Application of web services to heterogeneous networks of small devices

ANDREA BAGNASCO, DAVIDE CIPOLLA, DANIELE OCCHIPINTI, ANTONIO PREZIOSI, ANNA MARINA SCAPOLLA
Department of Biophysical and Electronic Engineering
University of Genoa
Via all’Opera Pia, 11a – 16145 Genova
ITALY

Abstract: Networks of cooperating low cost nodes are becoming more and more relevant components of modern information systems. This paper analyses how to apply a common web service paradigm to both small devices and Internet network nodes in order to promote systems interoperability and scalability. REST style and SOAP protocol are matched up with the typical resources of small devices and are applied to the development of a two layer service oriented architecture that merges different classes of nodes. Small devices perform basic tasks, such as environmental sensing and conditioning, and interface with nodes that offer more complex services on the base of the collected results. A prototype has been developed to validate the proposed architecture. It is based on very cheap hardware devices and demonstrates that a REST-based web service approach is possible.

Key-Words: web service, service oriented architecture, embedded device, microcontroller, SOAP, REST.

1 Introduction
Networks of cooperating, low-cost nodes are spreading fast. Enterprises, distributed on a geographical scale, need to control processes that are locally managed by autonomous networks of devices. These networks are part of the whole enterprise information system. In order to ease the communications among network-connected nodes, some proprietary standards have been proposed and developed. These include DCOM [1], CORBA [2], and OPC [3]. Despite being robust and widely tested, these technologies cannot fit all kinds of applications, and show some limits related to the architectural complexity of the underlying models and the required computational load. The implementation of these technologies on embedded nodes isn’t always possible [4,5]. Moreover, these technologies often use proprietary protocols and formats to transmit and to represent data: this requires the adoption, or the development, of ad-hoc software components for each node in the system. The use of custom TCP ports can cause problems in crossing firewalls or proxies inside the enterprise environment. The use of closed standards definitively affects the maintainability of the system: when the chosen technology is discarded by the promoting company, the system rapidly becomes outdated and often not compatible with next generation devices.

The need of developing standard and open communication mechanisms allowing overcoming these issues is strongly felt. Such mechanisms should promote interoperability, both inside the local networks of small devices from different vendors, and towards external entities. The development and the management should be simplified, and, at the same time, scalability and lifetime should be preserved.

The recent web service technology is a promising candidate to achieve these goals [6]. However, the implementation and the use of web services on limited-resources devices are not trivial. This paper proposes a solution based on a two-layer architecture, where a lower layer provides basic services to access heterogeneous hardware resources, and an upper layer aggregates and exposes the system functionalities as services on the Internet. Data exchange takes advantage of web service technology, on both layers.

The paper is structured as follows. The next section analyzes the feasibility of porting web services on small devices. The third section proposes a novel architecture to manage local networks of small devices as elements of distributed information systems, thanks to a common service oriented approach. The last section describes the prototype that has been developed to validate the proposed architecture. Finally, the conclusions are drawn.
2 Web services for embedded systems

Web services are devised to transform the document-oriented web into a service-oriented system where machines are able to access live data sources. This approach is characterized by the use of standard Internet protocols for inter-process communication: to the contrary of proprietary solutions, it offers platform-independence, and allows going through firewalls or other communication restrictions.

The most typical and widely used implementation of web services is based on the SOAP protocol [7]. SOAP is an open, standard protocol developed by the W3C and supported by the main IT players (Sun, Microsoft, HP, IBM, Oracle, BEA). SOAP adds a further application level to the TCP/IP stack, so that SOAP messages can be transported by any Internet application protocol (HTTP, SMTP, FTP, etc.).

The adoption of SOAP on embedded devices is limited by the verbosity of the communication and the overhead necessary to parse messages. Nevertheless, some attempts to port SOAP on embedded devices exist in literature. Worth mentioning are g-SOAP [8], eSOAP [9,10], kSOAP [11]. The ASP.NET family includes a web service framework for embedded devices, but being based on Windows CE, it can be considered heavily platform-dependent. Another interesting solution has been proposed by Thinkware [12]. It should be noted, however, that all the presented solutions are still demanding in term of resources. For example, kSOAP requires the presence of a Java Virtual Machine on the embedded system, while the Thinkware solution runs on the VxWorks operating system.

The heaviness and complexity coming from the usage of SOAP are experienced not only in case of embedded devices, but in computer networks environments too. A consequence is the diffusion of simpler and lighter web service based solutions, designed according to REST criteria [13]. The acronym “REST” stands for “REpresentational State Transfer”, an architectural style for the design of large-scale distributed software, inspired on a subset of World Wide Web functionality. REST can’t be considered a real standard, because it does not refer explicitly to technologies, protocols or formats; in fact, it states the roles for the entities making up a distributed system and the guidelines to manage the communications among them. REST uses stateless client-server architecture, based upon a minimum set of universally defined remote methods for complete resource management. The key abstraction of REST is the concept of resource: resources are the actual information, exchanged in the system under different representations. The representation of a resource consists of data and metadata, which describe data nature and format. It is possible to access the representation of a resource by means of an identifier, which addresses the resource regardless of its representation. In a so-called RESTful system, the client accesses a server-hosted resource using its identifier and calling the required handling methods, which return a representation suited to the nature of data, to the type of request, to its interpretation abilities.

The style of REST seems to be especially suited for implementing light-weight web services. The entities providing and consuming services can find in the Web infrastructure all the functionalities required to get and exchange information: namely the HTTP standard and the URI (Uniform Resource Identification) mechanism to the REST approach operates a resource modelling of the services provided by the server, thus creating a model based on a mapping between services/functionalities and resources that can be read and written by the clients. In order to make the services completely accessible by generic software entities, the server returns representations of the data in XML format. The RESTful implementation of a service, compared to the corresponding SOAP protocol solution, has an important difference: the design and implementation of an additional application layer is not required, because the HTTP protocol already provides the primitives for service access. URIs, natively supported by HTTP, provide an unique, uniform and universal addressing mechanism; at the same time, the system components are tied by loose coupling, granting a high degree of scalability. The architecture of a RESTful web service is simple because, at most, it involves the extension of the client-side and server-side functionalities for the management of the XML responses.

Currently, as discussed in [14] and [15], in the IT world there is a big debate on which approach, between SOAP and REST, has to be preferred for the implementation of web services. The question is still open, and the choice of the best approach for a specific system depends on several factors. The advantages coming from the usage of REST can be summarized in three words: simplicity, lightness and scalability. These are key aspects in software development, not only targeted to embedded devices.

Some important companies provide RESTful implementations of their web services: this is the case of eBay, Yahoo!, and Amazon. It is worth noting that Amazon also publishes the same
services through SOAP, but 85% of the clients access them using the RESTful implementation. The application of the REST style to small devices is reported in [16,17].

3 The architecture

In this section REST and SOAP are applied to the development of a service oriented architecture that merges different classes of devices and their functionalities. On one hand, there are small devices that perform basic tasks, such as environmental sensing and conditioning, on the other hand there are processes that need such information but do not know directly the small devices.

The resulting system can be characterized by the following assumptions:

- the data connection among the different elements of the system is not oriented to high performance data transfer, but to occasional transmissions of few bytes;
- no real-time constraints are associated to data transfer;
- the small devices are linked together in a local network context;
- the communication with external processes (Internet) is handled by a high-capability node.

These conditions fit many real-world scenarios. The resulting architecture features two layers, characterized by different complexities:

- on the upper layer, there is a Full Functionality Node (FFN), characterized by high availability of resource;
- on the lower layer, there are a number of Reduced Functionality Nodes (RFNs), characterized by scarce resources.

This configuration allows to move and centralize some functionality that require many resources and that don’t need to be supported by each node, but by the network as a whole. In particular, it is possible to centralize: access policies, support for multiple users with concurrent access to the resources, connection security (e.g. via encryption) and firewalling. Moreover, the FFN can also manage system diagnostic and be in charge of data logging/data repository. Collecting all these functionalities in the upper layer of the architecture reduces the low-level required resources and allows the usage of low-cost nodes.

The web service paradigm is common to both layers: the lower layer provides services for a single client, the FFN, while the upper layer behaves both as a service consumer towards the RFNs and as a service provider towards external applications. The services provided by the RFNs are not just encapsulated and forwarded by the FFN; they can also be aggregated and organized in special workflows, which are sequential, conditional invocations targeted at the execution of higher level operations involving more than one node.
According to the functionality and the computational capabilities of the nodes, FFN and RFNs, two different web service implementation strategies have been adopted.

Assuming that the RFNs are heterogeneous small devices, which provide basic services, they should run simple control software, possibly not specific to a particular hardware. In this case it is convenient a light and loosely coupled web service implementation based on the REST style. The resources are the functional parameters of the hardware devices controlled by the RFN. In case the RFN hosts an HTTP server, each request of access to a resource is handled via HTTP methods and can affect one resource at the time, identified through the URI mechanism.

The node replies to the requests returning the state of a resource or an error message, formatted as an XML document. In addition, the node is able to list its resources by publishing a WSDL document defined by a schema suited to the REST-based architecture in use. On the assumption that the configuration of the hardware devices managed by each node does not change, the WSDL document can be saved in a static way when the node starts its work. Consequently, it is possible to refrain from implementing a resource-demanding XML wrapper, and the firmware requires little more than the functionalities provided by a common web server.

The services provided by RFNs are available only inside the boundaries of the private network they belong to together with the FFN, which in this context plays the role of the only consumer. On the other side, the FFN provides a high level interface to external applications, hiding the details of RFNs. In this case the usage of SOAP-compliant web services is appropriate. These web services publish remote methods that can be invoked to access and modify the parameters of the system. In this context the FFN acts as a service provider, while the role of service consumers is played by external applications. The communication is carried on by means of SOAP messages. The FFN publishes a standard, SOAP-compliant WSDL document to list and describe the services. They may be high level services, designed ad hoc upon the specific system, and internally implemented as sequences of invocations of the basic services provided by the RFNs. This approach leads to an abstraction of the system. It is presented as a whole, regardless of the details of the single nodes. Obviously, it is possible to use the FFN as a simple gateway towards the internal network of small devices, by publishing the RFN basic services.

This two-layer architecture has been tested by developing the prototype that is presented in the next section.

4 The prototype

The prototype consists of a local network of RFNs connected to a FFN that interfaces with external applications. The goal is to publish simple access services to the devices embedded in the RFNs on the FFN.

RFNs have been implemented on PICDEM.net boards, provided by Microchip Technology [18]. Each board is equipped with a 8-bit PIC microcontroller, a 32Kbyte memory module, an Ethernet network interface and some simple hardware devices (LEDs, LCD display, potentiometers, serial port, buttons). A TCP/IP stack and a web server for PIC microcontroller are provided by the vendor, together with the source code. This code has represented the base for the development of the REST web services. The web server has been extended in order to make it compliant with the REST paradigm. In particular, the URI parser has been enhanced to allow resource identification and parameters passing. The dynamic generation and transmission of XML files enable
the node to answer to the service consumer requests. The code that interfaces web service resources with hardware devices has been carefully designed. This brought to the design of a very general framework, which is independent from the hardware devices mounted on the nodes used for the prototype. As far as the C language allows, a set of data structures and functions have been developed and these are integrated with tools for the automatic generation of boiler-plate code (function headers, data structure filling, etc.) based on description files of the hardware. These features improve the reusability of source code. The implementation of the web services has been kept as independent as possible from the rest of the firmware, so that porting such code on a different platform should require a small effort. Requirements for transporting the RFN code to a new board are the availability of a TCP/IP stack, and the possibility of programming the firmware using an ANSI C compliant language. In the demo nodes the TCP/IP stack and the HTTP server take 9.5Kbytes of program memory and 759 bytes of data memory. The implementation of the web service, including access to all the on-board devices (mapped as resources), adds about 2.5Kbytes of program memory and 31 bytes of data memory. FFN and RFNs are connected by an Ethernet LAN. For the implementation of the FFN a standard PC has been used. It is, equipped with two Ethernet adapters: one towards the Internet gateway, the other towards the private LAN of RFNs. The SOAP web service running on the FFN, has been developed using the C++ implementation of the AXIS framework.

5 Conclusions
Typical scenarios for the usage of the proposed architecture are industrial automation, domotics, or any application where cost is the key factor, and integration of devices and services is needed. The hardware that we have considered is very cheap and has very limited capabilities. Nevertheless, it demonstrates that a REST-based web service approach is possible. The proposed architecture allows setting up service-oriented applications that span small devices as well as e-services on the Internet.

References:
[12] www.thinkwarecorp.org
[18] www.microchip.com