Automatic Configuration and Service Discovery for Networked Smart Devices

Günter Obiltschnig
Applied Informatics Software Engineering GmbH

St. Peter 33
9184 St. Jakob im Rosental
Austria
Tel: +43 4253 32596 Fax: +43 4253 32096
guenter.obiltschnig@appinf.com
www.appinf.com

Electronica Embedded Conference Munich 2006

Abstract. A growing number of embedded devices are connected to TCP/IP networks. Configuring the network parameters of a device is a tedious task, as many devices do not have an appropriate user interface to do this comfortably, or the number of devices is simply too large to configure every single one manually. Furthermore, devices, to accomplish their job, more and more depend upon the services provided by other devices. Finding these services with no or minimal configuration is an important feature. This paper discusses the fundamental issues in automatic configuration and service discovery – address assignment, name resolution, service discovery, service description, service invocation and service presentation. Then, four popular technologies that solve some or all of these issues are presented – Zero Configuration Networking, Universal Plug and Play, Jini and JXTA.

1 Introduction

A growing number of embedded devices are connected to TCP/IP networks. And many of these devices are no longer passive network nodes, waiting for someone else to control them. They are full-blown network citizens, actively communicating with their peers and often relying on the network services provided by other devices to do their job. For this to work, all these devices must be configured properly. Configuring the network parameters of a device is a tedious task, as many devices do not have an appropriate user interface to do this comfortably. This is especially an issue with consumer devices, where the user might not even have the necessary technical knowledge to configure such a device. And as the number of devices in a network grows, configuring each device separately is no longer practical. There from comes the need for the automatic configuration of network devices and the automatic discovery of network services. In recent years, the industry has come up with a variety of different technologies and specifications to address this.

2 Usage Scenarios

The potential usage scenarios for automatic device configuration and service discovery are manifold and range from consumer electronics to industrial automation hardware. Following, a few examples are presented.

2.1 Network Printers

Most of the network-capable printers available on the market today support Zero Configuration Networking (Zeroconf). This allows them to automatically announce their presence to the network. Given that Zeroconf-capable software is installed on a PC (e.g., Apple's Bonjour software), printers supporting Zeroconf are discovered automatically and ready for printing as soon as they are plugged into the network and powered on. No further configuration of IP addresses, etc. is required.

2.2 Home Audio/Video Devices

Some of the never A/V receivers come with an Ethernet interface for connecting them to a home network. This makes it possible to stream music or video from a PC or media jukebox directly to the receiver, and even to control these media sources using the receiver's remote control. PCs and media servers must be able to advertise their media collections to the receiver, and the receiver must be able to control the playback on the media sources. Especially in such a setting it cannot be expected from the user that he configures IP addresses or sets up DHCP or DNS servers. The de-facto standard technology that makes all this work is Universal Plug and Play (UPnP).

2.3 Home, Building and Industrial Automation

Ethernet and TCP/IP are becoming increasingly popular in automation systems. These systems are often characterized by a large number of small devices talking to each other, or to one or more central servers. To make the life easier for maintenance technicians adding new devices to such an environment, new devices (e.g., sensors and actuators) should be automatically recognized by the system as soon as they are connected and powered up.

3 Fundamental Issues

Generally, there are three fundamental issues that must be dealt with for the automatic discovery of network devices and services, and three more issues that must be dealt with for the actual invocation or use of the discovered services.

3.1 Address Assignment

Every device must be assigned a unique network address. In case of TCP/IP networks this can be done with the help of a DHCP (Dynamic Host Configuration Protocol[1]) server. Should, however, no DHCP server be available (for example, in typical home user networks), another way of assigning an IP addresses must be found. Apart from manual configuration, which is often undesirable, a method called APIPA (Automatic Private IP Addressing, or AutoIP for short) is used. In this case, a device's TCP/IP stack randomly chooses an IP address in the private (link-local) IP address range 169.254.0.0 to 169.254.255.255. To prevent two or more

[1] The DHCP protocol is described in RFC 2131 – Dynamic Host Configuration Protocol.

devices from accidentally selecting the same address, each device must probe, using the ARP (Address Resolution Protocol[2]), whether the chosen address is available.[3]

3.2 Name Resolution

Whenever a user (or device) wants to access another device over the network, he usually wants to refer to the device by its name, not by its IP address. In a typical TCP/IP network, a DNS (Domain Name System[4]) server is used to map domain names to IP addresses. Again, if no DNS server is available, such as in a typical home network, another way of resolving names to IP addresses is required. Multicast DNS (mDNS), as used by Zeroconf, is such an alternative technology.

3.3 Service Discovery

A user or device must be able to find a service provided by one or more devices in the network. The important part here is that the user (or device) usually does not care which device implements the service, as long as the service with specific properties is available and accessible. A typical example for service discovery is: *I need to print a document in color. Give me an IP address and port number where I can send the print job to, using the Internet Printing Protocol (IPP), so that the document will be printed in color.*

What all technologies for service discovery have in common is, that they make use of IP multicasting. IP multicasting uses addresses in a special address range (224.0.0.0 to 239.255.255.255). A packet (typically, a UDP packet) sent to a multicast address is received by all hosts listening to that specific address.

Service discovery is implemented in the following way:

- An application or device that needs a certain service sends a request describing the required properties of the service to a specific multicast address (and port number).
- Other applications or devices on the same network receive the request, and if they provide the requested service themselves (or know another device that implements the service), respond with a message describing where the service can be found.
- The application or device searching for the service collects all responses, and from the responses chooses the one service provider it is going to use.

In addition, devices that join or leave a network can send announcements to other devices describing the availability of the services they provide.

3.4 Service Description

Once a certain service has been discovered, it may be necessary to obtain more information about the service. For example, if a service consists of multiple operations, it is necessary to find out exactly what operations are supported, and what arguments must be passed to them. This is the purpose of service description.

In case only well-defined service protocols are used (e.g., HTTP, Internet Printing, or media streaming), service description is not necessary, because the only information needed to access the service is the network address (IP address and port number, or URI), and this

[2] The ARP protocol is described in RFC 826 – Ethernet Address Resolution Protocol.
[3] The exact algorithm is described in RFC 3927 – Dynamic Configuration of IPv4 Link-Local Addresses.
[4] The DNS is described in various protocols, the most important being RFC 1034 – Domain Names – Concepts and Facilities, and RFC 1035 – Domain Names - Implementation and Specification.

information can be obtained by service discovery. In other cases, the information obtained via service discovery is insufficient to successfully access the service. In such a case, service discovery only returns the address of a network resource that provides detailed information about the capabilities of the service, and how to access them.

3.5 Service Invocation

After an application has obtained enough information about the services it wants to access – either via service discovery alone, or together with service description, the application will access or invoke them. This is usually beyond the scope of most service discovery technologies, and the domain of specialized technologies and protocols. Examples for such technologies are HTTP (HyperText Transfer Protocol) [1], SOAP (Simple Object Access Protocol) [2], Java RMI (Remote Method Invocation) [3], CORBA (Common Object Request Broker Architecture) [4], or media streaming protocols such as RTSP (Real Time Streaming Protocol) [5].

3.6 Service Presentation

Some technologies (UPnP and Jini) provide facilities to present a device specific user interface to the user, via another device (e.g., a central management workstation, the user's PC or TV set). Such a user interface can be used to allow the user to directly interact with a device, in order to configure it, or to use some of its functions not available otherwise.

This requires that the user interface is implemented in a device independent way. One way to do this is to implement the user interface in HTML (HyperText Markup Language), served by an embedded web server built into the device. Another way is to implement the user interface in Java, so that it can be run everywhere a Java Virtual Machine is available. The user interface code than talks to the device over a network connection, using a possibly proprietary protocol.

4 Technologies

Figure 1 gives an overview of the technologies presented in this paper, and the range of features they support.

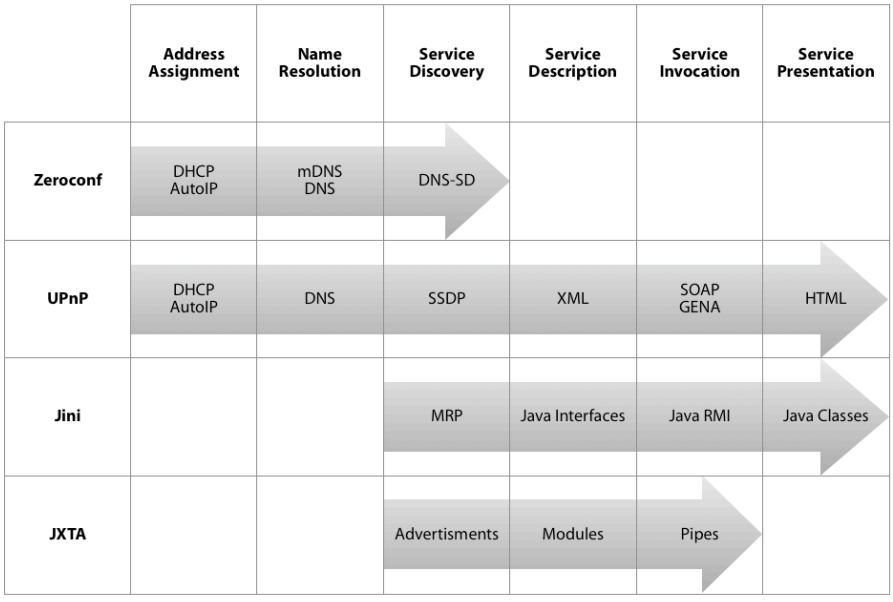

Figure 1: Service discovery technologies and their features

4.1 Zero Configuration Networking

The first technology presented is Zero Configuration Networking (Zeroconf) [6, 7], a technology developed by Apple and promoted under the trademark name Bonjour. Zeroconf is built upon technologies known as multicast DNS (mDNS) [8] and DNS Service Discovery (DNS-SD) [9], which themselves are based on the proven DNS protocol. Its most popular uses are in network printers, network cameras and for sharing music using Apple's iTunes music jukebox software. Open source implementations are available from Apple and others for all important platforms.

Zeroconf uses DHCP and AutoIP for address assignment if no DHCP server is available. A special variant of the well-known DNS protocol, multicast DNS, is used for name resolution in case no DNS server is available. In mDNS, a name resolution query is sent not directly to a DNS server, as with traditional DNS, but to a multicast address. All network devices supporting Zeroconf and listening to this multicast address respond to name resolution queries, if they have the requested information.

Finally, for service discovery, an extension of the DNS protocol called DNS Service Discovery is used. DNS-SD can be used both with multicast DNS (the usual way), or with traditional unicast queries to a DNS server. With the additional support for DNS Update and DNS Long Lived Queries, two extensions of the DNS protocol, Zeroconf can be used to announce services across the global internet. In this case, an ordinary DNS server is used to make the service information available. Periodic automatic updates of the DNS server's database ensure that its service information is up to date.

A major advantage of Zeroconf is that it is based on proven technology. Also, it can be implemented in a very resource efficient way, making it a good choice for resource constrained embedded devices.

Beside heavy use by Apple in many of its applications for Mac OS X and Windows (iTunes), Zeroconf is popular among manufacturers of printers and network cameras. In the home entertainment sector, it has lost the race to UPnP.

4.2 Universal Plug and Play

Universal Plug and Play [10], or UPnP for short, is a technology, backed by the UPnP Forum initiated by Microsoft and others. UPnP tries to achieve the same goals as Zeroconf, albeit on a different technological basis, the Simple Service Discovery Protocol (SSDP) [11]. Also, UPnP goes much further than Zeroconf in that it covers everything from address assignment to service presentation.

For address assignment and name resolution, UPnP uses the same technologies as Zeroconf: DHCP and AutoIP. For service discovery, UPnP uses a special variant of the HTTP protocol, using UDP instead of TCP [12]. This approach has some minor problems. First, UDP packets, in order to avoid packet fragmentation, must not exceed 512 bytes. Since traditional HTTP is rather verbose, and a request's metadata could easily alone eat up all the available space, UPnP uses extra short names for its HTTP header fields. This makes the protocol look somewhat awkward. Second parsing a HTTP message requires far more effort and thus code than parsing, for example, a DNS-SD message. In addition with the heavy use of XML in UPnP, this leads to higher memory and CPU performance requirements than Zeroconf. As the practice shows, however, this does not seem to be a real problem.

UPnP also supports service description, service invocation and service presentation. For service description, XML documents describing a device or service (thus called UPnP

device/service descriptions) can be downloaded from an embedded web server on the device. For service invocation, SOAP (the Simple Object Access Protocol) is used over HTTP. UPnP also supports event notifications that a device can use to notify interested parties of changes to its internal state. GENA (General Event Notification Architecture) [13], a simple HTTP-based protocol, is used for that purpose.

Finally, for service presentation, HTML pages served by the device's embedded web server are used.

UPnP is very often used for networked home audio and video devices and home networking equipment. Various implementations, both open source and commercial, are available.

4.3 Jini

Jini [14] is based on the Java platform and tries to accomplish similar goals as UPnP. Since Jini is tied heavily to the Java platform, its use is restricted to devices powerful enough to host a Java Virtual Machine. It supports service discovery, description, invocation and presentation, all implemented using standard Java facilities. Jini has been developed by Sun Microsystems.

For service discovery, Jini requires a central registry. Jini devices can find the registry via UDP multicasts using a protocol proprietary to Jini, which avoids manual configuration of Jini devices. However, the requirement of a central registry makes Jini harder to deploy than other technologies.

For service description and service invocation, Jini relies on Java RMI (Remote Method Invocation). A special feature of Jini is that is uses downloadable code to access services. An application or device wanting to access a Jini services downloads a piece of Java code from the device. This code then manages the communication between the two devices.

Downloadable code is also used for service presentation. In this case the code downloaded from the device implements a graphical user interface to access or configure the device.

The most important drawback of Jini is its reliance on the Java platform. It has not seen much use in embedded devices. However, it's sometimes being used in distributed applications built on the Java platform.

4.4 JXTA

Like Jini, JXTA [15, 16] has been developed by Sun Microsystems. However, unlike Jini, JXTA is not tied to the Java platform. Open source implementations of Jini in Java and C are available from Sun Microsystems. An implementation in C++ is currently in development by Applied Informatics.

JXTA is much more than a platform for service discovery. It is a complete middleware infrastructure for building peer-to-peer systems that scales from local area networks to the global internet.

JXTA relies heavily on XML. This alone makes it a bit of a heavyweight technology. It is basically independent of the underlying transport protocol – plain UDP and TCP, as well as HTTP and others are supported.

Central to JXTA is the concept of peers and peer groups. Peers are single hosts taking part in a JXTA network. In order to communicate with other peers, peers form peer groups.

For service discovery and service description, JXTA uses so-called advertisements. Advertisements are XML documents describing the availability and features of peers, peer

groups and their services. Services are implemented as so-called modules. Peers communicate with other peers over pipes, which are abstract communication channels. Different flavors of pipes are available, implemented on top of various communication protocols, such as UDP (multicast), TCP or HTTP.

JXTA is the most complex of the technologies introduced in this paper. As the only technology presented here, it has provisions for security. However, its complexity makes it inappropriate for resource-constrained systems. Nevertheless, its flexibility, extensibility, security, and scalability makes it a good choice for heavyweight applications. An example is the U.S. Army's Future Combat System (FCS) [17], which uses JXTA to provide the peer-to-peer discovery service.

5 Conclusion

Automatic configuration and service discovery for embedded devices is a solved problem. Mature technologies are available that cover all the typical requirements. The choice for a specific technology depends on many factors, such as scalability, implementation size, interoperability with devices from other manufacturers, and so on.

References

[1] R. Fielding, J. Gettys, et al., Hypertext Transfer Protocol – HTTP/1.1, RFC, 2616, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[2] N. Mitra, SOAP Version 1.2 Part 0: Primer, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[3] Sun Microsystems, Inc., Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi/

[4] Object Management Group, CORBA® Basics, http://www.omg.org/gettingstarted/corbafaq.htm

[5] Wikipedia, Real Time Streaming Protocol, http://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol

[6] S. Cheshire, Zero Configuration Networking, http://www.zeroconf.org/

[7] S. Cheshire, D. Steinberg, Zero Configuration Networking – The Definitive Guide, O'Reilly & Associates, 2005

[8] S. Cheshire, Multicast DNS, http://www.multicastdns.org/

[9] S. Cheshire, DNS Service Discovery (DNS-SD), http://www.dns-sd.org/

[10] The UPnP Forum, UPnP Device Architecture, http://www.upnp.org/download/UPnPDA10_20000613.htm

[11] Y. Goland, P. Leach, et al., Simple Service Discovery Protocol/1.0, Internet Draft, http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt

[12] Y. Goland, J. Schlimmer, Multicast and Unicast UDP HTTP Messages, Internet Draft, http://www.upnp.org/download/draft-goland-http-udp-04.txt

[13] J. Cohen, S. Aggarwal, Y. Goland, General Event Notification Architecture Base: Client to Arbiter, Internet Draft, http://www.upnp.org/download/draft-cohen-gena-client-01.txt

[14] Sun Microsystems, Inc., Introduction to Jini, http://www.jini.org/wiki/Category:Introduction_to_Jini

[15] Sun Microsystems, Inc., The JXTA Project Website, http://www.jxta.org/

[16] S. Oaks, B. Traversat, L. Gong, JXTA in a Nutshell, O'Reilly & Associates, 2002

[17] Sun Microsystems, Inc., Boeing Selects Sun's JXTA Technology For U.S. Army Future Combat Systems, Press Release, http://www.sun.com/smi/Press/sunflash/2005-06/sunflash.20050613.1.xml