An XML Web Services mobile interaction solution for X-10 Powerline Networking

Tarrini L., Miori V.

Abstract. In this article, we will outline the implementation of X10Services, an infrastructure for presenting services to mobile computing devices such as PDA or Smartphones. The infrastructure uses the X-10 powerline networking as communication protocol and the Web Services paradigm as the core technology for exposing household appliance across the network. Therefore X10Services is a software layer that is able to translate the requests of diverse mobile clients versus the X10 devices. In this application, we illustrate the main features about the X10 protocol communication focusing on Microsoft Framework .NET 2.0 for the interaction among the PC and serial IO port.

Keywords: Powerline Networking, X-10, C#, .NET, Home Automation, XML Web Services.

1. Introduction

With the inevitable arrival of broadband access, the demand for sending digital voice, video and Internet data within the home will increase continuously. In the context of networking environment, “no new wires” is the term applied to phoneline and powerline technologies that use existing wiring systems to distribute high-speed data and video throughout the house. The advantage of using existing infrastructure as data transmission medium is that every building or house are already equipped with several outlets and phone wires.
For many years, the powerline have been used for home automation. In fact the most important types of home automation applications include controlling lights, ventilators, security systems and temperature levels within the home. The main protocols for home automation and control are mainly based on one of these major powerline technologies namely, CEBus [1], LonWorks [2], Konnex [3], and X10 [4].
The X10 protocol is able to satisfy these entire scenario in a simple way enabling X10-compatible devices, which are electrical directly plugged into a wall outlets, to communicate each other, even though the lack of many services, as plug and play or ACK of prompt, represents a strong limitation. As a result, reliability remains as a major issue in X10 powerline networking.
In this article, we propose a new approach for domotics platform based on open XML Web Services standards. Our system is called X10Services and the goal is to design an infrastructure for providing services in standard way to diverse mobile clients, such as cell phones like Smart Phone 2003 and Personal Digital Assistants (PDAs), into home environment.

Then our system uses the Web Services as software layer to control X10 household appliance and the X10 protocol as powerline communications. In particular we have chosen to focus on one specific X10 product, the CM11A System [5]. It is an extremely inexpensive and easy to use product but it can give the most basic understanding of home automation potential.
The CM11A is connected to computer via serial cable on a COM port, allowing the computer to send X10 commands. The Microsoft Framework .NET 2.0 (beta version) provides System.IO.Ports namespace with the ability to access the serial ports on a computer, and to communicate with serial I/O devices. The final product will be a reusable class library wrapped into an ASP.NET Web Services for controlling devices within the house.
This paper is organized as follows. In the section 2 we are going to look at entire system architecture of the X10 communication describing the mechanisms of interaction among transmitter and receiver. Section 3 explains a typical scenario in which we develop a Home Automation System based on Web Services, called X10Services and how mobile phone can interact with the system. Section 4 talks about potential future work that can be done.

2. X-10 Protocol

2.1 Introduction

This section deal with the popular X-10 (also referred to as X10) communication protocol. Building control technology as provided by X10 is specialised and simple way of automated process control, dedicated to the needs of home applications. The X10 technology was first introduced in 1978 by X-10 Inc., for the Sears Home Control System and the Radio Shack.
X10 communications protocol allows to control lights, household appliances, thermostats and other devices in the house that are connected to X10 modules that receive signals over home's AC power lines. A device is assigned an "id" by input from the user, for example, setting a switch to a certain housecode and numbercode. Each unit has an address composed by a letter from A to P called the House Code, and a number from 1 to 16 called the Device Code. Using the combination of those two codes, it becomes possible to address (to plug) up to 256 devices on one power-supply network.

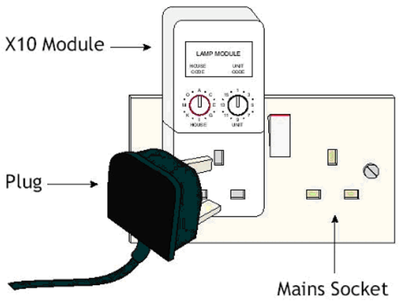


Figure 1 Typical X10 Module

But these devices are susceptible to damage by voltage spikes. The signal attenuation and line noises generated by external sources can transiently interfere with the typical X10 communications. As a result, reliability remains a major issue in X10 powerline networking [6].

Complex faults are unavoidable in X10 networking, and the faults manifest themselves as anomalous behaviour on the powerline in terms of illegal sequences of X10 commands.

A typical system consists of multiple X10 modules attached to the powerline communication medium. A CM11A PC interface attached to the PC via serial port can be used to generate and receive X10 commands on the powerline. Other modules act as receivers and they control the household appliance attached to them. A two-way receiver can also respond to commands. An RF transceiver module converts RF signals from remote controllers into X10 commands. A sample powerline network is provided below in figure 2.

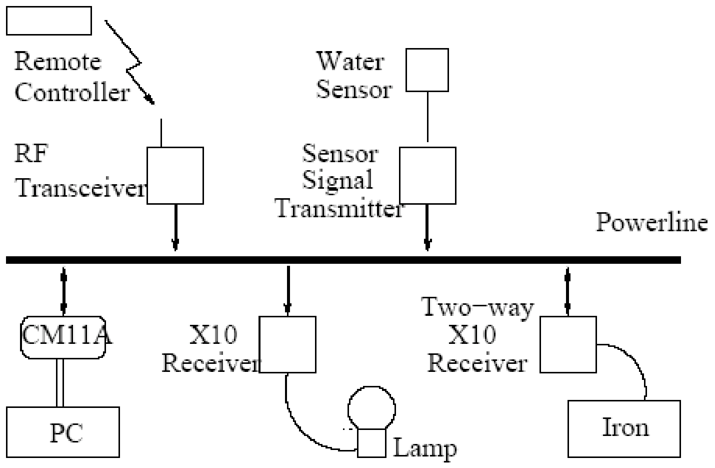


Figure 2. Typical Powerline Network

X10 originally started out as unidirectional only; however capability for bi-directional communication has also been added to it. Anyway the vast majority of X10 communication remains unidirectional only. There are two general classifications of devices: controller and receiver modules. The controller sends signals over existing AC wiring to receiver modules. The modules are adapters connected to outlets and controlling simple devices. X10 transmission rate is limited to only 60 bps, which makes it unsuitable for carrying Internet traffic around the house.

The next section illustrate in details the method of transmission and reception.

1.2 The technology

The most problematic part of this technology is the method in which the binary data are transmitted from one device (the transmitter) to another device (the receiver). Every device has an integral "zero crossing" detector that serve to synchronization.

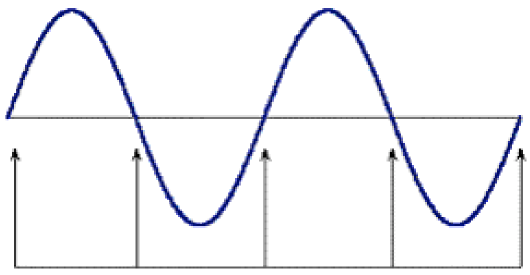


Figure 3. Point of synchronization

So the transmitter know when to send data and the receivers know when to look for data. Receivers "*sense*" the 0° point and then look for signal in a small window time, only 0.6 milli-seconds in duration. Since these devices don't have any direct wiring between them, it is necessary to devise a way of sending data over the existing electrical wiring. Then these signals involve short RF bursts which represent digital information. The goal should be to transmit as close as possible to the zero crossing point, and I have to do it within 200 micro-seconds of the zero crossing point.

The actual binary data is transmitted by sending 1ms bursts of 120kHz just past the zero crossing of the 60Hz power. It is also obvious that complementary bit pairs are necessary. Therefore, a binary "1" is defined as the presence of a pulse, immediately followed by the absence of a pulse. A binary "0" is defined as the absence of a pulse, immediately followed by the presence of a pulse.

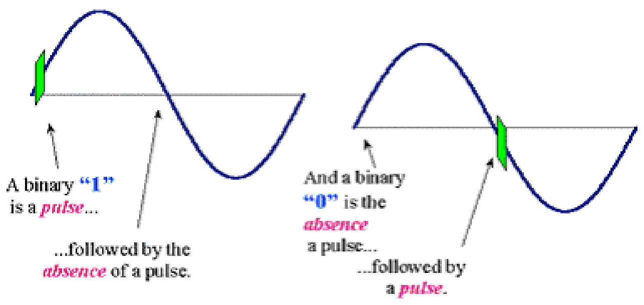

A binary "1" is a pulse...
...followed by the absence of a pulse.
And a binary "0" is the absence a pulse...
...followed by a pulse.

Figure 4. Binary data transmission

While the transmitted pulses are to be a full 1ms in duration, the receivers are designed to open a receive window of only 0.6ms. In order to provide a predictable start point, every data frame would always begin with at least 6 leading clear zero crossings, then a **start code** of "pulse", "pulse", "pulse", "absence of a pulse" (or 1110). The start code uses a different format. It's always the same two cycles sequence.

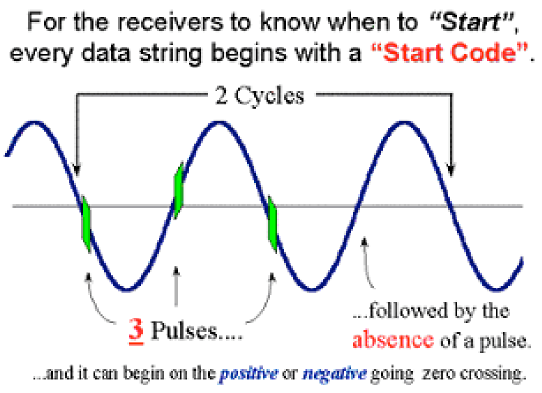
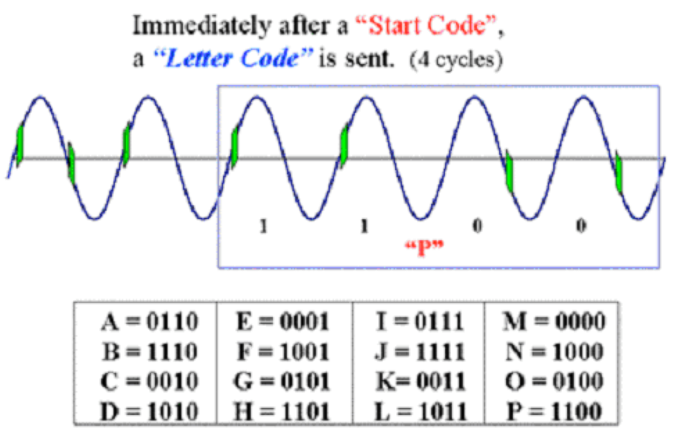

For the receivers to know when to "Start", every data string begins with a "Start Code".
2 Cycles
3 Pulses....
...followed by the absence of a pulse.
...and it can begin on the positive or negative going zero crossing.

Figure 5. Start Code

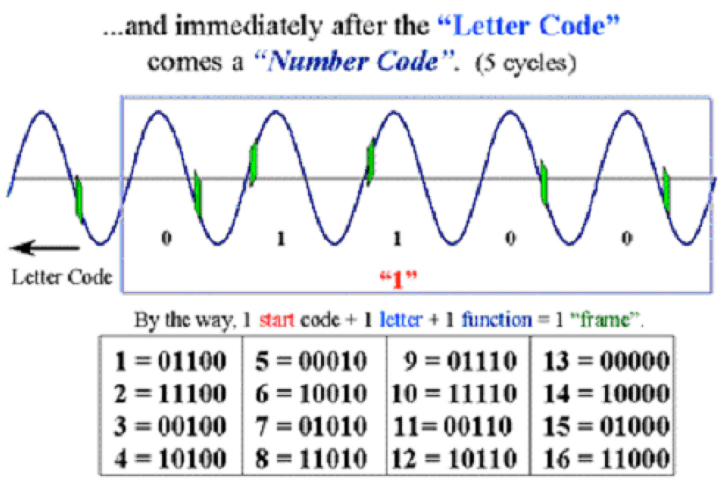
Once the **Start Code** has been transmitted, the first nibble (half a byte), is immediately followed by a "letter" code designations. It is also decided to randomly rearrange the patterns so that the "A", "B", "C" codes, etc., did not fall in the predicable binary pattern. It is easy to see that in reality, the "M" code is the first in the binary progression.


Immediately after a "Start Code", a "*Letter Code*" is sent. (4 cycles)

A = 0110	E = 0001	I = 0111	M = 0000
B = 1110	F = 1001	J = 1111	N = 1000
C = 0010	G = 0101	K= 0011	O = 0100
D = 1010	H = 1101	L = 1011	P = 1100

Figure 6. Letter Code

In one contiguous bit stream, the second nibble provides the second half of the address. The last bit appears to be a part of the "number" code but in reality it is a function bit. Whenever this function bit is setted to a "0", it designates that the preceding nibble as a number code and therefore it is a part of the address.


...and immediately after the "Letter Code" comes a "*Number Code*". (5 cycles)

By the way, 1 start code + 1 letter + 1 function = 1 "frame".

1 = 01100	5 = 00010	9 = 01110	13 = 00000
2 = 11100	6 = 10010	10 = 11110	14 = 10000
3 = 00100	7 = 01010	11= 00110	15 = 01000
4 = 10100	8 = 11010	12 = 10110	16 = 11000

Figure 7. Number Code

A complete code transmission encompasses eleven cycles of the power line. The first two cycles represent the Start Code, the next four cycles represent the House Code and the last five cycles represent the Number Code. For purposes of redundancy, reliability and to accommodate line repeaters, the X-10 protocol calls for every frame of data to be transmitted twice.

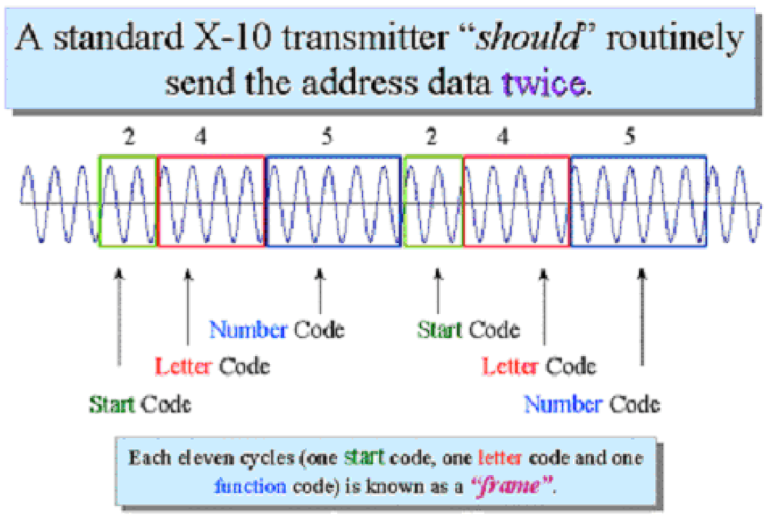

A standard X-10 transmitter "should" routinely send the address data twice.
2
4
5
2
4
5
Number Code
Start Code
Letter Code
Letter Code
Start Code
Number Code
Each eleven cycles (one start code, one letter code and one function code) is known as a "frame".

Figure 8. Standard X10 transmission

Whenever the data changes from one address to another address, from an address to a command, or from one command to another command, the data frames must be separated by at least 6 clear zero crossings (or "000000"). Actually, of course, the sequence of six "zero's" resets the shift registers.

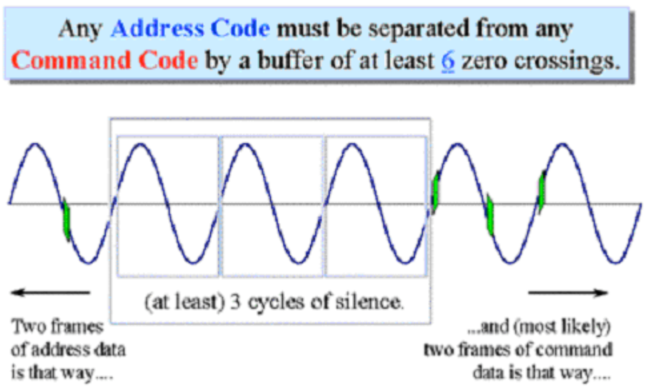
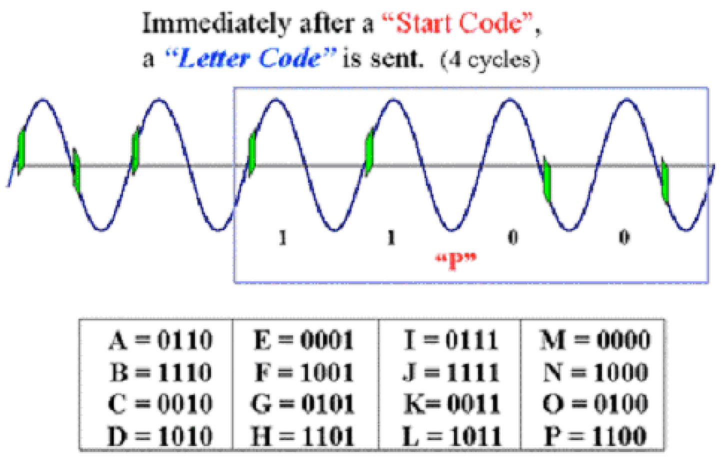

Any Address Code must be separated from any Command Code by a buffer of at least 6 zero crossings.
(at least) 3 cycles of silence.
Two frames of address data is that way....
...and (most likely) two frames of command data is that way....

Figure 9. Silente in the wire

Once a receiver has processed its address data, it is ready to receive a command. As before, all data frames must begin with a start code. The next nibbles gives the letter code and the next again is the command. Since the last bit is the function bit (b_f = 0 = address number, b_f = 1 = command) all the commands end with a binary 1.

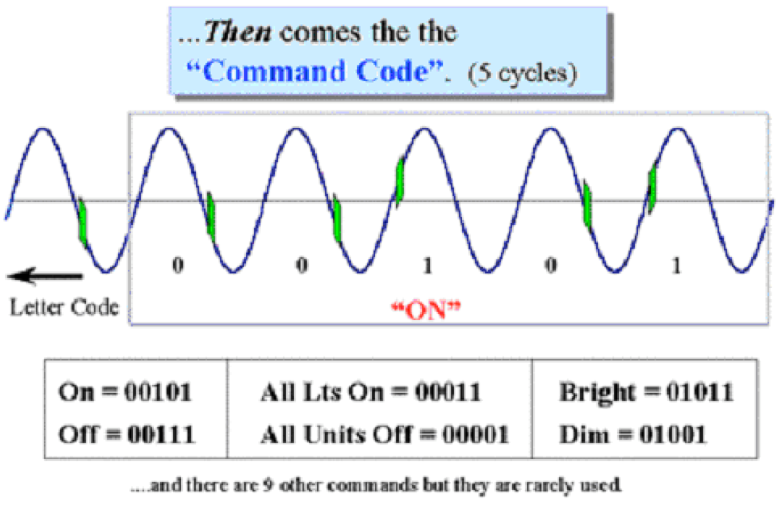


Immediately after a "Start Code", a "*Letter Code*" is sent. (4 cycles)

A = 0110	E = 0001	I = 0111	M = 0000
B = 1110	F = 1001	J = 1111	N = 1000
C = 0010	G = 0101	K= 0011	O = 0100
D = 1010	H = 1101	L = 1011	P = 1100

Figure 10. Letter Code

This diagram shows the six most often used commands only. As before, all X10 protocol transmitters send their data frames twice.



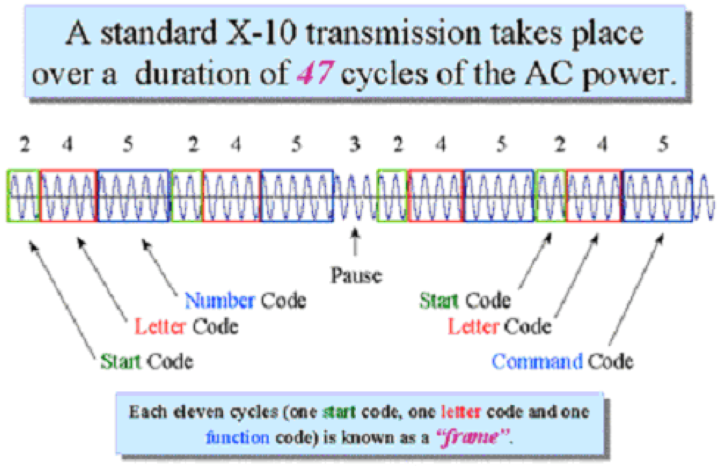
...*Then* comes the the "Command Code". (5 cycles)

On = 00101	All Lts On = 00011	Bright = 01011
Off = 00111	All Units Off = 00001	Dim = 01001

....and there are 9 other commands but they are rarely used.

Figure 11. Command Code

For example if the CM11A sends this information, A-On, would take 47 cycles of the 60Hz sine wave. That would equate to 0.7833 seconds, or in practical terms, just under 1 second.



A standard X-10 transmission takes place over a duration of *47* cycles of the AC power.

Each eleven cycles (one start code, one letter code and one function code) is known as a "*frame*".

Figure 12. Complete transmission

Of course, some commands take less time. When sending an "All-Lights-On" command, for example, no address needs to be sent. Therefore the entire two-frame sequence takes only one third of a second (actually, 0.3666 seconds, but who's quibbling). If your receivers react on the first frame, it could take mere two tenths of a second (0.1833 seconds).

Up to this time, all the diagrams have shown only one pulse but that is not entirely correct. In fact, our electrical power is a 3 phases system and so all X-10 compatible transmitters "should" send out 3 pulses.

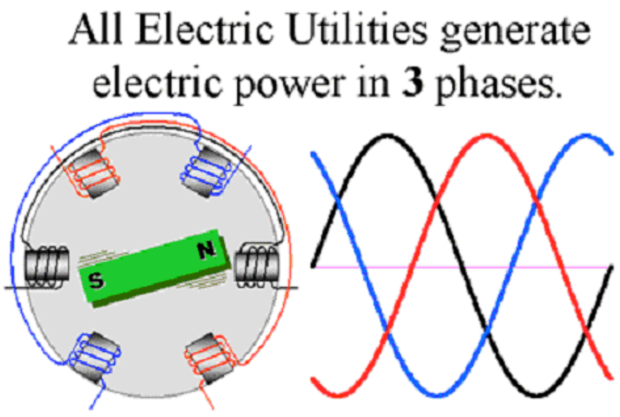


Figure 13. Three phases in the powerline

The transmitter releases a burst at its own zero crossing, then sends it again 60° later; the second burst coincides with the zero crossing of the third phase. Then another burst is sent 120° from the first, which corresponds with the zero crossing of the second phase.

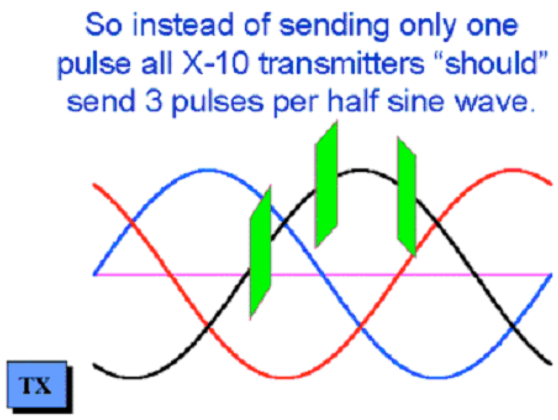

So instead of sending only one
pulse all X-10 transmitters "should"
send 3 pulses per half sine wave.
TX

Figure 14. X10 transmission in powerline

2. System Architecture

2.1 The scenario

The goal is to develop a home automation system, called ***X10Services***, based on Web Services paradigm that will allow users to control in standard way electric household appliances, e.g. turning on-off lights, into home environment. Then the idea is opening the home middleware versus Web Services Architecture in the sense that any devices, sensor, appliance communicate by this stack. Any computational element taking part into home environment must thus be able to expose its services as Web Services and the user interacts by using the mobile computing devices, such as smartphones or PDA.

First of all, turning phone into a SOAP client might have some performance costs related to slow data speeds and processing both HTTP commands and XML. However development environments such as J2ME or .NET Compact Framework represent robust platforms for developing advanced mobile clients. In this application we experiment the Compact as client for X10Services.

All the application are tested with Whidbey environment, last version of Visual Studio. Examples of possible scenarios are energy management, heating control, and lighting control.

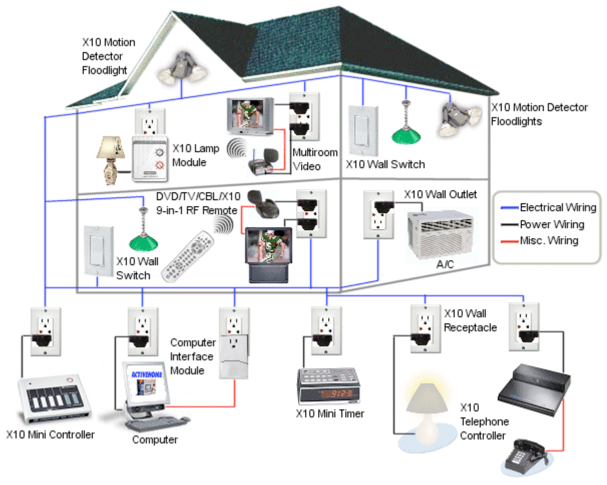


Figure 15. X10 Scenario

The typical X10 consists of transmitter module and several receiver modules. The transmitter module used in this project is the CM11A. This device is connected to serial cable on a COM port and allows the computer to send and receive commands.

Figure 16. CM11A Transmitter

The two standard types of receiver modules are:

- Appliance Module, AM12, which allows on-off control for an appliance.
- Lamp Module, LM12, which allows dimming of a lamp as well as on-off control.

Figure 17. Lamp Module

Figure 18 illustrates the proposed architecture model.

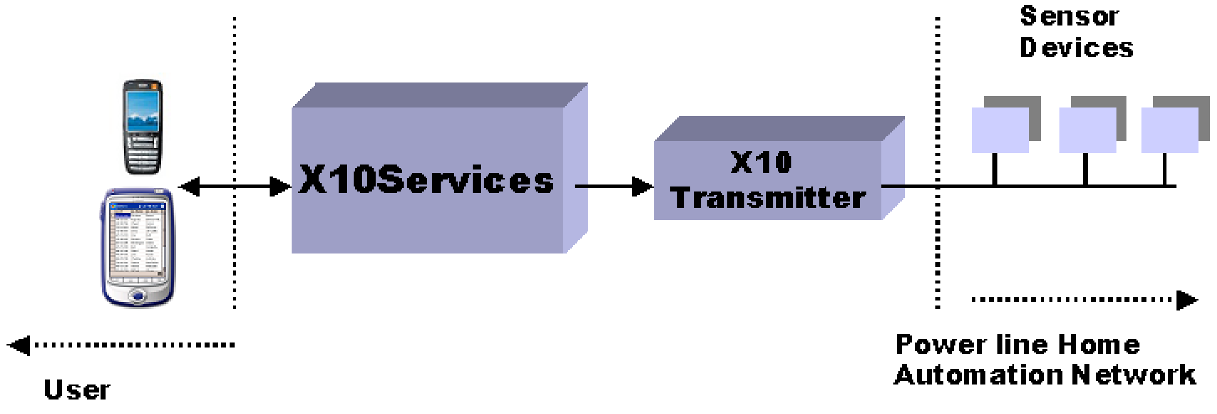


Figure 18. Architecture Model

In the next section we describe the single block of the architecture.

2.2 X10 Communication

The lower level of the architecture is the X10 transmitter, the CM11A module, which is connected to the computer via a COM port. It manages the communication in powerline with the sensor devices plugged into wall outlets. There are many implementations for controlling a CM11A and surely, the most important is the open source Java library unit written by Jesse Peterson [7]. This library is written in Java programming language and it provides a high level interface to the CM11A module.

Instead our solution is based on Microsoft Framework Class Library (FCL) .NET 2.0, beta version, developing a new library, based on C# language that it is available and downloadable [8].

Nowadays, Microsoft .NET 1.0 provides reasonably comprehensive coverage of the functionality of the underlying Win32 API. However, RS232 serial communications is an area that is conspicuously absent from this library. The only way of coding serial communications applications in .NET is to import the outdated and somewhat limited MSComm ActiceX control. Then the solution is to utilize Platform Invocation Services (P/Invoke) to interact with the Win32 API directly and enables managed code in the Common Language Runtime to make calls into unmanaged DLLs.

Instead the version 2.0 of .NET provides features for the serial communication by the *System.IO.Ports* namespace and in particular the *SerialPort* class. This class provides a framework for synchronous and event-driven I/O, access to pin and break states, and access to serial driver properties. It can be used to wrap Stream objects, allowing the serial port to be accessed by classes that use streams. That is, SerialPort class represents a serial port resource.

The methods of the SerialPort class that we use in this application are:

- *Open()*: opens a new serial port connection.
- *Write(byte[], int, int)*: writes an array of bytes to the communications resource file.
- *ReadByte()*: reads one byte from the port connection.

- *ReadTimeouts*: get or sets the number of milliseconds before a timeout occurs when a read operation does not finish.
- *Closes()*: closes the port connection.

The figure shows the serial parameters for the communications between the interface and PC.

```
comm = new SerialPort(comPort, 4800, Parity.None,
                      8, StopBits.One);
```

Figure 19. Constructor for the CM11A Class

Then the CM11A class must manage all the communication among the PC and CM11A device.

```
public void Send(byte[] buffer, int count)
{

        comm.DiscardInBuffer();
        comm.Write(buffer, 0, count);
        returnValue = comm.ReadByte();
}
```

Figure 20. CM11A write on the serial port

The X10 library used in this project is a file called CM11A.dll.

2.2 X10 Web Services

XML Web Services are a set of protocols to enable communication between independent software modules that offer their functionality in the form of services. The services are self-contained, modular applications, that can be described, published, located, and invoked over a network [9].

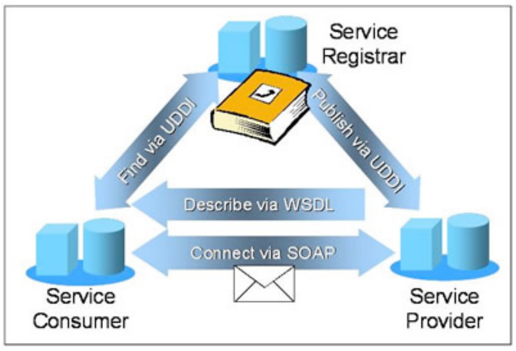


Figure 21. Publish, Find, and Bind

Then we propose to expose the services offered by the X10 network following the Web Services paradigm in the sense that any device, sensor or appliance implements the standard web service stack. In this away the mechanisms for the communication, service description, and discovery will be based on standardized protocols. In this article, we consider only the baseline of Web service specification: XML, SOAP, WSDL, and UDDI, figure 22.

This application is a part of a larger research program [10] of CNR-ISTI Domotics Lab, aimed at realizing a framework, service oriented, for the integration and interoperability among domotics middleware.

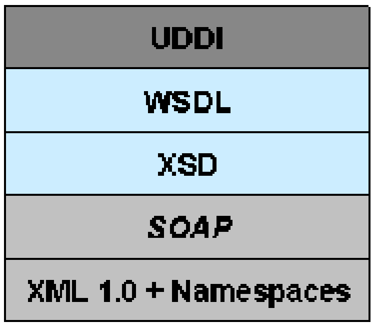


Figure 22. Web Services Stack

To avoid getting bogged down in the logic associated with CM11A control, we have provided an implementation in the CM11A dynamic link library, which encapsulates all the logic necessary to support the Web Services application. In fact the separation of the control logic from the XML Web Service implementation allow us to remain focused on the XML Web Service aspects of development. This approach is also representative of many real business scenario in which the logic is developed separately. In general, we think that separating the underlying logic from Web Services interface that is exposed to the outside world is good practice.

The *ASP.NET WebMethod* framework make it possible to built Web Services that communicate over HTTP. The calls for Web Services are always directed to URLs with .asmx extension. The IIS server intercepts these calls and passes all the related packets on the registered ASP.NET ISAPI filter (aspnet_isapi.dll). The figure shows the ASP.NET architecture to process Web Services.

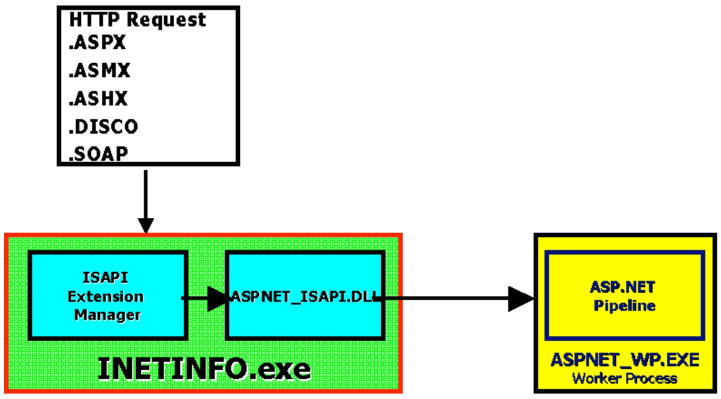


Figure 23. ASP.NET architecture

CM11AServices, as the name implies, is used to provide X10 services. The ***Control*** method has public accessibility and is annotated with *WebMethod* attribute, making it accessible to Web Services client. When invoked, the method instantiates a ***CM11AController*** object, implemented in the CM11A.dll library, and call the methods to control the household appliances.

The client do not need to know the details of the platform or language used to implement the service. The only requirement is that the client be able to formulate requests and process responses using the correct protocol and message structure.

2.3 Smartphones and Personal Digital Assistant Simulators

Microsoft .NET Compact Framework is a subset of .NET Framework that is designed to run on resource-constrained, providing support for managed code and XML Web Services. The

Compact is available for devices running Windows CE operative system and it is fully integrated into Visual Studio .NET. Then the X10 client, ***X10Mobile*** has been developed by this tools.
Although the structure and the syntax of SOAP are fundamentally simple, trying to manually encode complex data as SOAP messages, can result difficult. The use of proxies takes the complexity of SOAP processing out of our application. Proxy classes remove the need for the programmer to manipulate SOAP message directly. Then to invoke the Web Service's functionality, a client application simply calls the proxy class method.

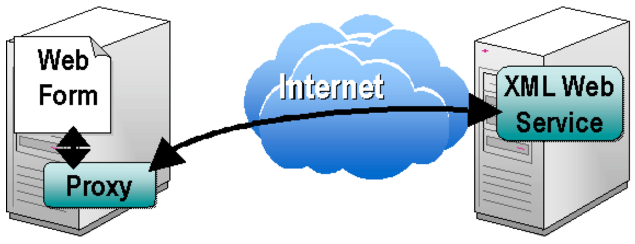


Figure 24. Proxy class in Web Services

To consume our Web Service, it needs to add a Web Reference to CM11AService and Visual Studio generates automatically a proxy class with methods that serve as proxies for each exposed method of the Web Service. In figure we show the .

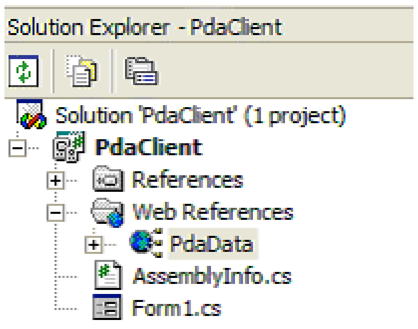


Figure 25. Web Reference in Visual Studio

In this way the X10Mobile receives a WSDL document that describes the service's operations in terms of messages and how they are bound to various protocols and endpoints. Thus the X10Mobile has enough information to interact with the X10 Web Services. In this application, the mobile devices knows where the WSDL lives and a discovery mechanism isn't needed.

3. Conclusions and Future Work

We have described the architecture of the X10Services, which could be used to control appliances within the home environment. This work also demonstrated how it is possible to expose the services provided by the X10 protocol and the interaction with mobile computing devices that support the Compact .NET Framework.
A good amount of work remains to be done on the interface of the application. The future idea is to extrapolate in XML the description of devices and automatically generate the proper interface. This is a difficult problem that can be addressed in the future. In addition to those problem, the appliance's remote controls have only unidirectional communication to the appliances wherein it can only change the current state of the appliance to another state, but

there is no feedback from the appliance back to its control. Then it needs to resolve this problem providing a two-way communication with the appliance.

References

[1] CEBus. http://www.cebus.org/

[2] LonWorks. http://www.echelon.com

[3] Konnex. http://www.konnex.org/

[4] X10. http://www.x10.org

[5] CM11A. ftp://ftp.x10.com/pub/manuals/

[6] Anish, A., Rajesh, J., Yi-King, W. "*Model-based Fault Detection in Powerline Networking*". Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'02), IEEE, 2002.

[7] Jesse, P. http://www.jpeterson.com/

[8] C#_CM11A. http://hats.isti.cnr.it/download

[9] Web Services Overview. http://www-106.ibm.com/developerworks/webservices/library/w-ovr/

[10] HATS Project. http://hats.isti.cnr.it