

Real-Time System Controlling based on CORBA Architecture and DAIS Standard

El Kamel Ali

Abstract—The Common Object Request Broker Architecture (CORBA) is intended to simplify the task of developing distributed applications. Although it is well-suited for conventional RPC style applications, several limitations become evident when CORBA is used for a broader range of performance-sensitive applications running in heterogeneous environments over high-speed networks.

This paper focuses on industrial system controlling using efficient architectures and standards such as CORBA (Common object request broker) and DAIS (Data acquisition from industrial system). The main objective is to implement an over-IP distributed application that is industrial device free. In fact, industrial device independence is ensured by the standard DAIS. CORBA principles are used to make a sophisticated solution supporting industrial device controlling and monitoring.

Keywords—CORBA, DAIS, ORB, C/S architecture.

I. INTRODUCTION

TODAY, heterogeneity is considered as the most future of actual wide networks such as the World Wide Web and extranets. In fact, several wide networks may be constituted from mainframes, servers, workstations and PCs which can use various operating systems (Microsoft Windows, IBM OS/2 or Apple MacOS) and can be equipped with miscellaneous devices such as Routing elements and robotic components.

Networks and associated protocols, used generally for interconnecting those components in order to ensure bi-communication of data, are also varied: Ethernet, FDDI, ATM, TCP/IP, X25, and more, can use multiple techniques such as RPC (Remote procedure call) technologies. The massive development of such networks has been enforced by the need of making useful information available and optimizing resource sharing.

Heterogeneity is generally resulting from many factors: technical agreements (rarely, complex problem can have unique solution), cost constraints and patrimonial applications riding (patrimonial applications are necessary applications for working of which the replacement is generally expensive or even useless).

Heterogeneity aims at using a combination of soft and hard technologies to ensure interoperability and portability of application. Such purpose can simplify new components integration with preservation of a coherent and operational system.

However, heterogeneity problems resolution is generally

laborious. Many solutions have been proposed in order to develop applications that can be defined as a unique homogeneous framework. Other solutions have been proposed to simplify components integration within existent frameworks without coherence and availability degradation. As an example, the foundation of Object Management Group (OMG) in 1989 by many concerned enterprises have the objective of defining standards of integration of distributed and heterogeneous applications based on three keywords: components reuse, interoperability and portability. CORBA (Common object request broker) is an architecture proposed and normalized by the OMG that ensures the non-homogeneous integration of applications without worrying about incoherence problems.

The Common Object Request Broker Architecture (CORBA) [5]) is a promising approach for improving the flexibility, reliability, and portability of communication software. CORBA is designed as an open standard for distributed object computing. It automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching.

This paper is structured as follows. The next section presents the architecture CORBA. Section III describes DAIS standard and gives an overview of included components. Section IV depicts problems within existing solutions and frameworks. The section V describes the proposed framework and denotes functionalities implementation. The last section concludes on actual work and presents an overview future work.

II. PROBLEM ANALYSIS

Many solutions have been proposed to monitor industrial systems in a distributed way. However, most of them have been made for appropriate use and are dedicated to a given kind of industrial systems and, further, most of them require having device drivers pre-installed and pre-configured.

Furthermore, data acquisition provides several values attached with mechanical or electrical criterion. Those values are generally presented in a specific way and require specific modifications and treatments to make them useful. Such problem has leading to develop various techniques and applications in order to satisfy heterogeneous devices which makes divergent solutions.

To deal with this problem, it was necessary to have a device-free application that can support many kinds of industrial systems and that can ensure acquisition of data values to offer them to applications in a useful way. Data acquisition should ensure a standard representation of values to the client interfaces. We opt thus for using DAIS[8] to have a homogeneous system for acquisition of data from various industrial devices without worry about heterogeneity and coherence issues.

Fig. 1 shows the main objective of the proposed framework. The system must be a distributed application based on the IP-infrastructure. We have opted for CORBA architecture due to several features which can be found in [5].

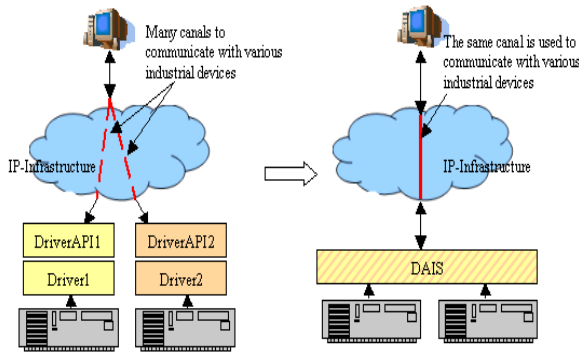


Fig. 1 Objective of the new framework

The proposed solution can be described from two sides. The first side is dedicated for application used to monitor and to control remote devices. The other side is associated with industrial devices. The first side is considered as the client side, the second one is described as the server one. The whole architecture is then implemented using the C/S (Client / Server) architecture. Such solution was implemented using CORBA technology.

Fig. 2 shows clearly several components associated with the proposed solution. First, the DAIS Server is dedicated to ensure data acquisition from various devices based on the standard DAIS. Furthermore, it should targeted data toward client application in a comprehensive way.

The infrastructure CORBA ensures data transmission from one side to other. Furthermore, the DAIS-Server ensures remote communication between industrial devices and client application. DAIS-Server provides specific interfaces to communicate with industrial devices and to make acquired data comprehensive for the client application. The common representation form of data is given by the standard DAIS.

The representation of data is accomplished by the server which should represent the internal data within the device in a hierarchical structure. The Tree structure is constructed by several nodes which one is considered the root and others are designed leaves. Every leaf presents measured criteria and has an absolute path indicating a hierarchical way of the component. Thus, a leaf denotes measured criterion to be controlled and monitored from the client application.

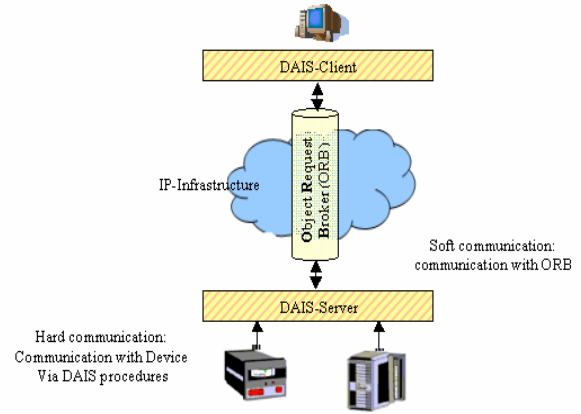


Fig. 2 General structure of the proposed framework

Within our system, leaves are denoted items, the entry point is designed root. Other elements are nominated nodes. The server should provide a tree representation toward the DAIS client which is used by the administrator. The DAIS client can access server database to find solicited information.

III. CORBA OVERVIEW

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them.

CORBA was introduced in 1991 by the Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). Furthermore, CORBA defines true interoperability by specifying how ORBs from different vendors can interoperate.

IV. DAIS OVERVIEW

The purpose of the DAIS is to support efficient real time transfer of large amount of data from an industrial process to a wide range of clients. It supports discovery of parameters and update of parameter values. The DAIS is intended for on-line data transfer and cannot be used to configure servers implementing the API.

Control systems used to monitor and control industrial processes consist of the following major pieces:

- Process instrumentation making sensor data and actuation capabilities available.
- Process stations or remote terminal units (RTUs) reading sensor data and controlling actuators.
- SCADA system making processed sensor data and control capabilities available to operators, applications, or other systems.
- Management systems using SCADA data to make further processing and control.

SCADA and management systems can be regarded having a server part where data processing is performed and an HMI part where visualization and command dialogs are made. The SCADA and management system may have common or different HMI. This results in a hierarchical structure as shown below.

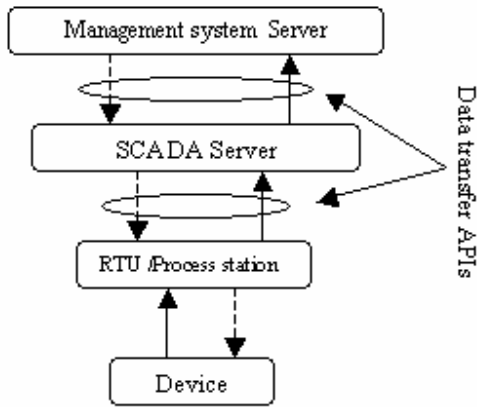


Fig. 3 Control system structure

The solid arrows in Fig. 1 correspond to transfer of data are to be transferred from servers to clients. The data can be state variables and parameters (for example, measured values, calculated values, limit values, dead bands, scan rates, engineering units). The dashed arrows correspond to data written to servers by clients. The data can be control variables and parameters. The API support simple writes operations. More complex controls like select and execute (for example, breaker on/off) or raise lower commands can be implemented combining multiple write and read operations. As indicated in Fig. 1, the DAIS API can be used at several levels in a system. For example, a DAIS server can be an RTU/Process station communication unit, a SCADA server, or even a Management system.

DAIS supports both subscription and read/write operations. A subscription means server has no a priori knowledge of clients and it is clients that establish connection with servers. Once connection is established, a server calls the clients back when data becomes available or updated. The callbacks mean the DAIS API also defines an interface that the client has to implement.

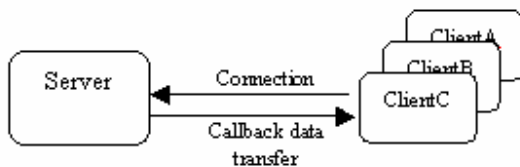


Fig. 4 Communication Client/Server

The DAIS API is intended for transfer of process data on subscription basis. Process data consists of quality tagged and time stamped scalar values. The API is intended to efficiently transfer large amounts of data simultaneously to many clients

(subscribers). Clients and servers involved in data exchange can be of many kinds (for example, HMI or management systems). A client may also appear as a server (for example, aggregating data from other servers or performing calculations). This creates hierarchical structures of DAIS servers.

The DAIS API is intended to be used for a wide range of industrial processes. For example:

- Power transmission
- Power generation
- Power distribution
- Water and sewage management
- Oil and gas
- District heating
- Pulp and paper
- Food manufacturing

The kinds of data that can be reached through the DAIS API are:

- Measurement data access
- Alarms & events access

This data is typically available from hardware units in the process (for example, RTUs, PLCs, distributed controllers) or other control centers (for example, SCADA systems). Refined or calculated data, parameters, and alarms & events might also come from applications (for example, custom calculations, state estimation, and optimization) in SCADA or Management systems. These data might be provided through the DAIS API as well.

V. THE PROPOSED FRAMEWORK

The proposed architecture is based on two sides: client side and server side (Fig. 3).

A. DAIS Server Overview

The DAIS server provides an efficient system which ensures representation of data values in a hierarchical way to make them available to client application. The DAIS server is structured as follows:

a) DAIS-Server API Module

It provides a management interface to server application. It presents, in an abstract way, functionalities provided by the DAIS-server. However, procedures and functions are not implemented within this module. This interface ensures the correspondence between called functionalities and the appropriate objects where those functions are implemented.

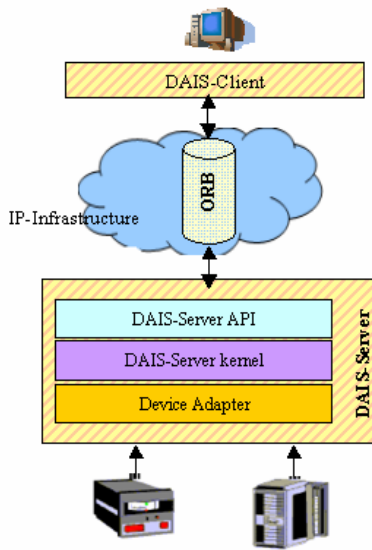


Fig. 5 The proposed framework

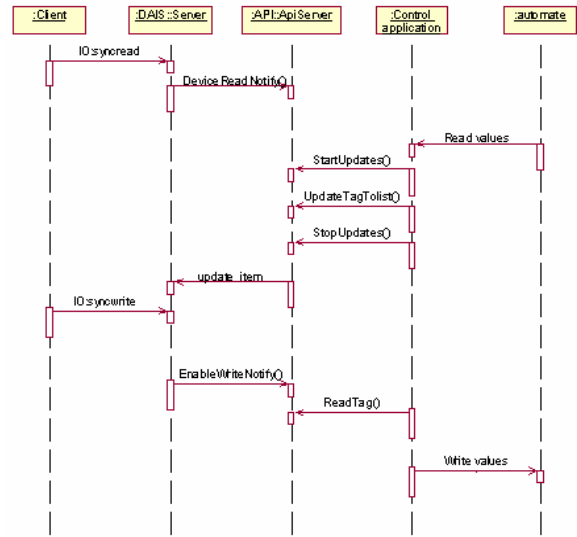


Fig. 6 Read/Write Process

b) DAIS-Server Module

The kernel of the DAIS-Server is an implementation of various functionalities. Most implementations are done using Java. The choice is made in order to have a portable application that is infrastructure-free and which is open source.

c) Device Adapter (Application Control)

The device adapter is designed to resolve the problem of communication with hard devices. It consists on a generic communication protocol which can discover industrial device based on specific message exchange with the peripheral.

1) Writing/Reading Situation

When a client requires information from device, it uses the method **syncread()**. Receiving such call, the server DAIS invokes the callback **DeviceReadNotify()** which is designed to inform the control application that a reading operation should be started.

Hence, the control application starts a process of values collection from the device using the function **ReadValues()**. The result is used to update tags within the interface of DAIS server. It uses functions **StartUpdates()** and **UpdateTagToIist()**.

Similarly, to write values toward the device, the client should use the function **syncwrite()**. The server DAIS transmits the request to the DAIS-Server API which uses the function **EnableWriteNotify()**. The control application uses the function **ReadTag()** to receive the value specified by the client request, then it invokes the method **WriteValues()** on the device. Read/Write operations are described by the sequence diagram (Fig. 6).

B. DAIS Client Overview

1) Access Session Creation

To start monitoring remote device, the client application should open a session on DAIS server. It should use the method **create_alarms_and_events_session()** to subscribe for alarm and event callbacks. As a result, this method returns eight objects home which are included in the object session.

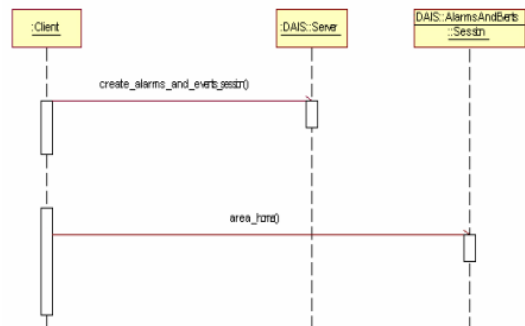


Fig. 7 Client Session creation

The most important object is **area_home**. It is used to identify the area and to have information about root area and sub-area.

2) Area Tree Discovery

To have the root identifier of the **area_home**, the client should call the function **get_root()**. For a given node, the

client should also invoke the function **find_by_parent()** to depict sub area. The result of this method is an iterator which describes identifiers of selected areas. Browsing the iterator is ensured by the function **next_n()**.



Fig. 8 Hierarchical Area Tree discovery

3) *Subscription for Alarms and Events*

The object **subscription_home** found within the object session is responsible for subscription mechanism. After invocation of the method **create_subscription()**, an object **subscription::Manager** is created.

The subscription manager is responsible for managing notifications when changes occur. Those notifications are transmitted due to states changing, error occurring or communication failure.

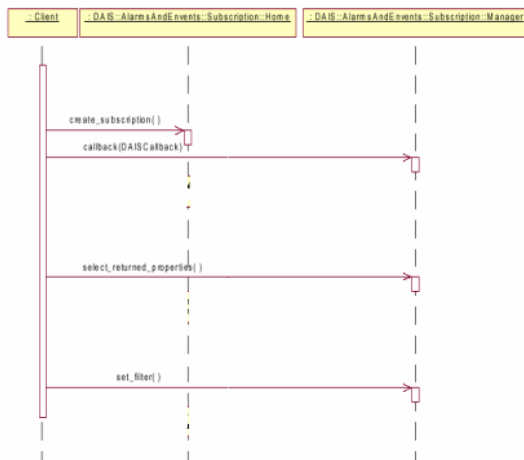


Fig. 9 Subscription steps

The client should transmit, to the subscription manager, the identifier of the object callback to use in notification process and the set of criteria to be attached to notification callbacks. This is done respectively by the function **callback(DAISCallback)** and **select_returned_properties()**.

Furthermore, the client should define alarms to be used by invocation of the method **set_filter()**. Many alarms can be defined accordingly to sources, areas, raisons and severity.

At this stage, the client has finished the subscription process and can be notified from server when any modification is observed.

4) *Initial State Determination*

When the client starts, it needs knowing the initial state of the remote device by evaluating the values of binary operands. In fact, events and alarms notifications transmit information only when changes are observed and can not evaluate initial state. Thus, the client should use the method **refresh()** to activate notifications and to have an evaluation of the initial device state.

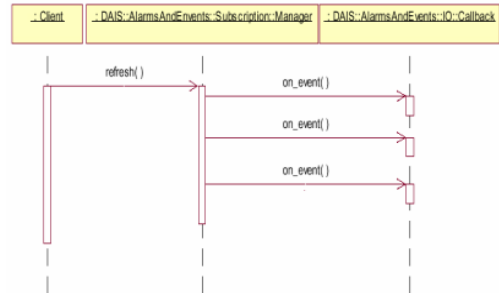


Fig. 10 Initialization of device state

C. *Framework Implementation*

The proposed framework has been implemented using the open source OpenORB which is conform to the standard CORBA 3.0. Interfaces and functions implementation has been done using Java which is able to ensure portability. The selected ORB is interoperable with various systems such as Orbix, Visibroker, Orbacus and JavaORB and can support common version of GIOP (IIOP1.2 and GIOP bi-directional).

Furthermore, the framework has been tested on a distributed environment. We have used a device of type Micro3 PLC to evaluate the efficiency of the approach. Figs. 11 and 12 show an example of the client application interface.

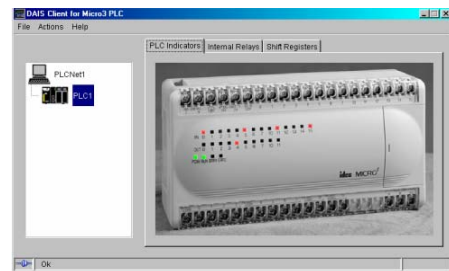


Fig. 11 Main interface

Users can select the appropriate device in the left side. Hierarchical tree of sub components can be viewed and selected. All selected properties can be viewed and monitored. To select an action, user can select the menu Actions from the interfaces.

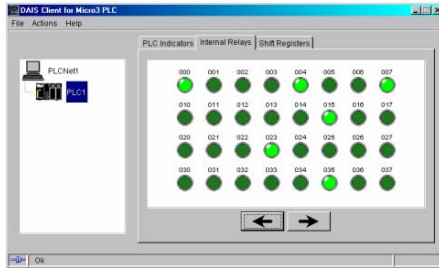


Fig. 12 Monitoring interface

Many other interfaces are used to make the implementation useful, efficient and reliable.

VI. CONCLUSION

This paper illustrates a CORBA framework coupled with the standard DAIS in order to conceive a reliable system used to control and monitor remote mechanical and electrical devices. The proposed framework uses features of CORBA like interoperability and portability to develop a distributed system that can be run over the infrastructure IP. This system provides a solution for remote real-time monitoring without need of specific infrastructure implementation.

The developed prototype of the framework has proved its efficiency. However, it can not take, in consideration, possible failures or the non-availability phenomenon. Such problem is crucial and should be treated in order to make the proposed framework able to support fault tolerance. This can be ensured due to the evolution of CORBA by introduction of the *ToF* modules. The proposed architecture has been called FT-CORBA.

REFERENCES

- [1] Object Management Group; Object Management Architecture Guide; Revision 3.0; OMG Document ab/97-05-05; <http://www.omg.org/cgi-bin/doc?ab/97-05-05.pdf>
- [2] Object Management Group; A Discussion of the Object Management Architecture Guide; OMG Document formal/00-06-41; <ftp://ftp.omg.org/pub/docs/formal/99-10-07.pdf>
- [3] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.
- [4] A. Gokhale and D. C. Schmidt, "Measuring and Optimizing CORBA Latency and Scalability Over High-speed Networks," *Transactions on Computing*, vol. 47, no. 4, 1998.
- [5] Object Management Group; The Common Object Request Broker: Architecture and Specification; Revision 2.6; OMG Document formal/01-12-01; <http://www.omg.org/cgi-bin/doc?formal/01-12-01.pdf>; Décembre 2001.
- [6] J. Ehrlich, A. Zerrouki, and N. Demassieux; Distributed Architecture for Data Acquisition: a Generic Model; in the proceedings "IEEE Instrumentation and Measurement Technology - IMTC'97"; Ottawa, Canada; 19-21 mai 1997.
- [7] Txomin NIEVA; Remote data acquisition of embedded systems using internet technologies: a role-based generic system specification; PHD Thesis presented in Lusanne federal polytechnic; 2000.
- [8] Object Management Group - Manufacturing Domain Task Force; Data Acquisition from Industrial Systems specification; OMG document dtc/01-09-03; <http://www.omg.org/cgi-bin/doc?dtc/01-09-03.pdf>; Septembre 2001.
- [9] Object Management Group - Utility Domain Task Force; Utility Management System (UMS) Data Access Facility; Version 1.0; OMG Document formal/01-06-01; <http://www.omg.org/cgi-bin/doc?formal/01-06-01.pdf>; Juin 2001.

- [10] Steve Vinoski; CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments; *IEEE Communications Magazine*; Vol. 35; No. 2; Février 1997