Available online at www.sciencedirect.com
ScienceDirect
Nuclear Instruments and Methods in Physics Research A 586 (2008) 444–451
NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH Section A
www.elsevier.com/locate/nima

An XML-based communication protocol for accelerator distributed controls

L. Catani*

INFN-Roma Tor Vergata, Roma, Italy

Received 30 October 2007; received in revised form 29 November 2007; accepted 8 December 2007
Available online 23 December 2007

Abstract

This paper presents the development of XMLvRPC, an RPC-like communication protocol based, for this particular application, on the TCP/IP and XML (eXtensible Markup Language) tools built-in in LabVIEW. XML is used to format commands and data passed between client and server while socket interface for communication uses either TCP or UDP transmission protocols. This implementation extends the features of these general purpose libraries and incorporates solutions that might provide, with limited modifications, full compatibility with well established and more general communication protocol, i.e. XML-RPC, while preserving portability to different platforms supported by LabVIEW. The XMLvRPC suite of software has been equipped with specific tools for its deployment in distributed control systems as, for instance, a quasi-automatic configuration and registration of the distributed components and a simple plug-and-play approach to the installation of new services. Key feature is the management of large binary arrays that allow coding of large binary data set, e.g. raw images, more efficiently with respect to the standard XML coding.

PACS: 07.05.Bx; 07.05.Dz; 07.05.Hd; 07.07.Hj

Keywords: Control systems; Communication protocols; XML

1. Introduction

At INFN-LNF (Laboratori Nazionali di Frascati of INFN) development of control systems for new accelerators under construction should be based on well-established and reliable technologies for communication while re-use of instrument drivers, sub-system controls and measurement applications, or at least part of them, already developed must be guaranteed. Existing control systems are mainly based on LabVIEW that is a very common development environment for controls. Limited size (and man power) projects, especially, take advantage of its ease of use and profit from the wide set of tools and libraries either built-in or developed by the large users community to interface and control instrumentation, create analysis program and display results effectively. LabVIEW also offers a number of tools to transfer, via network, data between distributed components of the control/acquisition system: DataSocket, VI Server, VI reference, TCP/IP and UDP and also interfaces to .NET and ActiveX. The above-mentioned communication tools are powerful and well suited for many applications but, with the exception of TCP/IP and UDP, are not flexible enough to allow implementation of a real communication protocol and, moreover, most of them are either proprietary or work only between LabVIEW applications. In addition the LabVIEV Internet Toolkit includes a HTTP server and the possibility to operate the VIs (Virtual Instruments, i.e. the LabVIEW applications or subroutines) as CGIs that a client can invoke using the HTTP protocol to execute particular procedure on the server side. This is relatively flexible solution but offers low performances and limited features. When compatibility and flexibility is an issue, TCP/IP and UDP protocols are, thus, the natural choice. The LabVIEW TCP/IP and UDP libraries provide

*Tel./fax: +39 06 72594544.
E-mail address: luciano.catani@roma2.infn.it

0168-9002/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.nima.2007.12.019

the basic tools for TCP and UDP data transmission over ethernet and they are compliant with standard socket communication being the basis for many data transfer protocols.

2. The XMLvRPC protocol

2.1. XML for communication in control systems

XML (eXtensible Markup Language) is becoming a very popular way of coding data especially when interoperability and compatibility between platforms and programming languages is an issue. Client/server communication protocols based on this coding exist, among these the more interesting are SOAP [1] and XML-RPC [2]. The latter, on which I put my attention, it is basically a remote procedure call [3] that uses HTTP, or other TCP/IP and UDP protocols, as the transport and XML as the coding allowing complex, and relatively large, data structures to be transmitted, processed and returned.

Services provided by the server are called *methods* that a client can invoke by issuing `methodCall` to the server. The latter, in turn, replies sending the result in the form of `methodResponse`.

Fig. 1 shows an example of messages passed between a client and a server in the XMLvRPC protocol. They include header with declarations and `methodCall` or `methodResponse` fields. Only part of fields in the header are actually used by the XMLvRPC protocol (e.g. Host, Content-length). The others are included for future compatibility with XML-RPC.

The `methodCall` contains, enclosed with the correspondent tags, the name of the method to be executed on the server side (`methodName`) and optional parameters (`params`). The `methodResponse`, being the reply message of the server to the client, contains the name of the client application that should receive the data on the client side (`methodName`) and data (`params`).

The specification of the `methodName` in the `methodResponse` introduces the first relevant difference of XMLvRPC with respect to XML-RPC.

`methodNames` are, usually, identical in both `methodCall` and `methodResponse` that means, for a particular service, there is an application that produce the data on the server and a correspondent application on the client that display, analyze, etc. the data received. This is not necessarily true in the XMLvRPC since the protocol allows also asymmetric call/response, as it will be described later in Section 3.1.

2.2. Implementation in LabVIEW

The implementation of XML-RPC communication protocol in LabVIEW, aimed at an accelerator control system, would present two main complications. First of all the XML code (of data) generated by LabVIEW built-in tools is not compatible with the XML-RPC specifications. Secondly, standard XML coding of binary arrays, that might be transferred between components in a control system (ADC buffered readout, raw images from digital camera, etc.), results in a significant increase of data size that makes the XML coding of large binaries impracticable.

It has been mentioned that LabVIEW provides a convenient set of tools to convert its data type to XML format according to the LabVIEW XML schema. Unfortunately, the LabVIEW XML schema, *LVXMLSchema.xsd*, cannot be customized or replaced by user. It should be noted that one can write LabVIEW virtual instruments to implement XML coding/decoding based on its own, or any other, schema instead of using built-in coding/parsing tools.

Anyway, if one is only interested (as we are for this particular application) in LabVIEW-based distributed control/acquisition systems, this is not a relevant limitation. In this case, actually, it is preferable to preserve full LabVIEW compatibility to take advantage of its XML

```
POST / HTTP/1.1
User-Agent: LabVIEW  (XMLvRPC)
Host: dummy.domain.com:6430
Content-Type: text/xml
Content-length: 155

<?xml version="1.0"?>
<methodCall>
  <methodName>get_image_data</methodName>
    <params>
      <name>cam_01</name>
      <dimH>8</dimH>
      <dimV>8</dimV>
    </params>
</methodCall>
```

```
HTTP/1.1 200 OK
Connection: close
Content-length:621
<?xml version="1.0"?>
<methodResponse>
  <methodName>get_image_data</methodName>
    <params>
    <String>
      <Name>image(2)(U8)</Name>
     <Val>
      □□□□□□□□
     □□□□□□□□□
     □□□□□□
     □□□□□□
     □□□□□□□□□□□□
     □□□□□
      </Val>
    </String>
    </params>
</methodResponse>
```

Fig. 1. `methodCall` (left) and `methodResponse` (right) in the XMLvRPC protocol. Non-printable characters enclosed with ⟨Val⟩ tags are the result of pre-processing of 2D-binary array.