, data and processes through a programmatic interface across a network. Developers can then add the Web service to a GUI (such as a Web page or an executable program) to offer specific functionality to users.

In a basic Web service communication, as shown in Figure 2.2, the following sequence of request through SOAP messages is used: 1) The service provider maintains a list of descriptions about Web services using WSDL in a directory, 2) The service consumer issues a query to the directory searching for a list of Web services, 3) The service consumer receives the query response from the directory using WSDL, 4) The service consumer issues a XML request based on WSDL to the service provider in order to invoke Web methods to

receive the required information, 5) Finally, the service provider sends the XML response using WSDL to the service consumer.

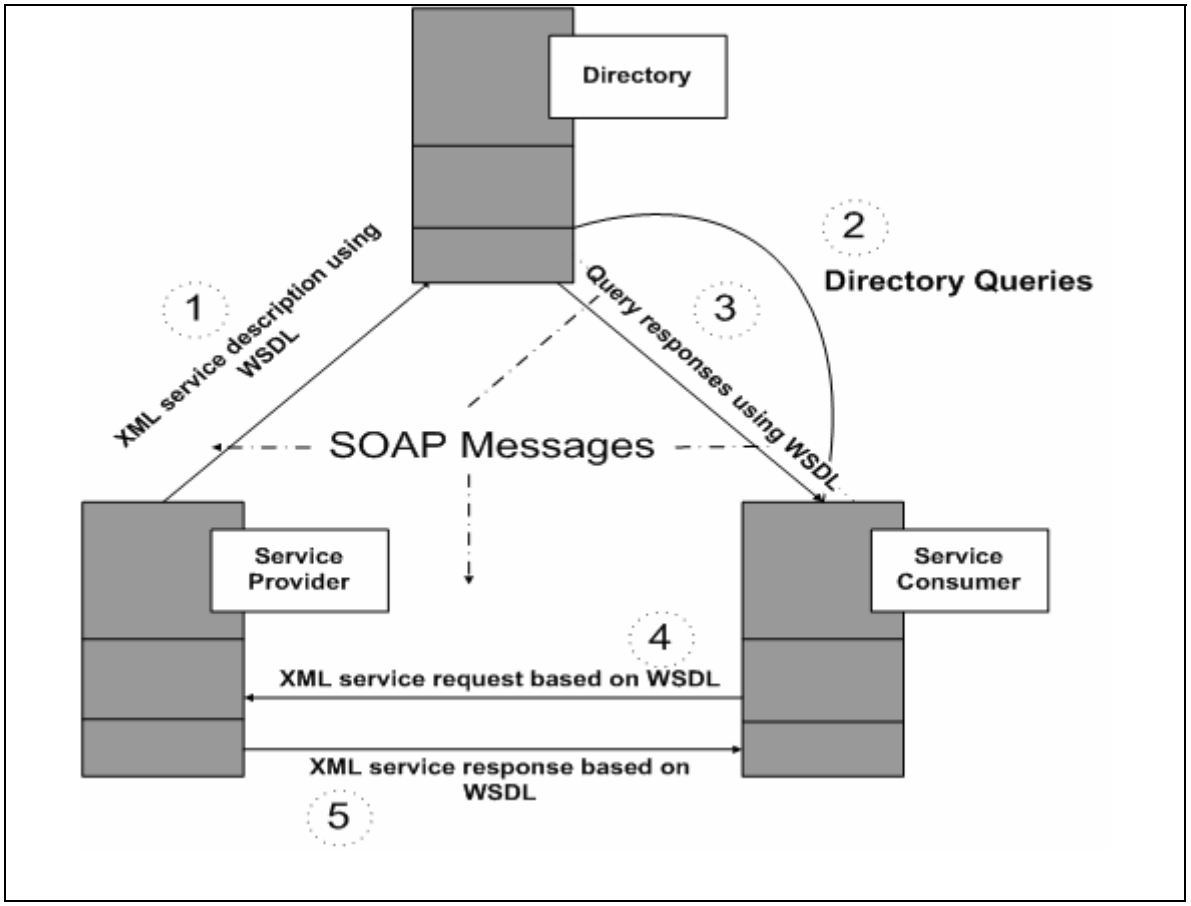

Figure 2.2: Web Services Communication Basics

Web services allow different applications from different sources to communicate with each other without time-consuming custom protocol coding. Moreover, because all communication is in XML, Web services are not tied to any operating system or programming language. For example, applications built with the Java programming language can communicate with applications built on languages such as Perl or C#. Likewise, applications deployed on machine running the Windows OS can communicate with other applications running in UNIX, MacOS, etc.

CHAPTER 3

OVERVIEW OF WAMDAS SYSTEM ARCHITECTURE

3.1 System Overview

WAMDAS is designed to manage dynamic environments in pharmaceutical plants, where different software architectures, hardware diversity and costly wiring solutions are a limiting factor to achieve effective systems integration. In WAMDAS, handheld and mobile devices are treated as data sources and monitoring tools, capable of delivering content to other sites in the system such as Internal Web servers and local authorized computers. At the same time, these handheld and mobile devices will be able to fetch data from other nodes connected to the middleware system. Thus, WAMDAS follows a Peer-to-Peer (P2P) operational model, because many components can interact with equivalent components running on different machines. In addition, WAMDAS allows mobile data sources to access local and remote resources such as sensor data from remote Human-Machine Interfaces (HMIs) or SCADAs, including the following devices: Water Treatment Pumps, Turbines, and other important sensor-enabled equipment.

3.2 System Architecture

WAMDAS is based on a decentralized architecture (see Figure 3.1) where all the clients can query and get information from remote data sources via a wireless environment. WAMDAS is based on XML Web services technology and SOAP protocol for communication between data source sites.

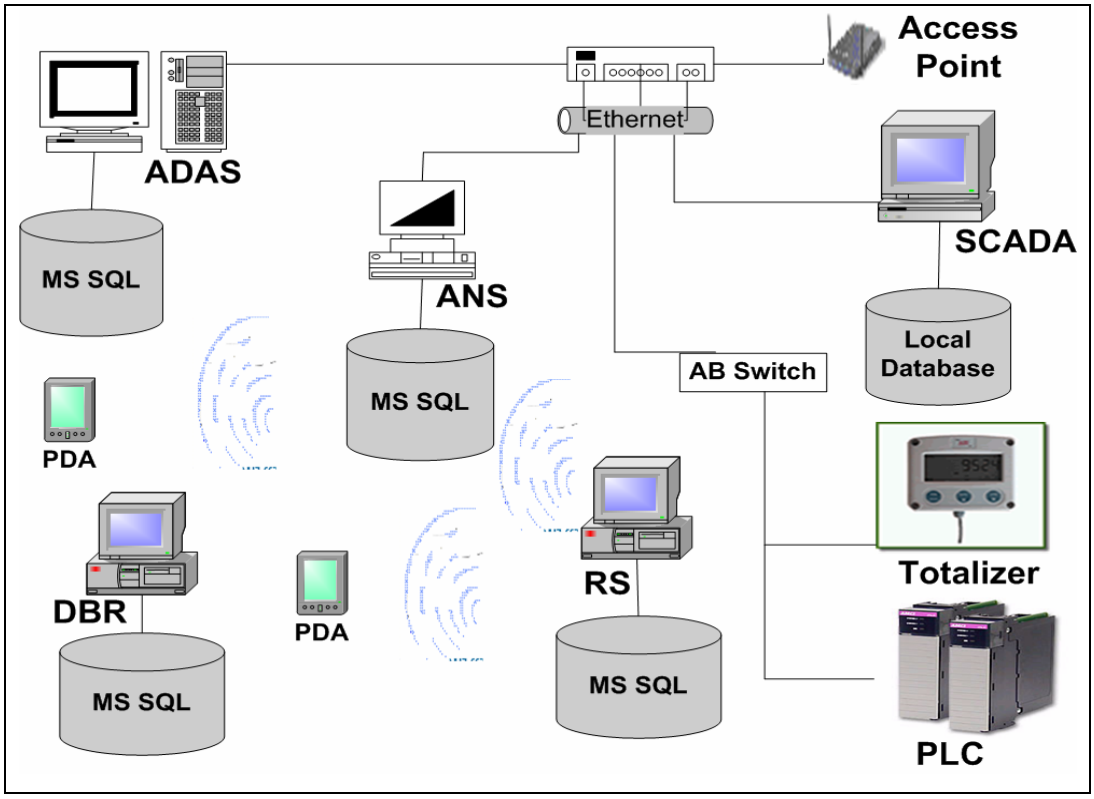

Figure 3.1: WAMDAS Architecture

WAMDAS has four types of components implemented as XML Web Services. Each Web Service has the necessary Web Methods to provide scalability and the proper functionality of our system**.** Web Methods are a key part of the messaging infrastructure employed by XML Web services. That is, a client and an XML Web service communicate using messages, specifically SOAP messages, by default. Clients send a SOAP request to an XML Web service and an XML Web service method returns a SOAP response. Each XML Web service defines the types of messages it can accept using operations, as defined by Web Services Description Language (WSDL). These operations correlate to each of the XML Web service methods within an XML Web service. The purpose of each of the XML Web services is as follows:

- **Data Broker (DBR)** – a Web service that works on behalf of the PDA clients to find the data and query processing capabilities necessary to solve a given query. This element has

responsibilities such as query plan generation, query coordination, and acquisition of results.

- **Registration Server (RS) –** this Web service is used to coordinate access to a federation and to the resources it provides. In addition, it handles all bookkeeping necessary to log events such as departure of a federation system user, or reappearance of a member at a different network or federation.. We deal with the details of federation management in section 3.3.1.
- **Alarm and Data Acquisition Service (ADAS) –** this Web service is responsible for monitoring any possible alarm information from all system components. When an alarm message is generated, the server will forward this message to the ANS for broadcast and acknowledge actions. Also, it acts as a wrapper for some data sources. When the client queries the DBR, the DBR will ask, the ADAS to contact the remote site from the data are to be retrieved.
- **Alarm Notification Service (ANS) –** this Web service is responsible for broadcasting of all critical alarm(s) or information to all interested clients. The Web service will require response from at least two clients after each issue of an alarm message.

The ADAS, which runs MS SQL Server for storage purposes, will be connected to the main Ethernet Switch. This switch is connected to others SCADAs workstations and trunk Interfaces that connect PLCs, IO cards and other sensors by using an Allen-Bradley Switch, all located at the Cogeneration Area of the plant. This is shown in Figure 3.1. The ADAS receives status information and alarms for the Turbines Control System, Water Injection System, Chillers Control System, Cooling Tower System, Compressors System, Boilers Control System, and the Warehouse building.

The RS Web service [14] will provide up-to-date status of any member of WAMDAS. This information correlates to the network ID, federation member ID, current URL address, and status of each Web service. Meanwhile, it can communicate with peers RS to resolve request of remote members.

In our prototype system, we implemented the ANS in order to manage all critical alarms for all the system components connected to a federation managed by WAMDAS. This Web service is at the heart of the WAMDAS architecture, since this component will define and provide the scalability and performance of WAMDAS. ANS will receive multiple alarms messages from any system equipment connected a WAMDAS federation. After an alarm message is received from the ADAS, ANS will be responsible for broadcasting the alarm message(s) to the interested clients that are connected to WAMDAS. Then, ANS will wait for a message of acknowledgement from one of the clients before considering the alarm handled. Thus, ADAS triggers the alarm notification message to the ANS; we decide to implement it as a different Web service in order to share computational resources by separating the status data query from the acknowledgement command and acknowledgement data query as well.

For the client side, a DBR Web service was implemented to receive the query requests from the mobile client. In addition, it receives the alarm notification from the ANS. It processes the alarm message by searching in the local DBMS for online PDA clients. After that, it proceeds to broadcasting the alarm notification to all PDA clients through a client socket. Also note that the DBR will contact peer DBRs to send alarms to interested clients located outside the federation in which the alarm was raised. Finally, it waits and processes the acknowledgement command from the PDA client.

In our implementation of WAMDAS, we simulate the data for the Potable Water Plant, which serves as a cluster on which wireless sensors for flow, status and temperature are connected for testing with the Alarm and Data Acquisition Server (ADAS) Web service. In addition, this cluster features HP wireless PDAs for the process control engineers, supervisors and operators, an Ethernet switch as the backbone for the ADAS. The IEEE 802.11b access point is connected to the Ethernet Switch.

We used Visual Studio.Net framework to implement our WAMDAS system. The RS, ADAS, ANS and DBR were implemented as XML Web services in Visual C#.net language.

Finally, the Pocket PC application client was implemented by using the Visual C#.net language for Smart Devices architecture. More details of this will be provided in chapter 4.

3.3 WAMDAS Communication Design

3.3.1 WAMDAS Federations

WAMDAS has four types of Web services that communicate with each other through SOAP messaging, using XML for data representation and query results. Each Web service will process each query request sent by the operator through a Pocket PC client application, obtaining a result from the diverse data sources in each federation to which WAMDAS has access to it. Again, the Web Services are the DBR, ANS, ADAS and RS.

We created the databases of each data source or Web site using Microsoft SQL 2000 Server. These data sources will store status information from system components like current operational status, temperature of the equipment, critical alarm information of any equipment's sensor.

Each Web site will serve data to all members of a federation as shown in Figure 3.2. A federation is a group of autonomous Web sites with local Web services and local DBMS located on different hosts. Federation Web sites on the manufacturing plant will be as follows:

1) Cogen Federation: federation representing the Cogeneration personnel.

2) Process Federation: federation representing the Process Control Engineers of the Utilities area.

3) Supervisor Federation: federation with supervisory privileges that shall be capable of monitoring all Web sites or federations' data sources within the plant.

Thus, each federation has its own XML Web Services that can communicate with the RS component; the DBR at the Supervisor federation can communicate with peers DBRs and ANSs in order to resolve PDA client's queries. As shown in Figure 3.2, WAMDAS exhibits

a Peer-to-Peer behavior since each Web Service can serve data to another peer Web service, as well as retrieve data from other federated Web services.

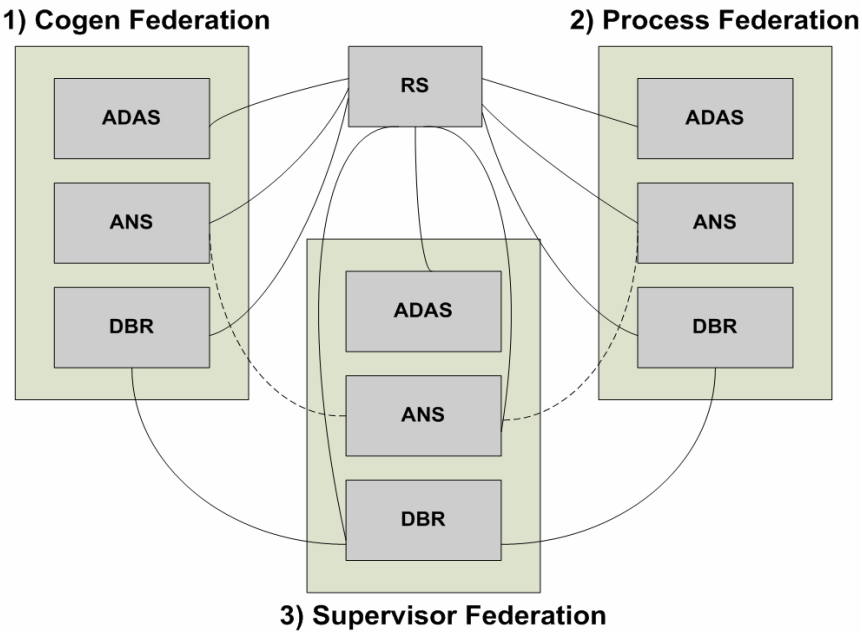

Figure 3.2: WAMDAS Federation Model

3.3.2 Wireless Communication

Authorized users can access WAMDAS within the manufacturing plant. WAMDAS has Web methods that will make it possible for the user (operators, supervisors or process control engineers) to search for status information of each of the equipments connected. In addition, the clients connected shall have real time knowledge of any alarm information generated by one or more equipments connected to WAMDAS. These equipments are Turbines, Boilers, Chillers, Cooling Tower, Potable Water Pumps and Water Injection system and the Warehouse building.

The main characteristic of WAMDAS is the capacity to use a wireless communication channel to request and process equipment status data and alarms information that are stored in multiple remote data sources. Operators with PDAs and wireless connectivity can access the critical information related to their jobs from any place within the manufacturing plant.

3.3.3 Queries, Notifications and Messaging schemes

WAMDAS has various query types implemented to obtain status and acknowledgement alarms information from remote data sources. The PDA client can query for status of equipment or a group of equipments. Likewise, PDA clients can query for acknowledgement alarms history from remote ANS sites. In addition, it provides a messaging scheme to handle alarms triggered by abnormal status of remote equipments. The management of that critical information is fulfilled with these queries and messages:

- **Status Data Query:** It is a query issued from the PDA client to search for status information from one or more equipment. This type of query can be used for specific equipment status information or a series of join operations with others equipments, in order to know more status information of related equipments.
- **Alarm Information Notification:** It is an alarm information message issued by any equipment connected, and that has to be acknowledged by one or more PDA clients. This alarm information can be from one or more equipments, and it is sent to all interested clients. The system will wait from an acknowledgement command from one or more clients.
- **Acknowledge Indication query:** It is an update query issued from the PDA client to acknowledge one or more alarms triggered by abnormal conditions of remote equipments.
- **Acknowledgement Data Query:** It is a query issued from the PDA client to search for acknowledgement information of previous alarms events. This type of query can be for acknowledgment information from their local federation or, in the case of a Supervisor client, status of any of the three federations implemented in our system.

The final query results are sent to the user by means of a Pocket PC client application that presents the status data and alarm information in a tabular form as shown in Figure 3.3. We shall go into more details in this client application in chapter 4. We used Microsoft SQL Server 2000 for the deployment of the databases used in the system. The main motivation for using Microsoft SQL Server relational database software is that it is used in most of the Windows-based computers that are standard in manufacturing companies like pharmaceutical plants. Also, the majority of SCADAs software and HMIs can represent data and have drivers to communicate with Microsoft SQL databases servers.

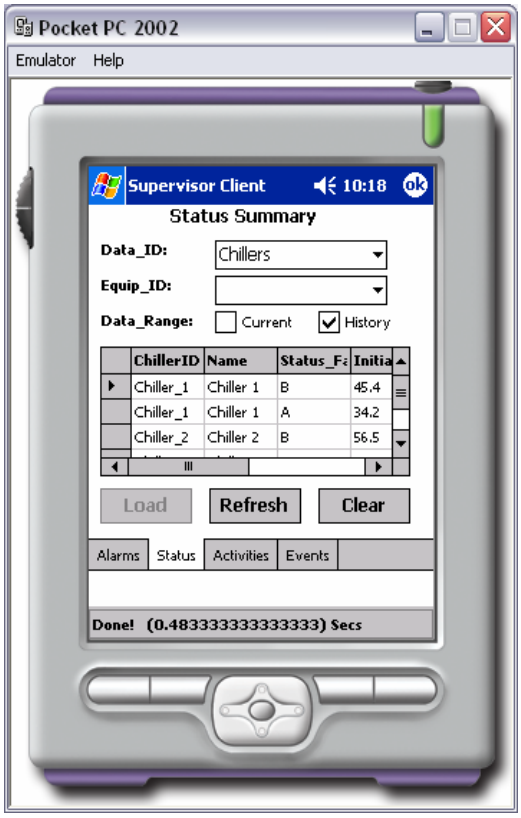

Figure 3.3: Pocket PC Client Application

The messaging schemes for data and control exchange are implemented using the SOAP protocol, since the standard data exchange format is XML language. *SOAP* is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols.

The framework has been designed to be independent of any particular programming model and other implementation specific semantics. The SOAP protocol encapsulates the XML data and query results, and is capable of exchange XML Schemas and other important information using wireless communication standards like 802.11x family Wi-Fi technology.

Schema of Request and Response in SOAP

```
POST /RSService/RSService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://localhost/RSService.asmx/GetOnlineADAS"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetOnlineADAS xmlns="http://localhost/RSService.asmx/">
      <RemoteNetID>string</RemoteNetID>
      <RemoteFedID>string</RemoteFedID>
    </GetOnlineADAS>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetOnlineADASResponse xmlns="http://localhost/RSService.asmx/">
      <GetOnlineADASResult>
        <xsd:schema>schema</xsd:schema>xml</GetOnlineADASResult>
    </GetOnlineADASResponse>
  </soap:Body>
</soap:Envelope>
POST /RSService/RSService.asmx/GetOnlineADAS HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

RemoteNetID=string&RemoteFedID=string
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://localhost/RSService.asmx/">
  <schema
xmlns="http://www.w3.org/2001/XMLSchema">schema</schema>xml</DataSet>
```

The SOAP message presented in the previous figure use HTTP as the transport protocol included in the first SOAP envelope tag. In addition, the first envelope tag contains the body tag that generates the request by invoking the Web method "GetOnlineADAS" from the Web service called "RSService" running on the localhost host server. The first envelope tag is responsible for handling the client request from the host server. Meanwhile, the second envelope tag consists of the SOAP body, which includes the information regarding the response from the Web service method, after processing the request from the client. Finally, the results generated by the Web method are returned to the client as XML data. The XML representation is illustrated with the "xsd:schema" tag.

3.4 Protocols for the Alarms and Status Information

We implemented a series of protocols to process the alarm information, and get the status of system equipments in our system. The description of the protocols is as follows:

- **Alarm Services Protocol –** The alarm services protocol works as follows (see Figure 3.4): After the ADAS receives an alarm information message from any system equipment of a remote data source; it will forward this message to the ANS. The ANS will communicate with the RS to find out the URLs of the DBR's clients that are interested in the alarms. Then, the ANS will broadcast the alarm information to all DBR's clients applications connected until an acknowledgment message is received back from one or more PDA clients. We offer a more comprehensive discussion of the Alarms Services Protocol in Chapter 4.

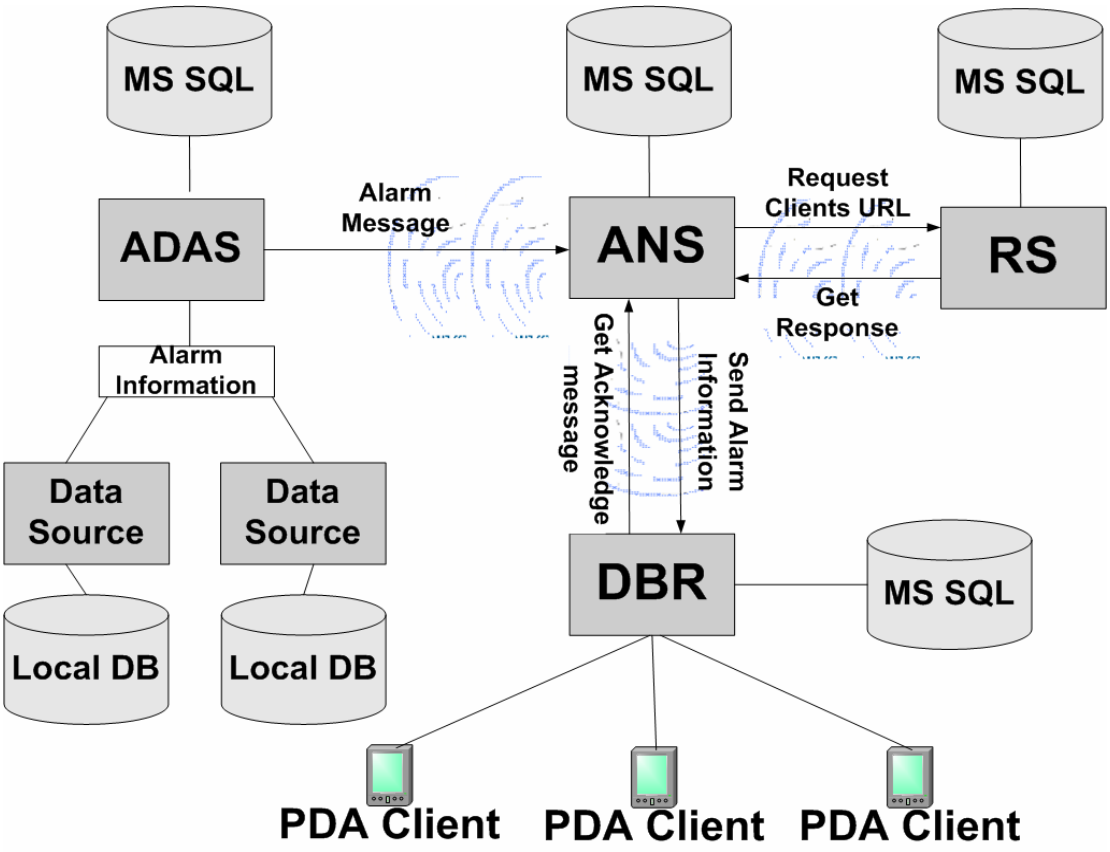

Figure 3.4: Alarm Services Protocol

- **-Status Services Protocol –** The status services protocol works as follows (see Figure 3.5): The PDA client issues a query that the DBR will handle. The DBR will send a message to the RS to find out the URL for each of the available data sources to resolve the query. When the URLs of the data sources that contains the information are sent back to the DBR from the RS, the DBR will be responsible to send the query to the ADAS running on the target data source site(s), collect the results from the data sources, and forward these to the PDA client. We offer a more comprehensive discussion of the Status Services Protocol in Chapter 4.

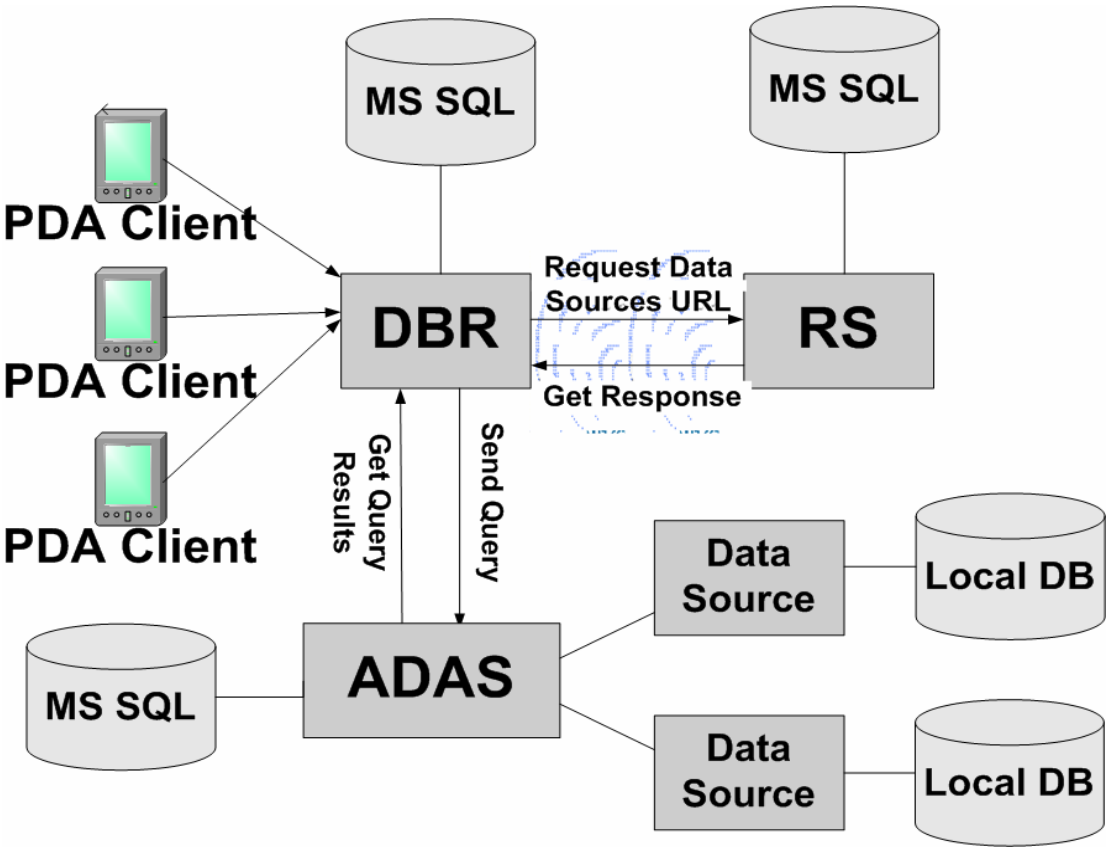

Figure 3.5: Status Services Protocol

We used XML schemas and SOAP to encode the exchange of the control information and query results for all the protocols used by all components in WAMDAS.

3.5 Central Paradigms

The central paradigm upon which WAMDAS is based is the idea that data and supporting environments set up on a given computing device, should accompany the user as he/she moves around his/her work environment. In WAMDAS, the operators, process control engineers and supervisors can monitor status from remote equipments and get near real-time information about critical alarms occurring from one or more equipments in near real- time.

3.6 Applications

WAMDAS allows mobile data sources to access local and remote resources such as sensor data from remote HMIs, including the following devices: Water Treatment Pumps, Turbines, SCADA Control Systems, and other important sensor-enabled equipment.

We can use WAMDAS to establish dynamic federations (workgroups) of devices at manufacturing plants. On a given federation, local data sources will be able to interoperate with each other. For example, Process Control Engineers for the Utilities Area can monitor the Water Treatment Plant control system, as well as other sensors values from the Purified Water System (USP) from their office via a wireless LAN. Likewise, operators from the Cogeneration Area might have PDAs to keep track of their systems including alarms, status of equipments or sensors, and other relevant conditions on their working environments.

During emergency or critical alarm activation, WAMDAS could form a dynamic content network that integrates the data from each unit, to give system administrators or owners clear and real time information of critical status or alarms of their production or utility systems. Moreover, administrators could get a first hand account of the situation before waiting for operator reports and make critical decisions more accurately and faster.

3.7 Chapter Summary

In this chapter, we have proposed WAMDAS to monitor alarms and status information from remote equipments at a pharmaceutical plant. We have shown the system architecture, communication scheme, queries and messages supported in our system. In addition, we have explained the behavior of both status and alarms protocols that provides the scalability and performance of this work. In summary, WAMDAS provides a robust Web Service-based, middleware system, capable of integrate data from remote equipments and seeks to optimize an operator job in a pharmaceutical plant.

Chapter 4

WAMDAS IMPLEMENTATION

This chapter presents a comprehensive description of our system. We present a detailed implementation of the Web methods that form the Web Service-based flavor of our system. In addition, we present a detailed description of our PDA application including the user interface. Finally, we present the PDA client application and Web services interaction in terms of the alarm and status services protocols implementation.

4.1 WAMDAS's Web Services Implementation

To improve the performance of WAMDAS's Web services, we implemented a series of Web Methods for the various WAMDAS services. For this, we implemented each service as asynchronous XML Web service methods. In .Net, each asynchronous call is placed in a background thread creating what is called a thread pool. A thread pool is a queue of threads, which has assigned to it a delegate pointer that signals the main application when the Web service request running in a thread is finished. This enhances the overall performance and scalability of our system by unblocking Web services call and placing them in the thread pool. Since our system is distributed, and each XML Web service communicates with other services, we need to unblock Web Service methods and separate each request in another thread. In Figure 4.1**,** we present an example of one of our asynchronous Web Methods, which is part of the DBR Web Service class implementation.

```
[WebMethod] // Begin Asynchronous Call.
public IAsyncResult BeginGetClientRequest(string RemoteFedID,string
RemoteDataID, string RemoteComponentID, string DateRange,AsyncCallback
callback, object asyncState)
[WebMethod] // End Asynchronous Call.
public DataSet EndGetClientRequest(IAsyncResult asyncResult)
```

Figure 4.1: Asynchronous Web Method

The asynchronous Web Method named *GetClientRequest* is an example of the implementation one of our web methods. We explain in detail the functionality of the Web method in section 4.1.2 as part of the DBR Web service implementation. In .Net, an asynchronous implementation of a Web method is always divided in two Web methods. As we show in Figure 4.1, the first method is the *BeginGetClientRequest*, and the second one is the *EndGetClientRequest*. The begin method always contains the parameters necessary to resolve the user request and is declared as an **IAsyncResult** type. The **IAsyncResult** type indicates in .Net, that a begin method correlates to an end method. In the *GetClientRequest* method example, the parameters lists are the RemoteFedID, RemoteDataID, RemoteComponentID, DateRange, AsyncCallback and asyncState. In our example, the first four parameters are necessary for the method's functionality.. The other two parameters are default in an asynchronous implementation of a Web method. The **AsyncCallback** is a callback type parameter. The **callback** parameter supplies a delegate, which is invoked when the method completes. In .Net, a delegate serves as a function pointer that signals when a thread execution completes. The last parameter is the asyncState. This parameter is an **Object** that allows a caller to supply state information to the method. It allows others Web methods to know the status of their service request to the asynchronous method. Meanwhile, when the EndGetClientRequest method finish, it returns a signal to the *EndGetClientRequest* method in order to return the results to the clients. The end method receives the results from the begin method as a parameter. Finally, the end method returns the results to the original caller, in our example as a XML File read from a dataset.

4.1.1 Web Services Method's Parameters

In our implementation of WAMDAS, we establish as default various parameters for the data exchange of queries and messages issued and requested by each of the Web services. These parameters are the following:

- **RemoteNetID:** The unique ID of the network used at the Registration Service (RS) communication level. This ID identifies the network on which all members of a given group of federation(s) are associated. In addition, it helps to communicate with peer RS components. In our implementation at the pharmaceutical plant, we used the following network ID: http://pfizernet.net.
- **RemoteFedID:** The unique ID of the federation. It identifies a member of the federation and it is part of each query and messages handled by the system. In our system, the federations IDs used are http://CogenFed_12.net, http://ProcessFed_10.net and http://SupervisorFed_11.net.
- **RemoteDataID:** The unique data ID for a group of components. This ID is used to identify the query type of the status data query. In addition, the data ID is used at the alarm notification message to identify the equipment family in which the equipment is part of. The following are some examples of data IDs for the Cogen and Process federations: http://BoilersData_12.net, http://ChillersData_10.net and http://CapsuleData_12.net.
- **RemoteComponentID:** The unique component ID that identifies particular equipment. The component ID is used to identify a single component query type in the status data query. In addition, is the component ID is used in the alarm notification message, and acknowledgement indication query to identify the equipment in alarm. Some examples are Boiler_1, Chiller_3 and RecPump_1.

We have implemented other parameters that are not defaults in all messages or queries, such as AlarmTime (the time at which the alarms was triggered) and AknTime (the time at

which the alarm was handled by the user) among others. We shall see these parameters in the discussion that follows.

4.1.2 The Data Broker Web Service (DBR)

DBR is a Web service that works on behalf of the PDA clients to find the data and query processing capabilities necessary to solve a given query. This element has responsibilities such as query plan generation, query coordination, and acquisition of results. DBR is responsible for processing client's request from operators, process control and supervisors at the pharmaceutical plant in order to get status of equipments and to process alarms messages from the ANS service back to the Pocket PC client. To fulfill this, we implemented a series of asynchronous Web methods as shown in Figure 4.2**.** These Web Methods are part of the DBR Web Service class ("DBRService.cs") public interface.

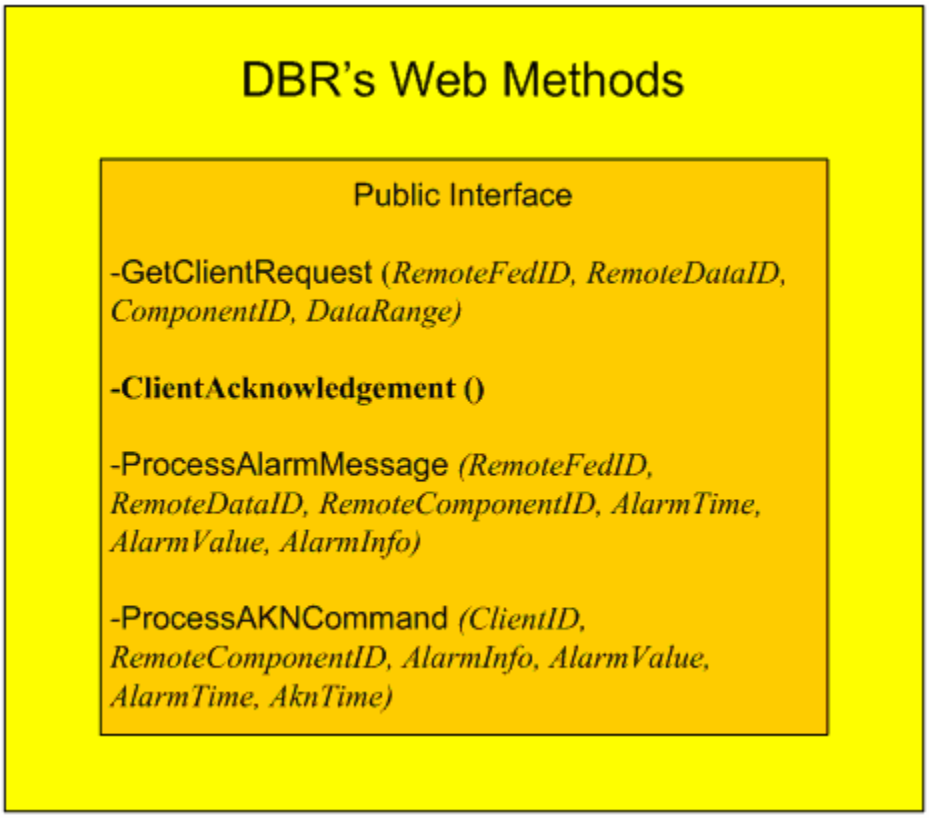

Figure 4.2: DBR Service Web Methods

The DBR has four important Asynchronous Web Methods to facilitate status information query and acknowledgement of alarms. The *GetClientRequest* method receives the PDA client query for status information from one or more equipments. In addition, it has the *ProcessAlarmMessage* method, which broadcasts the alarm message received from the ANS to the available clients connected to the network. Meanwhile, it has the *ClientAcknowledgement* method, which is used to retrieve alarms acknowledgement information from the ANS. It processes a query from the client to get the alarm history information including when, who and what alarm an operator, process control or a supervisor acknowledged. Finally, we implemented the *PorcessAknCommand* method. This Web Method receives the command of the PocketPC client for the acknowledgement notification query. It is used to process acknowledgement from operators, process control engineers and supervisors.

4.1.3 The Registration Web Service (RS)

RS is a Web Service used to coordinate access to a federation and to the resources, it provides. In addition, it handles all bookkeeping necessary to log events such as departure of a federation system user, or reappearance of a member at a different network or federation. RS provides important information about the availability of a remote DBR, ANS or ADAS component. It handles request from the DBR, ADAS and ANS each time a status query or alarm message is generated by the system. In our implementation, we partially implement the functionality of the RS adapting it to our system. We get a subset of methods from the Net Traveler's RS architecture. As part of our RS implementation, we implement the following Web Methods (see Figure 4.3). These Web Methods are part of the RS Web Service class ("RSService.cs") public interface. The first method is the *GetOnlineDBR;* it provides to the ANS the status of the available DBR to process the alarm notification message.

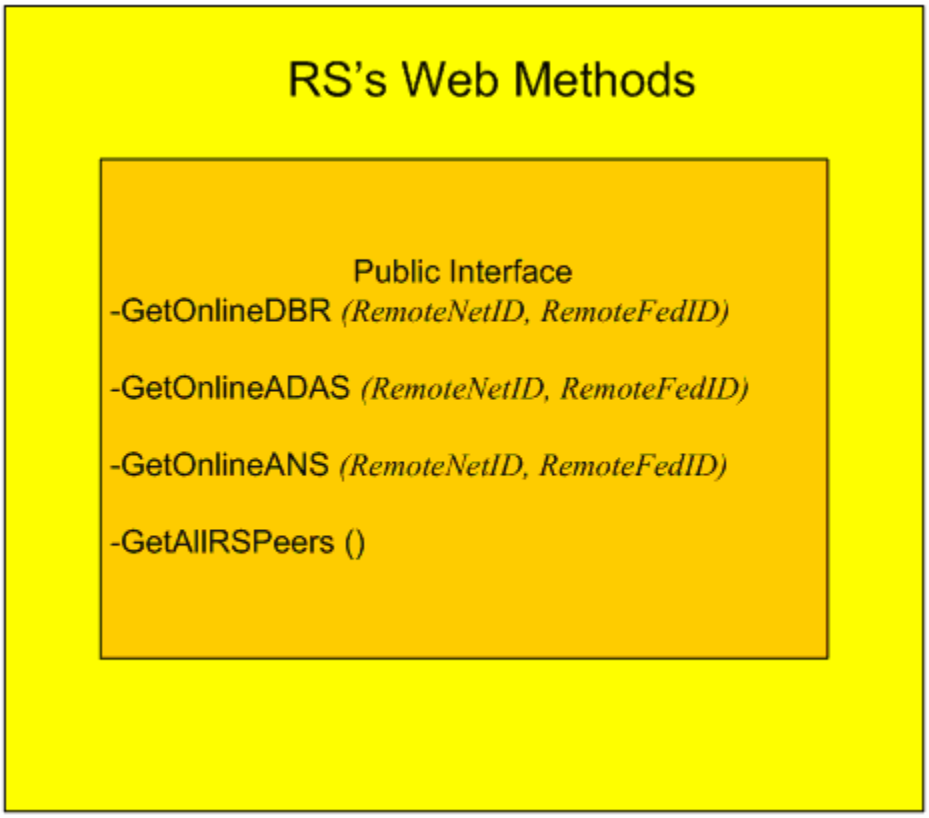

Figure 4.3: RS Service Web Methods

The second method is the *GetOnlineADAS,* which provides to the DBR the status of the available ADAS to process the status data query. The third method is the *GetOnlineANS* method. This method provides to the DBR the status of available ANS to process the acknowledge indication query. Likewise, it provides to the ADAS component, the status of the ANS in order to process the acknowledge data query. Finally, it has the *GetAllRSPeers* method. This method negotiates with peers RS in order to resolve a query in case that the requested data source or Web service component are not available in the local network.

4.1.4 The Alarm and Data Acquisition Web Service (ADAS)

ADAS is a Web service responsible for monitoring any possible alarm information from all system components. When an alarm message is generated, the server will forward this message to the ANS for broadcast and acknowledge actions. In addition, it acts like a wrapper handling relevant information of data sources in order to retrieve status information.

To implement this functionality, we implement a series of asynchronous Web methods as part of the ADAS Web service class (ADASService.cs) public interface (see Figure 4.4).

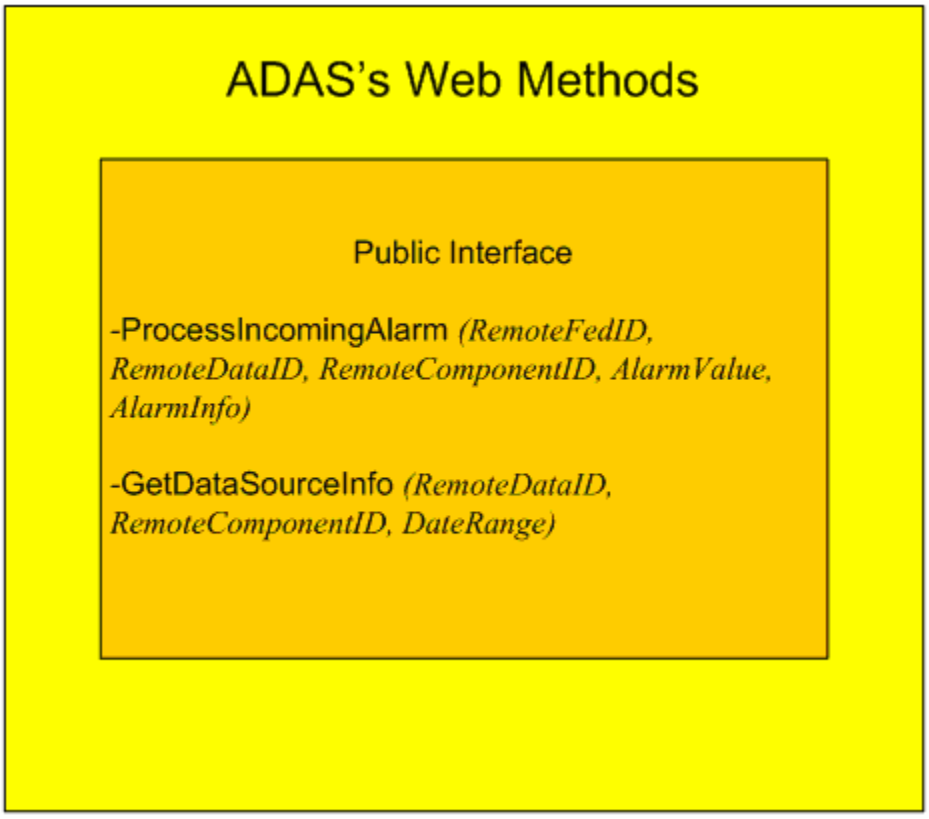

Figure 4.4: ADAS Service Web Methods

The *ProcessIncomingAlarm* method is responsible for broadcast the alarm message received from the remote data source to the available ANS connected to the network. It contacts a peer ANS site through the RS service to communicate the alarm message. In addition, it has the *GetDataSourceInfo* method. This method adds the wrapper functionality by providing valuable information of bonafide data sources that are part of the network. For this, it has the defaults parameters of the data and component IDs.

4.1.5 The Alarm Notification Web Service (ANS)

The ANS Web service is responsible for broadcasting of all critical alarm(s) or information to all interested clients. As we mentioned before, this component is the most

critical for the operation in a pharmaceutical plant. It is responsible of broadcasting critical alarms and process the acknowledgements indication from operators, supervisors and process control engineers. The Web service will require response from at least two clients after each issue of an alarm notification message. In order to implement this functionality, it has three asynchronous Web methods (see Figure 4.5). These methods are part of the ANS Web service class (ANSService.cs) public interface. The first one is the *GelAlarmMessageInfo*, which is responsible to search in its local database for acknowledgement information of previous events. The second one is the *AlarmBroadcast* method. This method has the functionality to broadcast the alarm message to the interested DBRs. The third one is the *ProcessAcknowledgemen*t method. This method receives and stores the acknowledge notification query from remote DBRs.

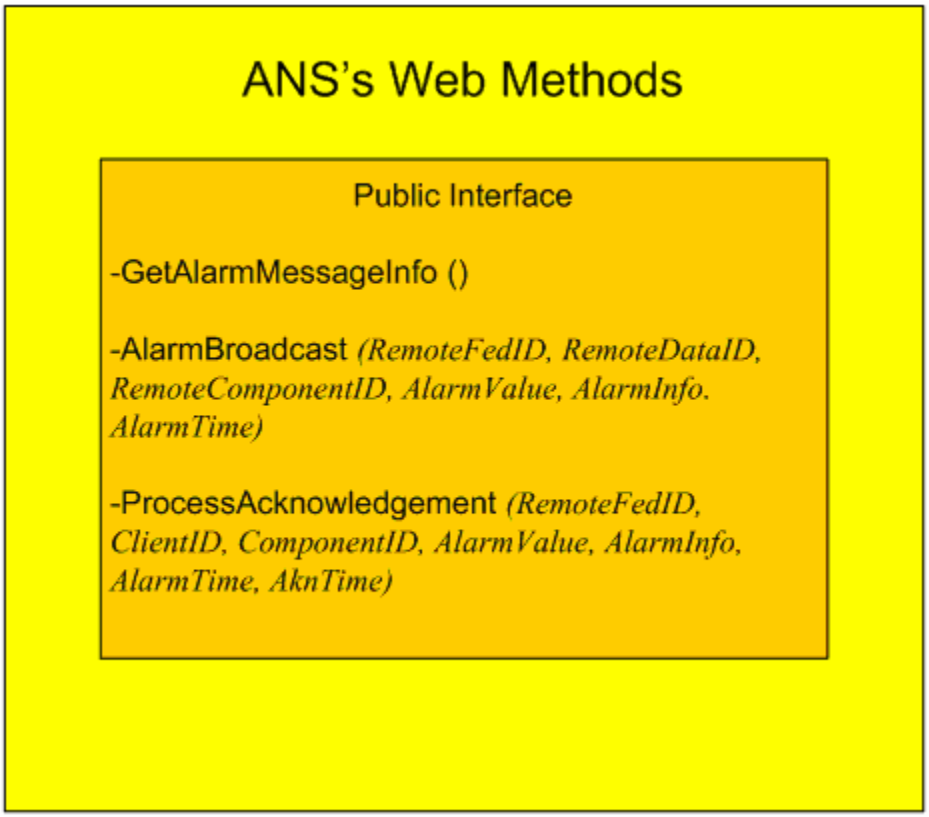

Figure 4.5: ANS Service Web Methods

4.2 PDA Client Application

The PDA client application forms an essential part of our implementation of WAMDAS. This application is a powerful tool that helps operators, supervisors and process control engineers to receive alarm notification messages, acknowledge those alarm messages, or even verify the acknowledgement information history for each of the alarms previously generated by real contingencies that occurred during the operation of the plant. In addition, it receives input request for status of equipment or a group of them. In Chapter 5, we present a user satisfaction evaluation of the PDA application and a comparison between it's functionality versus the actual system.

4.2.1 PDA Application Design

As we mentioned before, the PDA application is a tool that helps interested users to manage their relevant data such as alarms or status of equipments related to a pharmaceutical plant work environment. For this reason, we designed it to adapt and fulfill the desired functionality. In our implementation phase, we needed to make the application comply with two requirements. The first one was that it should be capable of acting like a client in order to request data from a Web service, in this case from the DBR component. The second one was the requirement of acting like a server in order to receive alarm(s) from the DBR component. These alarms are generated from remote data sources connected to the plant's equipments. For the first functionality, we implement the Web service calls for status data; acknowledgement indication and acknowledgement data queries as asynchronous call. Implementing the Web service calls asynchronously, helps to prevent the application from freezing on each call by placing it in a background thread, known as a delegate. When a client requests data through a Web service call, the application places that call in the background assigning it a delegate pointer responsible to process the client request and return the results of each request.

Meanwhile, for the server role, it was necessary to use TCP/IP sockets to handle incoming alarms notifications from the DBR. This was possible by using an asynchronous

server socket capable of handling multiple alarms in the background. It was necessary to use a socket because current PDA operating systems provide little support for hosting services like regular PCs and servers OS. When the PDA client application is loaded, it opens a socket listener as shown in Figure 4.6. This asynchronous socket will handle each alarm notification placing it in the background, allowing the user to manage the other application's functionality. In short, our PDA applications is robust multithread solution tool capable of handled multiple incoming alarm messages, acknowledgement indications and acknowledgement data queries. It is also capable of handling in the background status data queries requested by the user.

Figure 4.6: PDA Socket Listener

4.2.2 PDA Application User Interface

The development of the user interface was another important challenge for the PDA application implementation. We designed the user interface as simple as possible taking into consideration that Pocket PC devices have some limitations in comparison to normal PCs. These limitations include small screen, keyboardless interface and poor power management

among others. For this, we decide to minimize any possible data entry from the user. In our application, the user can interact without the need of data entry that requires using the soft key panel or other input method that requires writing information or graffiti.

(a) Alarms Summary Page

(b) System Activities Page

(c) Status Summary Page

(d) Acknowledgement History Page

Figure 4.7: User Interface Tab Pages

In addition, we split the user interface in tab pages using a Tab Control and each tab page correlates to the important information regarding alarms notifications and acknowledgement indications, status of equipments, and acknowledgement data queries as shown in Figure 4.7. The Alarms Summary Page (see Figure 4.7(a)) is related to the display of incomings alarms. In order for the alarm(s) to be placed in the screen, the user has to detect the alarm notification message. An example of an alarm notification message is shown in Figure 4.8. As we mentioned before, the alarm notification message is processed by an asynchronous socket, which place it in the background. This make it possible to receive this message no matter if the user is interacting with the Alarm tab or another one of the application tab pages, or even other PDAs applications. In the Alarm tab, users can acknowledge one alarm by selecting it from the List View control. In addition, interested users can acknowledge all alarms with one command.

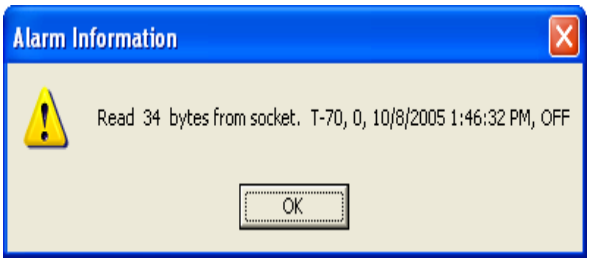

Figure 4.8:Alarm Notification Message Screen

The System Activities Page (see Figure 4.7(b)) provides the user with options to load acknowledgement historical data from previous events. It lets the user choose from a Combo Box list the Federation on which the acknowledgement information is wanted. The results are displayed in a tabular form in the Data Grid control. The Status Summary Page (see Figure 4.7(c)) lets the user select from two different combo boxes the equipments or group of equipment of which the status information is wanted. After selecting the equipment, users can load the status information to the Data Grid control. Finally, the Acknowledgement History Page (see Figure 4.7(d)) provides users with copies of receiving alarm notifications and acknowledgement indications. It helps the PDA user to recall what important actions he/she executed during the utilization of the application. This information resides in memory and is only related to the current user session.

As we have shown, the PDA application fulfills our needs in terms of simplicity, and at the same time, is a powerful solution for operators, supervisors and process control engineers in a pharmaceutical plant. It should be noticed that we used standard windows forms controls that are used in PC applications. In addition, we simplify the user interaction by eliminating typing or soft screen selections in favor of using the PDA stylus throughout the application.

4.3 Status Services Protocol Implementation

In our implementation, of the Status services protocol, we divide it in three application layers as shown in Figure 4.9. The first layer pertains to the user interface and includes the PDA client application.

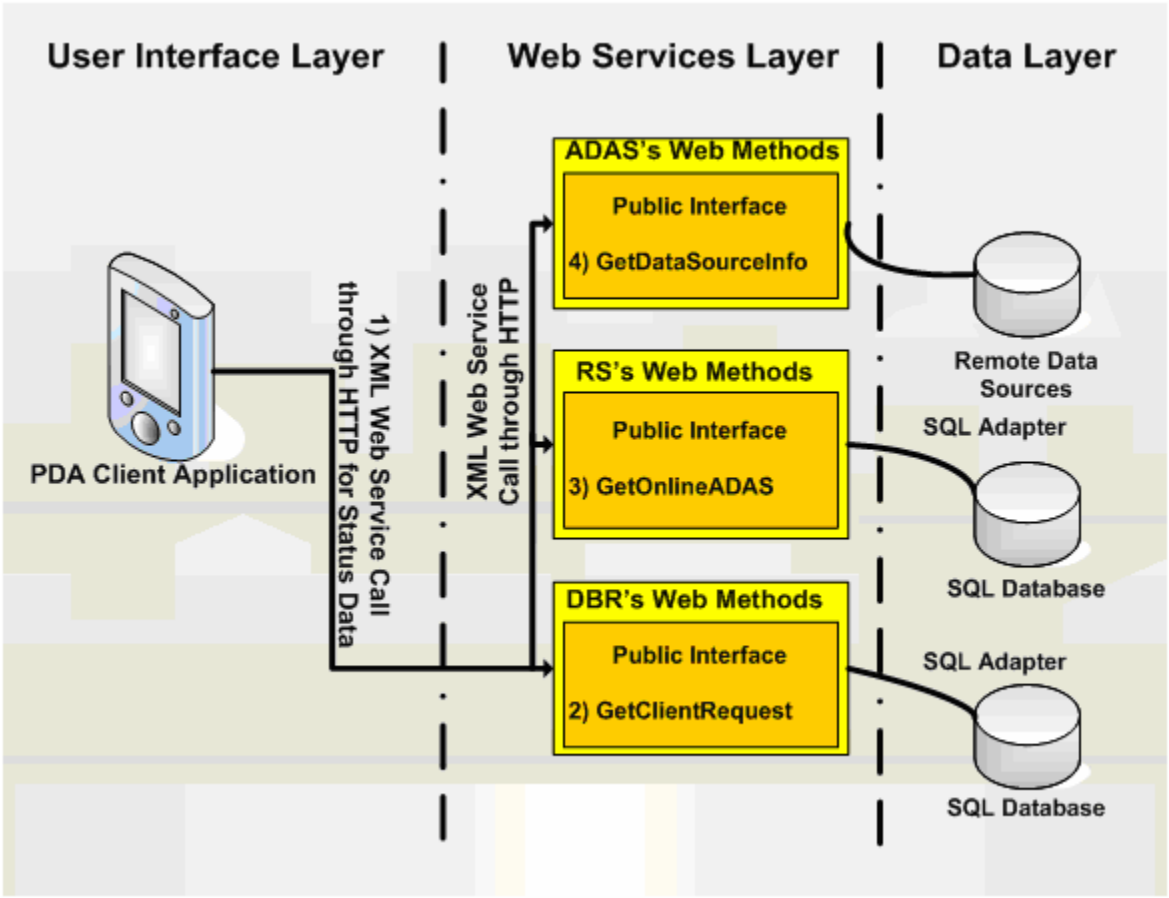

Figure 4.9: Status Services Protocol Implementation

The PDA client application provides the interface to the end-user, in this case operators, supervisor and process control engineers from a pharmaceutical plant. In the status services protocol, the following sequence of request through SOAP messages is used: 1) the PDA application request data from the Web services layer trough HTTP. It communicates with the DB*R's GetClientRequest* method. 2) The *GetClientRequest* method searches the SQL database for the network and federation IDs. After retrieving both IDs from the database, it makes an XML Web service call passing the two parameters to the RS's *GetOnlineADAS* method in order to get the ADAS addresses that are online in the network. 3) When the RS receives the DBR request for ADAS availability, it searches in a local SQL database catalog for the information regarding status and address information. After that, the RS proceeds to send the information to the DBR. When the DBR receives the information from the RS, it sends the status data query to the available ADAS by invoking the *GetDataSource*s method. 4) The ADAS starts to communicate to the remote data sources that match the Data ID, Component ID and data range, which identifies the query type related to the equipment(s). The ADAS retrieves the data and sends it back to the DBR component. Finally, the DBR sends the results to the PDA application. As we can see, the third layer is related to the databases of our Web services, and the data sources that are related to the HMIs and SCADAs systems.

For the data exchange, the SOAP protocol is used to envelop the data in all of the Web services calls via HTTP. On each call, the data are encapsulated in XML format as shown in Appendix A. Meanwhile, we used SQL adapter's objects from .Net to process all data request related to the SQL databases of each of the components. For the PDA client application, datasets objects were created to retrieve the status data query results from the DBR service.

4.4 Alarms Services Protocol Implementation

The Alarms services protocol exhibits a more complex implementation in comparison with the Status services protocol as shown in Figure 4.10. It still was divided in three application layers, the PDA client application not only requests data from the Web services, but also receives data from them, in both cases, the interaction occurs with the DBR service. In order to process incoming alarm notification messages, the Web services interaction is as follows: 1A) The ADAS receives the alarm message from a remote data source and process it with the *ProcessIncomingAlarm* method. Then, it communicates with the RS's *GetOnlineANS* method in order to retrieve available ANS services that are interested and related to the given federation and data IDs. 2A) From the RS, the ADAS receives the status and address of the ANS.

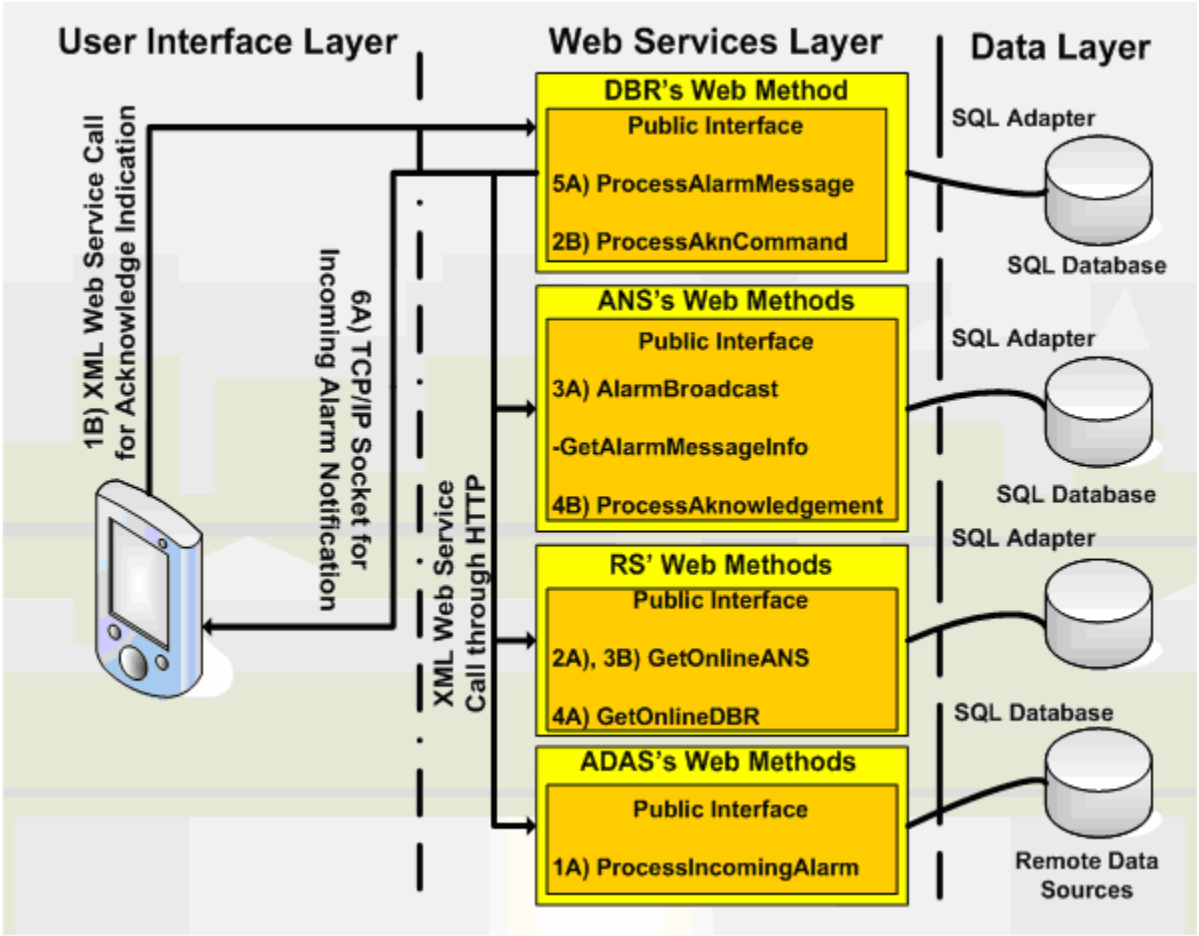

Figure 4.10: Alarm Services Protocol Implementation

After that, 3A) the ADAS proceeds to broadcast the alarm message to the ANS service by calling the *AlarmBroadcast* method of the ANS service. 4A) The ANS communicates with the RS service to get available DBRs. After receiving the information from the RS, the ANS proceeds to call the DBR' *ProcessAlarmMessage* method. 5A) The *ProcessAlarmMessage* method searches for available clients into the local SQL database. 6A) After retrieving IP addresses of online clients, it sends the alarm message through a TCP/IP socket client to PDA clients connected to the network, which is the final step before receiving an acknowledge indication query from the interested client(s).

Another important functionality of the Alarm services protocol is the acknowledgement indication after receiving an alarm notification message. This acknowledgement indication works as follows: The user submits an acknowledgement indication through the PDA client application of one or more alarms. 1B) The PDA application communicates with the DBR's *ProcessAknCommand* method. 2B) The DBR contacts the RS's GetOnlineANS method for available ANS services. 3B) After receiving the RS information, it proceeds to contact the ANS's *ProcessAcknowledgement* method. 4B) The ANS service will process the acknowledgement indication storing it in a local SQL database. Meanwhile, if the data related to previous acknowledgement indications are wanted for analysis or further investigations, the ANS provides the *GetAlarmMessageInfo* for these purposes.

We implement the data exchanged for the Web services interaction using HTTP, SOAP and XML (see Appendix B). However, for the alarms notifications, the implementation used TCP/IP sockets. For the Data level, all Web services local SQL databases were accessed using SQL adapter objects provided by the .Net framework.

4.5 Chapter Summary

In this chapter, we have provided a comprehensive description of our implementation of WAMDAS. We have shown how we implement our Web services methods capable of handling multiple requests from PDA clients. For accomplish this, we implement all relevant

services as asynchronous methods. We described the most relevant methods for each of the Web services. In addition, we presented a detailed description of the PDA client application including the user interface. Finally, we presented the implementation of the Alarms and Status services protocols including the interaction of each of the application levels. We show the three application levels of the Alarm and Status services protocol. These levels are the User Interface, Web Services and Data. These applications levels complement our case of a robust middleware, distributed system capable of integrating mobile devices with Web Services technologies to monitor status information and at the same time receive critical alarms from remote data sources that are connected to numerous equipment in a pharmaceutical plant.

CHAPTER 5

EXPERIMENTS AND RESULTS

We conducted a series of experiments on our WAMDAS prototype, which shows the scalability of our system in terms of alarms processing power and status request management. In section 5.1, we show information on measures of the performance of the Status and Alarm services protocols. The purpose of these tests was to measure how our WAMDAS prototype performs under extreme workload of alarms messages and acknowledgement, using an automated PDA client and an alarm simulator. We used an automated PDA client to acknowledge alarms and send status queries. Similarly, the alarm simulator was a program used to generate alarms at various rates. In addition, in section 5.2, we show results on a user study conducted based on WAMDAS to evaluate the usability and effectiveness of the system from the perspective of end-users. We prepare a scenario in which we generated several alarms and we asked end-users to use WAMDAS's PDA application. The end-users were real operators working on a regular shift at a pharmaceutical plant. At the end of the exercise, each user was required to complete a satisfaction survey that compares the actual system using existing HMIs and SCADAS systems versus our PDA application and WAMDAS's functionality.

5.1 Performance Evaluation

As we mentioned before, our performance evaluation of WAMDAS seeks to measure how the system reacts under a heavy workload of alarm messages and status data query requests. In addition, it shows how the three federations defined for a pharmaceutical plant environment interact with each of the Web services in a distributed environment. For this, purpose, we simulated three configurations at different fixed intervals of times.

5.1.1 Computer Equipments and Software Tools

Our performance evaluation of the Alarm and Status services protocols were executed on twenty-five workstations located at the ADMG laboratory at the University of Puerto Rico, Mayagüez, with the following characteristics: 2.40GHz Pentium IV with 1 GB of RAM, 80 GB HD, and all of them were networked by a 100 Mpbs Ethernet. All of these machines were running Windows XP Professional and Visual Studio.Net 2003. Meanwhile, we simulated the client applications on fifteen computer, using Windows PocketPC 2002 emulator. In addition, all computers running the WAMDAS' services used the Internet Information Services (IIS) Web server version 5.1 for Windows XP. Finally, for the database software, we used MS SQL Server 2000 for each of the Web services and data sources.

5.1.2 Configuration for the Experiments

The organization for the configuration of the Web services used in these experiments is shown in Figure 5.1. We placed the DBR, ANS and ADAS Web services for the Process Control, Supervisor and Cogen federations on nine different computers.

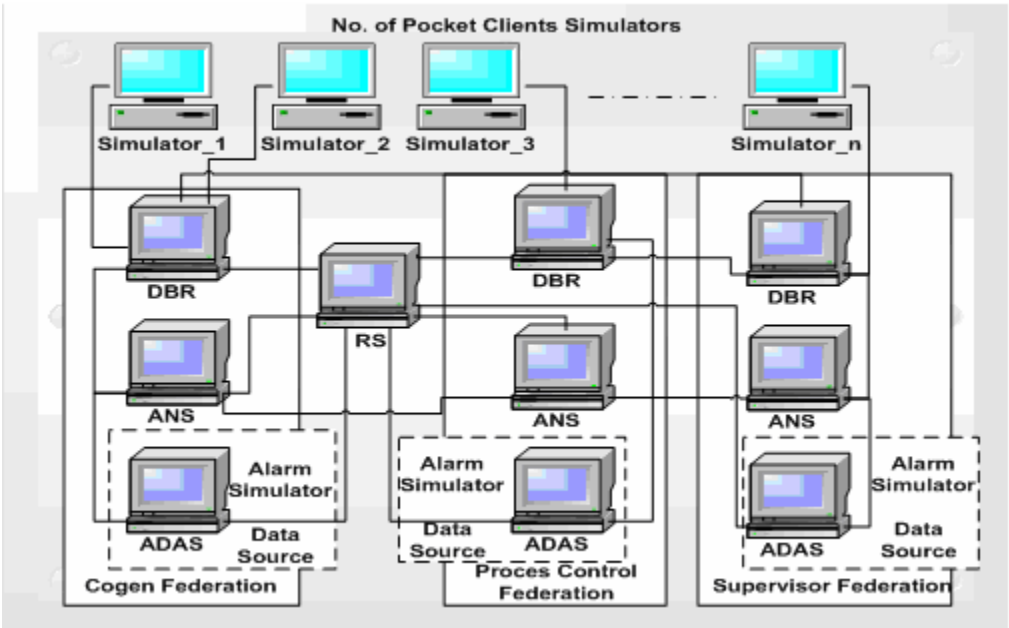

Figure 5.1: The configuration used to run our performance experiments

The RS component was placed on a different computer to handle all federations' requests. For the client simulators, we placed one simulator on each of the fifteen remaining computers, making fifteen clients. In addition, three different Alarm Simulators were placed on each ADAS computer for simulating alarms for a fixed period of time for each federation. Finally, we placed the data sources containing the alarms information and status of equipments on three computers, those where the ADAS was running.

5.1.3 Performance Evaluation of the Alarm Services Protocol

The first metric that we measured was acknowledgment time for alarms. For this purpose, we implemented the Alarm Services Protocol. We ran the system for various time intervals and generated alarms randomly from the alarm simulators at three different frequencies. The specific intervals and number of alarms were approximately as follows: a) ten minutes and at least from ten to one hundred twelve alarms ((10-15)-(100-112)), b) thirty minutes and at least thirty alarms to three hundred fifteen alarms ((30-40)-(300-315)), c) sixty minutes and at least sixty five to sixty seventy seven hundred alarms ((65-74)-(660-677)), and d) one hundred twenty minutes and at least one hundred thirty to twelve hundred fifty five alarms ((120-132)-(1240-1255)). For each alarm, each of the Web services processed fifteen acknowledgement messages. In Table 5.1, we present the tabulated results of each simulation time. For all simulation times, we repeat our experiment three times and average them. We show the total time spent in acknowledments and the total of acknowledgement messages from fifteen clients connected to the system. We show, the total of acknowledgement messages workload of the three configurations, for each of the simulation time, round in thousands. This shows that our system is capable of handling huge amounts of alarm notifications generated from several equipments in a pharmaceutical plant. In addition, the results evidence that our system can easily handle, thousands of acknowledgement messages from interested users.

Table 5.1: Alarm Services Protocol Results

Simulation Time (minutes)	Total Akn. Time (seconds)	Acknowledgement Messages
10	95.38972	2170 – 2180
30	240.7437	6350 - 6362
60	725.2505	13935 – 13955
120	1017.457	27148 - 27160

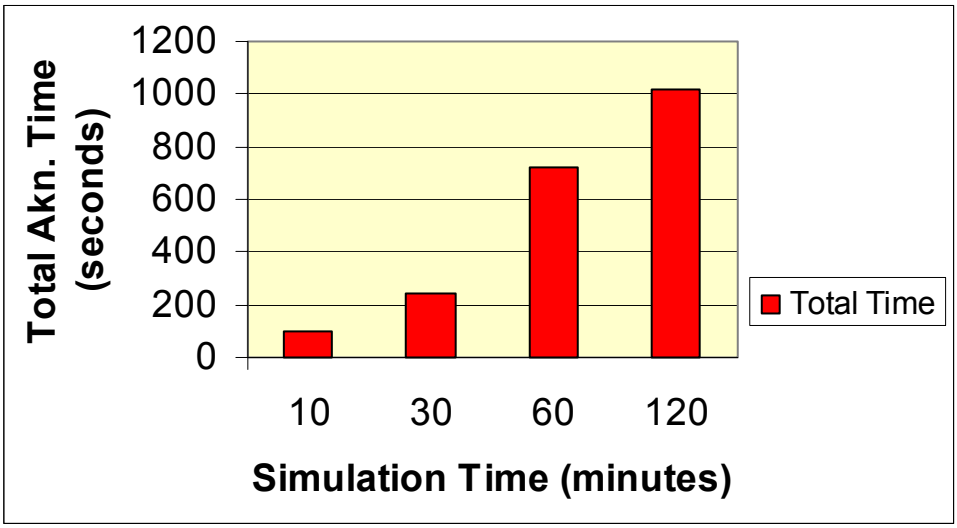

Figure 5.2: Total Acknowledgement Time (seconds) vs. Simulation Time (minutes)

Likewise, Figure 5.2 presents the total time spent in processing acknowledgment messages. As expected, this time increases with the time in which alarms are processed.

However, in our experiments we found that the average time to acknowledge alarms had a more complex behavior. In Figure 5.3, we showed the average acknowledgment time for the three configurations, and various time intervals. As we can see from the pictures, the average acknowledgment time starts to increase with time, but then decreases and levels out. This happens because in our implementation of Web services, the objects that implement the ANS are cached in memory and re-used by MS .NET rather than creating new ones. As the number of message to acknowledge increases, these objects can be found more frequently in memory, thus reducing overhead in object creation. Also, notice that our system does not reach saturation at alarms rate that are larger than the typical rate at which alarms are

generated in a plant. This shows that the system can grow to accommodate more load as sensors are added to automate more monitoring tasks. Also, noted, that in all simulation times, the average acknowledged time was less than a second. These results showed that our system performs well under heavy workload of alarm notification messages and acknowledgement indications.

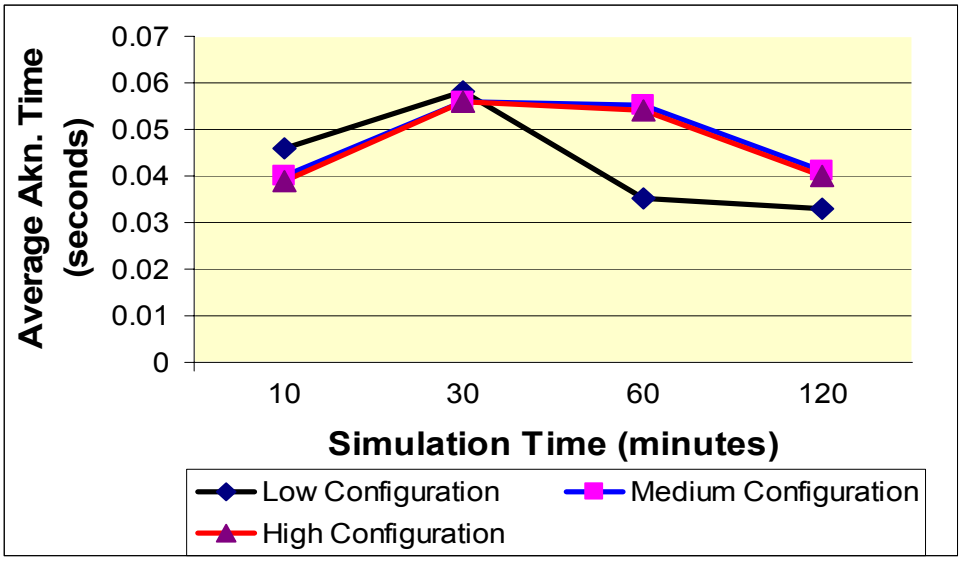

Figure 5.3: Average Acknowledgement Time (seconds) vs. Simulation Time (minutes)

5.1.4 Performance Evaluation of the Status Services Protocol

The other metric that we wanted to measure was the average times to execute a status request with different fixed times. For this test, we used the Status Services Protocol. We ran the system for various intervals as in per the alarm test. However, in this case we generated a status request workload consisting of two query types which ask for a) Type I - status of a given equipment instance (e.g. water pump number 11), and b) Type II - status of a class of equipment. We had seven binding values to choose from for query Type I, and twenty-five binding values for query Type III. Again, we let the simulators randomly chose query types and binding values for various time intervals. We present the tabulated results in Table 5.2. As in the Alarm Services Protocol, we repeat our test three times and average them. In the table, we show the total time spent in Status request and the total number of

status requests per client. As in the Alarm Status Protocol test, the results demonstrated that our system is capable of handling thousands of status requests from one or more equipments deployed in a pharmaceutical plant.

Table 5.2: Status Services Protocol Results

Simulation Time (minutes)	Status Request time (seconds)	Number of Status Request
10	103.043	2160 – 2165
30	247.24	6300 – 6320
60	737.091	13875 – 1396
120	1074.937	27120 – 27134

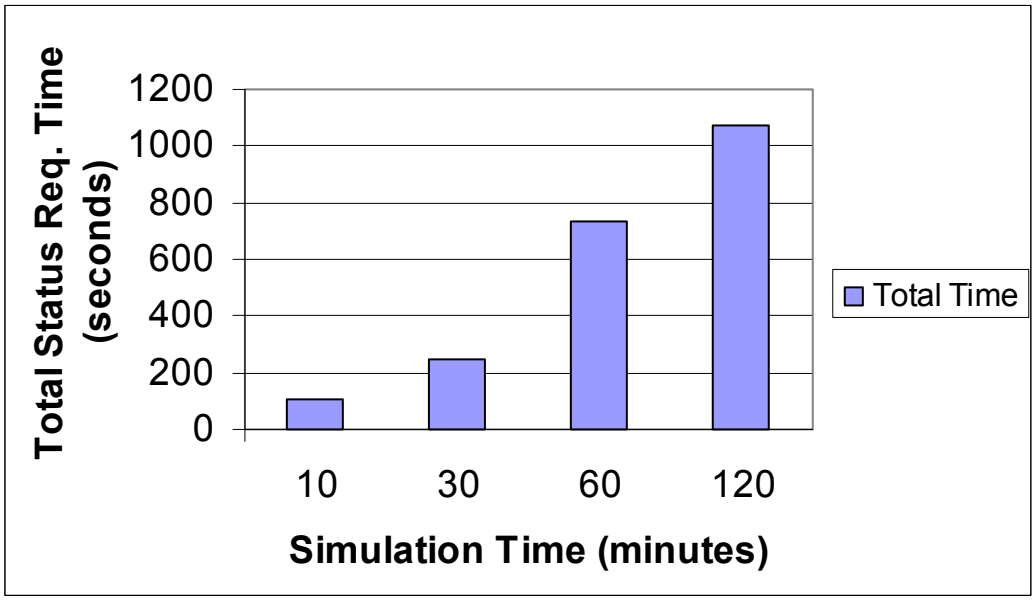

Figure 5.4: Total Status Request Time (seconds) vs. Simulation Time (minutes)

In Figure 5.4, we showed the total time spent in processing status request queries. As expected, this time increases with the time in which the status queries are processed.

Meanwhile, in Figure 5.5, we showed the average time to answer status queries. We noticed that the average time first decreases as time for simulation increases. This is due to the factor of caching and re-using the C# object that implements the Web service, since the overhead in object creation in diminished.

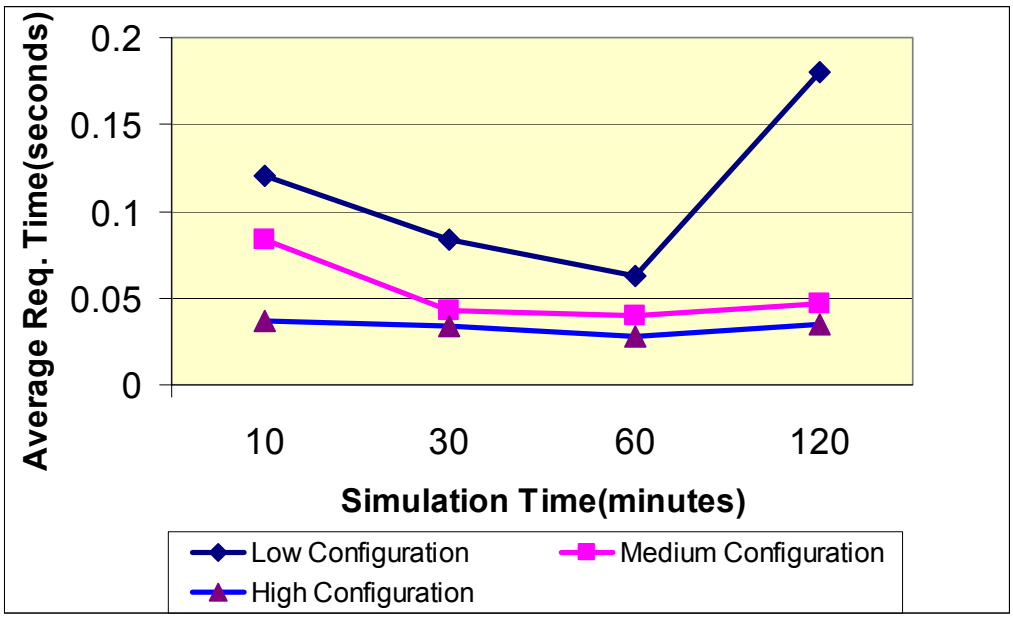

Figure 5.5: Average Status Request Time (seconds) vs. Simulation Time (minutes)

However, at sixty minutes we reached a saturation point in which the system takes more time to respond as the number of queries per unit of time increases. Clearly, it would be necessary to balance the load on the system by adding more DBR, RS, or more federations as the number of clients searching for status information increases. Thus, as our experiment for the Status Services protocol reaches a saturation point at the sixty-minute test, it still performs well with average values of less than a second. In addition, it is important to clarify that the data requested because of the query types, selected randomly, can cover one or more equipments. This can cause more loads to the system, since not all the status request issued by the simulators are the same.

5.2 User Evaluation at the Pharmaceutical Plant

We also conducted a user study of WAMDAS to evaluate the usability and effectiveness of the system from the perspective of end-users. As we mentioned before, we prepared various scenarios in which several alarms were generated and end-users were asked to utilize the PDA application to acknowledge these alarms. The test intends to monitor how the user

interacts with the system and how effective they can be when handling one or more alarms using our system. In addition, we let them use the system to get status of different equipments. The test shows how often the users make a status request and how effective they can be in comparison with the typical number of roundtrips to the field or equipment, twice per shift. For both tests, we instructed each user on how to use the PDA client application of WAMDAS.

5.2.1 Computer Equipments and Software Tools

Our evaluation of WAMDAS at the pharmaceutical plant was performed using the following computer equipments:

- 1 Dell Latitude Computer Pentium 4 1.60GHz with 512MB of system memory. In addition, the computer has an Airway wireless PMCI card.
- 1 HP Omnibook computer Pentium III 644MHz with 256MB of system memory. In addition, the computer has an Airway wireless PMCI card.
- 1 Dell Inspiron 600M computer Pentium M 1.60GHz with 512MB of system memory. The computer has an internal wireless LAN IntelPro card.
- 2 HP PocketPC Intel ® PXA250 with 62.91MB of system memory and an internal Wi-Fi anthena.
- 2 HP PocketPC Intel ® PXA255 with 123.63MB of system memory and an internal Wi-Fi anthena.
- 1 Wireless access point CISCO 350.

Meanwhile, the software tools used for the HP and both Dell computers were Windows XP Professional and Visual Studio.Net 2003. In addition, for the database software and Web server, we used MS SQL Server 2000 and Internet Information Service (IIS). On the other hand, both of the HP PocketPC PXA255 was running Windows PocketPC 2003 software. Likewise, both of the HP PocketPC PXA250 was running Windows PocketPC 2002. Finally, we used the CISCO 350 access point to provide the wireless communication channel for all computers and PocketPCs.

5.2.2 Configuration of the Experiments

The organization for the configuration of the Web services and PDA client applications used in these experiments is shown in Figure 5.6. We placed the DBR, ANS and ADAS Web services for the Process Control, Supervisor and Cogen federations on three different computers. The DBRs services were running on the HP Omnibook computer. For the ANSs and RS services, we placed them on the Dell Latitude computer. Meanwhile, we placed the ADAs, data sources and the alarm simulators on the Dell Inspiron 600M computer. In addition, we installed the Cisco 350 access point on the roof of the Cogeneration room. Finally, we handled the four PDAs running the client application to the users helping in the experiments.

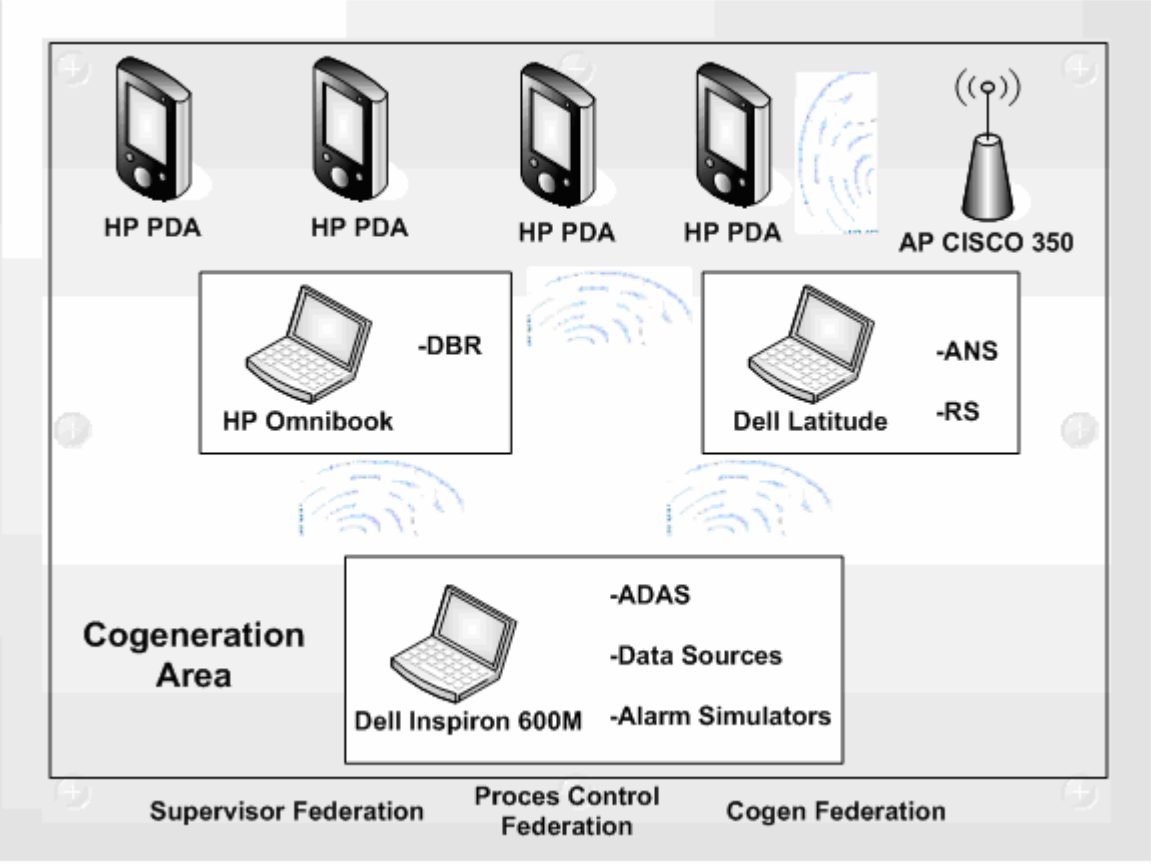

Figure 5.6: The configuration used to evaluate our system at the Pharmaceutical Plant

5.2.3 Alarm Monitoring Test

The first test that we performed was how users handle incoming alarms from the system. For this test, we recreated three cases from previous alarms events gathering from existing logbooks of the Cogeneration area as shown in Table 5.3. The first alarm case was related to an alarm occurred in the Clestra Room of the Warehouse building. The Clesta Room is an area for the storage of empty cases to store pills. We found in the logbook, with the help of the Cogeneration Supervisor, that an alarm of high humidity occurred at 3:00 PM but acknowledged at 6:00 PM. We found, that an official of the area alerted the operator of the alarm before his regular roundtrip to the area, but it took three hours to acknowledge the alarm message in the HMI. Then, the operator proceeded to troubleshoot the humidity sensor and resolved the alarm condition as stated in the comments section of the logbook. The second alarm case was related to a high temperature alarm triggered by power outage that occurred in the Cool Room of the Warehouse building. As in the humidity alarm case, some of the workers from the building alerted the operator. We found in the logbook, that the alarm occurred at 1:00 PM, but the operator acknowledged command registered at 3:00 PM. For this case, it took two hours for the operator to acknowledge the alarm. Then, he proceeded to correct the electrical problem. Finally, the third alarm case was related to a low chlorine level concentration occurred in a Potable Tank from the Pump House area. We found in the logbook, that the operator was notified from an independent contractor that an audible horn indicator was activated in the area. From the comments written by the operator, we found that the alarm occurred at 3:15 PM and he acknowledged it at 4:18 PM. In this case, it took one hour and three seconds for the operator to acknowledge this alarm.

Table 5.3: Real Alarm Cases

Event	Acknowledge Time Per Real Alarm Cases (minutes)		
Clients	Alarm Case 1	Alarm Case 2	Alarm Case 3
Cogen Operator	180	120	60.3

After collecting the information of real alarm cases, we configured our test with the help of real operators, supervisors and process control engineers from the Cogeneration area. For this test, we gave to each user a PDA device making a total of four. Two of the PDAs were handled by two operators. A Cogeneration supervisor used one of the PDAs, and the final one was handled to a process control engineer of the area. We ran our experiment in two shifts for three days per cases with the same configuration but with different users for shift. We identified the PDAs in the system as Pocket A, Pocket B, Pocket C and Pocket D. As mentioned before, we explained to the user how to use the PDA application including how to process an alarm notification and the acknowledgement indication. Meanwhile, we configured the alarm simulator to trigger every twenty-five minutes the alarms with the exact values and alarm information from the real events. To make our experiment more accurate and independent, we never told any of the users that those alarms were related to alarms events occurred in the past. We show the tabulated results in Table 5.4. As we can see, we collected the average acknowledge time of each of the alarm cases by user. In addition, we tabulated the average time for each of the cases. Clearly, from the results the users were far more effective using our system to acknowledge alarms messages. This appreciation can be seen more clearly in Figure 5.7.

Table 5.4: Alarm Cases Average Results

Shift I and II	**Average Acknowledge Time Per Alarm Cases (minutes)**		
User	**Alarm Case 1**	**Alarm Case 2**	**Alarm Case 3**
Pocket A	3.286	4.857	3.467
Pocket B	4.885	3.038	2.031
Pocket C	4.696	1.391	2.451
Pocket D	4.208	2.250	3.780
Average Time:	4.269	2.884	2.932

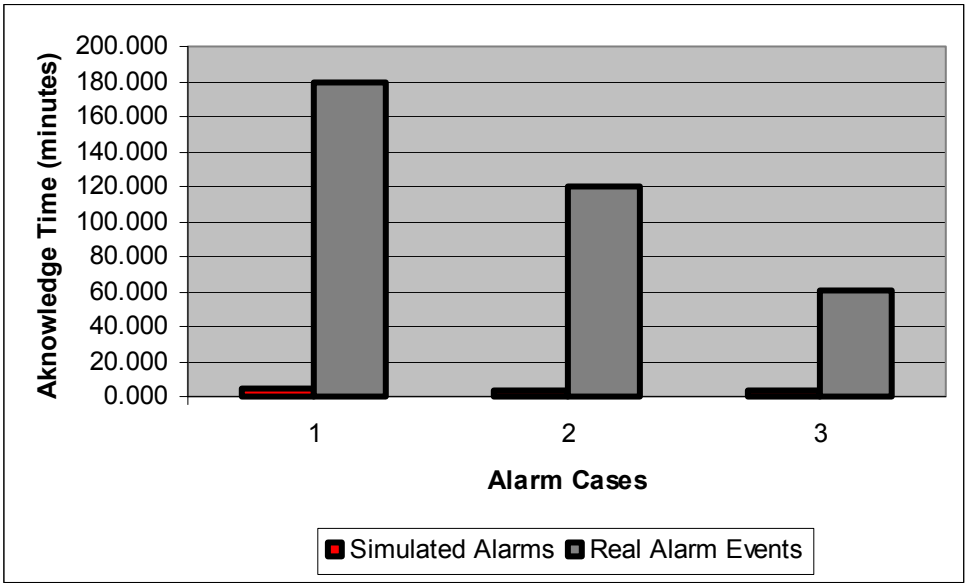

Figure 5.7: Alarm Cases Simulation versus Real Alarm Events

As we can see from the figure, the average results for the alarm events simulation are almost 97% better in comparison to the real alarm event for each of the cases. These results confirmed that our system could reduce from hours to minutes an acknowledgement indication of a real alarm. Interested users should be capable of easily handling alarms caused by equipment and sensors in an optimized fashion. This helps to reduce downtime of the equipments, environmental penalties or even life threating situation by letting an operator, supervisor or a process control engineer get first hand information of an alarm condition no matter what his/her location is in the pharmaceutical plant. It was also noticed, that even if the alarm events collected for the test took from three hours to one hour for the acknowledgement of an operator, in our simulation, the four users were capable of handling each of the events from three minutes to four minutes.

5.2.4 Status Monitoring Test

The second test measured was how the users perform Status requests from different equipments. For this test, we let the participants interact freely with the system. In the background, we gathered the information for each request made by the users. The duration of the test, covered three days with two shifts. For this test, we explained to the user how to use the system and what equipments were parts of the simulation. In addition, as in the Alarm test, we handle four PDAs for two operators, another to a process control engineer and the final one for the Cogeneration area supervisor. For the equipment status data, we simulated status of the following equipments/areas: Turbines, Compressors, Chillers, Boilers, CapsuleRoom, PotableHouse and the Cooling Towers. We show the average results for each status requested by user in Table 5.5. As we show, the average time per day for each status request took less than a second. This means that each user can request status information from one or a group of equipments in near real-time fashion. If we compare these results to an operator's normal job of gathering status twice by shift or every four hours for remote equipments, our results are more than a 100 % improvement. In addition, with a wireless setup, an operator can gather the status data from any part of the pharmaceutical plant, eliminating unnecessary roundtrips to the field.

Table 5.5: Status Simulation Average Results

Shift I and II	**Average Status Request Time Per Day (minutes)**		
Clients	**Day 1**	**Day 2**	**Day 3**
Pocket A	0.881	0.271	0.554
Pocket B	0.333	0.158	0.278
Pocket C	0.389	0.214	0.435
Pocket D	0.768	0.256	0.240
Average Time:	0.593	0.225	0.377

5.2.5 Satisfaction Evaluation

At the end of both Alarm and Status tests, each user was required to complete a satisfaction survey. This survey asked users about the usability of the PDA client tool, their ability to quickly receive and acknowledge alarm, how effective is the status information from one or more equipments, and the completion time for each task in our system versus the actual system. The users were asked to rate each of the system configurations using a 1 to 7 scale where 1 is not satisfied and 7 is very satisfied (see Appendix C). As we explained at the beginning of this document, the actual system is composed of HMIs and SCADAs dispersed in different locations through the plant. We ran the study on two of the three shifts running at the plant as in the Alarm and Status monitoring tests. We show the results in Figure 5.8.

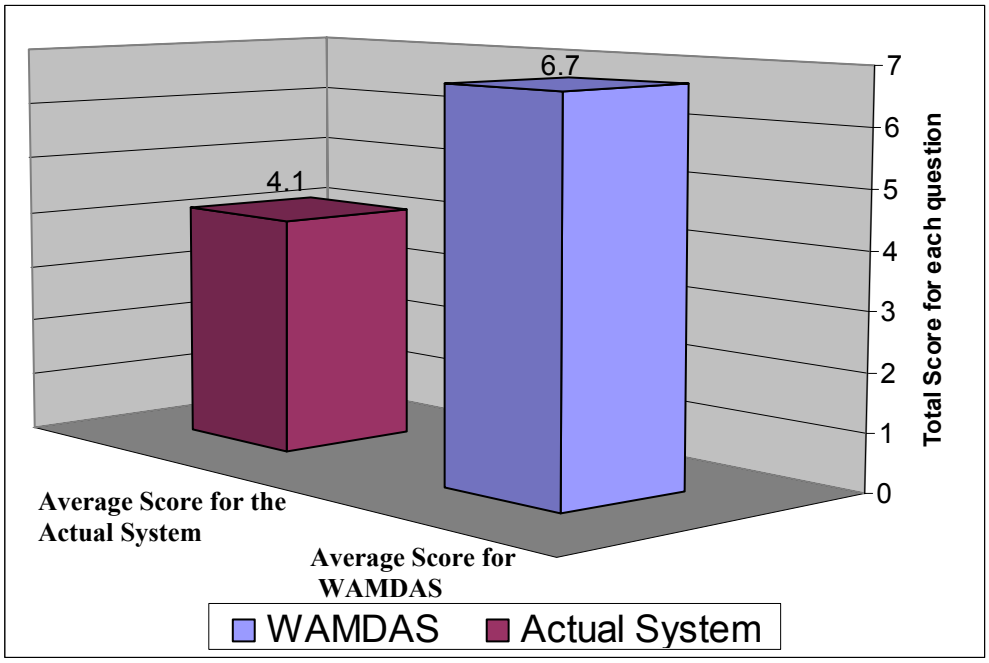

Figure 5.8: Average Preference Ratings for the WAMDAS Application and the Actual System

The above figure shows the average preference ratings between WAMDAS and existing HMIs and SCADAs applications. As we can see, the users preferred WAMDAS by a margin

of almost 2 to 1. This shows the effectiveness of the WAMDAS systems and its PDA client, in presenting a clear and efficient environment for the user to manage alarms and status requests.

The user satisfaction evaluation concludes our experiments of WAMDAS. We have shown in our performance evaluation and usability study of WAMDAS that our system is scalable, efficient, accurate, and provides a user-friendly environment for the employees to monitor alarms at the pharmaceutical plant.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we have presented WAMDAS, a Web Service-based, middleware system for alarm monitoring and status data acquisition of equipments at pharmaceutical plants. WAMDAS is based on a decentralized architecture where all the clients can query and at the same time, receive information from remote data sources. WAMDAS uses XML Web services technology and SOAP protocol for communication through a wireless channel between data source sites. We have presented the WAMDAS system architecture, and the relevant information regarding the implementation of each of the Web services. In addition, we presented our PDA client application showing the benefits of a simple user interface. Likewise, we have presented the Alarm and Status Services protocols created to help operators, supervisors and process control engineers optimize their job by obtaining status information and alarm notifications from remote equipments no matter their location in the pharmaceutical plant. We showed how each of the Web services interacts with the data sources, and the PDA client application. Meanwhile, we have presented the results of our performance evaluation of WAMDAS. Finally, we have presented our evaluation of WAMDAS in a pharmaceutical plant under a real wireless setup using operators, process control engineers and supervisors as real actors.

6.1 Research Conclusions

In our research, we show in our performance evaluation results, the capabilities of WAMDAS in terms of alarm messages processing and status request. Our results show that WAMDAS is capable of managing more than one thousand acknowledgement messages generated by fifteen clients in an average rate of less than a second. Also, our results for the Status test showed that our system is capable of handling, in less than a second, several status requests posted by the user. In addition, in our evaluation at the pharmaceutical plant, we

show that operators and engineers performs better in terms of managing alarms in comparison with existing alarms management tools. We conducted a survey of the PDA application including a comparison between the actual system and WAMDAS. The survey shows that operators, process control engineers, and supervisors were very satisfied with our prototype system.

The results of our experiments show that WAMDAS is a promising system for alarm and status monitoring of equipments in a pharmaceutical plant. WAMDAS should help operators, process control engineers and supervisors to control and organize better their critical information with a robust Web Service-based wireless system. Furthermore, WAMDAS could replace existing HMIs and SCADAs systems reducing operational cost by eliminating different software architectures and expensive implementations.

6.2 Future Work

We will address the following in a further revision of the WAMDAS prototype:

- We must add security to our system. The system has user name and password security for the client application but it is not a robust security configuration. The security shall cover the communication layer by encrypting all exchange of data and messages in the SOAP headers. In addition, it may have a message digest, also known as Digital Signature, in the XML file.
- Implement fault tolerance to our system. If a status request fails to execute, the system must be capable of restart the original query issued by any user. The same is applicable to the acknowledgement indication query. For example, if the PDA battery capacity is running low, it can cause an interruption of the Wi-Fi connectivity with the WLAN. Then the system must be able to indetify this event, and save the state of the original query related to the user on each of the Web services.
- Implement Quality of Services (QoS) for the Alarm Services Protocol. We should classify incoming alarms by the severity of the event in order to help users to react more

quickly. If an alarm is classified as critical, the system may reproduce the alarm notification continuosly until acknowledgements from one or more users are received. The alarm classification should be based on user responsibilities. For example, an operator may be the first person to be notified in case of a critical alarm from equipments. In another level of classification, supervisors and process control engineers may be contacted if no acknowledge were received from operators. Meanwhile, non-critical alarms may be treated like events requiring less attention from operators and engineers at the pharmaceutical plant. In this case, the system may not have to reproduce the non-critical alarm notification so continuously.

- Add doble verification for the acknowledgement indication query. For example, we may implement a scheme that requires a commitment-base communication veryfing first the availability of the end user. The verification could take place directly with the PDA client. For example, a pre-acknowledge query could be used to establish that the PDA client was connected and actually received the alarm notification from the system. Each time that an alarm is generated, the system will scan and process pre-acknowledge notifications from online PDAs, and then it will wait and process incoming acknowledgement indication from users. With the pre-acknowledgement verification, the system will enforce the acknowledgement indications from operators, engineers and supervisors at the pharmaceutical plant.
- Perform a series of trainings to promote the interaction of operators, engineers and supervisors with the PDA application. With more training, the end users may be more productive in terms of recognizing alarm notifications and the acknowledgement indication for each one. In this way, they can develop the habits necessary to keep monitoring the alarms received by the PDA.
- Perform further test with more DBRs, RSs, ANSs, ADASs and Federations to demonstrate that our system is capable of handling extreme workloads of alarm notifications, alarms indications, and status data queries.

BIBLIOGRAPHY

[1] Ahmed, R. and al., e. The Pegasus Heterogeneous Multidatabase System. in *IEEE Computer*, December 1991, 19-27.Ding, W., and Marchionini, G. *A Study on Video Browsing Strategies.* Technical Report UMIACS-TR-97-40, University of Maryland, College Park, MD, 1997.

[2] Bernstein, P.A and Chiu, D.W. Using Semi-Joins to solve Relational Queries. Journal of the ACM, 28(1).Lamport, L. *LaTeX User's Guide and Document Reference Manual.* Addison-Wesley, Reading, MA, 1986.

[3] Dessent, B. May 10, 2003. BitTorrent. Brian's BitTorrent FAQ and Guide. URL: http://dessent.net/btfaq. May 17, 2005.

[4] Carey, M.J., Franklin, M.J., Livny, M. and Shekita, E.J. Data Caching Tradeoffs in Client-Server DBMS Architectures. in Proc. ACS SIGMOD Conference, 1991, 357-366.

[5] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom, J. The TSIMMIS Project: Integration of Heterogeneous Information Sources. in *Proc. of IPSJ Conference*, Tokyo, Japan, 1994.

[6] Cluet, S., Delobel, C., Simeon, J. and Smaga, K. Your Mediators Need Data Conversion! in *Proc. ACM SIGMOD Conference*, Seattle, WA, USA, 1998, 177-188.

[7] Delis, A. and Roussopoulos, N. Performance and Scalability of Client-Server Database Architectures. in *Proc. 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992, 610-623.

[8] Franklin, M.J., Johnson, B.o.T.o. and Kossmann, D. Performance Tradeoffs for Client-Server Query Processing. in *Proc. ACM SIGMOD Conference*, Montreal, Quebec, Canada, 1996, 149-160.

[9] Mackert, L.F. and Lohman, G.M. R* Optimizer Validation and Performance Evaluation for Distributed Queries. in *Proc. 12th VLDB Conference*, Kyoto, 1986.

[10] Mohan, C., Lindsay, B.G. and Obermarck: R. Transaction Management in the R* Distributed Database Management System. *TODS*, *11* (4). 378-396.

[11] Rodriguez-Martinez, M. Automatic Deployment of Application-Specific Functionality in Database Middleware Systems *Department of Computer Science*, University of Maryland, College Park, 2001, 240.

[12] Rodriguez-Martinez, M. and Roussopoulos, N. Automatic Deployment of Application-Specific Metadata and Code in MOCHA. in *Proc. 7th EDBT Conference*, Konstanz, Germany, 2000.

[13] Rodriguez-Martinez, M. and Roussopoulos, N. MOCHA: A Self-Extensible Middleware System for Distributed Data Sources. in *Proc. ACM SIGMOD Conference*, Dallas, Texas, USA, May 2000, 213-224.

[14] Rodríguez-Martínez, M, Enseñat JF, Pagan, E, Arzola, JG, Sotomayor, M, Vargas, E. Registration and Discovery of Services in the NetTraveler Integration System for Mobile Devices in Proc. of International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2

[15] Stonebraker, M, Kemnitz, Aoki, Paul M., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A. Mariposa: AWide-area Distributed Database System. VLDB Journal, 1996.

[16] MIT, ERCIM, Keio. May 08, 2000. World Wide Web Consortium (WC3): SOAP Specification. URL: http://www.w3.org/TR/SOAP. April 10, 2005.

[17] MIT, ERCIM, Keio. March 15, 2001. World Wide Web Consortium (WC3): Web Services Description Languages WSDL URL: http://www.w3.org/TR/WSDL. April 10, 2005

APPENDIX A: XML FILE FOR THE ALARM SERVICES PROTOCOL

GetAknowledgementInfo Web Method

```
<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://localhost/DBRService.asmx/">
- <xs:schema id="NewDataSet" xmlns=""
      xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
      microsoft-com:xml-msdata">
- <xs:element name="NewDataSet" msdata:IsDataSet="true">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="Table">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="BoilerID" type="xs:string" minOccurs="0" />
  <xs:element name="Name" type="xs:string" minOccurs="0" />
  <xs:element name="Status" type="xs:string" minOccurs="0" />
  <xs:element name="WaterLevel" type="xs:float" minOccurs="0" />
  <xs:element name="HeaderPress" type="xs:string" minOccurs="0" />
  <xs:element name="StatusDate" type="xs:string" minOccurs="0" />
  <xs:element name="StatusTime" type="xs:string" minOccurs="0" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:choice>
  </xs:complexType>
  </xs:element>
  </xs:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
      xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <NewDataSet xmlns="">
- <Table diffgr:id="Table1" msdata:rowOrder="0">
  <BoilerID>Boiler_1</BoilerID>
  <Name>Johnston Boiler</Name>
  <Status>ON</Status>
  <WaterLevel>20</WaterLevel>
  <HeaderPress>34.2</HeaderPress>
  <StatusDate>4/25/2005</StatusDate>
  <StatusTime>10:05 PM</StatusTime>
  </Table>
- <Table diffgr:id="Table2" msdata:rowOrder="1">
  <BoilerID>Boiler_1</BoilerID>
  <Name>Johnston Boiler #1</Name>
  <Status>ON</Status>
  <WaterLevel>50</WaterLevel>
  <HeaderPress>35.6</HeaderPress>
  <StatusDate>5/1/2005</StatusDate>
  <StatusTime>7:40 PM</StatusTime>
  </Table>
  </NewDataSet>
  </diffgr:diffgram>
  </DataSet>
```

APPENDIX B: XML FILE FOR THE STATUS SERVICES PROTOCOL

GetClientRequest Web Method

```
<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://localhost/DBRService.asmx/">
- <xs:schema id="NewDataSet" xmlns=""
      xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
      microsoft-com:xml-msdata">
- <xs:element name="NewDataSet" msdata:IsDataSet="true">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="Table">
- <xs:complexType>
- <xs:sequence>
<xs:element name="BoilerID" type="xs:string" minOccurs="0" />
<xs:element name="Name" type="xs:string" minOccurs="0" />
<xs:element name="Status" type="xs:string" minOccurs="0" />
<xs:element name="WaterLevel" type="xs:float" minOccurs="0" />
<xs:element name="HeaderPress" type="xs:string" minOccurs="0" />
<xs:element name="StatusDate" type="xs:string" minOccurs="0" />
<xs:element name="StatusTime" type="xs:string" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
      xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <NewDataSet xmlns="">
<Table diffgr:id="Table1" msdata:rowOrder="0">
<BoilerID>Boiler_1</BoilerID>
Name>Johnston Boiler</Name>
<   Status>ON</Status>
<   WaterLevel>20</WaterLevel>
<HeaderPress>34.2</HeaderPress>
<StatusDate>4/25/2005</StatusDate>
<   StatusTime>10:05 PM</StatusTime>
</Table>
- <Table diffgr:id="Table2" msdata:rowOrder="1">
<    BoilerID>Boiler_1</BoilerID>
<Name>Johnston Boiler #1</Name>
<    Status>ON</Status>
<WaterLevel>50</WaterLevel>
<HeaderPress>35.6</HeaderPress>
<StatusDate>5/1/2005</StatusDate>
<StatusTime>7:40 PM</StatusTime>
</Table>
</NewDataSet>
</diffgr:diffgram>
</DataSet>
```

APPENDIX C: USER SATISFACTION SURVEY (WAMDAS CLIENT APPLICATION VS ACTUAL SYSTEM)

Evaluación de la aplicación cliente de WAMDAS

Por favor conteste las siguientes preguntas:

1) Posición que ocupa:_____
2) Tiempo que lleva realizando su presente posición: _____años.
3) Tiene experiencia utilizando computadoras: __ Si ___ No
4) Si contestó si a la pregunta anterior indique cuantos años de experiencia utilizando computadoras tiene: ____ años.
5) ¿Tiene experiencia o ha interactuado con computadoras portátiles inalámbricas (PDAs)? ___Si ___No
6) Si contestó si a la pregunta anterior indique que tipo de equipo o sistema ha utilizado: _____Pocket PC _____PALM Otro (especifique) _____
7) Necesita lentes para leer? ___ Si ___No

Seleccione con un círculo el número que represente su nivel de satisfacción con los siguientes aspectos (1 significa no satisfactorio, 7 significa muy satisfactorio):

Concepto generales de la aplicación	
Tamaños de letras	1 2 3 4 5 6 7
Organización de la información en las pantallas	1 2 3 4 5 6 7
Claridad en los botones de comando provistos en las pantallas	1 2 3 4 5 6 7
Manejo de alarmas	
Efectividad del aviso al recibir una alarma	1 2 3 4 5 6 7
Calidad de la información sobre la alarma	1 2 3 4 5 6 7
Manejo de múltiples alarmas	1 2 3 4 5 6 7
Efectividad en el "Aknowledgement" de una alarma	1 2 3 4 5 6 7
Efectividad en el "Aknowledgement" de múltiples alarmas	1 2 3 4 5 6 7
Manejo de estatus	
Efectividad en recibir estatus de un equipo	1 2 3 4 5 6 7
Efectividad en recibir estatus de varios equipos	1 2 3 4 5 6 7
Calidad de la información recibida sobre el estatus del(los) equipo(s)	1 2 3 4 5 6 7
Satisfacción general	
Con la aplicación WAMDAS	1 2 3 4 5 6 7
Con el sistema existente	1 2 3 4 5 6 7

Si tiene algún comentario u observación sobre la aplicación, por favor expréselo en el siguiente espacio:

_____
_____
_____

APPENDIX C (CONTINUED)

Evaluación del manejo de Alarmas y Estatus de Equipos

Por favor conteste las siguientes preguntas:

1) Posición que ocupa:_____
2) Describa brevemente la responsabilidad de su posición de trabajo:
   _____
   _____
3) Años de servicio: _____

Seleccione con un círculo el número que represente su nivel de satisfacción (1 significa no satisfactorio, 7 significa muy satisfactorio)

Manejo de alarmas	
Efectividad al detectar una alarma en los distintos sistemas de monitoreo y control distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Calidad de la información sobre la alarma.	**1 2 3 4 5 6 7**
Efectividad en detectar alarmas de múltiples equipos en los distintos sistemas de monitoreo y control distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Efectividad en el "Aknowledgement" de una alarma en los distintos sistemas distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Efectividad en el "Aknowledgement" de múltiples alarmas en los distintos sistemas de monitoreo y control distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Manejo de estatus	
Efectividad en monitorear estatus de un equipo en los distintos sistemas de monitoreo y control distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Efectividad en monitorear estatus de varios equipos en los distintos sistemas de monitoreo y control distribuidos en áreas remotas.	**1 2 3 4 5 6 7**
Calidad de la información recibida sobre el estatus del(los) equipo(s).	**1 2 3 4 5 6 7**

Si tiene algún comentario u observación sobre el sistema existente, por favor expréselo en el siguiente espacio:

_____
_____
_____