IMTC 2006 – Instrumentation and Measurement
Technology Conference
Sorrento, ITALIA, 24-27 April 2006

IEEE 1451.1 Standard and XML Web Services: a Powerful Combination to Build Distributed Measurement and Control Systems

Vítor Viegas[1,2], J.M. Dias Pereira[1,2], P. Silva Girão[2]
[1]ESTSetúbal-LabIM, Instituto Politécnico de Setúbal, 2910-761, Setúbal, Portugal
[2]Instituto de Telecomunicações, Av. Rovisco Pais, 1, 1049-001, Lisboa, Portugal
Phone: +351-265790000, Fax: +351-265721869, Email: vviegas@est.ips.pt

Abstract – *In 2005, we presented the NCAP/XML, a prototype of NCAP (Network Capable Application Processor) that runs under the .NET Framework and makes available its functionality through a set of Web Services using XML (eXtended Markup Language). Giving continuity to this project, it is time to explain how to use the NCAP/XML to build a Distributed Measurement and Control System (DMCS) compliant with the 1451.1 Std. This paper is divided in two main parts: in the first part, we present the new software architecture of NCAP/XML (which suffered some changes since the first version), and secondly, we describe the network configuration of a Web-enabled DMCS, which includes several NCAP/XML stations, a database and a Web Server.*

Keywords – *Smart Transducer, IEEE 1451.1, NCAP, .NET Framework, XML, Web Service.*

1. INTRODUCTION

The 1451.1 Std [1] defines an object model suitable to represent any networked smart transducer. The object model must be implemented in a processor with two communication ports, one that interfaces the network and another that interfaces the transducer. The network processor, denominated NCAP, acts like a bridge between the smart transducer and the network by exporting transducer functionalities over a standardized object model and hiding implementation details. As shown in figure 1, the object model is presented as a hierarchy of classes divided in three main groups:

- **Blocks**: Block classes are the working elements of the NCAP. Three block classes are specified: 1) the NCAP Block class that provides software interfaces for supporting network communications and system configuration; 2) the Transducer Block class that provides standard software interfaces between transducers and application functions; and 3) the Function Block class that encapsulates application-specific functionality.
- **Components**: Component classes provide common application building elements such as structured information (like measurements and files), collections of related application-specific objects, and actions with state where the action takes place over a relatively long period of time.
- **Services**: Service classes support communications between objects on distinct NCAPs and system-wide synchronization.

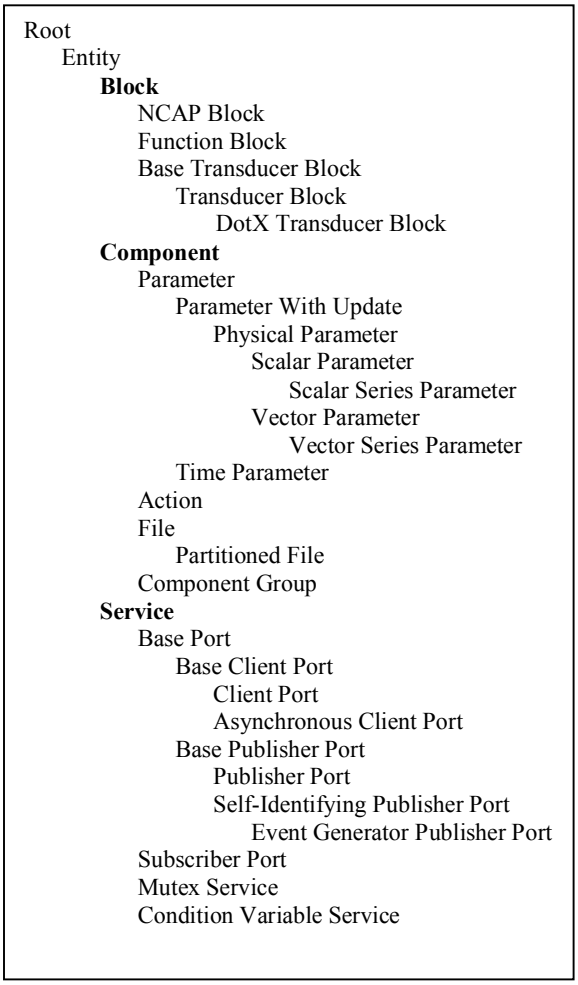

Fig. 1. IEEE 1451.1 Std class hierarchy.

In [2] we described the software implementation of an innovative NCAP prototype that runs under the .NET Framework [3]. We called this prototype NCAP/XML because it uses XML Web Services for inter-NCAP communication. Web Services [4-5] are, in simple terms, object methods made available via XML messages. These messages use a format known as SOAP (Simple Object Access Protocol) that is supported by the W3C [6] (World Wide Web Consortium), which includes all the major software companies around the world (such as Microsoft,

Sun Microsystems and IBM). This first version of NCAP/XML was composed by three software components, as depicted in figure 2:

- **NCAP Engine**: The NCAP Engine implements the IEEE 1451.1 object model, including Blocks and Components. Communication and synchronization services are implemented using .NET Framework and Web Services. The engine starts by creating the top-level NCAP Block and its child objects. The NCAP Block has its own thread that executes a script containing the user application (which implements control routines and provides "intelligence" to the system). All NCAP objects are published as singleton objects [7] in order to retain their execution state and share it with all connected clients.
- **NCAP Web Services**: This component is the Web interface of the NCAP application because it exposes NCAP objects as Web Services. Web Services are inherently stateless, but, in this case, they always return consistent results because, behind the scenes, they connect to the NCAP Engine that provides full-state objects. The communication between both components is supported by .NET Remoting [7], a distributed technology targeted for binary communications between applications over a private network or inside the same machine.
- **Web Server**: The Web Server acts like a "doorman", listening for HTTP requests on port 80 and routing them to the application that will serve them. It also handles security by preventing that unauthorized clients access Web pages and Web Services.

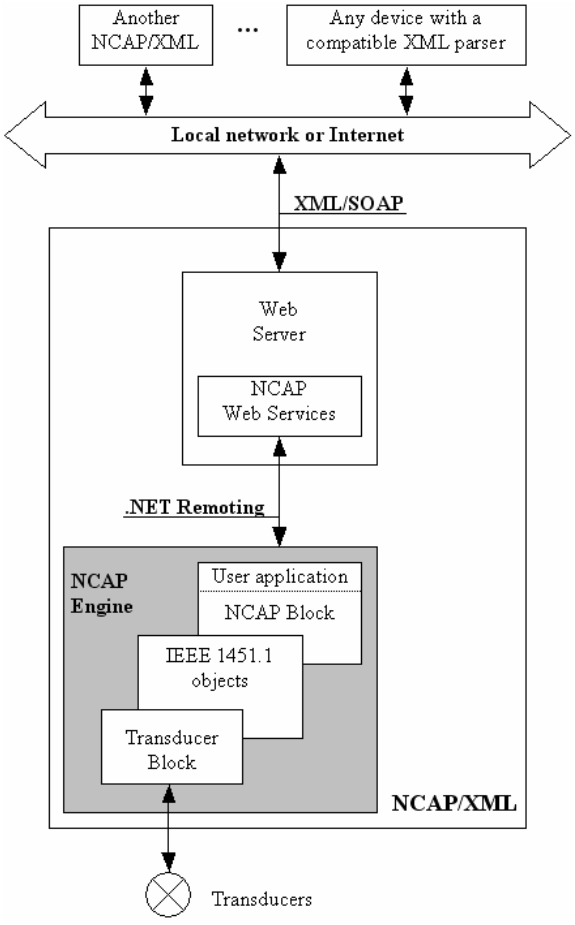

Fig. 2. First version of the NCAP/XML prototype.

NCAP/XML is completely open because any device with a compatible XML parser can communicate with it.

2. SOFTWARE UPDATE

As described in the previous section, the first version of NCAP/XML used two types of communication channels: an in-bound channel that uses .NET Remoting for fast communication between NCAP Web Services and the NCAP Engine; and an out-bound channel that uses Web Services for inter-NCAP communication. Although this works fine, we decided to simplify it.

Side by side with the binary formatter, the .NET Remoting technology offers a pre-built SOAP formatter, which has proprietary extensions that allow the serialization and deserialization of complex and platform-specific data types. Fortunately, this SOAP formatter is also compatible with W3C SOAP for basic data types and arrays of basic data types (by basic data types we mean strings, chars, booleans, integers, floats and time-stamps, among others). By this way, we can use this formatter for inter-NCAP communication, without any loss of interoperability, as long as we limit our application to use only basic data types, which is precisely the case of the 1451.1 Std restricted to block and component classes.

As shown in figure 3, the new software architecture of the NCAP/XML is a scale-down from the previous version: no NCAP Web Services are implemented and the Web Server is excluded, since all inter-NCAP communication is done through .NET Remoting using its SOAP formatter restricted to basic data types. This new arrangement has definitive advantages:

- The computational load is reduced, making easier its implementation in embedded devices with limited resources. Any device that runs the .NET Framework can host a NCAP/XML without the need of a Web Server. Such a host can be, for example, the industrial PC CX1001-0120 from the manufacturer Beckhoff [8].
- It doesn't compromise interoperability because inter-NCAP communication continues to be done with W3C SOAP compliance.
- It doesn't compromise security because access control can be built inside the NCAP Engine. If the prototype is to be used in a private network (which is the most probable

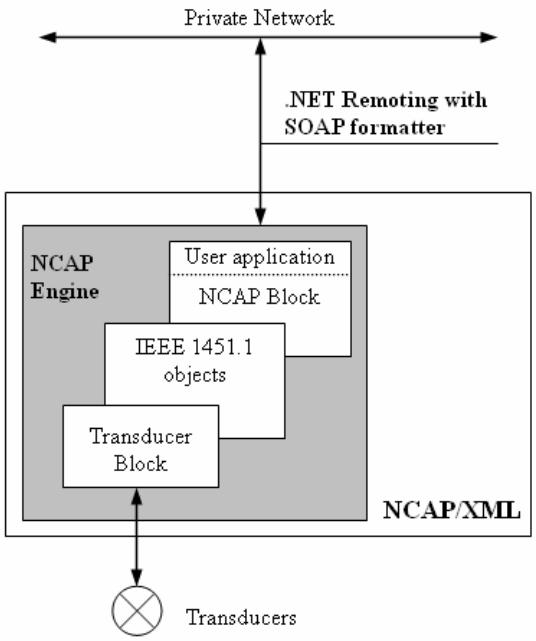

Fig. 3. NCAP/XML revised.

scenario), security can be enforced by using a firewall in the upstream.

3. NETWORK CONFIGURATION

It is possible to create a DMCS by networking several NCAPs/XML and let them "talk" each other using SOAP messages, without the need to buy and install any proprietary network drivers. This arrangement has three main advantages: 1) the communication is simplified; 2) the system is open to any device with a compatible XML parser; and 3) it makes easier the vertical integration of systems, from the field to the business, because many ERPs (Enterprise Resource Planning systems) are increasingly using Web Services.

Three main parts compose our DMCS proposal, as represented in figure 4.

- **NCAP/XML Stations**: Each station is responsible for the acquisition of field signals, their processing and the generation of actuation signals. Signals can come from local transducers or from remote stations. Examples of processing are: data logging, alarm management and control loops.
- **Information Server**: The Information Server (IS) stores all the information about the system. Static information is stored in read-only files, each file containing the settings for the corresponding NCAP/XML Station. Dynamic information is stored in an SQL database, each record representing a published object, including dynamic fields like measurement values, set-points and alarms.
- **HMI Application**: The Human Machine Interface (HMI) is a Web application that allows the operator to interact with the system. The HMI has three main purposes: 1) it allows an online configuration of the system; 2) it provides a graphic interface to visualize the state of the system using synoptic diagrams, trend graphics and alarm indicators; and 3) it allows the operator to actuate over the system gaining manual control over field variables.

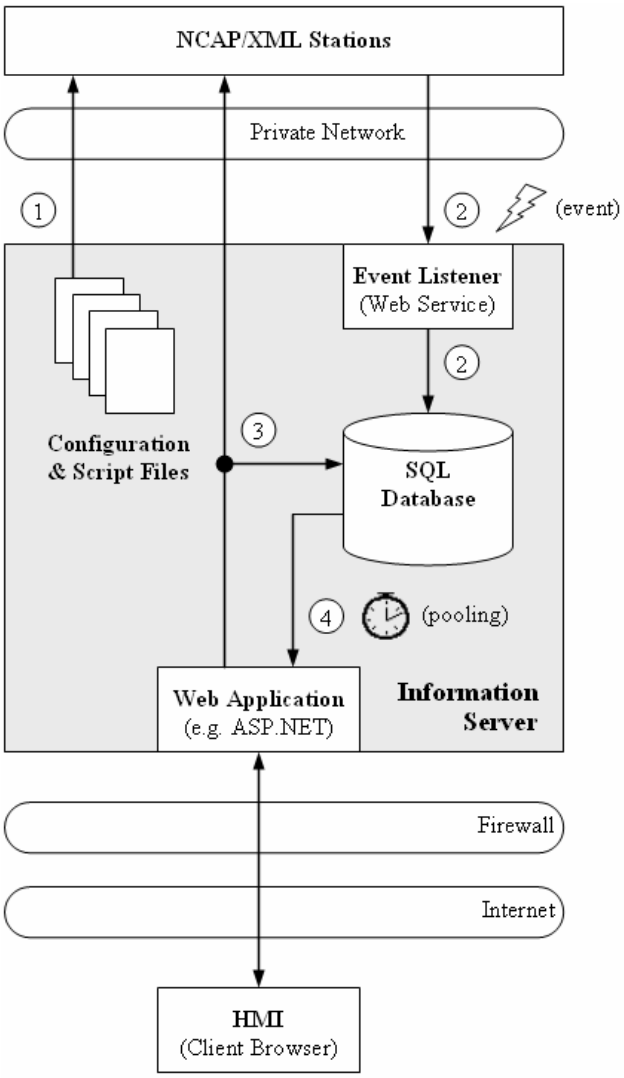

Fig. 4. DMCS proposal.

Given its complexity, the IS should be installed in a powerful workstation running a server operating system.

3.1. Configuration Files

When a NCAP/XML Station boots, it needs to receive initial information about itself and about the system where it

is integrated. This start-up information can be stored in a configuration file, resident on the IS, and downloaded by the NCAP/XML Station.

The configuration file is written in XML and includes a section dedicated to the IEEE 1451.1 object model. Inside that section, there is a text-based description about all the objects to be created and published by the NCAP/XML Station. The description includes the initial values of the object at the Entity level. As an example, let's consider an NCAPBlock object:

```
...
<!-- IEEE 1451.1 object model section -->
<IEEE1451Dot1>
   <!-- NCAPBlock instance -->
   <!-- properties filled with initial values -->
   <NCAPBlock
      ClassName="NCAPBlock"
      ClassID="1.1.1.1"
      ObjectTag="ST0.0.0"
      ObjectID=
         "272145CB-AA07-4c91-84CC-A91ABCC16F5D"
      ObjectName="NCAPBlock0"
      DispatchAddress=
         "http://10.41.0.50:6000/NCAPBlock0.soap"
      OwningBlockObjectTag="ST0.0.0"
      State="0"
   />
   ...
</IEEE1451Dot1>
...
```

Configuration files are saved in a pre-defined directory on the IS, accessible through FTP (File Transfer Protocol) and/or through a dedicated Web Service. Each NCAP/XML Station downloads its corresponding configuration file based on its name and interprets it using a XML reader. The use of XML helps interoperability and the centralized storage of all configuration files helps system maintenance.

3.2. Script Files

Script files are text files containing code to be executed by the NCAP/XML Station. This code adds "intelligence" to the station allowing it to execute control routines locally.

Script files are saved in a pre-defined directory on the IS, accessible through FTP and/or through a dedicated Web Service. Each NCAP/XML Station downloads its corresponding script file based on its name and interprets it using a script engine. The centralized storage of all script files helps system maintenance; on the other hand, scripting reduces overall performance because the code is interpreted, not compiled.

3.2. The Database

The state of the system is maintained in a database, which can be queried and updated in real time using SQL (Standard Query Language).

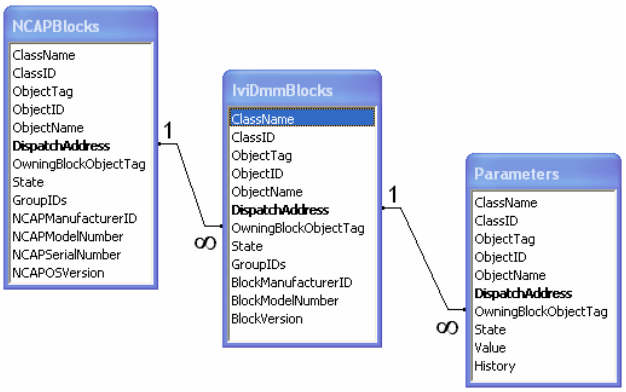

Fig. 5. Database snapshot.

The database contains one table for each non-abstract class of the IEEE 1451.1 object model. For example, a table named "Parameters" can describe all Parameter objects present in the system. The same principle is used for all other object classes.

Each table column, also known as field, describes a property of an object instance. For example, the field named "DispatchAddress" of the table named "Parameters" contains the dispatch addresses of all Parameter objects present in the system. Component tables contain two fields to store system variables: a field named "Value" that stores the instantaneous value of the variable (the last measured value of a temperature, for example), and a field named "History" to store all the past values of that variable. "History" fields are indispensable to display trend graphics.

Block tables and component tables are related through a field named "OwningBlockObjectTag". According to the 1451.1 Std, this property defines a relation parent/children between a block object (the parent) and many component objects (the children). This relation is shown in figure 5.

In the beginning, we considered the use of an UDDI Server to implement the database. UDDI [9] (Universal Description Discovery and Integration) is often called the "yellow pages" for Web Services, because it helps client applications to discover and consume Web Services by searches based on keywords, categories or interfaces. Unfortunately, UDDI revealed to be very restrictive, without margin for customization or extension, so we decided to implement an SQL database from the scratch.

3.3. Data Flow

An explanation about how the data flows in the system helps to understand how it works. The data flow is represented in figure 4 by numbered arrows:

- Arrow 1: At boot time, each NCAP/XML Station downloads its configuration file and script file. The download can be made through FTP and/or through a dedicated Web Service of the IS.

- Arrow 2: When a Parameter With Update object changes above a pre-defined dead band (5 per cent, for example), the NCAP/XML Station fires an event to the IS. The event is listened and the corresponding record in the database is updated using an SQL statement.
- Arrow 3: When the operator calls a function that changes the property value of an object, the corresponding record in the database is updated and the function is propagated to the remote object. Examples of such functions are: "SetObjectTag" (changes the tag of an Entity), "SetGroupIDs" (defines a new group of correlated blocks), "Write" (defines a new parameter value, for example, a new set-point of pressure), among others. The update of the database and the remote call are transactional: both are committed when both are successful. This way, we guarantee the consistence of data between the database and the NCAP/XML Station.
- Arrow 4: When the operator requests the instantaneous value of system variable, the corresponding record in the database is selected and the field "Value" is read. To refresh the value, this process is repeated periodically using a pooling mechanism. By this way, we guarantee the consistence of data between the database and the HMI application. The same approach applies to trend graphics, with the difference that the field "History" is the read one.

5. CONCLUSION

In this paper we describe the architecture of a complete DMCS compliant with the 1451.1 Std, which is composed by three main parts – NCAP/XML Stations, the Information Server and a Web-enabled HMI application.

In our opinion, the system accomplishes five main objectives:

- Compliance: The system behaviour was thought according to the directives of the 1451.1 Std.
- Interoperability: All data exchanged between NCAP/XML Stations and the IS is encapsulated in SOAP messages.
- Scalability: The system is highly scalable by allowing future extensions, not only in terms of devices (any device with an XML parser can be added), but also in terms of network protocols (which are abstracted by the 1451.1 object model).
- Performance: The new version of NCAP/XML is simpler, faster and targeted for embedded devices. In addition, the data flow between NCAP/XML Stations and the IS is done asynchronously, by means of events, not pooling.
- Easy to maintain: The centralized storage of all configuration files and script files in the IS helps system maintenance.

In a near future we will implement the proposed solution in an industrial PC in order to evaluate real system capabilities under field operation conditions.

REFERENCES

[1] "IEEE 1451.1 Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model", IEEE, New York – USA, April 2000

[2] Vítor Viegas, J. M. Dias Pereira, P. Silva Girão, "Using a Commercial Framework to Implement and Enhance the IEEE 1451.1 Standard", IMTC 2005 – Instrumentation and Measurement Technology Conference, Ottawa, Ontario, CANADA, May 2005

[3] David S. Platt, "Introducing Microsoft .NET", 2nd Edition, Microsoft Press, Washington – USA, 2002

[4] Mathew MacDonald, "Microsoft .NET, Distributed Applications: Integrating XML Web Services and .NET Remoting", Microsoft Press, US, 2003

[5] A. Russel Jones, "Mastering ASP.NET with C#", Sybex, US, 2002

[6] www.w3c.org

[7] David Curran, Andy Olsen, Jon Pinnock, "Visual Basic .NET, Remoting Handbook", Wrox Press, UK, 2002

[8] www.beckhoff.com

[9] www.uddi.org