Open Controller for Distributed Instrumentation Systems

Vítor Viegas[1,2], P. Silva Girão[2], J.M. Dias Pereira[1,2]
[1]ESTSetúbal – LabIM, Instituto Politécnico de Setúbal, 2910-761, Setúbal, Portugal
[2]Instituto de Telecomunicações, Av. Rovisco Pais, 1, 1049-001, Lisboa, Portugal
vviegas@est.ips.pt , joseper@est.ips.pt , p.girao@lx.it.pt

***Abstract* – The paper presents a controller designed to be highly interoperable in the context of distributed instrumentation systems. Interoperability is achieved by adopting strong standards – the IEEE 1451.1 Std to be more precise – and by using cross-platform, manufacturer-independent technologies such as Web Services. The 1451.1 Std contributes with its information model to represent data and organize functionalities through a well-defined hierarchy of objects. Web Services are used to implement both communication models, the client/server model for one-to-one communications, and the publish/subscribe model for one-to-many communications. Being supported by all the major software companies around the world, Web Services have the chance to become the first wide-used middleware solution and the answer for many interoperability problems. The controller was developed using the .NET Framework and tested in the Windows XP operating system.**

***Keywords* – IEEE 1451.1, Web Services, interoperability, distributed instrumentation**

I. INTRODUCTION

This section introduces the key technologies used to implement the proposed controller.

A. IEEE 1451.1 Standard

The IEEE 1451 project [1] aims at simplifying transducer connectivity to communication networks by proposing a set of hardware and software interfaces that act as *plugs* where heterogeneous components can connect and work together. The 1451.1 Std [2-3], in particular, proposes a generic object model to represent the data and the functionalities of any networked transducer. The model is composed by a hierarchy of classes divided in three main categories (figure 1):

- Blocks: Block classes are processing entities that manipulate data. Three types of Blocks are defined: (i) the Network Capable Application Processor Block (NCAP Block or PBlock for short), which represents the application as a whole and provides global resources to all underlying objects; (ii) Transducer Blocks, which provide high-level functions to interact with transducers (such as read sensors, write actuators, read status registers, read/write interrupt masks and read/write Transducer Electronic Data Sheet (TEDS) structures); and (iii) Function Blocks, which implement signal processing algorithms (such as mathematical formulas, digital filters, function generators and discrete Proportional Integral Derivative (PID) controllers).
- Components: Component classes are network-visible data repositories owned by Blocks. Two main types of Components are defined: (i) Parameters, which are used to store volatile data (such as field variables, set-points and computations); and (ii) Files, which are used to store persistent data (such as TEDS structures).
- Services: Service classes handle network communications. Two communication models are defined: (i) a tightly-coupled client/server model for one-to-one communications; and (ii) a loosely-coupled publish/subscribe model for one-to-many communications. Using the client/server model, a client can invoke a function on the server and consume the results returned (if any). Using the publish/subscribe model, the publisher can

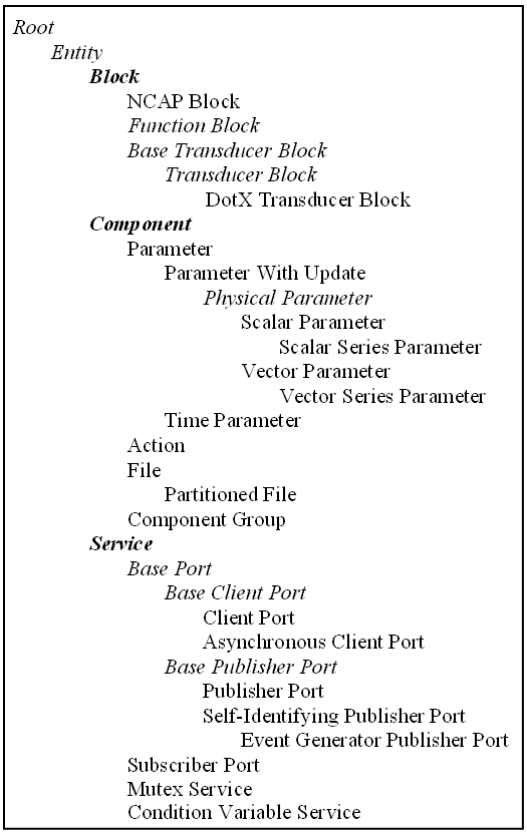

Fig 1. IEEE 1451.1 object model (classes listed in italic are abstract and shall not be instantiated).

broadcast messages on the network whenever it has some kind of public announcement to make. These messages, known as publications, contain metadata describing the syntax and the semantics of the information they carry. Based on this metadata, any subscriber can filter the publications of its interest and consume the attached data.

These classes, once deployed as a reusable software library, can be used to construct complex control applications. The model is extensible by adding non-1451.1 classes to satisfy particular requirements of a given application.

B. Web Services

Web Services [4-5] are, in simple terms, object methods exposed via eXtended Markup Language (XML) messages. These messages use a format protocol known as Simple Object Access Protocol (SOAP) [6] and travel across the network using a transport protocol like the Hyper Text Transport Protocol (HTTP). Every Web Service is described by its manifest, an XML document written according to the Web Service Definition Language (WSDL) [7]. The manifest describes the interface of the service, including the signatures of available methods, the data types used for input/output arguments and a list of supported communication paths. This information is all the client needs to consume the service without worrying about its underlying implementation. Interoperability is achieved by imposing strong standards (SOAP and WSDL) and by using ubiquitous technologies (XML and HTTP). Being supported by the World Wide Web Consortium (W3C) [8], which includes all the major software companies around the world, Web Services have the chance to become the first wide-used middleware solution and the answer for many interoperability problems.

C. .NET Framework

The .NET Framework [9-10] is a software infrastructure released by Microsoft to develop and execute Windows applications. The .NET Framework includes three main components: (i) a virtual machine that executes managed code in a secure and protected environment (conceptually similar to the Java virtual machine); (ii) an extensive class library that provides ready to use components to develop applications; and (iii) last-generation programming languages (Visual Basic .NET and C#) designed to increase productivity.

By the end of 2006, version 3.0 of the .NET Framework was released including a new software package called Windows Communication Foundation (WCF) [11]. The WCF contains pre-built classes designed to develop secure, reliable, and interoperable distributed applications. The WCF supports the last enhancements in terms of Web Services providing many useful facilities such as hosting, service instance management, asynchronous calls, reliability and security. The WCF provides developers with the essential off-the-shelf plumbing required by any service-based application, and as such, it greatly increases productivity.

2. CONTROLLER

This section covers the software architecture of the controller, including a detailed description about its transducer and network interfaces.

A. Architecture

The controller is an object-oriented application that creates and manipulates Blocks and Components in order to implement a desired behaviour. The control strategy is executed periodically by means of a high-priority software timer. On every timer-tick, data is acquired from sensors, control algorithms are executed and decision values are written to actuators, by this order as depicted in figure 2.

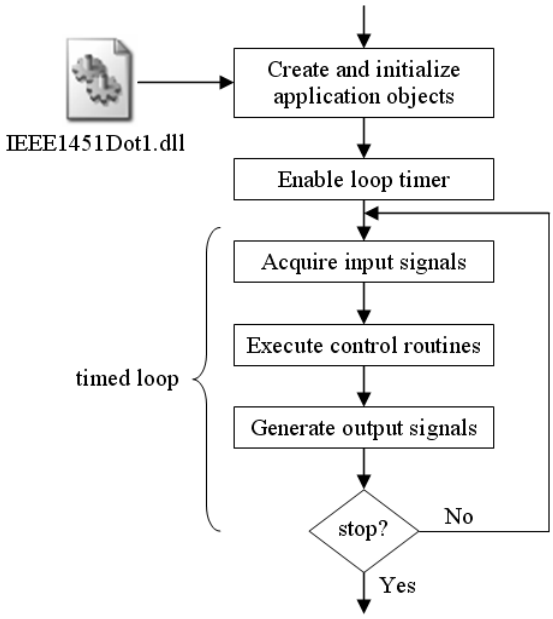

Fig 2. Flowchart of the controller application.

All 1451.1-objects used by the controller application were assembled as a Dynamic Link Library (DLL) under the name "IEEE1451Dot1.dll". The library was developed using Visual Basic .NET 2008 and can be reused by any Windows application. It implements a fully-functional subset of the 1451.1 object model composed by 16 classes (figure 3). Each class exposes a collection of methods according to the recommendations of the 1451.1 Std.

Blocks are processing entities that can create and own one or more Components to assist them. During the creation stage, the Component is informed about its owning Block by receiving a reference to it. This cross-reference mechanism was the way found to implement

owning relations. Ultimately, all Blocks and Components are owned by the top-level PBlock. For example, the HysteresisBlock, which implements a schmitt-trigger algorithm, owns three Components: two floating-point ScalarParameters, named "highThreshold" and "lowThreshold", that define the high/low thresholds of the hysteresis window, respectively; and a boolean ScalarParameter, named "output", that holds the decision of the comparator.

Blocks also have an internal state machine. They can switch from the INACTIVE state to the ACTIVE state and vice-versa. In the INACTIVE state, the Block does not process and becomes configurable by accepting *set* operations; in the ACTIVE state, the Block becomes fully-functional but only accepts *get* operations. These restrictions also apply to all Components owned by the Block. No Block can switch to the ACTIVE state if the top-level PBlock is INACTIVE.

Parameters are specialized Components that behave like network-visible data buffers with two levels of access: simple read/write operations only affect the internal buffer of the Parameter; but when the Parameter executes an *update* operation, it scratches its owning Block and accesses data inside it. For example, when a sensor Parameter is updated, it forces the DAQmxTBlock (its owning Block) to acquire a fresh sample from the field and return it back.

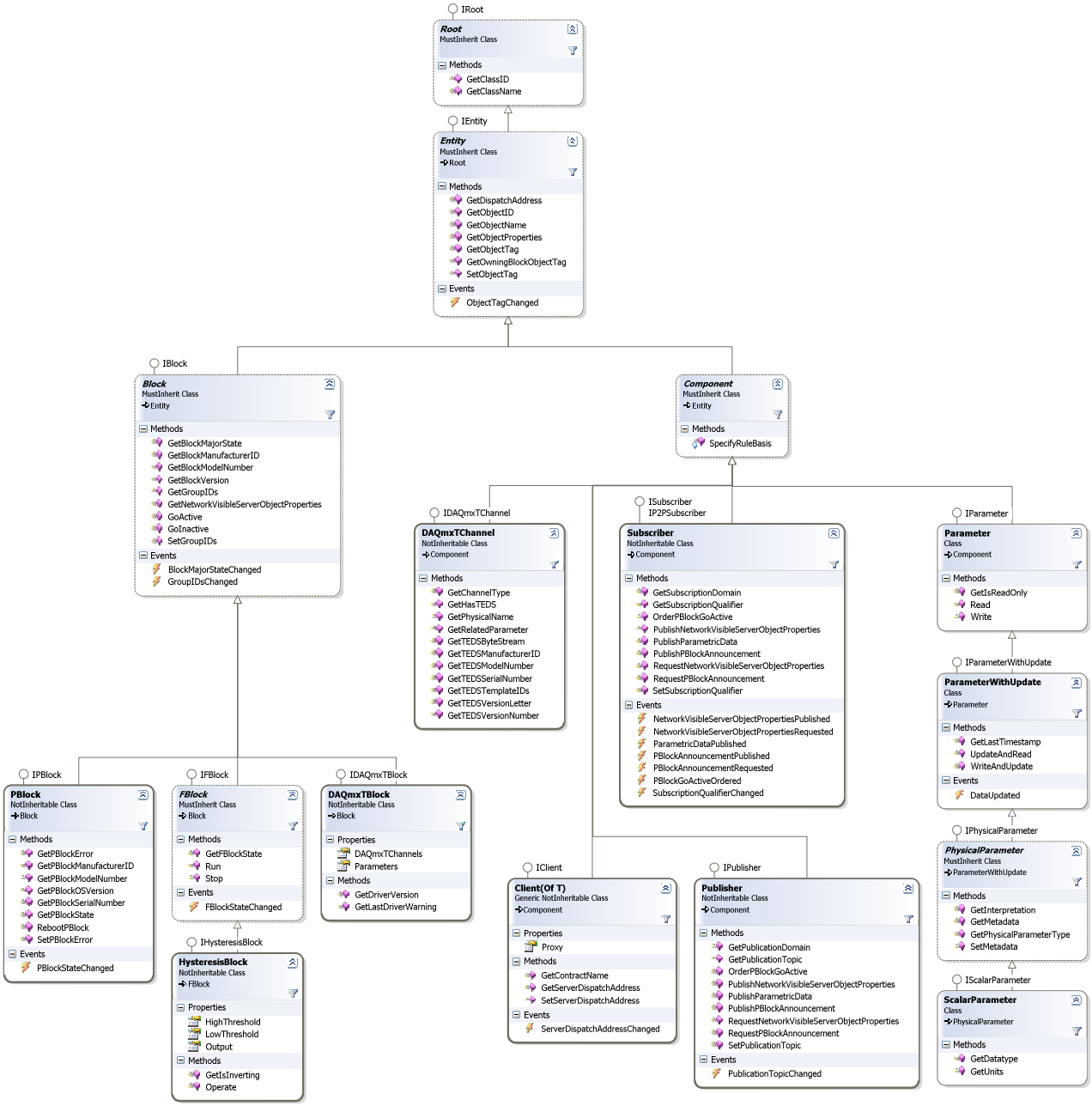

Fig 3. Diagram of implemented classes.

B. Transducer Interface

The controller works with Data AcQuisition (DAQ) boards compliant with the DAQmx driver from National Instruments (NI). The interface is assured by the DAQmxTBlock class that wraps the NI-supplied library DAQmx.dll, which contains the low-level functions that *indeed* communicate with the board.

The life-cycle of the DAQmxTBlock can be described as follows:

- At design-time, using the Measurement and Automation eXplorer (MAX), a software tool provided by NI, the developer configures and saves *tasks* involving one or more *channels*. A *task* is a data structure that depicts the process of acquiring/generating signals, including properties that describe the digitizer (such as hardware channels, sampling frequency, number of samples and trigger settings) and properties that describe the transducer itself (such as transducer type, range and units). A *channel* is a hardware input/output that participates on a task. Each channel can be automatically configured by reading the embedded TEDS of the attached transducer according to the directives of the 1451.4 Std [12]. If the transducer has no embedded TEDS, the developer can associate it a virtual TEDS in the form of a .ted file. Using MAX facilities, the developer can also add a custom scale to the task, which acts as a correction engine [13] by transforming raw field data into engineering units, and vice-versa. Detailed information about the configuration process can be found in [14-15].
- At run-time, the DAQmxTBlock is created, initialized and executed. During initialization, the DAQmxTBlock loads all pre-configured tasks and creates auxiliary objects to support them, more precisely one Parameter for each task and one DAQmxTChannel for each channel. The Parameter provides methods to read/write transducers, as well as a metadata structure that describes the meaning of the data exchanged. The DAQmxTChannel provides methods to get information about the channel where the transducer is attached, including methods to retrieve the underlying TEDS.

At the moment, the DAQmxTBlock only supports scalar tasks involving one analog or digital channel.

C. Network Interface

The network interface of the controller is completely based on Web Services. All non-abstract classes presented in figure 3 are implemented as singleton WCF Web Services following the best practices described in the literature. Whenever a service is created, it registers itself on a HTTP endpoint and exposes its methods on the network. If a client wants to invoke a method, it gets the dispatch address of the service, creates a proxy at run-time, executes the remote call and waits for results (if any). These are the typical steps of the client/server communication model.

The publish/subscribe communication model is mainly implement by the Subscriber class, which registers itself on a Peer-to-Peer (P2P) endpoint and listens to incoming publications on a multicast address. The P2P binding guarantees a reliable connection for one-to-many communications. If a publisher wants to issue a publication, it gets the multicast address, creates a proxy at run-time and executes the remote call with no return values. The Subscriber filters the publications of its interest and signals them to the controller application.

All remote calls execute on a dedicated thread without blocking the main loop of the controller.

3. Application Scenario

This section explains how the controller can be customized for a given scenario and presents a client application that performs configuration and monitoring tasks.

A. Controller Application

To demonstrate the effectiveness of the proposed solution, a simple ON/OFF controller with hysteresis (also known as schmitt-trigger) was built. The controller can be used to maintain the water level inside an open tank between two threshold values (figure 4). The water level is measured by a level transmitter and passed to the controller that turns the supply pump ON or OFF.

The controller application was developed using Visual Basic .NET 2008 and tested in the Windows XP operating system. The controller, shown in figure 5, loops every 100 ms and makes use of 13 objects as described in table 1. These objects and their owning relations are presented in the form of a tree list. Two

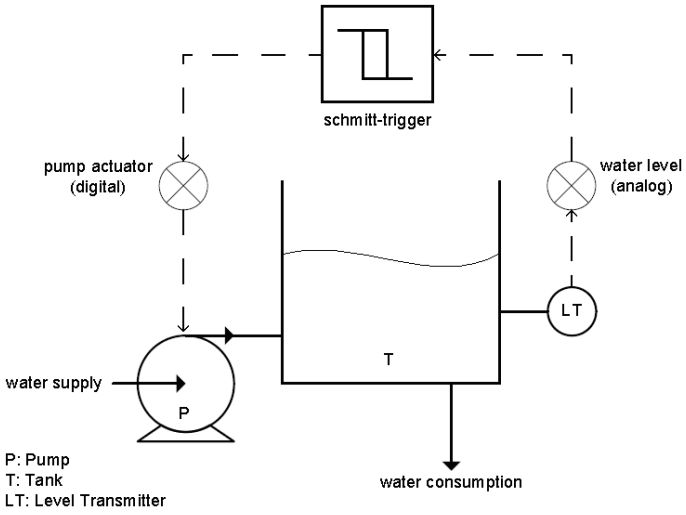

Fig 4. Water-supply pump system.

columns are provided to show the state of all Blocks and the value of all Parameters. An overview of the selected object is given in the text box at the bottom.

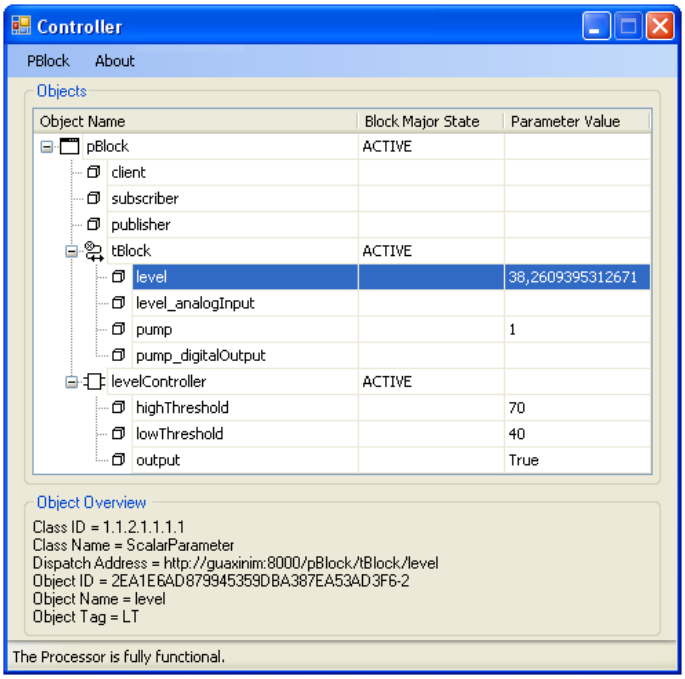

Fig 5. Controller application.

B. Client Application

A client was built to interact with the controller. Both applications – the controller and client – shall be connected to an Ethernet Local Area Network (LAN).

The client serves two main purposes:

- Configuration: The user can scan the network to find all registered services by issuing the publication RequestNetworkVisibleServerObjectProperties (using a Publisher object) and by intercepting all the replies in the form of PublishNetworkVisibleServerObjectProperties publications (using a Subscriber object). This way, the user gets a picture of all network-visible objects including their names, object IDs and dispatch addresses. With this information, the user can open a configuration console and invoke any method on any object using the client/server communication model.
- Monitoring: Parameter monitoring is done by listening to PublishParametricData publications (using a Subscriber object). The attached data, which includes the value and the identification of the publisher Parameter, is extracted, presented to the user and logged to a file. This way, the user gets a real-time picture of all system variables by taking full advantage of the publish/subscribe communication model.

Table 1. Controller objects.

Object Name	Class Name	Description
pBlock	PBlock	Is the top-level Block that represents the application as a whole and provides global resources to all underlying objects.
client	Client	Used to make remote calls to a remote server (which does not happen in the present case). Only created for testing purposes.
subscriber	Subscriber	Listens to all incoming publications on the multicast address net.p2p://myMesh/publications.
publisher	Publisher	Used to announce the presence of the controller on the network (by issuing the publication PublishPBlockAnnouncement) and to broadcast data from Parameters (by issuing the publication PublishParametricData).
tBlock	DAQmxTBlock	Acquires the signal from the level transmitter connected to an analog input of the DAQ board. The signal is converted to relative units (% of tank capacity) by the built-in correction engine. It also updates the digital output where the pump actuator is connected.
level	ScalarParameter	Holds a floating-point network-visible variable representing the water level.
level_analogInput	DAQmxTChannel	Represents the DAQ channel where the level transmitter is connected.
pump	ScalarParameter	Holds a boolean network-visible variable representing the state of the pump.
pump_digitalOutput	DAQmxTChannel	Represents the DAQ channel where the pump actuator is connected.
levelController	HysteresisBlock	Implements the control algorithm that decides the next state of the pump.
highThreshold	ScalarParameter	Holds a floating-point network-visible variable representing the high threshold of the control algorithm.
lowThreshold	ScalarParameter	Holds a floating-point network-visible variable representing the low threshold of the control algorithm.
output	ScalarParameter	Holds a boolean network-visible variable representing the result of the control algorithm.

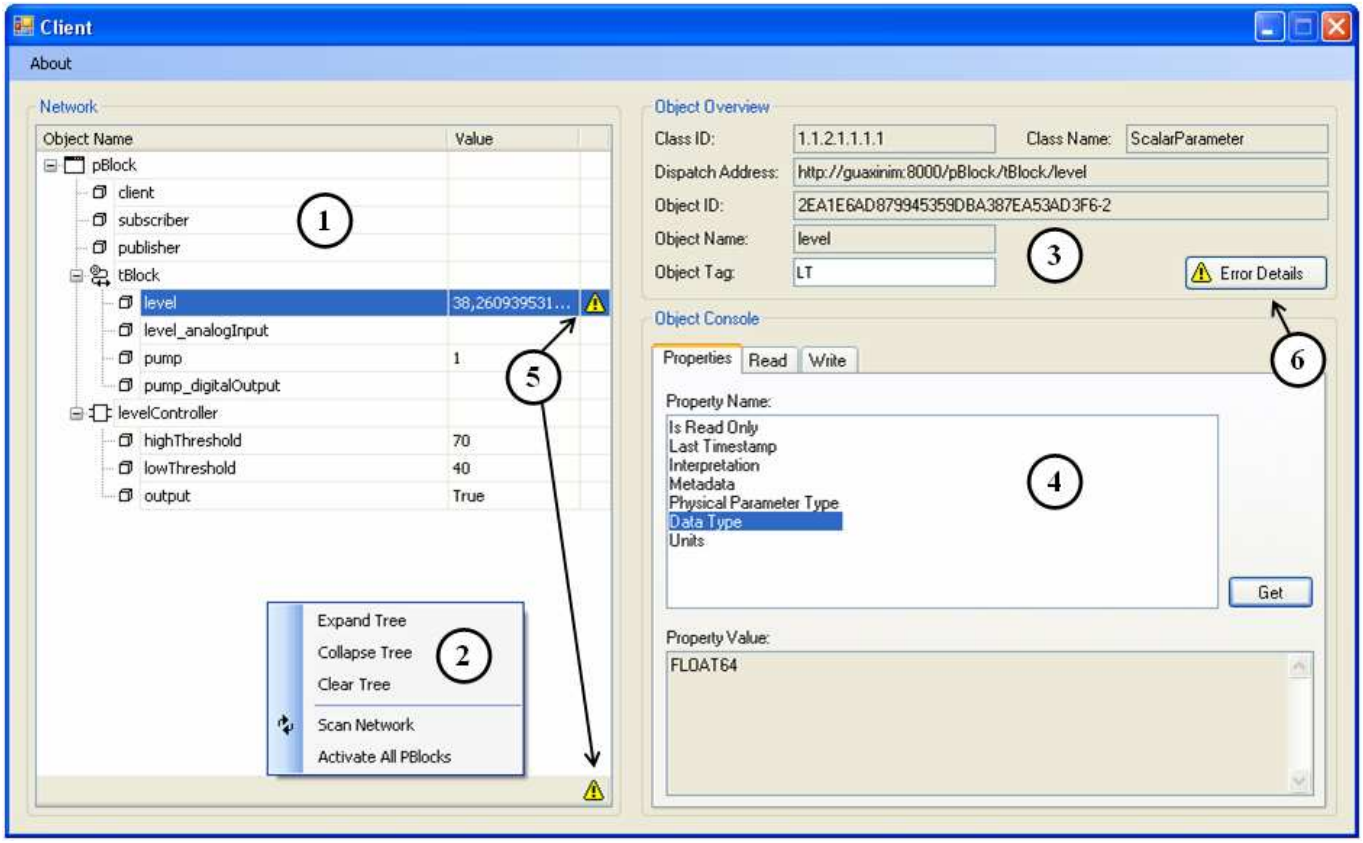

Fig 6. Client application.

The client application was developed using Visual Basic .NET 2008 and tested in the Windows XP operating system. The front panel of the client is composed by the following main elements (represented by numbered circles in figure 6):

1. The tree list provides most of the information to the user. It displays the hierarchy of all network-visible objects, the values of all monitored Parameters and eventual error conditions.
2. Using this context menu, the user can scan the network to find all registered services and load them into the tree list.
3. When the user selects a given object on the tree list, its main properties are presented in this group box. All properties are read-only except the object tag.
4. The configuration console of the selected object is dynamically loaded in this group box. Here, the user can make remote calls to the object, not only to get/set properties but also to invoke other methods as well.
5. If a remote call fails, the error is marked on the tree list. The icon at the bottom indicates the existence of at least one pending error.
6. By clicking this button, the user receives detailed information about the error that affects the selected object. The error can be acknowledged and erased if desired.

4. Results

Tests were conducted using the apparatus illustrated in figure 7, which is composed by the following main elements (indicated by numbered arrows):

1. The installation contains two cascaded open tanks but only the bottom-most is used to store water (the other one is put out of service).
2. The water circulates in a closed circuit pumped by a 12 VDC submersible pump (model 881 from Whale).
3. In the back, an enclosure contains the level transmitter, the DAQ board (NI-USB 6008), the power supply and some auxiliary signal conditioning circuits. The level transmitter is implemented by a pressure sensor (model ASDX001D44R from Honeywell), with its output range [0.62, 2.25] V mapped in the interval [0, 100] % of tank capacity.
4. A single Personal Computer (PC) is used to host the controller and its client (in a more realistic scenario, the two applications would run on separate machines).

The controller was started with high/low threshold values of 70% and 40%, respectively. In the client, a scan was made to the network and all controller objects were successfully discovered. Using configuration

consoles, remote calls were made to interact with the controller, in particular to adjust the high threshold to 60% and the low threshold to 50%, by this order. The most relevant system variables were monitored in real-time, logged to a file and plotted in figure 8.

5. Conclusion

This paper demonstrates that is possible to combine the 1451.1 Std with Web Services in order to build an open, interoperable, cross-platform controller. The client application revealed to be a useful tool to configure and monitor the controller. Both applications worked as expected and no failures were detected during experimental tests.

On the network side, the controller implements the client/server communication model by employing native WCF facilities. It also implements the publish/subscribe communication model by using a dedicated WCF Web Service connected to a multicast P2P endpoint. On the field side, the controller works with DAQ boards and takes full advantage of the DAQmx driver, in particular the built-in correction engine and the capability of reading 1451.4-TEDS.

A last word should be given to the .NET Framework, which provided a very productive environment for developing code.

References

[1] Eugene Y. Song, Kang Lee, "Understanding IEEE 1451 – Networked Smart Transducer Interface Standard", IEEE Instrumentation & Measurement Magazine, Vol. 11, No. 2, pp. 11-17, April 2008.

[2] IEEE Std. 1451.1-1999, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model".

[3] Vítor Viegas, J. M. Dias Pereira, P. Silva Girão, "A Brief Tutorial on the IEEE 1451.1 Standard", IEEE Instrumentation & Measurement Magazine, Vol. 11, No. 2, pp. 38-46, April 2008.

[4] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju, "Web Services – Concepts, Architectures and Applications", Springer, Germany, 2004, ISBN 3540440089.

[5] Adam Freeman, Allen Jones, "Microsoft .NET, XML Web Services Step by Step", Microsoft Press, USA, 2003, ISBN 0735617201.

[6] http://www.w3schools.com/soap/

[7] http://www.w3schools.com/wsdl/

[8] www.w3.org

[9] David S. Platt, "Introducing Microsoft .NET", 3rd Edition, Microsoft Press, USA, 2003, ISBN 0735619182.

[10] David Chappell, "Understanding .NET", 2nd Edition, Addison-Wesley, USA, 2006, ISBN 0321194047.

[11] Juval Lowy, "Programming WCF Services", O'Reilly, USA, 2007, ISBN 0596526997.

[12] IEEE Std. 1451.4-2004, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Datasheet (TEDS) Formats".

[13] IEEE Std. 1451.2-1998, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Electronic Datasheet (TEDS) Formats".

[14] "Upgrading Your System for Smart TEDS", NI Tutorial, http://zone.ni.com/devzone/cda/tut/p/id/2925.

[15] "Upgrading Your System for Virtual TEDS", NI Tutorial, http://zone.ni.com/devzone/cda/tut/p/id/4470.

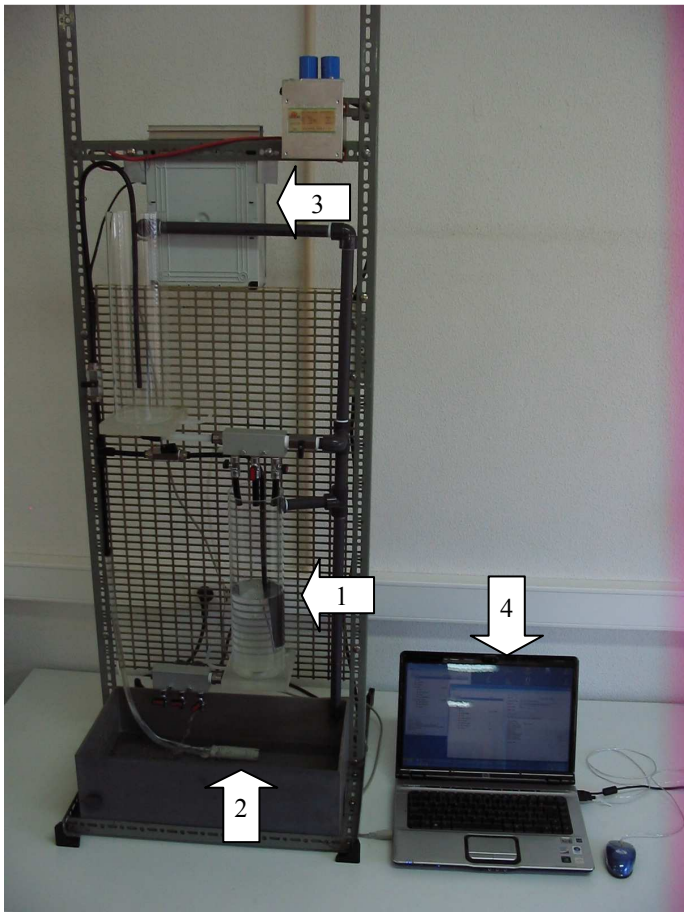

Fig 7. Experimental apparatus.

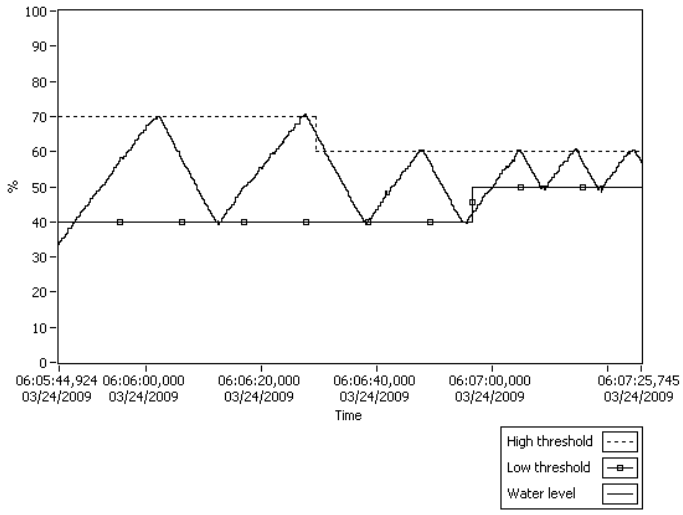

Fig 8. Experimental data.