A Multi-protocol Home Networking Implementation for HTML5

Clarke Stevens, CableLabs
August 11, 2011

Abstract

The W3C Home Networking Task Force (HNTF) was formed to consider the development of ways to provide access to resources on the home network (LAN) to HTML5 web browsers. This paper presents a low-level interface that provides the basic functionality required for multiple home networking technologies. An implementation of the API as a Java Applet that enables UPnP and Zeroconf protocols in multiple browsers will be described and demonstrated.

Architecture

The objective of this effort is to define an API to enable home networking protocols that can be implemented in all major web browsers. A direct implementation into the browser source code, however, takes time and in the case of proprietary browsers can only be done by the browser vendor. This approach was implemented in a previous version of the proposed architecture presented in the Web and TV IG meeting in Berlin, but due to the limitations described above, it was only implemented in one open-source browser. This time, in order to provide more flexibility and to immediately enable the architecture in all major browsers, we developed a user agent as a Java Applet. The applet implements a user agent with a simple, low-level API. JavaScript code to implement the particular protocol functions is built on top of the Java Applet user agent. Finally, HTML/CSS is used to build the user interface.

Applet

This has the advantage of working across different existing browsers immediately. It is downloaded with the web page, so no installation process is necessary. It can also be signed to provide some level of security or comfort for users.

API
The proposed API consists of two function calls and one or more callback functions. The discoveryControl() function call selects which protocols are to be discovered and identifies the discovery callbacks. The sendRequest() function sends commands to a service and specifies the callback to receive the response. The callback function has the same format regardless of the target protocol or service. It receives a single parameter that is a JSON object. The following is a list summarizing the API.

- discoveryControl(JSONString protocols);
- sendRequest(JSONString request, responseCallback);
- responsecallback(JSONObject response);

The free format of JSON allows for complete flexibility in specifying parameters. The protocol (e.g. “upnp,” “zeroconf,” etc.) is a required parameter for both requests and responses. The HTTP response code is required for the response callbacks. All other parameters are specific to the protocol and command.

JavaScript
The higher-level functions of the particular protocol are written in JavaScript. These JavaScript functions call the low-level communication APIs to enable discovery and communicate with home network devices. A simple application can call the Java APIs directly while more complex applications may want to provide a JavaScript library that completely implements the protocol.

HTML User Interface
HTML, JavaScript and CSS are used to implement the user interface. The user interface will typically provide a listing of the available devices and services as they are discovered. The devices and services discovered through multiple protocols can be listed together or managed separately depending on the needs of the application.

Demonstration
The demonstration implements two home networking protocols: UPnP and Zeroconf. UPnP is the basis of DLNA and is implemented in hundreds of millions of devices currently deployed. Zeroconf is the generic name of the technology used in Apple’s Bonjour products. Devices and services using either of these protocols will be discovered and presented in an HTML list. Content items will then be selected and played on different devices. The HTML user interface is simply a web page with a signed Java Applet that is downloaded from a web server on the Internet.

UPnP

The following example shows how UPnP is used in the discoveryControl() function. The discovery process is implemented in the user agent. When discoveryControl() is called with a protocol that is implemented, the discovery process begins. When a device using the specified protocol is discovered, the callback function is called. The callback function is called with a single argument that is a JSON object. It has two required parameters. The first is the “protocol” string. The second is the “status” string. All other parameters within the JSON object are particular to the specific protocol. For UPnP, the JSON object for a particular television is shown as an example below.

```
discoveryControl({“UPnP” : upnpDiscoveryCallback);

upnpDiscoveryCallback(jsonObject)
{
	//do something with the discovered service
};
```

Where:

```
jsonObject =
{
	“protocol”:”upnp”,
	“serviceType”:”urn:schemas-upnp-
org:service:ContentDirectory:1”,
	“uuid”:”00000000-0000-1010-8000-5442499C2FE3”,
	“deviceName”:”BRAVIA XBR-52LX900”,
	“response”:”information about the UPnP interface, actions,
events, etc.”,
	“status”:”found”
};
```

The information passed in the JSON object is used to keep track of the details of the discovered device. The status parameter indicates that the device is either found or lost.

Zeronconf

The implementation of discovery for the Zeroconf protocol is very similar to discovery of UPnP or any other protocol.

```
discoveryControl({“Zeroconf” : zeroconfDiscoveryCallback);

zeroconfDiscoveryCallback(jsonObject)
{
	//do something with the discovered service
};

Where:

jsonObject =
{
        “protocol”:”zeroconf”,
        “serviceType”:”DAAP”,
        “deviceName”:” Firefly Media Server on Windows_daap”,
        “status”:”found”
};
```

The discoveryProtocol() function and associated callbacks provide a service and protocol independent means to discover services that has sufficient flexibility to implement any protocols and services of interest. It allows for discovery to be event driven by the arrival and disappearance of devices.

Security

Security is an important issue when granting access to local resources. It is particularly important if access is grated to a program downloaded from the Internet. A badly-behaved application can get access to your content and control the networked devices in your home in annoying or even malicious ways.

In order to combat this threat, the Java Applet user agent code is signed. The signing verifies the source of the content. This can give some comfort, but it is still possible to get a malicious application from a trusted source.

An additional security measure is to require the user to explicitly authorize access to local resources. This is simulated in our sample implementation, but more work needs to be done to provide this as a standard feature that is known to be highly resistant to attack.

What Needs to be Standardized

The implementation of this architecture depends on a user agent implementation of the proposed discovery and messaging APIs. While the APIs can be implemented in a Java Applet (as demonstrated), there is a great benefit in the standardization of discovery and messaging protocols. Standardization allows applications to rely on the functionality to be included in major browsers and to behave in the same way.

Related W3C Work

In addition to the work being done in the W3C Web and TV IG, work on a discovery API specification has proposed in the W3C DAP WG. This proposal (or a modified version of this proposal) will be submitted to the DAP WG for consideration as a discovery API.

Sample Implementation

A sample implementation of the proposed architecture is available at www.somenewreference.com. The sample application will present a list of discovered content servers on your local network (UPnP or Zeroconf) as well as UPnP rendering devices. Content from the servers can be played on the selected renderer. However, please note that not all content will play on all rendering devices. This is a limitation of the rendering device.

Discussion Items

While we have demonstrated that the described API can be used to implement multiple home networking protocols in existing browsers, there are a number of issues that should be considered:

- What level of control do users have to ensure their content is not inadvertently exposed or misused?
- Should the API include higher-level functionality particular to the individual protocols?
- What vulnerabilities are exposed by the Java/JavaScript implementation?
- Can existing technologies (e.g. XHR or sockets) be used instead of a new messaging API?

References

1. Universal Plug-n-Play Device Architecture and DCPs (ISO/IEC 29341) (www.upnp.org).
2. RFC 3927, Dynamic Configuration of IPv4 Link-Local Addresses.
3. RFC 4795 (informational only), Link-Local Multicast Name Resolution (LLMNR).
4. RFC 4627, The application/json Media Type for JavaScript Object Notation (JSON).